Sécurisation du Contrôle d'Accès dans les Bases de Données

Luc Bouganim

Sécurisation du contrôle d'accès dans les bases de données

Luc BOUGANIM

Rapport scientifique pour l'obtention de l'

HABILITATION A DIRIGER LES RECHERCHES EN INFORMATIQUE

Université de Versailles Saint-Quentin-en-Yvelines

27 janvier 2006

Rapporteurs :	**Frédéric CUPPENS**	**(Professeur, ENST Bretagne, Rennes)**
	Pierre PARADINAS	**(Professeur, CNAM, Paris)**
	Dennis SHASHA	**(Professeur, New York University, New York, USA)**
Membres :	**Claude KIRCHNER**	**(Directeur de recherche, INRIA & LORIA, Nancy)**
	Philippe PUCHERAL	**(Professeur, UVSQ, Versailles – INRIA, Rocquencourt)**
	Patrick VALDURIEZ	**(Directeur de recherche, INRIA, Rennes)**

Remerciements

Table des matières

Préambule

Ce préambule a pour objectif de situer mes activités de recherche dans le temps. En effet, la suite de ce document structure l'ensemble des travaux de recherche que j'ai mené en fonction de leur problématique scientifique et non en fonction de leur chronologie ou des projets dans lesquels ils ont été intégrés.

J'ai obtenu ma thèse de doctorat [Bou96] de l'Université de Versailles en Décembre 1996, sous la direction de Patrick Valduriez, directeur de recherche et responsable du projet RODIN à l'INRIA. Cette thèse a débuté dans l'équipe 'bases de données avancées' de Bull SA (Les Clayes sous Bois) et s'est poursuivie[1] dans l'équipe RODIN. Le thème de ma thèse était l'équilibrage de charge lors de l'exécution parallèle de requêtes relationnelles. En effet, un obstacle majeur à l'obtention de bonnes performances réside dans l'équilibrage de la charge entre les différents processeurs exécutant une requête; le temps de réponse étant égal à celui du processeur le plus chargé. J'ai abordé ce problème en considérant successivement trois types d'architectures parallèles : (i) à mémoire partagée; (ii) hiérarchique; et (iii) à mémoire non uniforme. Ces travaux ont été fait dans le cadre du projet ESPRIT-II nommé EDS (European Declarative System) dont l'objectif était de réaliser une machine parallèle ainsi que le serveur base de données associé. Les résultats de mes travaux ont été implantés dans le prototype DBS3 (Database System for Shared Store).

J'obtiens, en 1997, un poste de Maître de Conférences à l'Université de Versailles - Saint Quentin et effectue mon enseignement à l'IUT de Mantes la Jolie. Je poursuis alors mes activités de recherche en collaboration avec l'équipe RODIN, toujours sur des problèmes liés à l'exécution de requêtes complexes, mais en me concentrant sur la ressource mémoire puis à la disponibilité des données lors de requêtes d'intégration dans le cadre du projet DISCO (système d'intégration de données dont sera issue la start-up Kelkoo). Ces travaux se font dans le cadre de l'encadrement de la thèse d'Olga Kapitskaia, 'Traitement de requêtes dans les systèmes d'intégration des sources de données distribuées' [Kap99].

---

[1] Le changement d'équipe a fait suite à l'arrêt de l'activité de recherche en bases de données aux Clayes sous Bois.

En 1999, je suis collaborateur extérieur à l'INRIA, mais dans l'équipe CARAVEL[2] et travaille alors au support de fonctions coûteuses dans l'exécution de requêtes distribuées dans le cadre du système de publication et d'intégration LeSelect. J'initie ces travaux avec Fabio Porto, étudiant brésilien en thèse 'sandwich', que j'encadre pendant l'année qu'il passe en France [Por01]. Je poursuis ces travaux en 2000, avec Ioana Manolescu, doctorante avec qui je travaille pendant environ 1 an [Man01]. Une partie de ces travaux sont actuellement implantés dans la version industrielle de LeSelect, distribuée par la start-up Medience (issue de l'équipe CARAVEL).

Dans la même période (i.e., à partir de 1999), nous entamons, au PRiSM, avec Philippe Pucheral, des recherches sur l'ubiquité et la confidentialité des données, motivées par une prise de conscience de l'importance croissante des problèmes liés à la confidentialité dans le domaine des bases de données. Je participe à la création d'une nouvelle équipe sur ces thèmes au PRiSM dont Philippe Pucheral sera le directeur.

En 2000, nous proposons PicoDBMS, le premier SGBD complet embarqué dans une carte à puce, destiné à la sécurisation de dossiers portables. Ce travail est récompensé par un *Best Paper Award* de la conférence VLDB'00 et donne lieu à l'implantation de plusieurs prototypes (sur des cartes Schlumberger obtenues grâce à un accord de coopération avec Schlumberger CP8). J'encadre alors (depuis le DEA), Nicolas Anciaux sur le thème des bases de données embarquées sur des calculateurs ultralégers.

En 2002, je postule à l'INRIA et suis recruté en qualité de chargé de recherche 1ère classe dans le projet CARAVEL. Nous montons alors une équipe commune (PRiSM / INRIA), qui, suite à la création de Medience et au détachement de Philippe Pucheral à l'INRIA, devient une équipe INRIA en janvier 2003. L'équipe SMIS (Secured and Mobile Information Systems) dont je suis le responsable permanent depuis sa création est centrée sur les deux thématiques suivantes : (i) la gestion de données embarquées sur des calculateurs ultralégers et (ii) la préservation de la confidentialité des données par le biais de calculateurs sécurisés. SMIS deviendra un projet INRIA en octobre 2004.

Dans ce contexte, j'encadre depuis 2002, François Dang Ngoc sur le thème de la sécurisation du contrôle d'accès à des documents XML. Ces travaux donnent lieu à

[2] Le projet RODIN se terminant, il céda la place au projet CARAVEL dirigé par Eric Simon.

l’implantation de plusieurs prototypes dans des contextes applicatifs très différents (e.g. travail collaboratif, contrôle parental). Ils furent récompensés par deux prix logiciels (e-gate 2004 et SIMagine 2005).

Il est tout à fait entendu que la majorité des résultats présentés dans ce document sont le fruit de collaborations multiples, comme en atteste la liste de mes publications. J'en profite pour exprimer ici ma reconnaissance à tous ceux et celles qui ont apportés leur concours à ces travaux et pour cette raison, j'utiliserai dans la suite du document la première personne du pluriel.

Travaux initiaux

Mon activité de recherche a été, et est toujours centrée sur les noyaux des Systèmes de Gestion de Bases de Données (SGBD), et notamment sur l'exécution et l'optimisation de requêtes. L'objectif de ce chapitre est de présenter rapidement mes premiers travaux antérieurs à ceux sur lesquels se concentre ce manuscrit. En effet, bien qu'ils s'attaquent à des problématiques différentes, l'approche utilisée est une approche système des bases de données, tout comme dans les études menées sur l'ubiquité et la confidentialité des données. Ce chapitre expose tout d'abord le contexte général considéré puis présente successivement les différents travaux que j'ai menés entre 1993 et 2002 sur les SGBD parallèles, les modèles d'exécution adaptatifs et enfin sur la prise en compte de fonctions coûteuses.

1. Contexte

Un SGBD est un système permettant de stocker, d'organiser et de rendre facilement accessible de grands ensembles de données. Lors de l'accès à un SGBD, un utilisateur exprime sa question sous la forme d'une requête déclarative ; à savoir, la requête exprime ce que l'utilisateur désire plutôt que la façon de l'obtenir. Le SGBD doit alors produire un plan d'exécution (Query Execution Plan ou QEP) optimisé correspondant à la meilleure façon de répondre à la requête. Le terme 'meilleure' se traduit souvent en terme de temps de réponse, mais cela peut différer suivant le contexte (e.g. consommation de ressources, délivrance des premiers résultats, etc.). Le QEP est alors exécuté, par le moteur d'exécution, qui déclenche et contrôle l'exécution des différents opérateurs (e.g. sélection par index, jointure par hachage, etc.) qui le composent. Les opérateurs sont, eux même, définis au dessus des couches basses du SGBD, à savoir, le gestionnaire de stockage et d'indexation, le gestionnaire de mémoire et le gestionnaire transactionnel [CBB93]. La complexité des couches basses du SGBD ainsi que les opportunités d'optimisation font qu'elles se substituent généralement au système d'exploitation. En fait, ne pouvant désactiver le système d'exploitation, les SGBDs cherchent à ne pas déclencher certains de ses mécanismes. Par exemple, pour garder le contrôle de la mémoire, les SGBDs essayent de ne pas utiliser plus de mémoire que la mémoire physique.

La figure 1 présente l'architecture en couche des SGBD. Mes contributions se situent dans les couches systèmes des SGBD, à savoir entre le système d'exploitation et l'optimisation. Elles sont décrites rapidement dans les sections suivantes.

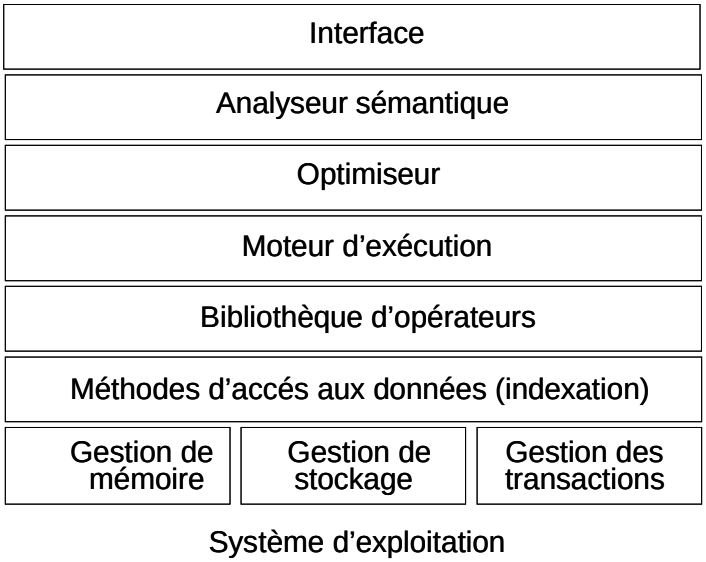


Figure 1 : *Architecture en couche d'un SGBD*

2. Exécution parallèle de requêtes : l'équilibrage de charge (thèse, 1993–1996)

Un des obstacles majeurs à l'obtention de bonnes performances lors de l'exécution parallèle de requêtes réside dans l'équilibrage de la charge entre les différents processeurs. En effet, le temps de réponse d'une exécution parallèle est égal au temps de réponse du processeur le plus chargé. Une mauvaise répartition de la charge de travail peut donc entraîner une chute de performance importante [Bou97].

Plusieurs remarques peuvent être faites sur les solutions proposées pour la répartition de charge: (i) à notre connaissance, aucune étude n'a cherché à résoudre le problème de la répartition de charge dans son ensemble, c'est à dire, au niveau intra opérateur (problème de mauvaises distributions des données) et inter opérateurs (problème d'allocation des processeurs aux opérateurs). (ii) les solutions au problème des mauvaises distributions de données sont généralement des solutions dynamiques (i.e. durant l'exécution), cherchant à redistribuer la charge équitablement. Le problème d'allocation des processeurs aux opérateurs est traité

statiquement ou juste avant l'exécution. (iii) enfin, aucune étude n'a considéré les architectures hybrides comme les architectures hiérarchiques et NUMA qui tentent d'associer la flexibilité de l'architecture à mémoire partagée et l'extensibilité des architectures distribuées.

Ce sont ces trois points qui ont motivé notre travail dans le cadre de ma thèse. Notre approche est totalement différente des études existantes. D'une part, nous nous intéressons à des architectures hybrides et cherchons à exploiter toutes leurs possibilités. D'autre part, les mécanismes développés sont entièrement dynamiques. Enfin, nous considérons d'une manière intégrée les différentes formes de parallélisme intra requête.

Nous avons successivement abordé trois types d'architectures parallèles : (i) à mémoire partagée [BDV96, BFD96]; (ii) hiérarchique [BFV96]; et (iii) à mémoire non uniforme (NUMA) [BFV99]. Pour chacune, nous avons proposé des modèles d'exécution redistribuant dynamiquement la charge au niveau intra et inter opérateurs, en minimisant les surcoûts entraînés par cette redistribution. Pour l'architecture hiérarchique, par exemple, le modèle d'exécution proposé permet une répartition dynamique de la charge à deux niveaux (locale sur un nœud à mémoire partagée puis globale entre les nœuds). Ce modèle permet de maximiser la répartition de charge locale, réduisant ainsi le besoin de répartition de charge globale, source de surcoûts. Ces travaux ont été validés par des mesures sur le prototype DBS3 [CBB93, CDB95] et par des simulations.

3. Modèles d'exécution adaptatifs (INRIA, 1997–1999)

Les plans d'exécution produits par les optimiseurs de requêtes traditionnels peuvent être peu performants pour plusieurs raisons : les estimations de coûts peuvent être inexactes; la mémoire disponible lors de l'exécution peut s'avérer insuffisante; et les données, si elles proviennent de sources distantes, peuvent ne pas être disponibles immédiatement, lorsqu'elles sont demandées. Classiquement, le moteur d'exécution n'a aucune marge de manœuvre et se borne à exécuter le plan d'exécution produit par l'optimiseur. C'est pourtant lors de l'exécution que l'on peut constater des problèmes comme une surconsommation de mémoire, des erreurs d'estimation, des problèmes de disponibilité des données, etc. Ces considérations motivent le développement de modèles d'exécution adaptatifs.

Dans un premier temps, nous nous sommes attachés au problème de la gestion de la mémoire lors de l'exécution de requêtes complexes dans le contexte plus simple d'un système

centralisé et proposé deux solutions [BKV98] : l'allocation statique de mémoire appliquée au lancement de l'exécution de la requête et un modèle d'exécution dynamique qui adapte l'ordonnancement de la requête en fonction de la consommation mémoire. Notre modèle d'exécution dynamique se révèle très efficace : il gère bien les débordements de mémoire et résiste aux erreurs d'estimation de coûts. Nous avons comparé nos deux solutions grâce à une combinaison d'implémentation et de simulation (pour la génération de requêtes). Les résultats sur un grand nombre de requêtes montrent des gains significatifs en performances pour notre modèle dynamique, de 50 à 85%, par rapport au modèle statique.

Nous nous sommes ensuite intéressés au problème de disponibilité des données lors de l'exécution de requêtes d'intégration de données distantes [BFV+00, BFM+00]. L'approche est à la fois prévisionnelle, en produisant un ordonnancement pas à pas de plusieurs fragments de requêtes, et réactionnelle, en exécutant ces fragments en fonction de l'arrivée des données distantes. Dans notre contexte réparti, les méta-données nécessaires à l'élaboration d'un plan d'exécution efficace peuvent être regroupées en quatre classes: les paramètres connus statiquement, avec plus ou moins d'exactitude (e.g. requête, statistiques, etc.), les paramètres qui ne sont connus qu'au début de la phase d'exécution (e.g. ressources disponibles), les paramètres qui ne seront connus avec exactitude qu'en cours d'exécution (e.g. taille des résultats intermédiaires, etc.), enfin les paramètres qui varient continuellement (e.g. débit du réseau, sources de données disponibles, taux d'arrivée des données, etc.). Nous avons développé une méthode d'optimisation et d'exécution adaptée à ce contexte : nous proposons d'exploiter les méta-données dès qu'elles deviennent disponibles. Ainsi, un plan d'exécution initial (ou du moins, un ensemble de pré-calculs qui pourront faciliter l'élaboration de ce plan) est produit avec les méta-données connues statiquement, au début de la phase d'exécution. Ce plan fixe les grandes options pour l'exécution mais laisse certains degrés de liberté, ceux qui dépendent de paramètres inconnus à ce moment. Ensuite, l'exécution se déroule en plusieurs étapes entrecoupées par des phases de planification. Une phase de planification fournit un plan d'exécution conforme aux méta-données connues à ce moment précis. Elle est suivie par une phase d'exécution qui applique le plan et qui réagit conformément à ce qui a été prévu par le planificateur. Lorsqu'un événement susceptible de remettre en cause le plan se produit, la phase d'exécution se termine et cède sa place à une nouvelle phase de planification. Cette méthode fournit un cadre générique pour l'optimisation dynamique. Son adaptation à un contexte donné demande de définir les heuristiques utilisées lors des phases de planification, les degrés de

liberté par rapport au plan initial, ainsi que les événements remettant le plan en cause. Nous avons développé plusieurs algorithmes, qui suivent cette méthode afin d'optimiser dynamiquement l'exécution des requêtes globales dans un médiateur d'accès aux données.

4. Prise en compte de fonctions coûteuses (INRIA, 1999–2002)

Internet rend possible le partage de données et de programmes entre des groupes de scientifiques. LeSelect est un système d'intégration de données, permettant d'une part la publication de ressources (données et programmes) et d'autre part, l'interrogation de ces ressources de manière transparente. Dans le cadre des applications environnementales que nous envisageons avec Le Select, les scientifiques peuvent typiquement avoir à poser des requêtes impliquant des données et des programmes (par exemple, un programme d'extraction de motifs dans une image) publiés en divers points du réseau. Même s'il est possible d'exprimer ces requêtes en SQL, les techniques classiques d'optimisation et d'exécution de requêtes sont insuffisantes pour deux raisons. Tout d'abord les optimiseurs de requêtes SQL considérant que le facteur prédominant est le coût des opérations de jointure, cherchent à minimiser ce coût en jouant sur l'ordre dans lequel sont exécutées les jointures. A contrario, dans notre contexte, le coût prédominant est celui de l'exécution des programmes et du transport des données volumineuses (telles que des images) depuis le site publiant ces données vers les lieux de traitement. D'où la nécessité d'établir des techniques d'optimisation spécifiques qui minimisent le nombre d'exécutions de programmes et la quantité de données volumineuses transférées. Dans [BFP+01], nous proposons deux techniques qui exploitent les possibilités d'optimisation inhérentes à l'architecture distribuée de systèmes comme Le Select. La première consiste à exécuter les traitements chers (appel des fonctions et transfert des données volumineuses) le plus tard possible: l'optimiseur planifie donc d'exécuter d'abord les opérations standard (sélections, jointures, etc.). La seconde consiste à utiliser un cache pour éviter des traitements redondants. Le coût total en temps d'exécution est minimisé en parallélisant l'exécution des traitements: parallélisme entre le transfert des données et l'exécution des programmes, exécution concurrente de plusieurs programmes. Le problème pour l'optimiseur est de décider de l'ordonnancement des programmes entre eux, et de décider la forme de parallélisme inter programme: parallélisme pipeline, ou parallélisme indépendant. Nous proposons des algorithmes de planification dynamique permettant de s'adapter au flot de données constaté en

cours d'exécution et montrons que l'approche dynamique peut apporter des gains considérables en termes de temps de réponse.

Dans [MBF02], nous montrons comment le modèle de relations à patterns d'accès peut être utilisé pour modéliser uniformément des sources de données contenant des fonctions ainsi que des blobs. Les patterns d'accès peuvent être utilisés de manière naturelle pour modéliser des fonctions. Nous proposons la même modélisation pour des relations contenant des objets binaires de grande taille (blobs). L'opérateur logique BindJoin est une variante de l'opérateur relationnel de jointure, utilisé pour accéder aux relations ayant des patterns d'accès. Nous analysons alors l'effet de la présence des fonctions coûteuses et des blobs sur la conception de l'opérateur de BindJoin, et sur son intégration dans un plan d'exécution de requête. Premièrement, le travail total et le temps de réponse des requêtes nécessitant des transferts de blobs et des appels de fonctions coûteuses doivent être réduits, en utilisant des techniques de cache et d'asynchronisme. Deuxièmement, nous montrons l'importance de maximiser le taux de délivrance des résultats au début de l'exécution. L'aspect le plus spécifique de l'opérateur proposé est qu'il exploite la présence de doublons dans son entrée afin de fournir à l'utilisateur la plupart des résultats de la requête assez vite. Puisque l'opérateur de BindJoin proposé inclut toutes les optimisations, la tâche du publieur est considérablement réduite, tout en fournissant des bonnes performances.

5. Conclusion

Mon activité de recherche a donc débuté par l'étude des noyaux des SGBD, notamment l'exécution et l'optimisation de requêtes les différents contextes mentionnés précédemment. Depuis 7 ans environs, j'ai orienté mes recherches sur les thèmes de l'ubiquité et de la confidentialité et c'est sur cette période que se concentre la suite de ce manuscrit, justifiant ainsi son titre : 'Sécurisation du contrôle d'accès dans les bases de données'. Mon objectif, dans la suite de ce document, est de mener une analyse de différentes études que nous avons conduites, en les situant par rapport à l'état de l'art et en indiquant leur limites.

Sécurisation du contrôle d’accès dans les bases de données

Chapitre 1

Introduction

La préservation de la confidentialité est devenue une priorité pour les citoyens ainsi que pour les administrations. Le besoin d'accumuler, de partager et d'analyser des données personnelles est multiple : pour l'amélioration de la qualité des soins grâce au dossier médical électronique (Electronic Health Record - EHR), pour rendre plus simples et efficaces les procédures administratives, pour personnaliser les services rendus par une grande quantité d'objets électroniques dans un environnement d'intelligence ambiante (e.g. [ImN02], projet 'Aware Home' [AWA03]) ou même pour la lutte contre le terrorisme (croisement de bases de données commerciales et gouvernementales pour la recherche de suspects [EFF]). Bien que le traitement de données personnelles a généralement un but louable, il constitue une menace sans précédent aux droits élémentaires à la protection de la vie privée[3].

Partout dans le monde, les gouvernements adoptent des lois spécifiques pour cadrer l'utilisation de données personnelles comme le 'Federal Privacy Act' aux USA [Pri74] ou la directive pour la protection des données en Europe [Eur85]. Il est cependant difficile de traduire ces lois en moyens technologiques convaincants garantissant leur application.

Comme l'atteste le rapport 'Computer Crime and Security Survey' établi par le Computer Security Institute et le FBI [CSI04], le nombre d'attaques de serveurs de bases de données est croissant malgré la mise en place de politiques de sécurité de plus en plus drastiques. Pire encore, presque la moitié de ces attaques sont conduites par des employés ayant légalement accès à tout ou partie des données. Ceci montre la vulnérabilité des techniques traditionnelles de sécurisation des serveurs bases de données [BPS96].

La sécurité des bases de données inclus trois principales propriétés : la confidentialité, l'intégrité et la disponibilité [Cup00]. Grossièrement, la propriété de confidentialité garantie que les données protégées ne seront jamais accédées par une personne ou un programme non autorisé. La propriété d'intégrité garantie la détection d'une quelconque modification des

données, qu'elle soit accidentelle ou malicieuse. Enfin, la propriété de disponibilité protège le système contre les attaques de déni de service. Ce document est centré sur la confidentialité des données. Il aborde cependant aussi l'intégrité des données car la corruption de données peut entraîner des fuites d'information (par exemple, le contrôle d'accès du SGBD peut être trompé par la modification de données sur lesquelles est basé ce contrôle).

La préservation de la confidentialité des données nécessite de mettre en application les politiques de contrôle d'accès définies au niveau du SGBD. Une politique de contrôle d'accès, i.e., un ensemble d'autorisations, peut prendre différentes formes selon le modèle de données sous jacent. Par exemple, une autorisation dans une base de données relationnelle est habituellement exprimée comme le droit d'exécuter une action donnée (e.g. sélection) sur une table relationnelle ou sur une vue (i.e., une table virtuelle calculée par une requête SQL) [Mes93]. Une autorisation sur un document XML est généralement exprimée par une combinaison de règles positives (resp. négatives) permettant de sélectionner dans le document les sous arbres autorisés (resp. interdits) grâce à des expressions XPath [BCF00, GaB01, DDP+02]. Une autre dimension des modèles de contrôle d'accès est le mode d'administration de ces autorisations, en suivant soit un modèle discrétionnaire (Discretionary Access Control - DAC) [HRU76], basé sur des rôles (Role-Based Access Control - RBAC) [SCF+96], ou sur un mode obligatoire (Mandatory Access Control - MAC) [BeL76]. Dans ce document, nous ne faisons pas d'hypothèse sur le mode d'administration de ces autorisations si ce n'est dans un but illustratif.

Indépendamment du modèle de contrôle d'accès, les limitations mises en place par le serveur de bases de données peuvent être outrepassées de plusieurs façons. Un intrus peut, par exemple, infiltrer le système d'information et examiner l'empreinte disque de la base de données. Une autre source d'attaque vient du fait que beaucoup de bases de données sont aujourd'hui externalisées chez des fournisseurs de service base de données (Database Service Providers ou DSP [CaB02, eCr02, Qck02]). Ainsi, le propriétaire des données n'a pas d'autre choix que de croire aux garanties de confidentialité des DSPs, soutenant que leur système est totalement sûr et que leurs employés sont au dessus de tout soupçon, un argumentaire

[3] Remarquons que la confidentialité des données est bien sur de première importance pour la protection des données d'entreprises (stratégie commerciale, savoir faire, fichiers clients, etc.) contre l'espionnage industriel et commercial.

fréquemment contredit par des faits [HIM02]. Finalement, un administrateur de bases de données a suffisamment de pouvoirs pour modifier le mécanisme de contrôle d'accès et pour espionner le comportement du SGBD. Bien que ce constat ne soit pas nouveau ni spécifique aux systèmes électroniques de gestion de données[4], il doit être considéré avec une attention particulière.

L'objectif de ce document est de présenter les différents moyens de lutte contre ces différentes formes d'attaque. La communauté bases de données s'est récemment intéressée à ce problème, considérant l'utilisation de techniques cryptographiques et de puces sécurisées pour compléter et renforcer les techniques de contrôle d'accès. Dans ce document, nous analysons et comparons différentes méthodes à base de chiffrement, hachage cryptographique, signatures ou de composants matériels sécurisés pour accroître la confiance accordée aux systèmes de gestion de bases de données afin de garantir la protection de la vie privée. Nous dégageons alors de cette étude d'importantes perspectives de recherche

Ce document est organisé comme suit. Le chapitre 2 introduit une classification des différentes attaques pouvant être conduite sur un SGBD, puis présente les instruments de bases permettant de se prémunir contre ces attaques, à savoir, les techniques cryptographiques et les composants matériels sécurisés. Le chapitre 3 présente PicoDBMS, un SGBD relationnel entièrement embarqué dans une carte à puce sécurisée. L'approche proposée dans PicoDBMS est représentative des méthodes de protection basées uniquement sur un composant matériel sécurisé. Dans le chapitre 4, nous décrivons tout d'abord les approches basées uniquement sur l'utilisation de techniques de chiffrement pour les SGBD relationnels puis présentons C-SDA, une solution combinant un serveur relationnel travaillant sur des données chiffrées et un composant embarqué dans une carte à puce. Le chapitre 5 décrit deux approches existantes utilisant des techniques cryptographiques (chiffrement, intégrité) et des composants matériels sécurisés puis présente notre approche pour l'exécution de requêtes sur des données stockées sur un serveur non sécurisé. Enfin le chapitre 6 aborde le problème de la protection de données XML, présente l'état de l'art du domaine et décrit notre approche. Le chapitre 7 conclut cette étude et présente nos perspectives de recherche.

---

[4] Une règle spécifiant par exemple que 'seuls les médecins peuvent accéder aux dossiers des patients' est en fait traduite dans le cas d'archives papier par 'les médecins et l'archiviste peuvent accéder aux dossiers des patients'.

Chapitre 2

Attaques, attaquants et instruments

Ce chapitre introduit tout d'abord quatre classes d'attaques pouvant compromettre la confidentialité des données puis présente les différents types d'attaquants pouvant conduire ces attaques. Dans une deuxième section, les instruments de base permettant de parer ces attaques sont exposés : techniques cryptographiques et composants matériel sécurisés. L'objectif est de clarifier le problème et les éléments de solution.

1. Attaques et attaquants

Comme indiqué dans l'introduction, nous nous intéressons ici aux attaques pouvant compromettre la confidentialité des données, c'est-à-dire, aux attaques pouvant entraîner la lecture de données non autorisées. Cependant, les attaques entraînant la corruption de données doivent aussi être considérées puisqu'elles peuvent permettre de modifier les autorisations des utilisateurs. Sur des données relationnelles, ces attaques peuvent être conduites en modifiant les définitions de vues, les tables de privilèges ou les données participant à l'évaluation d'une vue autorisée. De même, dans un contexte XML, la modification de la structure du document ou simplement de la valeur de certains nœuds peut changer l'évaluation de règles positives ou négatives et donner l'accès à des données non autorisées.

Nous considérons dans la suite que l'attaquant a éventuellement accès à des données autorisées et a obtenu de manière licite ou illicite un *certain accès* à des données non autorisées. C'est le cas, par exemple, d'un utilisateur qui aurait accès aux fichiers de la base de données stockés sur un serveur, voire sur un terminal portable lui appartenant[5]. Dans ce cas, il est impératif de protéger de manière cryptographique ces fichiers afin d'éviter que l'attaquant puisse facilement les consulter ou les modifier. Le chiffrement des fichiers est donc nécessaire et nous montrons ci-dessous quatre classes d'attaques pouvant être menées sur les fichiers de la base de données même quand ces derniers sont chiffrés.

---

[5] Remarquons que l'utilisateur peut héberger la base de données sans toutefois avoir accès à son intégralité. C'est le cas du dossier médical personnalisé (qui gagnerait à être stocké sur un terminal portable) ou par exemple d'une base de données stockant des informations utilisées pour le contrôle d'accès à des données digitales (DRM).

- Examen des données chiffrées : L'attaquant examine les données chiffrées et déduit des informations confidentielles. En effet, suivant le mode de chiffrement, certains motifs peuvent apparaître dans les données chiffrées et peuvent laisser transparaître des informations (voir section 2).

- Corruption de données chiffrées : L'attaquant supprime ou modifie, même aléatoirement, des données dans le but de corrompre le mécanisme de contrôle d'accès. Par exemple, la modification aléatoire d'un champ 'age' chiffré modifie avec une forte probabilité une règle qui autoriserait l'accès aux personnes majeures.

- Substitution de données chiffrées : L'attaquant substitue des blocs de données valides par d'autres blocs de données valides. Cette attaque, relativement simple à mener, permet de créer une information invalide à partir de données valides et donc de corrompre le contrôle d'accès.

- Rejeu de données : L'attaquant substitue des blocs de données valides par une ancienne version de ces mêmes blocs. Cette attaque permet par exemple de conserver l'accès à des données pour lesquelles les droits ont été supprimés.

Remarquons que ces attaques peuvent être menées sur les données stockées sur le disque du SGBD, sur les données en cours de traitement dans le SGBD (notamment dans les blocs mémoire du serveur) et enfin sur les données échangées sur le réseau entre le client et le serveur.

Trois classes de pirates susceptibles de mener ces attaques peuvent être distinguées en fonction de leurs privilèges initiaux :

- Le pirate *externe* est un intrus qui s'infiltre sur un système informatique et récupère l'empreinte disque de la base de données. [App03, Eru01] montrent que la majorité des attaques sur des systèmes informatiques ciblent les données stockées sur les disques des serveurs.

- Le pirate *interne* est un usager reconnu par le système d'exploitation et le SGBD. Il possède des droits sur une partie de la base de données et veut accéder à des données outrepassant ses droits. Ce pirate a potentiellement le même pouvoir qu'un pirate

externe et peut en plus exploiter ses droits restreints. [CSI04] montre qu'environ 50% des vols d'informations proviennent d'attaques internes.

- Le pirate *administrateur* a suffisamment de privilèges (généralement tous) pour administrer le système informatique (administrateur système) ou la base de données (administrateur de base de données ou DBA). Ces privilèges lui permettent d'accéder aux fichiers du SGBD et d'espionner son comportement.

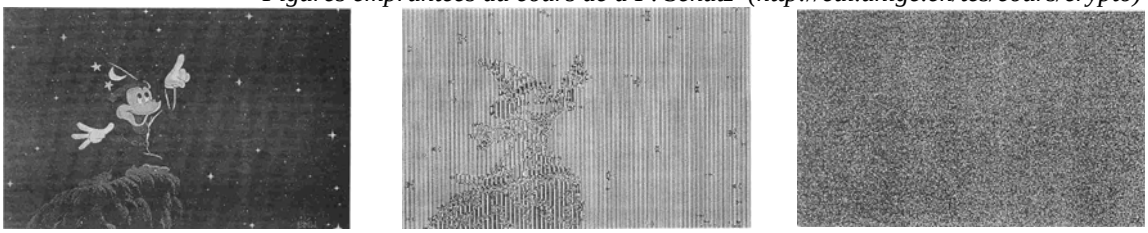
L'intersection entre ces différentes classes est non vide puisque, par exemple, un pirate externe peut arriver à s'octroyer les droits d'administration. L'objectif de cette classification est simplement d'associer un nom aux différentes classes d'attaquants de dangerosité croissante.

2. Instruments : Techniques cryptographiques et composants matériels sécurisés

Les techniques logicielles (techniques cryptographiques) ou matérielles (e.g. puces sécurisées) décrites sommairement ci-dessous forment les outils de base pour la réalisation de systèmes de bases de données résistants aux attaques exposées dans la section précédente.

Chiffrement : Les techniques de chiffrement (ou cryptage) prennent en entrée une donnée en clair et une clé et rendent en sortie une donnée indéchiffrable à l'échelle humaine pour quiconque ne connaît pas la clé de déchiffrement. On distingue deux grandes classes d'algorithmes : (1) les algorithmes à clé secrète (e.g. DES, 3DES, AES) utilisent la même clé pour le chiffrement et le déchiffrement ; (2) les algorithmes asymétriques (e.g. RSA [RSA78]) utilisent un couple de clés, privée et publique, cette dernière pouvant être divulguée librement. Les algorithmes asymétriques permettent de ne pas partager de secret mais sont cependant beaucoup plus lents. L'objectif du chiffrement est donc de garantir l'opacité des données (protection contre l'examen des données) en cachant l'information aux personnes non autorisées (e.g. pirate externe). Cependant, pour éviter qu'un attaquant puisse analyser les motifs répétitifs apparaissant dans les données chiffrées, il est essentiel de choisir un mode de chiffrement cachant ces motifs. La figure 2 donne un aperçu de ce problème en montrant le résultat d'un chiffrement 3-DES sur une image avec un mode ECB (chaque bloc de 8 octets est chiffré séparément) et avec un mode CBC (le résultat du chiffrement d'un bloc de 8 octets dépend des blocs précédents).

Figures empruntées au cours de à F. Schutz (http://cui.unige.ch/tcs/cours/crypto)



(a) Image originale *(b) chiffrement 3-DES-ECB* *(c) chiffrement 3-DES-CBC*

Figure 2. *Opacité du mode de chiffrement.*

Fonctions de hachage cryptographique (FHC) : Une FHC est une fonction mathématique qui prend en entrée un texte d'une longueur quelconque et calcule une valeur de hachage, i.e., une empreinte de taille fixe (e.g. 20 octets), utilisée pour détecter toute modification accidentelle ou malicieuse du texte initial. Une FHC garantie qu'il est techniquement impossible de retrouver le texte initial à partir de la valeur de hachage ou de trouver deux textes distincts produisant la même valeur de hachage. Message Digest 5 (MD5) [Riv92] et Secure Hash Algorithm 1 (SHA-1) [NIS95] sont deux exemples classiques de FHC. Le résultat d'une FHC peut être utilisé pour détecter si une donnée a été modifiée ; il est alors appelé *Code de Détection de Modification (CDM)* et permet de détecter une modification d'un bit quelconque dans un texte de longueur arbitraire.

Authentification de l'origine [MOV97] : Lorsque l'on associe une clé secrète à une FHC, le résultat est appelé un *Code d'Authentification de Message (CAM)*. Il permet de garantir que le message n'a pas été modifié lors d'une communication entre deux parties détenant la clé secrète. L'utilisation d'algorithmes asymétriques permet d'obtenir une signature électronique, prouvant l'origine des données (le détenteur de la clé privée) et garantissant ainsi la non répudiation.

Garantie de fraîcheur : Bien que les CDM, CAM et les signatures électroniques puissent être utilisées pour prouver que les données n'ont pas été corrompues depuis leur production, ils ne donnent pas de garantie de fraîcheur. Ainsi, ces techniques, à elles seules, ne permettent pas d'éviter les attaques basées sur le rejeu. L'ajout de paramètres dépendant du temps (TVP ou time-variant parameters) permet de parer ce type d'attaques. Ces TVP peuvent être des nombres aléatoires dans des protocoles de type challenge-réponse, des numéros de séquences ou des estampilles.

Arbres de hachage de Merkle [Mer90] : Un arbre de hachage de Merkle, ou arbre de Merkle est un arbre virtuel, calculé dynamiquement à partir des blocs de données chiffrés (feuilles de l'arbre). Il permet de garantir l'intégrité de n'importe quel bloc ou groupe de blocs en fonction d'une seule valeur de hachage correspondant à la racine de l'arbre. Comme indiqué sur la figure 3, chaque valeur de hachage intermédiaire est calculée en hachant la concaténation des valeurs de hachage des fils. Chaque valeur de hachage, au niveau des feuilles, est obtenue en hachant le bloc de données correspondant. Pour vérifier l'intégrité d'un des blocs, il est nécessaire de calculer son hachage et de la combiner avec quelques valeurs de hachage afin de recalculer la valeur racine à comparer avec la valeur espérée. Dans l'exemple de la figure 3, pour prouver l'intégrité du bloc C4, il est nécessaire de récupérer les valeurs de hachage H3, H9 et H14 afin de calculer H15'=H (H14||H (H9||H (H3||H(C4))) et de le comparer avec H15.

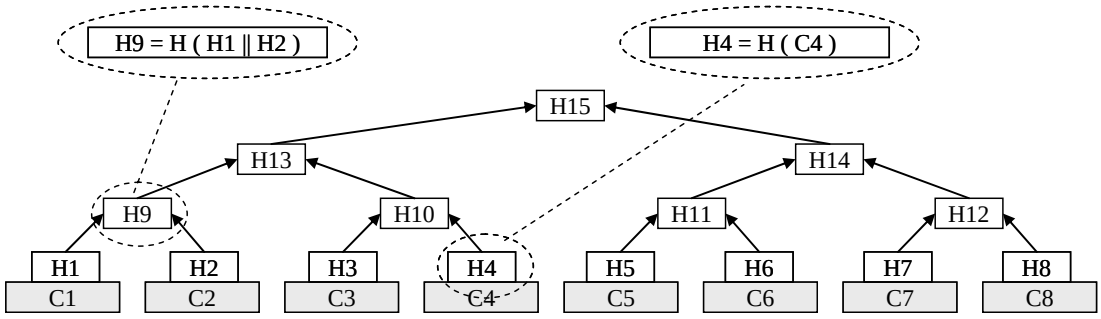


Figure 3. *Arbre de hachage de Merkle.*

Composants matériels sécurisés : Les coprocesseurs sécurisés [Swe99, DLP+01] et les cartes à puces [Tua99] sont aujourd'hui les composants matériels sécurisés les plus largement utilisés. Ils garantissent que le code et les données embarqués ne peuvent être corrompus ni accédés de manière illicite grâce à des mécanismes de sécurité matériels et logiciels. Dans les cartes à puces, par exemple, des couches de métal couvrent la puce et permettent de détecter toute tentative d'intrusion, les radiations produites par le processeur pendant le traitement sont limitées, la consommation d'énergie ainsi que la température de la puce sont maintenue constantes afin de contrer toute analyse, les comportements anormaux sont évités grâce à des détecteur de basse fréquence et de voltage anormal désactivant la puce. Ainsi les données et/ou programmes chargés sur ces composants sont stockés/exécutés correctement ou sont détruits en cas d'intrusion.

Chapitre 3

PicoDBMS, une approche basée sur un composant matériel sécurisé

Une solution drastique pour garantir la sécurité du code du SGBD et des données consiste à embarquer l'ensemble dans un environnement d'exécution sécurisé (Secured Operating Environment ou SOE [HeW01]), héritant ainsi de sa sécurité intrinsèque. La carte à puce est un très bon exemple de SOE ; elle présente une sécurité inégalable [AnK96] pour les données et le code embarqué, permettant de résister aux attaques identifiées dans le chapitre 2. Ce constat nous a conduit à proposer PicoDBMS, le premier SGBD relationnel complet embarqué dans une carte à puce [BBP00, PBV+01]. Ce chapitre résume tous d'abord l'état de l'art, détaille les contraintes des cartes à puce et leur influence sur la conception d'un SGBD puis présente succinctement notre contribution. Les limitations de PicoDBMS et de l'approche 'tout embarqué' en général sont alors discutées. Finalement, les résultats les plus significatifs de cette étude sont présentés.

1. Etat de l'art

Peu de travaux ont porté jusqu'à présent sur la gestion de données embarquées au sein de la communauté bases de données. Les premiers travaux émanent des éditeurs de SGBD qui dérivent aujourd'hui des versions allégées de leurs systèmes pour des assistants personnels (e.g. Sybase Adaptive Server Anywhere [Gig01], Oracle 9i Lite [Ora02], SQLServer for Windows CE [SeG01] or DB2 Everyplace [KLL+01]). Leurs préoccupations concernent la réduction de l'empreinte disque du SGBD [Gra98] ainsi que sa portabilité, mais le problème spécifique et plus complexe d'embarquer un SGBD dans une puce n'est pas adressé.

Les membres de l'équipe RD2P du laboratoire LIFL, associés à Gemplus ont pour la première fois suggéré l'idée d'une carte à puce à interface SQL, aboutissant à la carte CQL (Card Query Language) [Gri92, Gem92, PaV94a, PaV94b]. Ces travaux ont conduit à la

définition du standard ISO SCQL [ISO99] spécifiant un langage bases de données pour carte à puce et à un produit industriel (ISOL's SQLJava Machine [Car99]). Ces propositions considéraient une génération de cartes disposant de 8 KB de mémoire persistante. Bien que leur conception soit limitée à des requêtes mono-table, ces travaux illustrent l'intérêt de concevoir un SGBD dédié pour carte à puce. Plus récemment, MODS, l'initiative de Mastercard [Mas02], offre une API commune permettant aux commerçants, banques et autres organisations d'accéder et de stocker des données sur les cartes à puces des utilisateurs avec une sécurité accrue pour le porteur de carte. Cependant, MODS est basé sur des fichiers plats (et donc non compacts), des droits d'accès grossiers (au niveau du fichier) et n'offre aucune possibilité d'exécution de requête. Néanmoins, cette initiative montre l'intérêt du développement de techniques bases de données pour ces environnements.

Une étude récente propose des techniques de stockage spécifiques pour gérer des données dans une puce contenant de la mémoire FLASH [BSS+03]. Là encore, la conception se limite à des requêtes mono-table et est fortement orientée par l'architecture du type de puce ciblé. Ainsi, ce travail propose de stocker les données dans de la mémoire FLASH de type NOR, généralement substituée à la ROM et dédiée au stockage des programmes. Vu que les modifications dans une FLASH NOR sont très coûteuses (une simple modification impose l'effacement très coûteux d'un large bloc de données), les techniques envisagées sont orientées vers la minimisation du coût de modification. Bien que cette étude montre l'impact des propriétés matérielles sur le noyau du SGBD, elle ne satisfait pas les contraintes des cartes et ne traite pas de l'exécution des requêtes complexes, nécessaires au calcul des données autorisées.

2. Approche

Les cartes à puce actuelles disposent d'un processeur 32 bit cadencé à 50 MHz, de modules mémoire dotés de 96 KB de ROM, 4 KB de RAM et 64 KB d'EEPROM, et de modules de sécurité (générateur de nombres aléatoires, co-processeur cryptographique, etc.). La ROM stocke le système d'exploitation, des données statiques et des routines standard. La RAM sert de mémoire de travail (pile et tas). L'EEPROM stocke les informations persistantes, les données et les applications téléchargées (dans le cas de cartes multi-applications).

Les ressources internes de la puce présentent des asymétries originales. Le processeur, calibré pour effectuer des calculs cryptographiques et assurer la sécurité de la puce, est

surdimensionné par rapport au volume de données embarquées. De plus, la mémoire persistante (EEPROM) partage les caractéristiques d'une mémoire RAM en terme de granularité (accès direct à chaque mot en mémoire) et de performance de lecture (60 à 100 ns par mot), mais souffre d'un temps d'écriture extrêmement lent (environ 10 ms par mot). Enfin, seules quelques centaines d'octets de RAM sont disponibles pour les applications embarquées, la majeure partie de la RAM étant réservée au système d'exploitation et à la machine virtuelle (cartes Java). Ces caractéristiques entraînent une reconsidération en profondeur de l'ensemble des techniques de gestion de bases de données et nous amènent à suivre les six règles de conception suivantes :

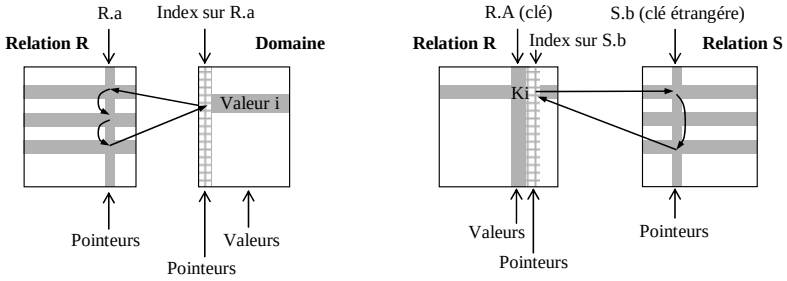
- *Règle de compacité :* minimiser l'empreinte des données, des index, et du code de PicoDBMS pour s'adapter à la quantité réduite de mémoire persistante ;
- *Règle de la RAM :* minimiser la consommation RAM des opérateurs vu sa taille extrêmement limitée ;
- *Règle d'écriture :* minimiser les écritures en mémoire persistante vu leur coût très élevé (≈10 ms/mot) ;
- *Règle de lecture :* tirer parti des lectures rapides en mémoire persistante (≈100 ns/mot) ;
- *Règle d'accès :* tirer parti de la fine granularité et de l'accès direct à la mémoire persistante pour les opérations de lecture et d'écriture ;
- *Règle de sécurité :* ne jamais externaliser de donnée sensible hors de la puce et minimiser la complexité algorithmique du code pour éviter les trous de sécurité ;
- *Règle du CPU :* tirer parti de la puissance surdimensionnée du CPU, comparée au volume de données embarquées.

3. Contribution

Les modules embarqués dans la puce pour garantir la confidentialité sont : un module de stockage organisant les données et les index associés dans la mémoire persistante de la puce, un évaluateur de requêtes capable de construire et d'exécuter des plans d'exécution complexes (composés d'opérateurs de sélection, projection, jointures et calculs d'agrégats), un module de droits d'accès offrant la possibilité de donner/retirer des droits sur les vues définies (évitant ainsi d'externaliser des données non autorisées), et finalement un moteur transactionnel assurant l'atomicité de séquences de modifications. Dans la suite, nous présentons succinctement le modèle de stockage et les techniques d'évaluations de requêtes associées. Pour plus de détails, le lecteur peut se référer à l'article joint en annexe A.

Le modèle de stockage d'un PicoDBMS doit garantir au mieux la compacité des données et des index. Dès lors que la localité des données n'est pas un problème (règle d'accès), un modèle de stockage basé sur l'utilisation intensive de pointeurs inspiré des SGBD grande mémoire [MiS83, AHK85, PTV90] favorise la compacité. Pour éliminer les doublons, nous regroupons les valeurs dans des domaines. Les enregistrements référencent leurs valeurs d'attributs avec des pointeurs. Ce stockage complexifie la mise à jour mais diminue son coût simplement parce qu'elle génère moins d'écritures (règle d'écriture).

Les index doivent eux aussi être compacts. En supposant que l'attribut indexé prenne valeur sur un domaine, nous proposons de construire une structure accélératrice en forme d'anneau partant de la valeur de domaine vers les enregistrements, comme le montre la figure 4.a. Le coût de ce stockage indexé est réduit à son minimum, c'est à dire un pointeur par valeur de domaine, quelque soit la cardinalité de la relation indexée, au prix d'un surcoût de projection des enregistrements, dû au parcours, en moyenne, de la moitié de la chaîne de pointeurs pour retrouver la valeur d'attribut.



(a) RS sur un attribut régulier. *(b) RS sur un attribut clé étrangère.*

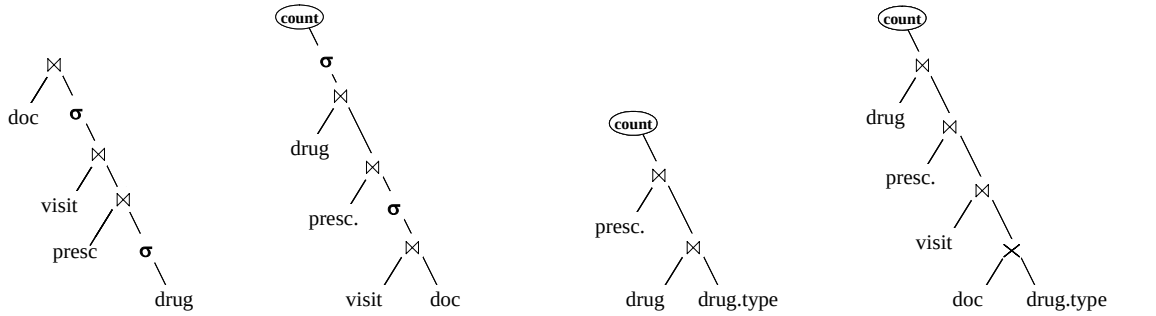
Figure 4. *Stockage en anneau (RS).*

Les index de jointure [Val87] peuvent être construits d'une manière similaire. La figure 4.b montre un index de jointure en anneau entre les relations R et S. R.a est une clé stockée 'à plat' et un anneau est construit partant de R.a vers tous les tuples de S pour lesquels S.b = R.a.

L'exécution traditionnelle de requêtes exploite la mémoire de travail pour stocker des structures temporaires (e.g. tables de hachage) et des résultats intermédiaires, et en cas de débordement mémoire matérialise des résultats sur disque. Ces algorithmes ne peuvent être utilisés dans un PicoDBMS (règles de la RAM et d'écriture). Nous proposons d'exécuter les requêtes de type SPJ (sélection, projection, jointure) en 'pipeline extrême', i.e., tous les opérateurs travaillent en pipeline, tuple par tuple, y compris les sélections (cf. figure 5). Les

opérandes de gauche étant toujours les relations de base, elles sont déjà matérialisées en mémoire persistante, ce qui permet d'exécuter la requête sans consommer de RAM. L'exécution pipeline peut être réalisée en utilisant le modèle *itérateur* [Gra93], où chaque opérateur supporte trois appels de procédures : *open* prépare l'opérateur à produire un résultat, *next* produit un résultat, et *close* libère les structures et termine l'opérateur. L'exécution commence à la racine de l'arbre et se propage jusqu'aux feuilles, le flux de données étant constitué des tuples passés par un opérateur fils à son père en réponse à un appel *next*. Bien que les plans d'exécution ainsi générés soit clairement sous optimaux (e.g. une requête contenant plusieurs sélections ne pourra profiter pleinement de leur sélectivité), les performances restent correctes du fait du surdimensionnement du CPU (règle du CPU) et du modèle de stockage en anneau.

Le problème se complexifie lorsqu'il s'agit d'exécuter des requêtes contenant des opérateurs de calculs d'agrégats. En effet, l'exécution pipeline parait incompatible avec ces opérateurs, s'exécutant traditionnellement sur des résultats intermédiaires matérialisés. Nous proposons une solution au problème exploitant deux propriétés : (1) les calculs d'agrégat peuvent être réalisés en pipeline sur un flot de tuples groupés par valeur distincte, et (2) les opérateurs fonctionnant en pipeline préservent l'ordre des tuples puisqu'ils consomment (et produisent) les tuples dans l'ordre d'arrivée. Ainsi, la consommation dans un ordre adéquat des tuples aux feuilles de l'arbre permet d'effectuer les calculs d'agrégat en pipeline. Ces techniques permettent d'exécuter des requêtes contenant des agrégats sans consommation de RAM au prix d'itérations multiples sur les données. Considérons maintenant la requête Q2 de la figure 5. L'évaluation de cette requête en pipeline nécessite de former l'arbre de manière à parcourir Doctor en premier. Vu que la relation Doctor contient des docteurs distincts, les tuples arrivant à l'opérateur *count* sont naturellement groupés par docteur. Le cas de Q3 est complexe. Les données devant être groupées par type de Drug, une jointure additionnelle est nécessaire entre la relation Drug et le domaine Drug.type. Les valeurs du domaine étant uniques, cette jointure produit les tuples dans l'ordre adéquat. Le cas de Q4 est plus complexe encore. Le résultat doit être groupé sur deux attributs (Doctor.id et Drug.type), nécessitant de commencer le parcours à partir des deux relations. La solution est d'insérer un produit cartésien en bas de l'arbre pour produire les tuples ordonnés par Doctor.id et Drug.type. Dans ce cas, le temps de réponse de la requête est environ n fois plus grand (n étant le nombre de types de Drug distincts) que celui de la même requête sans clause de groupement.



Q1: *Qui a prescrit des antibiotiques depuis 2000 ?*

Q2: *Nombre d'antibiotiques prescrits par docteur en 1999.*

Q3: *Nombre de prescriptions par type de médicament*

Q4: *Nombre de prescriptions par docteur et type de médicament*

Figure 5. *Plans d'exécution 'pipeline extrême droit'*

4. Conclusion

PicoDBMS apporte donc une solution effective pour la protection de données confidentielles contre les attaques mentionnées dans le chapitre 2. Ce haut degré de sécurité est obtenu grâce aux 3 propriétés suivantes : (1) la protection matérielle de la puce ; (2) les données et le code sont embarqués sur la puce ; et (3) PicoDBMS est suffisamment simple pour ne pas nécessiter d'administration.

Les limites de PicoDBMS, et plus généralement de l'approche 'tout embarqué' résident précisément en ces trois points. En effet, la protection matérielle est obtenue au prix de fortes contraintes sur les ressources. Par exemple, les prototypes de cartes à puces les plus avancés sont limités à 1 mégaoctet d'espace de stockage. De plus, l'accès aux données embarquées ne peut évidemment se faire que lorsque la carte est physiquement connectée au réseau, limitant ainsi l'applicabilité de l'approche aux applications de type dossiers portables sécurisés. Il est raisonnable de penser qu'à long terme ces contraintes pourraient disparaître, permettant d'embarquer une base de donnée de taille importante ainsi qu'un SGBD plus puissant qui pourrait même être connecté au réseau, comme un véritable serveur de bases de données. Malheureusement, cette évolution conduirait inévitablement à un SGBD plus complexe nécessitant un administrateur, et par là même, réintroduirait le problème des attaques de l'administrateur.

5. Résultats

Le design initial de PicoDBMS a été publié en 2000 [BBP00] et a été récompensé par le Best Paper Award de VLDB [PBV+01]. Un prototype complet de PicoDBMS a été initialement développé en JavaCard [Sun99] et démontré à la conférence VLDB'2001 [ABB+01]. Ce prototype validait le fonctionnement des techniques proposées mais ses performances étaient très faibles. Il a été réécrit en langage C (et optimisé dans un souci de performance), et récemment adapté à 'Zeplatform', le système d'exploitation servant de base à la prochaine génération de cartes à puce d'Axalto. Trois ans ont été nécessaires à l'obtention de ces cartes et à l'adaptation (avec l'aide d'Axalto) du noyau du système d'exploitation pour satisfaire les exigences de performances des applications embarquées orientées données. Un banc d'essai dédié aux bases de données de type PicoDBMS a été défini et utilisé pour juger des performances relatives des différentes structures de stockage et d'indexation des données [ABP05a]. Un article concluant cette étude a été récemment soumis à la revue TODS [ABP05b] dans lequel nous reconsidérons le problème initial à la lumière des évolutions matérielles des cartes et des nouveaux besoins applicatifs. Une analyse de performance détaillée de notre prototype y est présentée ainsi que d'importantes perspectives de recherche dans le domaine de la gestion de données dans les puces sécurisées.

Les travaux relatifs à PicoDBMS ont constitué une partie du travail de thèse de Christophe Bobineau [Bob02] et de Nicolas Anciaux [Anc04].

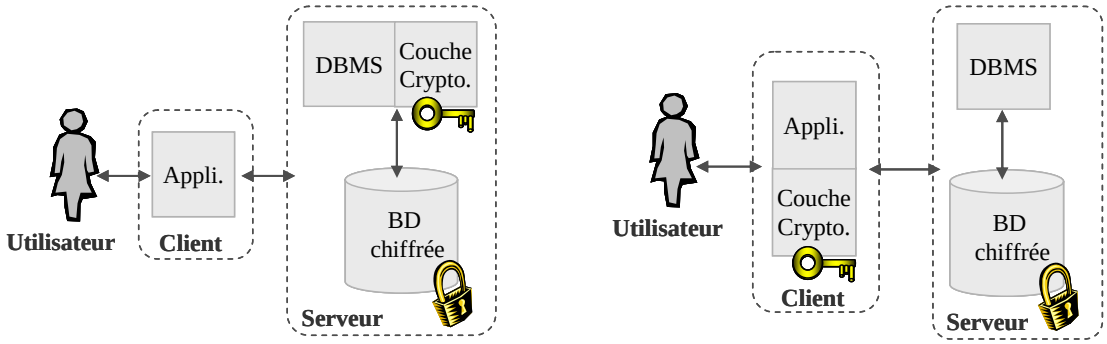
Chapitre 4

C-SDA, une approche hybride utilisant un SGBD relationnel

Ce chapitre présente Chip-Secured Data Access (C-SDA), une solution basée sur le chiffrement de la base de données côté serveur et sur un traitement sécurisé dans une puce, côté client. Dans une première section, nous étudions les techniques existantes, à base de chiffrement seul, et montrons qu'elles ne permettent pas de se protéger contre les attaques d'un administrateur malveillant, ou qu'elles ne permettent pas le partage de données entre plusieurs clients. Nous présentons alors C-SDA qui suit une approche hybride (chiffrement + puce sécurisée). Son principe général est présenté dans la section 2 et les éléments les plus significatifs de la solution sont détaillés dans la section 3. Les limitations de C-SDA sont discutées dans la section 4. Finalement, nous présentons les résultats les plus significatifs de cette étude.

1. Etat de l'art

Les éditeurs de SGBD proposent des outils cryptographiques pour chiffrer les données de la base. DB2 UDB inclut des fonctions de chiffrement permettant de chiffrer les tuples au niveau de chaque attribut [IBM03]. Oracle fournit un package PL/SQL 'Oracle Obfuscation Toolkit' permettant de chiffrer des chaînes de caractères et des données binaires [Ora01, Ora02]. Finalement, Microsoft SQL Server 2000 permet de chiffrer les communications réseau ainsi que les métadonnées (e.g. la définition des vues). L'architecture fonctionnelle présentée figure 6.a est représentative de ces méthodes. Le chiffrement permet de se protéger contre des attaques externes. Cependant, les données sont déchiffrées sur le serveur lors du traitement des requêtes, les clés étant soit transmises par le client, soit conservées sur le serveur. Ainsi, ces techniques ne permettent pas de résister aux attaques d'un DBA car celui-ci possède le moyen d'intercepter les clés, de substituer le package de chiffrement par un autre [HeW01], ou encore, peut espionner la mémoire du serveur lors de l'exécution. Dans [IMM+04], un modèle de stockage optimisé pour supporter un chiffrement partiel des données est présenté. Ce modèle, inspiré du modèle PAX [ADH+01] décompose les tuples verticalement, chaque colonne étant chiffrée séparément afin de minimiser la quantité de données à déchiffrer. Cette solution souffre cependant des mêmes défauts que les approches commerciales.



(a) Chiffrement côté serveur. *(b) Chiffrement côté client.*

Figure 6. *Architectures de SGBD exploitant le chiffrement*

Une solution complémentaire au chiffrement a été récemment proposée pour protéger la base de données des attaques de l'administrateur. Protegrity [Mat04] introduit une distinction claire entre le rôle du DBA, qui administre les ressources de la base de données et le rôle du SA (Administrateur de Sécurité), qui administre les privilèges des utilisateurs, les clés, et d'autres paramètres ayant trait à la sécurité. Cette distinction est aussi faite au niveau système en séparant physiquement le serveur de bases de données et le serveur de sécurité. La sécurité de la solution repose donc sur une séparation stricte des serveurs et des rôles (DBA / SA), une attaque nécessitant alors une association entre le DBA et le SA. Bien que Protegrity soit compatible avec la plupart des SGBD, il n'apporte qu'une protection relative puisque les données sont toujours déchiffrées sur le serveur lors du traitement des requêtes.

Pour contourner cette faiblesse, plusieurs études académiques proposent d'effectuer le déchiffrement du côté client [HIM02, HIL+02, DDJ+03], correspondant alors à l'architecture fonctionnelle de la figure 6.b. Ainsi, la couche cryptographique ainsi qu'une partie du traitement des requêtes sont déportés sur le client. La requête initiale est divisée en deux sous requêtes. La première est évaluée sur le serveur, directement sur les données chiffrées, grâce à des index flous ajoutés aux données chiffrées, et retourne un sur ensemble du résultat. La seconde sous requête est traitée sur le client, après déchiffrement des résultats de la première sous requête. Remarquons que l'utilisation d'index plus précis permet d'optimiser les performances mais augmente le risque d'inférence d'information non autorisée à partir des données chiffrées. [DDJ+03] étudie l'équilibre entre précision et confidentialité. Remarquons que ces approches ont été proposées pour protéger des bases de données externalisées chez un DSP (Database Service Provider ou fournisseur de services bases de données) mais n'ayant pas vocation à être partagées entre plusieurs utilisateurs ayant des droits différents.

2. Approche

Ainsi, les techniques déchiffrant les données sur le serveur ne permettent pas de se protéger contre un administrateur malveillant, ou contre un intrus ayant acquis les droits de l'administrateur. Celles, déchiffrant les données sur le client, ne permettent pas le partage (car la connaissance des index permettrait d'inférer des informations non autorisées) et nécessitent la présence d'index à partir desquels il est possible d'inférer certaines informations.

Nous avons proposé, dans [BoP02], de déporter du SGBD l'ensemble des fonctions de sécurité (gestion des droits, des vues des clés, déchiffrement, etc.) dans un environnement sécurisé (SOE), qui s'interface entre le client et le serveur. En effet, le déchiffrement des données ne peut s'effectuer sur le serveur (risques d'attaques du DBA), ni sur le client (risques d'attaques du client ou pas de partage). Ce module de sécurité appelé C-SDA est embarqué dans une carte à puce et agit comme un médiateur incorruptible entre le client et le serveur.

Le schéma de la base de données ainsi que les données sont chiffrées par C-SDA au niveau de chaque attribut en respectant la contrainte suivante : $\forall a, \forall b, a=b \Leftrightarrow Chiffre(a)=Chiffre(b)$. Lorsque le client soumet une requête, celle-ci est interceptée par C-SDA. Les droits sont alors vérifiés. La requête, si elle met en jeu des vues, est traduite en une requête portant sur les relations de base. C-SDA transmet au serveur une partie de la requête, qui effectue le maximum de traitement sur les données chiffrées. Le résultat est transmis à la carte à puce, déchiffré puis traité par C-SDA qui complète éventuellement l'exécution de la requête, délivrant uniquement les données autorisées au client. La difficulté est d'obtenir au final, un processus suffisamment performant malgré les fortes contraintes des cartes à puce. La section suivante détaille certains éléments de la solution, le lecteur pouvant se référer à l'article initial en annexe B.

3. Contribution

La requête Q doit être décomposée en une partie Q_S évaluable par le serveur sur les données chiffrées et son complément Q_C à évaluer par la puce sur le résultat partiel, après déchiffrement. Il est fondamental que Q_C puisse être exécutée par la puce et non par le terminal pour éviter d'externaliser des données non autorisées. Par exemple, un utilisateur peut avoir le droit de consulter le résultat d'un calcul d'agrégat sans pour autant posséder le droit de consulter les données élémentaires à partir desquelles cet agrégat est calculé.

Pour que ce principe soit applicable sur une carte à puce, il faut que l'évaluation de Q_C par C-SDA puisse se faire en respectant les contraintes inhérentes à la carte (cf. chapitre 3, section 2). Ces contraintes imposent d'évaluer Q_C sans jamais matérialiser de résultats temporaires car la très faible capacité de la RAM ne permet pas une telle matérialisation et le coût d'écriture de l'EEPROM rend également cette matérialisation impossible en mémoire stable. La solution proposée consiste à exécuter l'intégralité de Q_C en mode *pipeline* (c.à.d, tuple à tuple). Bien que ce mode d'évaluation ne soit pas traditionnel, surtout pour les requêtes faisant intervenir des groupements et calculs d'agrégats, nous pouvons montrer que toute requête Q_C est évaluable en pipeline en suivant le principe d'exécution suivant.

La forme générale d'une requête SQL est :

Select	liste d'attributs, fonctions
From	liste de relations
Where	qualification
Group by	liste d'attributs
Having	qualification sur fonction agrégat
Order by	critère de tri

La sémantique opérationnelle de SQL précise que le traitement doit être équivalent à :

1. Calcul du produit cartésien des relations impliquées dans la clause **From**
2. Restriction aux seuls tuples produits en (1) satisfaisant la clause **Where**
3. **Groupement** des tuples issus de (2) ayant même valeur pour la liste d'attributs spécifiée
4. Calcul des fonctions agrégats apparaissant dans les clauses Select, Having et Order by
5. Restriction aux groupes produits en (4) satisfaisant la qualification de la clause **Having**
6. Projection des tuples issus de (5) sur la liste d'attributs et fonctions de la clause **Select**
7. Tri des tuples issus de (6) en fonction du critère de tri spécifié par la clause **Order by**

Le serveur est capable d'exécuter sur des données chiffrées les étapes suivantes, soit Q_S :

1. Calcul du produit cartésien des relations impliquées dans la clause **From**

2s. Restriction aux seuls tuples produits en (1) satisfaisant les prédicats d'égalité présents dans la qualification de la clause **Where**

3. **Groupement** des tuples issus de (2s) ayant même valeur pour la liste d'attributs spécifiée

6s. Projection des tuples issus de (3) sur la liste d'attributs résultat de l'union des listes d'attributs des clauses Select, Group by, Order by ainsi que de la liste d'attributs nécessaires à l'évaluation des prédicats non traités en 2s de la clause Where et des prédicats de la clause Having.

C-SDA est pour sa part contraint d'exécuter sur la carte à puce en pipeline les étapes suivantes :

Pour chaque tuple *t* résultat de Q_S, faire :

2c. Vérifier les prédicats de la clause **Where** non traités en 2s
4. Agréger dans une variable tampon la valeur de(s) attribut(s) de *t* impliqué(s) dans l'évaluation d'une fonction agrégat (apparaissant dans les clauses Select, Having et Order by) et ce, tant que *t* appartient au même groupe que le tuple précédent.
5. Lorsque la fin de groupe est détectée et que les fonctions agrégats sont calculées, évaluer la qualification de la clause **Having**.
6c. S'il est sélectionné, projeter le tuple issu de (5) sur la liste d'attributs et de fonctions spécifiée dans la clause **Select**

A noter que la clause **Order by** peut être systématiquement déportée sur le terminal car elle est sans effet vis à vis de la gestion des droits et constitue donc une troisième requête Q_t.

L'évaluation d'une requête SQL Q se fait de la façon suivante : $Q = Q_t(Q_C(Q_S))$. Ce principe d'évaluation respecte bien les contraintes de la carte puisque l'espace mémoire consommé se réduit à la mémorisation d'un tuple courant et de quelques variables tampons destinées au calcul des fonctions agrégats.

Remarques sur la performance des exécutions

Le mode d'évaluation présenté ci-dessus est sous-optimal pour deux raisons :

(1) L'évaluation de Q_S, telle que présentée précédemment, laisse à penser que les produits cartésiens sont exécutés en premier (étape 1). En fait, l'évaluation de tout ou partie de 2s et de 6s peut être réalisée avant l'étape 1 en suivant les techniques traditionnelles d'optimisation de requêtes implantées sur le serveur.

(2) Les prédicats d'inégalité (restrictions, inéqui-jointures) ne pouvant être évalués directement par le serveur, le volume de données transférées vers C-SDA peut être très supérieur au volume du résultat final. Il est nécessaire d'évaluer les prédicats d'inégalité au plus tôt afin de profiter de leur sélectivité pour simplifier l'ensemble de la chaîne de traitements. Cela nécessite un pré-traitement coopératif entre C-SDA et le serveur. Soit une requête Q contenant un prédicat de la forme $\sigma_{ai\ \theta\ valeur}(T)$, où σ dénote l'opérateur de restriction, $\theta \in \{<, >, \leq, \geq\}$ et a_i est un attribut quelconque de la table T. Le principe d'optimisation consiste pour C-SDA à envoyer au serveur une requête de pré-traitement $Q_S^P = \Pi_{ai}(T)$. C-SDA exécute alors $Q_C^P = \sigma_{ai\ \theta\ valeur}(\Pi_{Déchiffre(ai)}$

(Q_S^P))) et renvoie le résultat R au serveur. C-SDA transforme ensuite la requête Q en remplaçant le prédicat initial $\sigma_{ai\ \theta\ valeur}(T)$ par le prédicat $\propto(T, R)$, où $\propto$ dénote l'opérateur de semi-jointure. Ce principe s'applique de façon similaire pour les inéqui-jointures. Le principe général d'évaluation d'une requête contenant des prédicats d'inégalité devient alors : $Q = Q_t(Q_C(Q_S(*[Q_C^P\ (Q_S^P)])))$, la composition $Q_C^P\ (Q_S^P)$ se répétant pour chaque prédicat d'inégalité.

Remarques sur la robustesse de l'approche

Pour augmenter la robustesse de l'approche, nous proposons dans [BoP02] (1) d'utiliser plusieurs clés de chiffrement en fragmentant les tables verticalement (i.e., différents attributs sont chiffrés avec différentes clés) et/ou horizontalement (i.e., plusieurs clés sont utilisées pour un même attribut) ; (2) d'utiliser un protocole de communication sécurisé entre le client et le serveur pour éviter une attaque de l'utilisateur qui 'écouterait' les communications ; et (3) de stocker les données particulièrement sensibles dans la puce. L'idée consiste à supprimer les données sensibles de la base stockée sur le serveur, en les remplaçant par des indices dans un *domaine sensible* (ensemble de données sans doublons), stocké lui, sur la carte à puce. Ce 'codage' des données sensibles peut être vu comme un *chiffrement particulier* de ces données, chiffrement dont la particularité est d'être définitivement incassable sans la carte à puce.

Remarques sur la gestion des droits d'accès et des vues

La gestion des droits faisant partie intégrante du mécanisme de sécurité, celle-ci ne peut en aucun cas être déléguée au serveur de données. Autrement, le DBA n'aura aucune difficulté à s'octroyer tous les droits sur l'ensemble des données. La gestion des droits doit donc être assurée par C-SDA. Si la carte à puce est responsable de la vérification des droits, elle ne peut par contre pas stocker elle-même la définition de ces droits du fait du partage entre les utilisateurs et de la dynamicité de ces derniers. Par conséquent, la définition des droits est stockée chiffrée sur le serveur et est chargée dynamiquement (ou simplement rafraîchie) sur la carte lors de la connexion de l'utilisateur au serveur.

La puissance du mécanisme de droits des SGBD relationnels provient de la capacité d'affecter un droit à un utilisateur sur des *vues*, c'est à dire des tables virtuelles calculées par une requête SQL. La conséquence en est que C-SDA doit intégrer la gestion des vues au même titre que la gestion des droits.

4. Conclusion

Chip-Secured Data Access (C-SDA) agit donc comme un médiateur incorruptible, embarqué dans une carte à puce, entre un client et un serveur de base de données. Du fait des contraintes physiques de la carte à puce, nous avons proposé un processus d'exécution basé sur un mode de chiffrement assurant que deux valeurs égales soient chiffrées de la même manière. Cette contrainte permet d'obtenir un processus d'évaluation relativement simple et efficace.

Cette contrainte sur le mode de chiffrement peut cependant permettre de mener des attaques statistiques sur les données chiffrées (e.g. la valeur de l'attribut 'ville' la plus fréquente dans une tables de personnes correspondra certainement à 'Paris'). Cette solution suppose, de plus, qu'il n'y a pas d'association entre l'utilisateur et l'administrateur du serveur, sans quoi, un utilisateur malicieux pourrait modifier, même aléatoirement les données chiffrées pour essayer d'obtenir des données non autorisées.

L'approche proposée est cependant prometteuse : (1) le volume de données dans la base n'est pas contraint ; (2) C-SDA permet de partager des données, d'affecter dynamiquement des droits différents à plusieurs utilisateurs en dissociant complètement l'aspect chiffrement des considérations sur les droits ; (3) C-SDA permet d'obtenir des performances raisonnables. L'amélioration de cette approche nécessite de développer de nouvelles techniques pour assurer la sécurité des traitements délégués au serveur.

5. Résultats

Le design de C-SDA a été publié en 2002 [BoP02]. Un prototype a été réalisé en JavaCard avec l'aide de l'Agence Nationale pour la VAlorisation de la Recherche (ANVAR) et a été démontré à la conférence VLDB'2003 [BDP+03]. La démonstration, faite sur l'exemple d'une base de données d'entreprise hébergée par un DSP, illustre les différentes fonctionnalités implantées par C-SDA et notamment l'optimisation de requêtes complexes. Le procédé mis en œuvre par C-SDA a été breveté par le CNRS en août 2001 [BoP01] et une demande de PCT (brevet international) a été faite en août 2002 pour les USA, l'Europe, le Canada et le Japon.

Chapitre 5

Approches hybrides basées sur un stockage externe non sécurisé

Ce chapitre reconsidère l'approche précédente en généralisant l'architecture et en étendant le spectre des attaques considérées à l'ensemble des attaques présentées dans le chapitre 2. Dans cette architecture, aucune confiance n'est accordée au serveur, limitant les traitements qui lui sont délégués et nécessitant d'intégrer des vérifications au niveau du composant sécurisé pour garantir que les données n'ont pas été altérées, substituées ou rejouées par un utilisateur malicieux ou par un pirate. Dans une première section, nous présentons cette architecture générique en caractérisant la confiance accordée à chaque élément. Nous présentons alors deux approches qui exploitent des techniques de chiffrement, des composants matériels sécurisés ainsi que des techniques de contrôle d'intégrité. Ces deux approches sont cependant extrêmement restreintes en termes de traitement de requêtes. Finalement, nous présentons nos travaux en cours dans ce contexte et esquissons une approche possible pour supporter le traitement de requêtes SQL en conservant le même degré de sécurité. Cette dernière partie est moins précise que les précédentes, étant basée sur des travaux en cours de réalisation.

1. Architecture générique

L'architecture de référence, présentée figure 7 inclus 3 éléments, chacun avec un degré de confiance différent. L'environnement d'exécution sécurisé (SOE) est l'unique élément de confiance. Il assure un stockage sécurisé et un environnement d'exécution de confiance, ne laissant transparaître aucune information. Il possède cependant des ressources limitées (e.g. carte à puce). A l'inverse, l'environnement de non confiance (Untrusted Operating Environment ou UOE) peut être la cible des attaquants. Le UOE n'offre aucune garantie de sécurité sur les données qu'il détient, ni sur les traitements effectués et nécessite donc d'utiliser des techniques de chiffrement et d'intégrité. Finalement le terminal utilisateur (Rendering Terminal ou RT) est le moyen par lequel le résultat est délivré à l'utilisateur. Ainsi, le RT a le même degré de confiance et les mêmes droits que l'utilisateur sans quoi, il ne pourrait délivrer les résultats. Le

RT ne doit pas avoir accès à des données non autorisées, ni à des résultats temporaires car l'utilisateur peut facilement attaquer le RT.

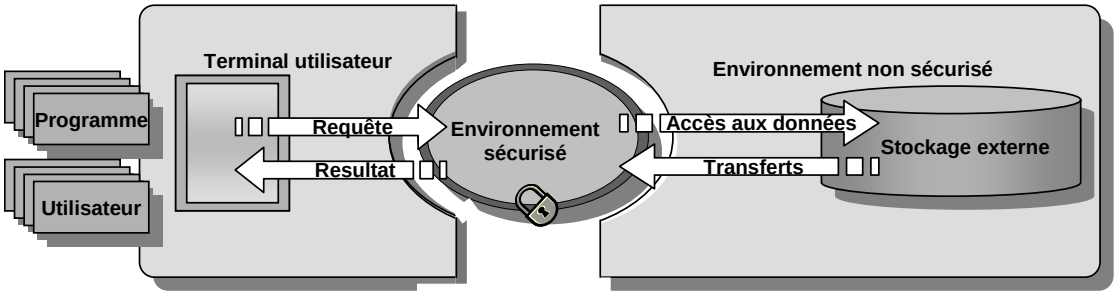


Figure 7. *Architecture générique.*

2. Etat de l'art

Deux solutions, appelées Trusted-DataBase (TDB) [VMS02, MVS00] et GnatDB [Vin02], ont été proposées (par la même équipe). Ces solutions ciblent les applications nécessitant un certain de degré de confiance pour des programmes s'exécutant sur des hôtes de non confiance. Par exemple, pour la gestion de droits digitaux (Digital Right Management ou DRM), les distributeurs de contenu digital doivent contrôler l'usage que font les utilisateurs du contenu délivré. La politique de gestion des droits digitaux est implantée par un programme (un interpréteur de licences) qui, bien que s'exécutant sur un terminal non sécurisé, doit être fiable.

La première solution, TDB, vise une architecture dotée d'un environnement sécurisé (SOE) de type co-processeur sécurisé comme le co-processeur IBM4758 [Swe99, DLP+01] (CPU puissant, performance cryptographiques améliorées, quelques mégaoctets de RAM et une mémoire stable de type FLASH). La seconde solution est une version limitée de TDB et vise des architectures équipées de SOE de type carte à puce.

Dans TDB, les données sont stockées chiffrées dans le UOE sous la forme de grands blocs (100 octets à 100 kilo-octets) structurés sous forme d'un journal (séquentiellement). L'intégrité des données est assurée grâce à des arbres de hachage de Merkle. Les clés de chiffrement ainsi que les racines de chaque arbre de Merkle sont stockées dans le SOE. Les données sont chiffrées et déchiffrées par le SOE, l'intégrité vérifiée grâce aux arbres de Merkle, enfin, les attaques de type rejeu sont évitées par l'utilisation de compteurs croissants associés aux données. TDB implante quelques mécanismes transactionnels (plusieurs mises à jour peuvent être réalisées de manière atomique). Les requêtes supportées par TDB sont équivalentes à des

requêtes de sélection mono table (Il a en fait une interface d'accès orientée objet). L'analyse de performance basée sur le banc d'essai TPC-B [TPC] montre la faisabilité de l'approche.

GnatDB est une version légère de TDB, qui s'accommode des contraintes des cartes à puces. GnatDB assure le même degré de sécurité que TDB mais n'utilise pas d'arbres de Merkle afin de réduire l'empreinte du code. D'autre part, GnatDB minimise l'utilisation de la RAM et inclus un noyau transactionnel léger (réduit à l'atomicité et à la durabilité). La carte de localisation des données n'est pas hiérarchique, comme dans TDB, mais est stockée dans un tableau, et est écrite séquentiellement à chaque mise à jour. Cette technique limite GnatDB à la gestion de petites collections de données (pas de passage à l'échelle) mais permet de réaliser des mises à jour atomiques, sans surcoût. De même que pour TDB, GnatDB ne permet pas l'exécution de requêtes.

Ces deux solutions utilisent un environnement d'exécution sécurisé, des techniques de chiffrement et d'intégrité, pour assurer un haut degré de sécurité, bien plus élevé en fait, que toutes les approches décrites précédemment (excepté PicoDBMS). Cependant, bien que quelques fonctionnalités bases de données soient supportées, les capacités d'interrogation sont fortement limitées et limitent de la même façon la puissance d'expression des droits d'accès. La section suivante présente une première analyse du problème de l'exécution de requêtes dans le même contexte.

3. Approche

La considération de l'exécution de requêtes, et du même coup, de droits d'accès à grain fin, soulève deux questions importantes : (1) quelle partie du traitement de requêtes peut être déléguée aux éléments de non confiance sans réduire le degré de sécurité ; et (2) comment effectuer le reste du traitement au vu des contraintes du SOE. Ces questions sont abordées dans la suite et quelques pistes sont proposées.

Si l'on compare à l'architecture de type PicoDBMS ou le code du SGBD et la base de données sont embarquées dans le SOE, le degré de sécurité est forcément réduit, dès qu'une partie du traitement ou du stockage des données se fait sur l'UOE, géré par un attaquant potentiel (e.g. l'administrateur de l'UOE). Aussi, les données externalisées et les traitements délégués (i.e., traitements effectués sur le UOE, messages échangés entre le SOE et l'UOE) doivent être opaques et résistant aux attaques.

Bien que les techniques cryptographiques puissent apporter une telle opacité et résistance aux attaques pour les données externalisées, garantir ces propriétés pour les traitements délégués est aujourd'hui un problème ouvert. D'une part, assurer la résistance aux attaques implique que tout traitement délégué à l'UOE doit être vérifié dans le SOE. En fait, cette vérification peut être aussi coûteuse que le traitement lui-même ! Quelques techniques existent cependant pour de simples sélections [GKM+04] mais le problème reste ouvert pour des opérations plus complexes. D'autre part, une opacité totale des traitements délégués semble inatteignable car tout traitement délégué révèle des informations (même si ce traitement est réalisé sur des données chiffrées) qui peuvent être exploitées par un attaquant. L'assurance d'une opacité totale sur des traitements délégués s'avère être un problème de PIR (Private Information Retrieval) [CGK+95].

Une solution possible, mais drastique, pour résoudre ce problème est de réduire les traitements délégués à leur minimum. Cela peut être réalisé en réduisant le UOE à un serveur de blocs chiffrés, chacun caractérisé par une adresse et une taille. Les avantages de cette option sont d'obtenir le plus haut degré d'opacité et de résoudre le problème de la résistance aux attaques. En effet, une telle approche ne révèle que les accès aux blocs de données, le nombre et la taille des messages échangés. De plus, la vérification de l'intégrité des traitements délégués revient à vérifier que chaque bloc récupéré n'a pas été altéré, substitué, ni rejoué. L'intégrité d'un bloc de données peut être facilement vérifiée à l'aide de fonctions de hachage cryptographiques (e.g. CAM). La résistance aux attaques de type substitution et rejeu peut être obtenue en incluant dans le bloc son adresse et une indication de version, informations devant être vérifiées lors de la récupération du bloc.

Le traitement de requête est donc confiné dans le SOE, excepté pour la récupération de blocs de données externalisés, forcément déléguée à l'UOE. La stratégie d'exécution est fortement influencée par le coût d'accès aux données et par la quantité limitée de RAM disponible dans le SOE. En effet, les données externalisées doivent être déchiffrées et contrôlées (intégrité, non substitution, non rejeu) au sein du SOE, augmentant significativement le coût d'accès (par exemple, le coût de déchiffrement et de contrôle est supérieur d'environ deux ordres de magnitude au coût d'accès à la mémoire interne d'une carte à puce). Il semble donc de première importance de réduire l'accès aux seules données utiles au calcul du résultat. A l'extrême, il faudrait ne pouvoir accéder qu'aux données faisant parti du résultat ! Bien que cela ne soit évidemment pas réalisable, une indexation massive des données et un grain fin d'accès

(et du coup, grain fin pour le chiffrement, le contrôle d'intégrité, etc.) peuvent permettre de minimiser les accès inutiles.

La quantité limitée de RAM du SOE a aussi une large influence sur l'exécution de requêtes. En effet, les techniques proposées dans PicoDBMS ou dans [ABP03] ne peuvent être utilisées car elles entraînent un grand nombre d'itérations sur les données, non compatible avec le coût d'accès élevé aux données. De nouveau, une indexation massive est clairement requise dans ce contexte mais ne résout pas totalement le problème à moins que chaque résultat de requête (ou du moins, de vue, pour ce qui est des droits d'accès) ne soit matérialisé, ce qui n'est clairement pas envisageable !

Une approche possible est d'utiliser la petite quantité de RAM disponible pour l'exécution de requêtes et de swapper sur le UOE ou le RT lorsque la RAM est saturée, en adaptant les algorithmes de l'état de l'art à notre contexte (e.g. jointure par tri-fusion). Les données swappées devront cependant avoir le même degré d'opacité et de résistance aux attaques que les données de bases. Ainsi, le coût de swap d'une donnée est grossièrement le double du coût d'accès. Afin de maximiser l'utilisation de la RAM, nous pouvons adopter le principe suivant : à chaque étape de l'exécution, le SOE doit seulement récupérer le sous ensemble de données strictement nécessaire à l'exécution de la tâche courante. Bien que cela semble naturel, cette stratégie est opposée aux techniques classiques d'exécution pour lesquelles des stratégies basées sur la lecture par lot (prefetching) et la gestion de cache sont utilisées pour accroître les performances.

4. Conclusion

L'approche présentée ci-dessus peut donc être vue comme une 'extension' de PicoDBMS, où le stockage des données est délégué à un serveur de non confiance afin de palier aux limitations de PicoDBMS, à savoir, le volume de données et le partage (partage physique de la carte). Les données, stockées sur le serveur de non confiance, sont protégées par des techniques cryptographiques contre les attaques présentées dans le chapitre 2. Le serveur agit comme un serveur de blocs de données et n'effectue aucun traitement sur les données. Au vu du surcoût d'accès aux données, du au déchiffrement et aux contrôles cryptographiques, il apparaît judicieux d'utiliser une indexation massive et de n'accéder qu'aux portions utiles des données (grains d'accès fin). Une adaptation des algorithmes de l'état de l'art doit être réalisée pour

pouvoir travailler avec une quantité très réduite de RAM, à l'aide de swap sécurisé sur l'UOE ou le RT.

Ces travaux ont été initiés avec Nicolas Anciaux durant sa dernière année de thèse. Ils sont donc décris avec plus de précision dans [Anc04] sur la base de l'architecture générique décrite en section 1. Si les bases de cette solution sont déjà présentes dans ce document, il reste beaucoup de chemin à parcourir afin d'obtenir une solution qui soit à la fois sûre et efficace. Ce travail d'investigation se situe à plusieurs niveaux : (1) aux niveau des techniques cryptographiques et de leur utilisation ; (2) au niveau des techniques de stockage (des données chiffrées) et de leur indexation (chiffrée) ; (3) au niveau des communications entre le SOE et l'environnement extérieur ; (4) au niveau des algorithmes implantés dans le SOE pour réaliser les opérations de l'algèbre relationnelle ; (5) au niveau de l'optimisation des requêtes, cette dernière pouvant avoir comme objectif la performance ou une confidentialité accrue (i.e., quels algorithme ou quel plan d'exécution donne le moins d'information au pirate). Enfin, et dès lors que l'on considère un environnement multi-utilisateur, il faut prendre en compte les aspects transactionnels complexifiant significativement l'approche. Il est, par exemple, difficile pour plusieurs SOE de connaître, avec certitude (i.e., même en cas d'attaque) les numéros des dernières versions de données stockées sur un serveur distant.

Chapitre 6

Une approche hybride pour des documents XML

Si le modèle relationnel est le plus utilisé aujourd'hui en bases de données, on ne peut ignorer la poussée considérable de XML, devenu standard de-facto pour décrire, diffuser et partager tout type de données. La prise en compte d'un modèle de données XML (données arborescentes semi-structurées, interrogation XPath ou XQuery) change radicalement le problème de définition et de contrôle des droits d'accès. Les règles de droits prennent la forme d'expressions de chemin régulières qualifiées, elles s'appliquent à des sous-arbres, ont des relations d'inclusion et peuvent rentrer en conflit. Dans la suite de ce chapitre, nous présentons tout d'abord la sémantique des modèles de contrôle d'accès pour les documents XML. Les solutions existantes sont alors rapidement présentées et le besoin de dynamicité, incompatible avec ces solutions est motivé. Nous présentons alors une solution combinant chiffrement et SOE afin de pouvoir évaluer des règles d'accès dynamiques sur un document XML chiffré.

1. Sémantique du contrôle d'accès sur des documents XML

Nous introduisons ci-dessous un modèle simplifié de contrôle d'accès pour XML, inspiré du modèle de Bertino [BCF00] et de celui de Samarati [DDP+02] qui partagent globalement les mêmes principes. Dans ce modèle, les règles de contrôle d'accès prennent la forme d'un triplet <signe, sujet, objet>. *Signe* désigne soit une permission (règle positive), soit une interdiction (règle négative) pour l'opération de lecture. *Sujet* représente le destinataire de la règle. *Objet* correspond aux éléments ou sous-arbres du document XML, identifiés par une expression XPath.

La propagation en cascade des règles est implicite dans ce modèle, ce qui signifie qu'une règle se propage d'un objet à tous ses descendants dans la hiérarchie XML. Etant donné ce mécanisme de propagation et le fait que plusieurs règles peuvent être définies pour un même utilisateur sur un même document, un principe de résolution de conflit est nécessaire. Les conflits sont résolus en utilisant deux principes : L'*Interdiction-Est-Prioritaire* et *Le-Plus-Spécifique-Est-Prioritaire*. Considérons deux règles R1 et R2 de signes opposés. Ces règles peuvent être en conflit soit parce qu'elles sont définies sur le même objet, soit parce qu'elles

sont définies respectivement sur deux objets différents O1 et O2, reliés par une relation ancêtre-descendant (e.g. O1 est l'ancêtre de O2). Dans le premier cas, le principe l'Interdiction-Est-Prioritaire donne priorité à la règle négative. Dans le second cas, le principe Le-Plus-Spécifique-Est-Prioritaire donne priorité à la règle qui s'applique directement sur l'objet par rapport à celle qui est héritée (i.e., R2 est prioritaire sur R1 pour l'objet O2).

Les règles associées à un sujet pour un document définissent une vue autorisée de ce document, qui, suivant le contexte applicatif, peut être interrogée. Du point de vue sémantique, le résultat d'une requête est calculé à partir de la vue autorisée du document considéré (e.g. les prédicats ne peuvent s'appliquer sur des éléments non-autorisés même si ceux-ci n'apparaissent pas dans le résultat de la requête). Cependant, les prédicats de règles d'accès peuvent s'appliquer sur n'importe quelle partie du document initial[6].

2. Etat de l'art

Plusieurs solutions de contrôle d'accès basées sur le chiffrement, ont été proposées afin de résister à d'éventuelles attaques sur les serveurs ou pour permettre la distribution sélective de données. Dans ce dernier contexte, [BCF01] suggère de chiffrer chaque portion du document en fonction des utilisateurs pouvant y accéder, menant à la génération d'un nombre potentiellement très grand de clés de chiffrement (i.e., au pire, 2^n où n est le nombres d'utilisateurs). Ce problème peut être résolu par l'utilisation de clés compatibles [RRN02] qui permettent de déchiffrer une donner avec plusieurs clés. Cependant, les clés compatibles sont basées sur un chiffrement asymétrique très coûteux. [Mis02, Mis03] propose un schéma de chiffrement plus subtil, basé sur une connaissance préalable, pouvant être une clé de déchiffrement ou d'autres informations. Il est par exemple possible de déchiffrer des données sur un patient à partir du moment où l'utilisateur connaît son nom. Les données déchiffrées peuvent, elle-même, contenir des clés permettant de déchiffrer d'autres parties du document. Plus récemment, une approche similaire à [BoP02, HIL02] a été proposée dans un contexte XML [CFB04]. Dans cette approche, le document XML est chiffré sur le serveur et les requêtes XPath sont chiffrées sur le terminal client (au niveau des tags) en utilisant un mode de chiffrement conservant l'égalité. Une partie de la requête peut donc être traitée sur le serveur. Pour ce qui est des valeurs, des index flous sont ajoutés aux données chiffrées dans le même esprit que [HIL02], permettant de

retourner un sur ensemble du résultat. Cette approche est étendue avec des techniques, basées sur les arbres de hachage de Merkle, assurant l'intégrité, l'authenticité et la complétude du résultat.

Ces modèles diffèrent sur plusieurs points : le modèle d'accès aux données (interrogation vs. diffusion), le modèle de droit d'accès, le schéma de chiffrement, le mécanisme de distribution des clés et la granularité de partage. Cependant, ils ont en commun de minimiser la confiance requise sur le terminal client au prix d'un partage statique des données. En effet, quelle que soit la granularité du partage considéré, l'ensemble des données est découpé en sous-parties qui suivent le schéma de partage et chacune d'elle est chiffrée avec une clé différente ou une composition de clés différentes. Ainsi les intersections des règles de contrôle d'accès sont précompilées par le chiffrement. Une fois l'ensemble des données chiffré, toute modification de la définition des règles de contrôle d'accès peut entraîner une modification de la frontière entre les différentes sous-parties et aboutir à un rechiffrement partiel de l'ensemble des données et éventuellement à une redistribution des clés.

Cependant, il y a de nombreuses situations où les règles de contrôle d'accès sont spécifiques à chaque utilisateur, dynamiques et donc difficile à prédire. Considérons par exemple une communauté d'utilisateurs (famille, amis, équipe de recherche) partageant des données via un DSP ou d'une manière pair à pair (agendas, carnets d'adresses, profils d'utilisateurs, travaux de recherche, etc.). Il est très probable que les politiques de partage changeront au fur et à mesure que la situation initiale évolue (changement des relations entre les utilisateurs, nouveaux partenaires, nouveaux projets avec des intérêts divergents, etc.). Traditionnellement, l'échange d'information médicale est dicté par des politiques strictes de partage pour protéger la vie privée des patients mais ces règles peuvent subir des exceptions dans des situations particulières (e.g. en cas d'urgence) [EBB+03, EBM+03, CuM03], peuvent évoluer avec le temps (e.g. en fonction du traitement en cours du patient) et peuvent être sujet à des autorisations temporaires [KuH00]. De la même manière, rien ne justifie qu'une base de données hébergée ait des règles de contrôle d'accès plus statiques qu'une base de données gérée localement [BoP02]. En ce qui concerne le contrôle parental [PIC], ni les gestionnaires de sites Web ni les fournisseurs d'accès Internet ne peuvent prédire la diversité des règles de contrôle

[6] Remarquons que les règles d'accès (en fait, leur définition) doivent être cachées à l'utilisateur pour éviter toute

d'accès que les parents, avec leurs sensibilités différentes, veulent voir appliquer à leurs enfants. Finalement, la diversité des modèles de publication (lucratifs et non lucratifs) amène à définir des langages de contrôle d'accès complexes comme XrML ou ODRL [XrM, ODR]. Les règles de contrôle d'accès étant plus complexes, le contenu chiffré et les licences sont gérés par des canaux différents, permettant à différents utilisateurs de jouir de différents privilèges sur le même contenu chiffré.

3. Approche

L'objectif est de concevoir de meilleures solutions de gestion du contrôle d'accès sur le client grâce à l'utilisation d'un composant sécurisé (SOE). Le but poursuivi est d'évaluer des règles de contrôle d'accès personnalisées et dynamiques sur un document chiffré passé en entrée, avec comme bénéfice de dissocier les droits d'accès du schéma de chiffrement. Dans ce contexte, nous définissons le problème de la manière suivante :

- *Proposer une évaluation en flux efficace de règles de contrôle d'accès* : L'évaluation doit, en effet, s'adapter aux contraintes mémoire du SOE, interdisant de ce fait toute matérialisation (e.g. construire une représentation DOM du document).

- *Garantir que les informations non autorisées ne sont jamais révélées* : Le contrôle d'accès étant réalisé sur le terminal client, seules les parties autorisées doivent être rendues accessibles aux éléments non sûrs du terminal client.

- *Protéger le document d'entrée contre toute forme de modification illicite* : Sous l'hypothèse que le SOE est sûr, la seule manière de tromper l'évaluateur de contrôle d'accès est de modifier illégalement le document en entrée, par exemple en substituant ou en modifiant des blocs chiffrés de ce document.

4. Contribution

La solution que nous proposons au problème décrit précédemment peut se résumer en quatre points :

1. *Evaluation en flux de règles de contrôle d'accès*

---

inférence.

A première vue, la gestion en flux du contrôle d'accès ressemble au problème bien connu du traitement de requêtes XPath sur des documents en flux, largement étudié dans le contexte de filtrage XML [DiF03, GMO+03, CFG+02]. Ces études considèrent un grand nombre de requêtes XPath avec comme objectif principal de trouver le sous-ensemble produisant une réponse pour un document donné (quelque soit la réponse). L'accent est mis sur l'indexation ou la manière de combiner l'ensemble de ces requêtes. L'un des premiers travaux adressant le problème précis de l'évaluation d'expressions XPath complexes sur des flux XML a été réalisé par [PeC03] qui propose une solution pour délivrer les parties du document qui satisfont une requête XPath unique. La nature de notre problème diffère largement des travaux précédents. En effet, le principe de propagation des règles et les politiques de résolution de conflits associées rendent les règles d'accès dépendantes entre elles. L'interférence entre les règles introduit deux problèmes importants :

- *Evaluation de règles d'accès* : Pour chaque nœud du document d'entrée, l'évaluateur doit être capable de déterminer l'ensemble des règles qui s'y applique et pour chaque règle déterminer si elle s'applique directement ou si elle est héritée. L'imbrication de la portée des règles d'accès détermine si ce nœud sera ou non autorisé.

- *Optimisation du contrôle d'accès* : L'imbrication de la portée des règles associé à la résolution de conflits peut inhiber l'effet de certaines règles. L'évaluateur de règles doit pouvoir tirer avantage de cette inhibition pour suspendre l'évaluation de ces règles et éventuellement de toutes les règles si une décision globale peut être établie pour un sous-arbre donné.

Comme nous considérons des documents arrivant en flux, nous supposons que l'évaluateur de règles est alimenté par un parseur basé sur des évènements (e.g. SAX [SAX]) qui déclenchent les évènements ouverture, valeur et fermeture respectivement pour chaque ouverture, texte et fermeture de tag du document en entrée. Chaque règle d'accès (i.e., expression XPath) est représentée par un automate non-déterministe à états finis (NFA) [HoU79] composé d'un *chemin de navigation*, et éventuellement d'un ou plusieurs *chemins de prédicats.*

Très grossièrement, chaque automate passe à l'état suivant lorsqu'un évènement d'ouverture ou une valeur le concernant est reçu. Il revient à l'état précédent lors d'évènements de fermeture. Afin de gérer efficacement ces automates, nous utilisons une pile qui stocke tous

les états actifs, matérialisant les chemins qui peuvent être suivis dans les automates non-déterministes. Pour qu'une règle s'applique, il faut que tous les états finaux soient atteints (i.e., de navigation et de prédicats). Une règle est dite *en attente* si l'état final du chemin de navigation est atteint alors que l'état final d'un des chemins de prédicat n'est pas encore atteint, signifiant que la valeur d'un des prédicat de la règle n'est pas encore connue au moment où sa cible (sous arbre concerné par la règle) est rencontrée. Finalement, la propagation des règles et les conflits sont gérés grâce à une pile d'autorisation, stockant en haut de pile, le signe courant qui est propagé si aucune autre règle ne s'applique.

Ces techniques, détaillées dans l'article joint en annexe C, permettent d'évaluer en flux des règles de contrôle d'accès supportant un sous-ensemble conséquent du langage XPath.

2. Index de Saut

Nous proposons une structure compacte d'indexation (voir [ABC04, BGK03]) pour les données en flux qui permet (i) de converger rapidement vers les parties autorisées du document d'entrée en sautant les parties non autorisées, et (ii) de calculer les intersections avec une requête potentiellement appliquée au document (dans un contexte pull). Indexer est particulièrement important compte tenu du fait que les deux facteurs limitants de l'architecture cible sont le coût du déchiffrement dans le SOE et le coût de communication entre le SOE, le client et le serveur. Cette seconde contribution est complémentaire de la première pour atteindre l'objectif de performance.

En gardant à l'esprit la nécessité d'avoir un index compact, l'information structurelle minimale requise pour atteindre notre objectif est l'ensemble des tags qui sont présents dans chaque sous-arbre ainsi que la taille de ce dernier (nécessaire pour pouvoir le sauter le cas échéant). Bien que ces métadonnées ne prennent pas en compte l'imbrication des tags, elles se révèlent comme étant une manière très efficace de filtrer les expressions XPath non pertinentes. Le codage de ces informations est réalisé de manière récursive. Pour chaque noeud, l'ensemble des tags présents est construit grâce à un tableau de bits se référant soit au dictionnaire de tags du document [ABC04, TPh02] pour le premier noeud soit au tableau de bits du noeud père. Le tag du nœud est lui-même codé par une référence au tableau de bits du nœud père, codé sur un nombre de bits minimal. Enfin, le même principe est utilisé pour la taille du sous-arbre : elle est codée sur le nombre de bits minimum par rapport à la taille du sous arbre du noeud père. Ces techniques permettent d'obtenir un index ayant un surcoût très faible (cf. étude de performance

dans l'article joint) puisque la taille des noeuds diminue avec leur profondeur (encodage sur un nombre de bits de plus en plus petit).

L'index permet de désactiver rapidement les règles qui n'atteindront pas un état final, à savoir, dès que les tags nécessaires pour atteindre l'état final ne sont pas présents dans la liste de tags du sous arbre, matérialisée par le tableau de bits. Lorsque l'ensemble de règles actives restantes permet de prendre une décision à propos d'un sous arbre, ce dernier est soit délivré (autorisation) soit sauté (interdiction ou décision dépendante d'un prédicat en attente (cf. ci-dessous)) grâce à l'information de taille présente dans l'index.

3. Gestion des prédicats en attente

Les prédicats en attente (i.e., un prédicat conditionnant une autorisation ou une interdiction pour un sous-arbre S mais qui est rencontré après S lors de l'analyse du document) sont difficiles à gérer en flux. Nous proposons une stratégie permettant de détecter les parties en attente du document, de les sauter lors de l'analyse (grâce à l'index de saut) puis de réassembler au bon endroit celles qui sont pertinentes vis à vis du résultat final. La manière dont les prédicats en attente sont gérés garantit que les données non autorisées ne sont jamais révélées sur le terminal client. Plus de détails sont donnés dans l'article joint.

4. Vérification de l'intégrité sur des fragments du document

Afin de résister aux attaques par examen des données, altération ou substitution, mentionnées dans le chapitre 2, le document est chiffré de manière opaque (i.e., mode Cipher Block Chaining ou CBC [Sch96]) et des informations d'intégrité et d'authenticité sont ajoutées. Le problème est compliqué par les accès aléatoires en avant et en arrière générés par l'utilisation de l'index de Saut et par la gestion des prédicats en attente. Des arbres de hachage de Merkle [Mer90] sont utilisés pour supporter efficacement ce type d'accès sans avoir à systématiquement vérifier l'intégrité de l'intégralité du document.

5. Conclusion

Au niveau de l'approche, cette proposition est similaire à celle proposée dans le chapitre précédent; à savoir : une protection logicielle utilisant du chiffrement et du hachage alliée à une protection matérielle permettant de déchiffrer et vérifier l'intégrité sur le terminal client dans un environnement sécurisé. Elle s'en distingue par la structure arborescente des données XML et

par la spécificité du modèle de contrôle d'accès, entraînant une évaluation totalement différente à base d'automates non-déterministes à états finis et d'index sur la structure du document.

Ces techniques pourraient être améliorées par une meilleure utilisation des techniques d'inclusion de requêtes [ACL01] afin de raffiner les optimisations avant et pendant l'évaluation des règles d'accès ainsi que par la définition de méthodes plus précises d'indexation en flux (notamment en ce qui concerne les valeurs). Le support d'attaque à base de rejeu doit être aussi pris en compte dans le modèle de protection cryptographique.

6. Résultats

Ces travaux ont constitués le corps de la thèse de François Dang Ngoc [Dan05]. Un premier design a été publié au VLDB 2004 [BDP04a]. Plusieurs prototypes ont été développés pour valider l'approche. Pour les études de performances, un prototype écrit en C et tournant sur un simulateur matériel de carte à puce fourni par Axalto a permis de faire des mesures prometteuses et démontrant la viabilité de notre solution. Depuis, deux prototypes Javacard, tournant sur de vraies cartes à puce (e-gate [Axa04b] et SIMera [Axa04c]), ont été développés dans le but de démontrer l'intérêt applicatif de notre technologie. Le premier, appelé C-SXA [BDP04b] illustre le partage sécurisé de document sur l'exemple d'un agenda partagé. Le deuxième, appelé MobiDiQ, a été proposé dans un contexte de téléphonie mobile. MobiDiQ est un composant embarqué dans une carte SIM qui garanti un modèle d'accès équitable à des contenus digitaux, préservant l'intérêt des utilisateurs et des fournisseurs de contenu avec de fortes assurances de sécurité et de confidentialité. Le contrôle d'accès est basé sur le profil de l'utilisateur et sur les descripteurs de contenus, exprimés en XML, et permet de mettre en place des politiques d'accès évoluées. Ces prototypes ont respectivement reçus les médailles d'argent et d'or des concours e-gate open 2004 [Axa04a] et SIMagine 2005 [Axa05] montrant ainsi l'intérêt du monde industriel pour les solutions basées sur un élément sécurisé. Dans [BDP05a], nous décrivons ces expériences applicatives et illustrons notre approche par divers scénarios pour des applications mobiles. Enfin, un dernier prototype Javacard a été développé pour mettre en exergue les aspects techniques de notre solution et a été présenté à la conférence ACM Sigmod 2005 [BCD+05]. Un article complétant et concluant ces travaux vient récemment d'être soumisse au journal ACM TISSEC [BDP05b]

Chapitre 7

Conclusion et travaux futurs

Dans ce document, nous avons donc présenté une comparaison des différentes alternatives pour garantir la confidentialité des données, à l'aide de techniques cryptographiques et/ou de composants matériels spécialisés.

Les approches reposant uniquement sur le chiffrement ont un degré de confidentialité et un niveau de fonctionnalité dépendant de l'endroit où le chiffrement/déchiffrement est effectué. Lorsque le déchiffrement est fait sur le serveur, l'ensemble des fonctionnalités bases de données sont conservées mais l'approche n'est pas résistante à des attaques de la part d'administrateurs. Le degré de sécurité est augmenté lorsque le déchiffrement est effectué sur le client rendant cependant impossible un partage contrôlé des données (i.e., différents utilisateurs avec différents droits). Ainsi, cette approche est limitée à la gestion sécurisée de bases de données privées.

Un niveau de sécurité inégalé est obtenu lorsque le SGBD et la base de données sont embarqués dans un environnement sécurisé. Actuellement, des approches basées sur des cartes à puces ont été proposées. Bien que limitées par les contraintes des cartes à puces, ces solutions sont bien adaptées pour la gestion sécurisée de dossiers personnels, comme le dossier médical, ainsi que pour un ensemble plus large d'applications interagissant avec des objets électroniques intelligents dans un environnement d'intelligence ambiante. A moyen terme, l'évolution des puces sécurisés, en termes de ressources (e.g. débit de communication, mémoire de stockage, etc.) devrait élargir encore le spectre applicatif de cette approche.

La combinaison de techniques cryptographiques et de composants matériels sécurisés permet de garantir un haut degré de confidentialité tout en préservant d'importantes propriétés des SGBD comme le partage contrôlé de données, de grands volumes de données et de bonnes performances. Peu de travaux suivant cette approche ont été proposés. C-SDA est une de ces propositions et utilise un mode de chiffrement particulier permettant d'effectuer une partie des traitements directement sur les données chiffrées en les déléguant au serveur. Cette méthode donne de bonnes performances mais peut révéler des informations sensibles par une étude

statistique des données chiffrées, notamment pour un utilisateur connaissant une partie des données. Bien que l'approche soit prometteuse, elle est donc limitée à des scénarios où il n'y a pas de collusion entre le client et le serveur.

Lorsque le serveur se trouve dans un environnement non sécurisé, il faut protéger les données stockées contre la corruption, la substitution ou le rejeu. Deux systèmes, représentatifs de cette approche existent actuellement mais sont limités à des interrogations sommaires (sélections mono table), restreignant du même coup la finesse d'expression des droits d'accès aux données. La complexité du problème augmente considérablement dès que le traitement de requête est pris en compte. Nous avons présenté des idées préliminaires permettant de réaliser des traitements complexes, sans réduire pour autant le degré de sécurité.

Finalement, nous avons décris une proposition similaire au niveau de l'approche mais dans un contexte XML. La structure arborescente des données XML et la spécificité du modèle de contrôle d'accès entraînent une évaluation totalement différente à base d'automates non-déterministes à états finis et d'index sur la structure du document.

La suite de ce chapitre de conclusion présente nos perspectives de recherches liées à nos travaux sur la confidentialité des données, en incluant ceux sur l'ubiquité qui, relativement aux approches proposées, sont intimement liés. Ces perspectives sont structurées en trois aspects, à savoir, (1) l'évolution matérielle des composants sécurisés ; (2) le type de contrôle que l'on veut effectuer sur les données ; et (3) les différentes approches au niveau architectural (i.e., quel élément fait quel traitement).

1. Evolution matérielle des composants sécurisés

Environnement sans contact : les constructeurs et les organisations gouvernementales [SIN] portent un intérêt grandissant aux puces sans contact lié à leur facilité d'utilisation. Bien que les techniques proposées dans ce document puisse être intégrées tel quel dans une carte sans contact, l'environnement diffère par de multiples aspects, ce qui peut conduire à une conception différente. Par exemple, le temps d'exposition au lecteur est extrêmement court dans un environnement sans contact. La remise en cause de l'objectif en terme de temps de réponse (considéré jusqu'alors comme étant de l'ordre de la seconde), impose de nouvelles techniques d'exécution. Ceci peut conduire à d'intéressantes problématiques visant à exécuter des requêtes

très rapidement, quitte à fournir un résultat approché (évitant au porteur de stationner devant le lecteur).

Utilisation de nouvelles technologies de mémoire persistante : nous avons considéré dans ce document des puces basées sur une technologie de mémoire EEPROM. Cependant, l'EEPROM ayant atteint sa limite de scalabilité d'après certains constructeurs, de nouvelles technologies comme la FLASH font leur apparition dans les puces. La mémoire de type FLASH présente ses propres caractéristiques qui doivent être prises en compte dans le design de composants embarqués. L'étude mentionnée dans le chapitre 3 [BSS+03] s'intéresse exclusivement au stockage d'un petit volume de données (centaines de tuples) dans une mémoire FLASH. Une étude considérant de plus larges volumes et orientée vers l'évaluation de droits d'accès complexes est nécessaire. De même, les technologies alternatives à long terme (PCM, OUM ou Millipèdes), pourraient être envisagées. Ces mémoires exhibent aussi des spécificités ayant un impact direct sur le modèle de stockage et d'indexation des données. Des travaux existent dans un contexte plus général sur les mémoires Millipèdes [YAE03]. La combinaison de ces propriétés mémoires avec les autres contraintes des puces constitue clairement un problème intéressant.

Smart Secure Mass Storage Cards (SSMSC) : La SSMSC (Smart Secure Mass Storage Cards) est un nouveau composant qui peut-être vu comme la conjonction sur une même plate-forme matérielle d'un microprocesseur hautement sécurisé de type carte à puce et d'une mémoire de stockage de masse de type FLASH, pouvant atteindre plusieurs Gigaoctets. Ces principes peuvent être mis en œuvre dans différents contextes : carte à puce classique dans laquelle on insère un composant de mémoire de masse FLASH, clé USB ou une carte MMC dans laquelle on insère une puce sécurisée classique. La mémoire FLASH de la SSMSC ne jouit pas de la protection matérielle de celle-ci et les données stockées dans cette mémoire doivent donc être protégées. Il apparaît donc clairement que la SSMSC n'est autre qu'une instanciation de l'architecture générique présentée dans le chapitre 5. Des réflexions avancées sont menées sur cette architecture et sur la mise en place des principes proposés en chapitre 5, dans le cadre d'un projet avec le conseil général des Yvelines, visant à réaliser un dossier médico-social embarqué sur une SSMSC. Remarquons tout d'abord que malgré l'augmentation des capacités du processeur et de la taille de la mémoire de travail (RAM) de la puce sécurisée de la SSMSC, les contraintes matérielles subsistent car ces augmentations sont à mettre en regard avec l'augmentation considérable du volume de données à traiter (potentiellement plusieurs

gigaoctets de données!). La technologie mémoire de la SSMSC vient compliquer le problème car la mémoire FLASH a des caractéristiques très particulières en termes de temps d'accès en lecture et en écriture et surtout de mode d'utilisation (e.g. écriture en séquence, effacement par gros blocs pour la mise à jour) qui doivent être prises en compte lors du stockage, de l'indexation et de l'interrogation des données.

Calibrage des ressources : les ressources matérielles embarquées ont un impact immédiat sur le coût de la puce, surtout lorsque celle-ci concerne un marché de masse. Il est donc particulièrement important de calibrer au plus juste les ressources matérielles à intégrer. Nous avons mené dans ce contexte une première étude du calibrage de la RAM [ABP03]. En effet, la RAM présente une cellule de très faible densité, occupant en moyenne le tiers de la puce, et représentant le tiers de son prix (le coût d'une puce est directement lié à sa taille). Cette ressource est donc cruciale. Les recherches futures menées sur ce point devraient viser à calibrer la puce selon ses trois dimensions principales (processeur, RAM, mémoire persistante) selon le besoin de l'application. La consommation processeur peut être minimisée par l'utilisation intensive d'index et la matérialisation, et la quantité de mémoire persistante peut être modulée par compression des données de bases et des structures accélératrices. Bien sûr, les différentes dimensions du problème ne sont pas indépendantes. Tous cela fait de la co-conception de la puce face au besoin de l'application un défi particulièrement intéressant.

2. Type de contrôle

Modèle de contrôle d'accès : Si jusqu'à présent, nous nous sommes focalisés sur les modèles de contrôle d'accès de type DAC (Discretionary Access Control), beaucoup de travail reste à faire afin de mieux appréhender l'ensemble des modèles et leur mise en œuvre. En particulier, leur mise en œuvre très souvent centralisée ne permet pas de refléter la complexité d'organisations largement distribuées (fédération de bases de données, architectures pair-à-pair). Nous avons démarré des collaborations avec l'ENST Bretagne, qui a défini un modèle décentralisé intitulé ORBAC (ORganization Based Access Control) mieux adapté à ces contraintes [EBM+03]. Le modèle ORBAC permet ainsi d'exprimer des règles basées sur trois abstractions (les rôles, les vues et les activités). Les règles peuvent spécifier des permissions, des interdictions, des obligations ou des recommandations. Par exemple, un médecin pourra consulter le dossier d'un

patient d'un collègue, à condition qu'il en indique le motif. Nous souhaitons nous intéresser notamment à la sécurisation de l'administration d'un tel modèle de contrôle d'accès, à l'aide de composants matériels sécurisés.

Sémantique du contrôle d'accès XML : [CCS05, FMP05] proposent des extensions du modèle de contrôles d'accès présenté dans le chapitre 6. Comme indiqué précédemment, les règles de droits prennent généralement la forme d'expressions de chemin régulières qualifiées. Les règles peuvent ainsi s'appliquer à des sous arbres ayant des relations d'inclusion et rentrer en conflit entre elles. La sémantique des modèles de contrôle d'accès pour XML n'est pas claire dans un certain nombre de situations (ex : documents récursifs, nœuds autorisés accessibles par un chemin prohibé, etc.). Par ailleurs, les modèles actuels ne permettent pas d'exprimer de façon explicite des règles d'autorisation portant sur les associations entre nœuds. Pour combler ces limites, [FMP05] propose trois classes de règles d'autorisations pour les associations entre nœuds (la dépersonnalisation d'un ancêtre, la réduction de chemins et la décorrélation d'informations entre frères). Ce modèle de contrôle d'accès basé sur des règles permet de masquer les relations existant entre un nœud et – certains de – ses ancêtres et frères. Ce modèle possède un fort pouvoir d'expression, tout en gardant un fort degré de concision et une sémantique non ambiguë. Le support d'une telle sémantique dans une puce pose des problèmes délicats, relatifs aux contraintes de la puce pendant le calcul, plus complexe, de la vue autorisée.

Puces hippocratiques : Le concept de SGBD Hippocratique, à savoir de SGBD donnant l'assurance du respect d'un serment de confidentialité, a été introduit dans [AKS+02]. Un tel SGBD se doit de respecter un ensemble de principes fondateurs parmi lesquels : préciser l'objectif d'utilisation de chaque donnée collectée sur un utilisateur et recueillir l'assentiment de l'utilisateur sur cet objectif, ne stocker que l'information strictement nécessaire à l'atteinte de cet objectif et uniquement pendant le laps de temps strictement nécessaire, ne pas divulguer cette information à des tiers sans autorisation préalable de l'utilisateur, donner à l'utilisateur la possibilité de consulter les informations qui le concernent et enfin offrir des outils permettant à un tiers de contrôler que ces principes sont bien respectés. Ces principes sont tous très séduisants et gagneraient à être intégrés dans les dispositifs informatiques disséminés dans un environnement d'intelligence ambiante. Ainsi les différents capteurs participant à cet environnement donneraient des garanties de confidentialité en rapport avec leur tâche et avec l'objectif des requêtes qui lui seraient adressées. L'intérêt de considérer ce problème dans le contexte de puces sécurisées est que des solutions semblent possibles puisque ces puces peuvent

exécuter des traitements sans que l'utilisateur (ou le pirate) ne puisse interférer. Les contributions possibles dans ce cadre seraient d'ordre sémantique (e.g, comment définir l'objectif, l'assentiment de l'utilisateur, etc.), algorithmique (e.g. comment vérifier efficacement les principes hippocratiques, quelles données doivent être conservées, comment , etc.) ou au niveau des protocoles (e.g, qui garantie quoi, comment, etc.).

3. Architectures

Modèles d'accès aux données chiffrées : Différents modèles d'accès à des données chiffrées peuvent être considérés. On peut par exemple distinguer les modèles d'accès de type pull, dans lesquels un client émet une requête à destination d'un serveur contenant des données chiffrées, des modèles d'accès de type push, dans lesquels un serveur diffuse des données chiffrées à destination d'une population de clients susceptibles de les déchiffrer. Chaque modèle d'accès pose des problèmes spécifiques de décomposition et d'optimisation des traitements (partie évaluable sur données chiffrées, partie évaluable uniquement sur données en clair, affectation de ces différentes parties aux unités de calcul disponibles dans l'architecture en fonction de leur niveau de sécurité). Nous souhaitons également nous intéresser au problème de confidentialité dans les architectures pair-à-pair dont la caractéristique est d'imposer un contrôle fortement décentralisé dans un environnement très dynamique.

Délégation de traitements : Dans le chapitre 5, nous avons pris une approche drastique en réduisant les traitements délégués à leur minimum. Une perspective intéressante est cependant de proposer des techniques permettant de déléguer, de manière sure, des traitements. L'intérêt d'une telle approche serait d'améliorer les performances, notamment lors de la manipulation de grandes bases de données, en profitant de la puissance de calcul des serveurs. Cependant, cet approche pose plusieurs problèmes intéressants. D'une part, l'utilisation de modes de chiffrement particuliers (e.g. comme dans [BoP02]) ou l'ajout d'index aux données chiffrées (e.g. comme dans [HIL+02]) peuvent révéler de l'information. Ce problème est délicat car, dès que le serveur a les moyens de faire un traitement quelconque sur les données chiffrées, il est probable qu'un pirate puisse faire de même, et utilise ces possibilités pour extraire de l'information non autorisée. D'autre part, la justesse des traitements délégués doit être vérifiée puisque le serveur n'est pas de confiance. Bien que des techniques existent pour des sélections

simples [GKM+04] (à base d'arbre de hachage de Merkle), le problème reste entier pour des requêtes plus complexes. Une possibilité serait d'ajouter du côté serveur un composant sécurisé, plus puissant, comme le co-processeur sécurisé IBM4758. Le fait de considérer des composants sécurisés du côté serveur ouvre certainement la voie à des contributions intéressantes en termes de répartition des traitements.

Bibliographie

[ABB+01] N. Anciaux, C. Bobineau, L. Bouganim, P. Pucheral, P. Valduriez, 'PicoDBMS: Validation and Experience'. *27th International Conference on Very Large Data Bases (VLDB)*, September 2001.

[ABC04] A. Arion, A. Bonifati, G. Costa, S. D'Aguanno, I. Manolescu, A. Puglies, 'Efficient Query Evaluation over Compressed Data', *9th International Conference on Extending Database Technology (EDBT)*, March 2004.

[ABP03] N. Anciaux, L. Bouganim, P. Pucheral: 'Memory Requirements for Query Execution in Highly Constrained Devices'. *29th International Conference on Very Large Data Bases (VLDB)*, September 2003.

[ABP05a] N. Anciaux, L. Bouganim, P. Pucheral, 'SGBD Embarqué dans une Puce – Etude de PicoDBMS', Submitted to *Techniques et Sciences Informatiques (TSI)*, 2005.

[ABP05b] N. Anciaux, L. Bouganim, P. Pucheral, 'Smart Card DBMS: where are we now?', Submitted to *ACM Transactions on Database Systems (ACM TODS)*, 2005.

[ACL01] S. Amer-Yahia, S. Cho, L. Lakshmanan, and D. Srivastava, "Minimization of tree pattern queries", *27th International Conference on Management of Data (SIGMOD)*, June 2001.

[ADH+01] A. G. Ailamaki, D. J. DeWitt, M. D. Hill, M. Skounakis, 'Weaving Relations for Cache Performance', *27th International Conference on Very Large Data Bases (VLDB)*, September 2001.

[AHK85] A. Ammann, M. Hanrahan, and R. Krishnamurthy. 'Design of a Memory Resident DBMS', *IEEE COMPCON*, 1985.

[AKS+02] R. Agrawal, J. Kiernan, R. Srikant, Y. Xu, 'Hippocratic Databases'. *28th International Conference on Very Large Data Bases (VLDB)*, September 2002.

[Anc04] N. Anciaux, 'Database Systems on Chip', Thèse de doctorat de l'Université de Versailles – Saint Quentin en Yvelines, Décembre 2004

[AnK96] R. Anderson, M. Kuhn, "Tamper Resistance – a Cautionary Note", *USENIX Workshop on Electronic Commerce*, 1996.

[App03] Application Security Inc., 'Encryption of Data at Rest - Database Encryption', *White Paper*, 2002. http://www.appsecinc.com

[AWA03] The Aware Home Research Initiative – Georgia Institute of Technology - http://www.cc.gatech.edu/fce/ahri/

[Axa04a] Axalto, e-gate open 2004, Worldwide USB smart card developer contest. Second edition, held at CTST, Washington DC, USA, http://www.egateopen.axalto.com.

[Axa04b] Axalto. e-gate USB smart card. http://www.axalto.com/infosec/egate.asp . 2004

[Axa04c] Axalto. SIMera - Classic SIM Card. http://www.axalto.com/wireless/classic.asp. 2004.

[Axa05] Axalto SIMagine 2005, Worldwide Mobile Communication and Java Card™ developer contest. Sixth edition, held at 3GSM, Cannes, France, http://www.simagine.axalto.com.

[BBP00] C. Bobineau, L. Bouganim, P. Pucheral, P. Valduriez, 'PicoDBMS: Scaling down Database Techniques for the Smartcard', *26th International Conference on Very Large Data Bases (VLDB)*, September 2000.

[BCD+05] L. Bouganim, C. Cremarenco, F. Dang Ngoc, N. Dieu, P. Pucheral : 'Safe Data Sharing and Data Dissemination on Smart Devices'. *31th International Conference on Management of Data (SIGMOD)*, June 2005.

[BCF00] E.Bertino, S.Castano, E.Ferrari, M.Mesiti, 'Specifying and Enforcing Access Control Policies for XML Document Sources', WWW Journal, (3) 3, 2000.

[BCF01] E. Bertino, S. Castano, E. Ferrari, 'Securing XML documents with Author-X', IEEE Internet Computing, 2001.

[BDP+03] L. Bouganim, F. Dang Ngoc, P. Pucheral, L. Wu : 'Chip-Secured Data Access: Reconciling Access Rights with Data Encryption'. *29th International Conference on Very Large Data Bases (VLDB)*, September 2003.

[BDP04a] L. Bouganim, F. Dang Ngoc, P. Pucheral : 'Client-Based Access Control Management for XML documents'. *30th International Conference on Very Large Data Bases (VLDB)*, September 2004.

[BDP04b] L. Bouganim, F. Dang Ngoc, P. Pucheral : 'A Smart XML Access Right Controller for Mobile Applications'. *5th e-Smart Conference*, September 2004.

[BDP05a] L. Bouganim, F. Dang Ngoc, P. Pucheral, 'Tamper-Resistant Ubiquitous Data Management', *Int. Journal of Computer Systems Science and Engineering*,20(2), 2005.

[BDP05b] L. Bouganim, F. Dang Ngoc, P. Pucheral : 'Client-Based Access Control Management for XML documents'. Submitted to *ACM Transactions on Information and System Security (ACM TISSEC)*, 2005.

[BDV96] L. Bouganim, B. Dageville, P. Valduriez: 'Adaptive Parallel Query Execution in DBS3'. *5th International Conference on Extending Database Technology (EDBT)*, March 1996.

[BeL76] D. E. Bell, L. J. LaPadula, 'Secure computer systems: Unified exposition and multics interpretation', Technical Report ESD-TR-73-306, The MITRE Corporation, 1976.

[BFD96] L. Bouganim, D. Florescu, B. Dageville: 'Skew handling in the DBS3 Parallel Database System'. *3rd International Conference of the Austrian Center for Parallel Computation, ACPC'96*, Klagenfurt, September 1996.

[BFM+00] L. Bouganim, F. Fabret, C. Mohan, P. Valduriez, 'A Dynamic Query Processing Architecture for Data Integration Systems'. *IEEE Data Engineering Bulletin*, 23(2), 2000.

[BFP+01] L. Bouganim, F. Fabret, F. Porto, P. Valduriez, 'Processing Queries with Expensive Functions and Large Objects in Distributed Mediator Systems', *17th International Conference on Data Engineering (ICDE)*, April 2001.

[BFV+00] L. Bouganim, F. Fabret, P. Valduriez, C. Mohan, 'Dynamic Query Scheduling in Data Integration Systems', *16th International Conference on Data Engineering (ICDE)*, March 2000.

[BFV96] L. Bouganim, D. Florescu, P. Valduriez: 'Dynamic Load Balancing in Hierarchical Parallel Database Systems', *22nd International Conference on Very Large Data Bases (VLDB)*, September 1996.

[BFV99] L. Bouganim, D. Florescu, P. Valduriez, 'Load Balancing for Parallel Query Execution on NUMA Multiprocessors', *Distributed and Parallel Databases, DAPD*, 7(1), 1999.

[BGK03] P. Buneman, M. Grobe, C. Koch, 'Path Queries on Compressed XML', *29th International Conference on Very Large Data Bases (VLDB)*, September 2003.

[BKV98] L. Bouganim, O. Kapitskaia, P. Valduriez: 'Memory Adaptive Scheduling for Large Query Execution', *7th International Conference on Information and Knowledge Management (CIKM)*, November 1998.

[Bob02] C. Bobineau, 'Gestion de transactions en environnement mobile', Thèse de doctorat de l'Université de Versailles – Saint Quentin en Yvelines, Décembre 2002.

[BoP01] L. Bouganim, P. Pucheral, 'Procédé de sécurisation de bases de données', Dépôt par le CNRS du brevet français n°01/10552 le 07/08/2002, délivré le 30/01/04. Demande de PCT (brevet international) 'Method for Making Database Secure' n° PCT/FR02/02824 pour USA, Europe, Canada, Japon, 07/08/02.

[BoP02] L. Bouganim, P. Pucheral: 'Chip-Secured Data Access: Confidential Data on Untrusted Servers'. *28th International Conference on Very Large Data Bases (VLDB)*, August 2002.

[Bou96] L. Bouganim, 'Equilibrage de charge lors de l'exécution parallèle de requêtes sur des architectures multiprocesseurs hybrides', Thèse de doctorat de l'Université de Versailles – Saint Quentin en Yvelines, Décembre 1996.

[Bou97] L. Bouganim, 'Exécution parallèle de requêtes relationnelles et équilibrage de charge', *Calculateurs Parallèles*, 9(3), 1997.

[BPS96] A. Baraani, J. Pieprzyk, R. Safavi-Naini, 'Security In Databases: A Survey Study', 1996. citeseer.nj.nec.com/baraani-dastjerdi96security.html

[BSS+03] C. Bolchini, F. Salice, F. Schreiber, L. Tanca, 'Logical and Physical Design Issues for Smart Card Databases', *ACM Transactions on Information Systems (TOIS)*, 2003.

[CaB02] The Caspio Bridge DSP. www.caspio.com/bridge.htm

[Car99] L. C. Carrasco, 'RDBMS's for Java Cards ? What a Senseless Idea !', 1999. http://www.sqlmachine.com

[CBB93] P. Casadessus, P. Borla-Salamet, L. Bouganim: 'Une Expérience de Conception d'un Gestionnaire Transactionnel dans un Environnement Parallèle', *9èmes journées Bases de Données Avancées (BDA)*, Août 1993.

[CCS05] F. Cuppens, N. Cuppens-Boulahia, T. Sans, 'Protection of relationships in XML documents with the XML-BB model', *1st International Conference on Information Systems Security (ICISS)*, 2005.

[CDB95] P. Casadessus, B. Dageville, L. Bouganim, 'Performance du SGBD parallèle DBS3 sur la machine KSR1', *Ingénierie des Systèmes d'Information, ISI*, 3(1), 1995.

[CFB04] B. Carminati, E. Ferrari, E. Bertino, 'Assuring Security Properties in Third-party Architectures', Technical Report UNINSUBRIA.D.2.

[CFG+02] C. Chan, P. Felber, M. Garofalakis, R. Rastogi, 'Efficient Filtering of XML Documents with Xpath Expressions', *18th International Conference on Data Engineering (ICDE)*, March 2002.

[CGK+95] B. Chor, O. Goldreich, E. Kushilevitz, M. Sudan, 'Private information retrieval ', *Symposium on Foundations of Computer Science (FOCS)*, 1995.

[CSI04] Computer Security Institute. 'CSI/FBI Computer Crime and Security Survey', 2004. http://www.gocsi.com/forms/fbi/pdf.html.

[CuM03] F. Cuppens, A. Miège, 'Modelling contexts in the Or-BAC model' . *19th Annual Computer Security Applications Conference*, December 2003.

[Cup00] F. Cuppens, 'Modélisation formelle de la sécurité des systèmes d'informations', Habilitation à Diriger les Recherches, Université Paul Sabatier, 2000.

[Dan05] F. Dang-Ngoc, 'Sécurisation matérielle du contrôle d'accès à des documents XML', Thèse de doctorat de l'Université de Versailles – Saint Quentin en Yvelines, 2005

[DDJ+03] E. Damiani, S. De Capitani Vimercati, S. Jajodia, S. Paraboschi, P. Samarati, 'Balancing Confidentiality and Efficiency in Untrusted Relational DBMSs', *ACM Conference on Computer and Communications Security (CCS)*, 2003.

[DDP+02] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, 'A Fine-Grained Access Control System for XML Documents', *ACM Transactions on Information and System Security (ACM TISSEC)*, (5)2, 2002.

[DiF03] Y. Diao, M. Franklin, 'High-Performance XML Filtering: An Overview of YFilter', *19th International Conference on Data Engineering (ICDE)*, April 2003.

[DLP+01] J. G. Dyer, M. Lindemann, R. Perez, R. Sailer, L. van Doorn, S. W. Smith, S. Weingart, 'Building the IBM 4758 Secure Coprocessor', *IEEE Computer*, 2001.

[EBB+03] A. El Kalam, P. Balbiani, S. Benferhat, F. Cuppens, Y. Deswarte, R. El Baida, A. Miège, C. Saurel et G. Trouessin, 'Modèles et politiques de sécurité des systèmes de santé'. *1ère conférence francophone en gestion et ingénierie des systèmes hospitaliers*, janvier 2003.

[EBM+03] A. El Kalam, S. Benferhat, A. Miege, R. Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, G. Trouessin, 'Organization based access control', *IEEE 4th International Workshop on Policies for Distributed Systems and Networks*, 2003.

[eCr02] The eCriteria DSP. www.ecriteria.net

[EFF] Electronic Frontier Foundation, 'Unintended Consequences: Five Years under the DMCA'. http://www.eff.org/IP/DMCA/

[Eru01] Eruces Inc., 'Securing Data Storage: Protecting Data at Rest', In Dell Power Solutions magazine, Issue 4, 2001. http://ftp.dell.com/app/4q01-Eru.pdf

[Eur85] European Directive 95/46/EC, 'Protection of individuals with regard the processing of personal data', Official Journal L 281, 1985.

[FMP05] B.Finance, S. Medjdoub, P. Pucheral, 'The Case for Access Control on XML Relationships', *14th ACM International Conference on Information and Knowledge Management (CIKM)*, November 2005.

[GaB01] A. Gabillon and E. Bruno, “Regulating access to XML documents. *IFIP Working Conference on Database and Application Security*, 2001.

[Gem92] Gemplus. CQL Language Reference Manual, Gemplus documentation, 1992.

[GIG01] E. Giguère, “Mobile Data Management: Challenges of Wireless and Offline Data Access”, *17th International Conference on Data Engineering (ICDE)*, April 2001.

[GKM+04] M. Gertz, A. Kwong, C. Martel, G. Nuckolls, P. Devanbu, S. Stubblebine, ‘Databases that tell the Truth: Authentic Data Publication’, *Bulletin of the Technical Committee on Data Engineering*, 2004.

[GMO+03] T. Green, G. Micklau, M. Onizuka, D. Suciu, ‘Processing XML streams with Deterministic Automata’, *9th International Conference on Database Theory (ICDT)*, 2003.

[Gra93] G. Graefe. “Query Evaluation Techniques for Large Databases”, *ACM Computing Surveys*, 25(2), 1993.

[Gra98] G. Graefe. “The New Database Imperatives”, *14th International Conference on Data Engineering (ICDE)*, April 1998.

[Gri92] G. Grimonprez,. "Etude et réalisation d’une carte à microprocesseur intégrée aux SGBD", Mémoire d’Habilitation à diriger la Recherche, Université de Lille I, France, 1992.

[HeW01] J. He, M. Wang, “Cryptography and Relational Database Management Systems”, *Int. Database and Engineering and Application Symposium (IDEAS)*, 2001.

[HIL+02] H. Hacigumus, B. Iyer , C. Li, S. Mehrotra, ‘Executing SQL over Encrypted Data in the Database-Service-Provider Model’, *28th International Conference on Management of Data (SIGMOD)*, June 2002.

[HIM02] H. Hacigumus, B. Iyer , S. Mehrotra, ‘Providing Database as a Service’, *18th International Conference on Data Engineering (ICDE)*, April 2002.

[HoU79] J. Hopcroft, J. Ullman, ‘Introduction to Automata Theory, Languages and Computation’, Addison-Wesley, 1979.

[HRU76] M. A. Harrison, W. L. Ruzzo, J. D. Ullman, ‘Protection in Operating Systems’, *Communication of the ACM*, 19(8):461-471, 1976.

[IBM03] IBM corporation, ‘IBM Data Encryption for IMS and DB2 Databases v. 1.1’, 2003. http://www-306.ibm.com/software/data/db2imstools/html/ibmdataencryp.html.

[IMM+04] B. Iyer , S. Mehrotra, E. Mykletun, G. Tsudik, Y. Wu, ‘A Framework for Efficient Storage Security in RDBMS’, International Conference on Extending Database Technology, 2004.

[ImN02] T. Imielinski, B. Nath, “Wireless Graffiti – Data, data everywhere”, *28th International Conference on Very Large Data Bases, 28th International Conference on Very Large Data Bases (VLDB)*, August 2002.

[ISO99] International Standardization Organization (ISO), Integrated Circuit(s) Cards with Contacts – Part 7: Interindustry Commands for Structured Card Query Language (SCQL), ISO/IEC 7816-7, 1999.

[Kap99] O. Kapitskaia, ‘Traitement de requêtes dans les systèmes d'intégration des sources de données distribuées’, Thèse de doctorat de l'Université de Paris 6, Novembre 1999.

[KLL+01] J. S. Karlsson, A. Lal, C. Leung, T. Pham, “IBM DB2 Everyplace: A Small Footprint Relational Database System”, *17th International Conference on Data Engineering (ICDE)*, April 2001.

[KuH00] M. Kudo, S. Hada, ‘XML document security based on provisional authorization’, ACM CCS, 2000.

[Man01] I. Manolescu, ‘Techniques d'optimisation pour l'intégration de données distribuées et hétérogènes’, Thèse de doctorat de l'Université de Versailles, Décembre 2001.

[Mas02] MasterCard, ‘MasterCard Open Data Storage (MODS)’, 2002. https://hsm2stl101.mastercard.net/public/login/ebusiness/smart_cards/one_smart_card/biz_opportunity/mods

[Mat04] U. Mattsson, ‘Transparent Encryption and Separation of Duties for Enterprise Databases -A Solution for Field Level Privacy in Databases’, Protegrity Technical Paper, 2004. http://www.protegrity.com/

[MBF02] I. Manolescu, L. Bouganim, F. Fabret, E. Simon : 'Efficient Querying of Distributed Resources in Mediator Systems’. *10 th International Conference on Cooperative Information Systems, COOPIS 2002*, November 2002.

[Mer90] R. Merkle, ‘A Certified Digital Signature’, *Advances in Cryptology (Crypto'89)*, LNCS, vol.435, Springer--Verlag, 1990.

[MeS93] J. Melton, A. R. Simon, ‘*Understanding the new SQL: A Complete Guide*’, Morgan Kaufmann, 1993.

[MiS02] G. Miklau and D. Suciu, “Containment and equivalence for an XPath fragment”, *ACM Symposium on Principles of Database Systems (PODS)*, 2002.

[MiS03] G. Micklau, D. Suciu, ‘Controlling Access to Published Data Using Cryptography’, *29th International Conference on Very Large Data Bases (VLDB)*, September 2003.

[MiS83] M. Missikov, M. Scholl. 'Relational Queries in a Domain Based DBMS', *9th International Conference on Management of Data (SIGMOD)*, June 1983.

[MOV97] A. Menezes, P. Van Oorschot, S. Vanstone, '*Handbook of Applied Cryptography*', CRC Press, 1997. www.cacr.math.uwaterloo.ca/hac.

[MVS00] U. Maheshwari, R. Vingralek, W. Shapiro, 'How to build a trusted database system on untrusted storage', *Symposium on Operating Systems Design and Implementation (OSDI)*, 2000.

[NIS95] NIST, 'Secure hash standard', FIPS Publication 180-1, 1995.

[ODR] The Open Digital Rights Language Initiative, http://odrl.net/.

[Ora01] Oracle Corporation, 'Database Encryption in Oracle9i', 2001. otn.oracle.com/deploy/security/oracle9i.

[Ora02] Oracle Corporation, Oracle 9i Lite - Oracle Lite SQL Reference", Oracle Documentation, 2002.

[PaV94a] P. Paradinas, J. J. Vandewalle, "A Personal and Portable Database Server: the CQL Card", Proceedings of Application of Databases (ADB'94), June 1994.

[PaV94b] P. Paradinas, J. J. Vandewalle, "How to integrate smart cards in standard software without writing specific code?", CardTech/SecurTech, 1994.

[PBV+01] P. Pucheral, L. Bouganim, P. Valduriez, C. Bobineau, 'PicoDBMS: Scaling down Database Techniques for the Smartcard', *Very Large Data Bases Journal, VLDBJ, 10(2-3)*, 2001. Special issue on the best papers from VLDB'2000.

[PeC03] F. Peng, S. Chawathe, 'XPath Queries on Streaming Data', *29th International Conference on Management of Data (SIGMOD)*, June 2003.

[PIC] W3C consortium, "PICS: Platform for Internet Content Selection", http://www.w3.org/PICS.

[Por01] F. Porto, 'Strategies for parallel execution of queries in distributed scientific database', Thèse de doctorat de la Pontifícia Universidade Católica do Rio de Janeiro (PUC Rio), avril 2001.

[Pri74] The Privacy Act, 5 U.S.C.§552a, 1974. http://www.usdoj.gov/04foia/privstat.htm

[PTV90] P. Pucheral, J. M. Thévenin, P. Valduriez, 'Efficient Main Memory Data Management Using the DBGraph Storage Model', 16th *International Conference on Very Large Data Bases (VLDB)*, August 1990.

[Qck02] The Quickbase DSP. https://www.quickbase.com/

[Riv92] R.L. Rivest, 'The MD5 message-digest algorithm', RFC 1321, 1992.

[RRN02] I. Ray, I. Ray, N. Narasimhamurthi, “A Cryptographic Solution to Implement Access Control in a Hierarchy and More”, *ACM Symposium on Access Control Models and Technologies (SACMAT)* , 2002.

[RSA78] R. L. Rivest, A. Shamir, L. A. Adelman, “A method for obtaining digital signatures and public-key cryptosystems”; Communications of the ACM, 21(2), 1978.

[SAX] Simple API for XML, http://www.saxproject.org/.

[SCF+96] R. Sandhu, E. J. Coyne, H. L. Feinstein, C. E. Youman, ‘Role-based access control models’, *IEEE Computer*, 29(2):38-47, 1996.

[Sch96] B. Schneier, “Applied Cryptography”, 2nd Edition, John Wiley & Sons, 1996.

[SeG01] P. Seshadri, P. Garrett: “SQLServer for Windows CE - A Database Engine for Mobile and Embedded Platforms”, *17th International Conference on Data Engineering (ICDE)*, April 2001.

[SIN] SINCE project. http://www.eurosmart.com/since/index.htm

[Sun99] Sun Microsystems, JavaCard 2.1 Application Programming Interface Specification, JavaSoft documentation, 1999.

[Swe99] S.W. Smith, S.H. Weingart, Building a High-Performance, Programmable, Secure Coprocessor, *Computer Networks* (31) - 1999

[TPC] Transaction Processing Performance Council, http://www.tpc.org/

[TpH02] P. Tolani, J. Haritsa, ‘XGRIND: A Query-Friendly XML Compressor’, *18th International Conference on Data Engineering (ICDE)*, April 2002.

[Tua99] J.-P. Tual, “MASSC: A Generic Architecture for Multiapplication Smart Cards”, *IEEE Micro Journal*, N° 0272-1739/99, 1999.

[Val87] P. Valduriez, ‘Join Indices’, *ACM Transactions on Database Systems (ACM TODS)*, 12(2), 1987.

[Vin02] R. Vingralek, ‘Gnatdb: A small-footprint, secure database system’, *28th International Conference on Very Large Data Bases (VLDB)*, August 2002.

[VMS02] R. Vingralek, U. Maheshwari, W. Shapiro, ‘TDB: A Database System for Digital Rights Management’, *8th International Conference on Extending Database Technology (EDBT)*, March 2002.

[XrM] XrML eXtendible rights Markup Language, www.xrml.org/

[YAE03] H. Yu, D. Agrawal, A. El Abbadi, ‘Tabular Placement of Relational Data on MEMS-based Storage Devices’, *29th International Conference on Very Large Data Bases (VLDB)*, September 2003.

Annexe A

PicoDBMS: Scaling down Database Techniques for the Smartcard

Philippe Pucheral, Luc Bouganim, Patrick Valduriez, Christophe Bobineau

Very Large Data Bases Journal, VLDBJ, 10(2-3), 2001.

Special issue on the best papers from VLDB'2000

The VLDB Journal (2001) 10: 120–132 / Digital Object Identifier (DOI) 10.1007/s007780100047

PicoDBMS: Scaling down database techniques for the smartcard

Philippe Pucheral[1], **Luc Bouganim**[1], **Patrick Valduriez**[2], **Christophe Bobineau**[1]

[1] University of Versailles, PRiSM Laboratory, Versailles, France;
E-mail: {philippe.pucheral;luc.bouganim;christophe.bobineau}@prism.uvsq.fr
[2] University Paris 6, LIP6 Laboratory, Paris, France; E-mail: patrick.valduriez@lip6.fr

Edited by A. El Abbadi, G. Schlageter, K.-Y. Whang. Received: 15 October 2000 / Accepted: 15 April 2001
Published online: 23 July 2001 –

Abstract. Smartcards are the most secure portable computing device today. They have been used successfully in applications involving money, and proprietary and personal data (such as banking, healthcare, insurance, etc.). As smartcards get more powerful (with 32-bit CPU and more than 1 MB of stable memory in the next versions) and become multi-application, the need for database management arises. However, smartcards have severe hardware limitations (very slow write, very little RAM, constrained stable memory, no autonomy, etc.) which make traditional database technology irrelevant. The major problem is scaling down database techniques so they perform well under these limitations. In this paper, we give an in-depth analysis of this problem and propose a PicoDBMS solution based on highly compact data structures, query execution without RAM, and specific techniques for atomicity and durability. We show the effectiveness of our techniques through performance evaluation.

Key words: Smartcard applications – PicoDBMS – Storage model – Execution model – Query optimization – Atomicity – Durability

1 Introduction

Smartcards are the most secure portable computing device today. The first smartcard was developed by Bull for the French banking system in the 1980s to significantly reduce the losses associated with magnetic stripe credit card fraud. Since then, smartcards have been used successfully around the world in various applications involving money, proprietary data, and personal data (such as banking, pay-TV or GSM subscriber identification, loyalty, healthcare, insurance, etc.). While today's smartcards handle a single issuer-dependent application, the trend is toward multi-application smartcards[1]. Standards for multi-application support, like the JavaCard [36] and Microsoft's SmartCard for Windows [26], ensure that the card be universally accepted and be able to interact with several service providers. This should make smartcards one of the world's highest-volume markets for semiconductors [14].

As smartcards become more and more versatile, multi-application, and powerful (32-bit processor, more than 1 MB of stable storage), the need for database techniques arises. Let us consider a health card storing a complete medical folder including the holder's doctors, blood type, allergies, prescriptions, etc. The volume of data can be important and the queries fairly complex (select, join, aggregate). Sophisticated access rights management using views and aggregate functions are required to preserve the holder's data privacy. Transaction atomicity and durability are also needed to enforce data consistency. More generally, database management helps to separate data management code from application code, thereby simplifying and making application code smaller. Finally, new applications can be envisioned, like computing statistics on a large number of cards, in an asynchronous and distributed way. Supporting database management on the card itself rather than on an external device is the only way to achieve very high security, high availability (anywhere, anytime, on any terminal), and acceptable performance.

However, smartcards have severe hardware limitations which stem from the obvious constraints of small size (to fit on a flexible plastic card and to increase hardware security) and low cost (to be sold in large volumes). Today's microcontrollers contain a CPU, memory – including about 96 kB of ROM, 4 kB of RAM, and up to 128 kB of stable storage like EEPROM – and security modules [39]. EEPROM is used to store persistent information; it has very fast read time (60–100 ns) comparable to old-fashion RAM but very slow write time (more than 1 ms/word). Following Moore's law for processor and memory capacities, smartcards will get rapidly more powerful. Existing prototypes, like Gemplus's Pinocchio card [16], bypass the current memory bottleneck by connecting an additional chip of 2 MB of Flash memory to the microcontroller. Although a significant improvement over today's cards, this is still very restricted compared to other portable, less secure, devices such as Personal Digital Assistants (PDA). Furthermore, smartcards are not autonomous, i.e., have no independent power supply, thereby precluding asynchronous and disconnected processing.

[1] Everyone would probably enjoy carrying far fewer cards.

These limitations (tiny RAM, little stable storage, very costly write, and lack of autonomy) make traditional database techniques irrelevant. Typically, traditional DBMS exploit significant amounts of RAM and use caching and asynchronous I/Os to reduce disk access overhead as much as possible. With the extreme constraints of the smartcard, the major problem is scaling down database techniques. While there has been much excellent work on scaling up to deal with very large databases, e.g., using parallelism, scaling down has not received much attention by the database research community. However, scaling down in general is becoming very important for commodity computing and is quite difficult [18].

Some DBMS designs have addressed the problem of scaling down. Light versions of popular DBMS like Sybase Adaptive Server Anywhere [37], Oracle 8i Lite [30] or DB2 Everywhere [20] have been primarily designed for portable computers and PDA. They have a small footprint which they obtain by simplifying and componentizing the DBMS code. However, they use relatively high RAM and stable memory and do not address the more severe limitations of smartcards. ISOL's SQLJava Machine DBMS [13] is the first attempt towards a smartcard DBMS while SCQL [24], the standard for smartcard database language, emerges. While both designs are limited to single select, they exemplify the strong interest for dedicated smartcard DBMS.

In this paper, we address the problem of scaling down database techniques and propose the design of what we call a PicoDBMS. This work is done in the context of a new project with Bull Smart Cards and Terminals. The design has been made with smartcard applications in mind, but its scope extends as well to any ultra-light computer device based on a secured monolithic chip. This paper makes the following contributions:

- We analyze the requirements for a PicoDBMS based on a typical healthcare application and justify its minimal functionality.
- We give an in-depth analysis of the problem by considering the smartcard hardware trends and derive design principles for a PicoDBMS.
- We propose a new pointer-based storage model that integrates data and indices in a unique compact data structure.
- We propose query execution techniques which handle complex query plans (including joins and aggregates) with no RAM consumption.
- We propose transaction techniques for atomicity and durability that reduce the logging cost to its lowest bound and enable a smartcard to participate in distributed transactions.
- We show the effectiveness of each technique through performance evaluation.

This paper is an extended version of [7]. In particular, the section on transaction management is new. The paper is organized as follows. Section 2 illustrates the use of take-away databases in various classes of smartcard applications and presents in more detail the requirements of the health card application. Section 3 analyzes the smartcard hardware constraints and gives the problem definition. Sections 4–6 present and assess the PicoDBMS' storage model, query execution model, and transaction model, respectively. Section 7 concludes.

2 Smartcard applications

In this section, we discuss the major classes of emerging smartcard applications and their database requirements. Then, we illustrate these requirements in further detail with the health card application, which we will use as reference example in the rest of the paper.

2.1 Database management requirements

Table 1 summarizes the database management requirements of the following typical classes of smartcard applications:

- *Money and identification*: examples of such applications are credit cards, e-purse, SIM for GSM, phone cards, transportation cards. They are representative of today's applications, with very few data (typically the holder's identifier and some status information). Querying is not a concern and access rights are irrelevant since cards are protected by PIN-codes. Their unique database management requirement is update atomicity.
- *Downloadable databases*: these are predefined packages of confidential data (e.g., diplomatic, military or business information) that can be downloaded on the card – for example, before traveling – and be accessed from any terminal. Data availability and security are the major concerns here. The volume of data can be important and the queries complex. The data are typically read-only.
- *User environment*: the objective is to store in a smartcard an extended profile of the card's holder including, among others, data regarding the computing environment (PC's configuration, passwords, cookies, bookmarks, software licenses, etc.), an address book as well as an agenda. The user environment can thus be dynamically recovered from the profile on any terminal. Queries remain simple, as data are not related. However, some of the data are highly private and must be protected by sophisticated access rights (e.g., the card's holder may want to share a subset of her/his address book or bookmark list with a subset of persons). Transaction atomicity and durability are also required.
- *Personal folders*: personal folders may be of a different nature: scholastic, healthcare, car maintenance history, loyalty. They roughly share the same requirements, which we illustrate next with the healthcare example. Note that queries involving data issued from different folders can make sense. For instance, one may be interested in discovering associations between some disease and the scholastic level of the card holder. This raises the interesting issue of maintaining statistics on a population of cards or mining their content asynchronously.

2.2 The health card application

The health card is very representative of personal folder applications and has strong database requirements. Several countries (France, Germany, USA, Russia, Korea, etc.) are developing healthcare applications on smartcards [11]. The initial idea was to give to each citizen a smartcard containing her/his identification and insurance data. As smartcard storage capacity increases, the information stored in the card can be

Table 1. Typical applications' profiles

Applications	Volume	Select/project	Join	Group by / Distinct	Access rights / views	Atomicity	Durability	Statistics
Money & identification	tiny					✓		
Downloadable DB	high	✓	✓	✓				
User environment	medium	✓			✓	✓	✓	
Personal folder	high	✓	✓	✓	✓	✓	✓	✓

extended to the holder's doctors, emergency data (blood type, allergies, vaccination, etc.), surgical operations, prescriptions, insurance data and even links to heavier data (e.g., X-ray examination, scanner images, etc.) stored on hospital servers. Different users may query, modify, and create data in the holder's folder: the doctors who consult the patient's past records and prescribe drugs, the surgeons who perform exams and operations, the pharmacists who deliver drugs, the insurance agents who refund the patient, public organizations which maintain statistics or study the impact of drugs correlation in population samples, and finally the holder her/himself.

We can easily observe that: (i) the amount of data is significant (more in terms of cardinality than in terms of volume because most data can be encoded); (ii) queries can be rather complex (e.g., a doctor asks for the last antibiotics prescribed to the patient); (iii) sophisticated access rights management using views and aggregate functions are highly required (e.g., a statistical organization may access aggregate values only but not the raw data); (iv) atomicity must be preserved (e.g., when the pharmacist delivers drugs); and (v) durability is mandatory, without compromising data privacy (logged data stored outside the card must be protected).

One may wonder whether the holder's health data ought to be stored in a smartcard or in a centralized database. The benefit of distributing the healthcare database on smartcards is threefold. First, health data must be made highly available (anywhere, anytime, on any terminal, and without requiring a network connection). Second, storing sensitive data on a centralized server may damage privacy. Third, maintaining a centralized database is fairly complex due to the variety of data sources. Assuming the health data is stored in the smartcard, the next question is why the aforementioned database capabilities need to be hosted in the smartcard rather than the terminals. The answer is again availability (the data must be exploited on any terminal) and privacy. Regarding privacy, since the data must be confined in the chip, so must the query engine and the view manager. As the smartcard is the unique trusted part of the system, access rights and transaction management cannot be delegated to an untrusted terminal.

3 Problem formulation

In this section, we make clear the smartcard constraints in order to derive design rules for the PicoDBMS and state the problem. Our analysis is based on the characteristics of both existing smartcard products and current prototypes [16, 39], and thus, should be valid for a while. We also discuss how the main constraints of the smartcard will evolve in a near future.

3.1 Smartcard constraints

Current smartcards include in a monolithic chip, a 32 bits RISC processor at about 30 MIPS, memory modules (of about 96 kB of ROM, 4 kB of static RAM, and 128 kB of EEPROM), security components (to prevent tampering), and take their electrical energy from the terminal [39]. ROM is used to store the operating system, the JavaCard virtual machine, fixed data, and standard routines. RAM is used as working memory for maintaining an execution stack and calculating results. EEPROM is used to store persistent information. EEPROM has very fast read time (60–100 ns/word) comparable to old-fashion RAM, but a dramatically slow write time (more than 1 ms/word).

The main constraints of current smartcards are therefore: (i) the very limited storage capacity; (ii) the very slow write time in EEPROM; (iii) the extremely reduced size of the RAM; (iv) the lack of autonomy; and (v) a high security level that must be preserved in all situations. These constraints strongly distinguish smartcards from any other computing devices, including lightweight computers like PDA.

Let us now consider how hardware advances can impact on these constraints, in particular, memory size. Current smartcards rely on a well-established and slightly out-of-date hardware technology (0.35μm) in order to minimize the production cost (less than five dollars) and increase security [34]. Furthermore, up to now, there was no real need for large memories in smartcard applications such as the holder's identification. According to major smartcard providers, the market pressure generated by emerging large storage demanding applications will lead to a rapid increase of the smartcard storage capacity. This evolution is however constrained by the smartcard tiny die size fixed to 25 mm^2 in the ISO standard [23], which pushes for more integration. This limited size is due to security considerations (to minimize the risk of physical attack [5]) and practical constraints (e.g., the chip should not break when the smartcard is flexed). Another solution to relax the storage limit is to extend the smartcard storage capacity with external memory modules. This is being done by Gemplus which recently announced Pinocchio [16], a smartcard equipped with 2 MB of Flash memory linked to the microcontroller by a bus. Since hardware security can no longer be provided on this memory, its content must be either non-sensitive or encrypted.

Another important issue is the performance of stable memory. Possible alternatives to the EEPROM are Flash memory and Ferroelectric RAM (FeRAM) [15] (see Table 2 for performance comparisons). Flash is more compact than EEPROM and represents a good candidate for high capacity smartcards [16]. However, Flash banks need to be erased before writing, which is extremely slow. This makes Flash memory appropriate for applications with a high read/write ratio (e.g., address books). FeRAM is undoubtedly an interesting option for smartcards as read and write times are both fast. Although its theoretical foundation was set in the early 1950s, FeRAM is just emerging as an industrial solution. Therefore, FeRAM is expensive, less secure than EEPROM or Flash, and its integration with traditional technologies (such as CPUs) remains an

Table 2. Performance of stable memories for the smartcard

Memory type	EEPROM	FLASH	FeRAM
Read time (/word)	60 to 150 ns	70 to 200 ns	150 to 200 ns
Write time (/word)	1 to 5 ms	5 to 10 μs	150 to 200 ns
Erase time (/bank)	None	500 to 800 ms	None
Lifetime $^{(*)}$ (/cell)	10^5 write cycles	10^5 erase cycles	10^{10} to 10^{12} write cycles

* A memory cell can be overwritten a finite number of time.

issue. Thus FeRAM could be considered a serious alternative only in the very long term [15].

Given these considerations, we assume in this paper a smartcard with a reasonable stable storage area (a few megabytes of EEPROM[2]) and a small RAM area (some kilobytes). Indeed, there is no clear interest in having a large RAM area, given that the smartcard is not autonomous, thus precluding asynchronous write operations. Moreover, more RAM means less EEPROM as the chip size is limited.

3.2 Impact on the PicoDBMS architecture

We now analyze the impact of the smartcard constraints on the PicoDBMS architecture, thus justifying why traditional database techniques, and even lightweight DBMS techniques, are irrelevant. The smartcard's properties and their impact are:

- *Highly secure*: smartcard's hardware security makes it the ideal storage support for private data. The PicoDBMS must contribute to the data security by providing access right management and a view mechanism that allows complex view definitions (i.e., supporting data composition and aggregation). The PicoDBMS code must not present security holes due to the use of sophisticated algorithms[3].
- *Highly portable*: the smartcard is undoubtedly the most portable personal computer (the wallet computer). The data located on the smartcard are thus highly available. They are also highly vulnerable since the smartcard can be lost, stolen or accidentally destroyed. The main consequence is that durability cannot be enforced locally.
- *Limited storage resources*: despite the foreseen increase in storage capacity, the smartcard will remain the lightest representative of personal computers for a long time. This means that specific storage models and execution techniques must be devised to minimize the volume of persistent data (i.e., the database) and the memory consumption during execution. In addition, the functionalities of the PicoDBMS must be carefully selected and their implementation must be as light as possible. The lightest the PicoDBMS, the biggest the onboard database.
- *Stable storage is main memory*: smartcard stable memory provides the read speed and direct access granularity of a main memory. Thus, a PicoDBMS can be considered as a *main memory DBMS* (*MMDBMS*). However the dramatic cost of writes distinguishes a PicoDBMS from a traditional MMDBMS. This impacts on the storage and access methods of the PicoDBMS as well as the way transaction atomicity is achieved.
- *Non-autonomous*: compared to other computers, the smartcard has no independent power supply, thereby precluding disconnected and asynchronous processing. Thus, all transactions must be completed while the card is inserted in a terminal (unlike PDA, write operations cannot be cached in RAM and reported on stable storage asynchronously).

[2] Considering Flash instead of EEPROM will not change our conclusions. It will just exacerbate them.

[3] Most security holes are the results of software bugs [34].

3.3 Problem statement

To summarize, our goal is to design a PicoDBMS including the following components:

- *Storage manager*: manages the storage of the database and the associated indices.
- *Query manager*: processes query plans composed of select, project, join, and aggregates.
- *Transaction manager*: enforces the ACID properties and participates to distributed transactions.
- *Access right manager*: provides access rights on base data and on complex user-defined views.

Thus, the PicoDBMS hosted in the chip provides the minimal subset of functionality that is strictly needed to manage in a secure way the data shared by all onboard applications. Other components (e.g., the GUI, a sort operator, etc.) can be hosted in the terminal or be dynamically downloaded when needed, without threatening security. In the rest of this paper, we concentrate on the components which require non-traditional techniques (storage manager, query manager, and transaction manager) and ignore the access right manager for which traditional techniques can be used.

When designing the PicoDBMS's components, we must follow several design rules derived from the smartcard's properties:

- *Compactness rule*: minimize the size of data structures and the PicoDBMS code to cope with the limited stable memory area (a few megabytes).
- *RAM rule*: minimize the RAM usage given its extremely limited size (some kilobytes).
- *Write rule*: minimize write operations given their dramatic cost ($\approx$ 1 ms/word).
- *Read rule*: take advantage of the fast read operations ($\approx$ 100 ns/word).
- *Access rule*: take advantage of the low granularity and direct access capability of the stable memory for both read and write operations.
- *Security rule*: never externalize private data from the chip and minimize the algorithms' complexity to avoid security holes.

4 PicoDBMS storage model

In this section, following the design rules for a PicoDBMS, we discuss the storage issues and propose a very compact model based on a combination of flat storage, domain storage, and ring storage. We also evaluate the storage cost of our storage model.

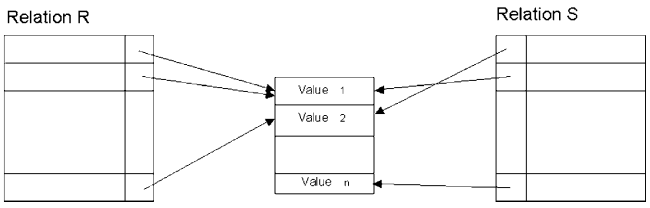


Fig. 1. Domain storage

4.1 Flat storage

The simplest way to organize data is *flat storage (FS)*, where tuples are stored sequentially and attribute values are embedded in the tuples. Although it does not impose it, the SCQL standard [24] considers FS as the reference storage model for smartcards. The main advantage of FS is access locality. However, in our context, FS has two main drawbacks:

- *Space consuming*: while normalization rules preclude attributes conjunction redundancy to occur, they do not avoid attribute value duplicates (e.g., the attribute *Doctor.Specialty* may contain many duplicates).
- *Inefficient*: in the absence of index structures, all operations are computed sequentially. While this is convenient for old fashion cards (some kilobytes of storage and a mono-relation select operator), this is no longer acceptable for future cards where storage capacity is likely to exceed 1 MB and queries can be rather complex.

Adding index structures to FS may solve the second problem while worsening the first one. Thus, FS alone is not appropriate for a PicoDBMS.

4.2 Domain storage

Based on the critique of FS, it follows that a PicoDBMS storage model should guarantee both data and index compactness. Let us first deal with data compactness. Since locality is no longer an issue in our context, pointer-based storage models inspired by MMDBMS [3, 27, 31] can help reducing the data storage cost. The basic idea is to preclude any duplicate value from occuring. This can be achieved by grouping values in domains (sets of unique values). We call this model *domain storage (DS)*. As shown in Fig. 1, tuples reference their attribute values by means of pointers. Furthermore, a domain can be shared among several attributes. This is particularly efficient for enumerated types, which vary on a small and determined set of values[4].

One may wonder about the cost of tuple creation, update, and deletion since they may generate insertion and deletion of values in domains. While these actions are more complex than their FS counterpart, their implementation remains more efficient in the smartcard context, simply because the amount of data to be written is much smaller. To amortize the slight overhead of domain storage, we only store by domain all large attributes (i.e., greater than a pointer size) containing duplicates. Obviously, attributes with no duplicates (e.g., keys) need not be stored by domain but with FS. Variable-size attributes – generally larger than a pointer – can also be advantageously stored in domains even if they do not contain duplicates. The benefit is not storage savings but memory management simplicity (all tuples of all relations become fixed-size) and log compactness (see Sect. 6).

[4] Compression techniques can be advantageously used in conjunction with DS to increase compactness [17].

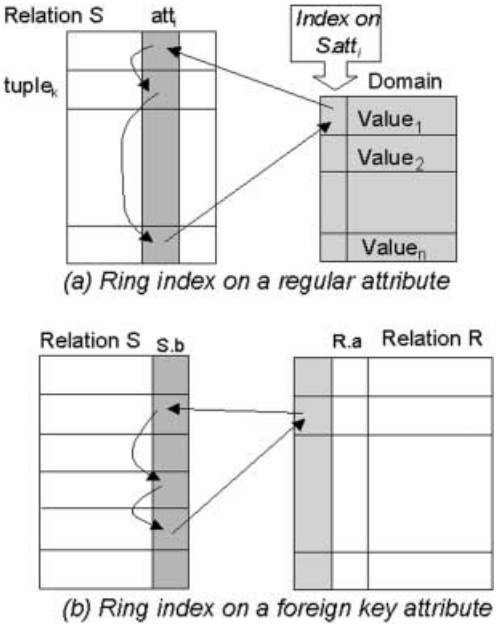


Fig. 2. Ring storage

4.3 Ring storage

We now address index compactness along with data compactness. Unlike disk-based DBMS that favor indices which preserve access locality, smartcards should make intensive use of secondary (i.e., pointer-based) indices. The issue here is to make these indices as compact as possible. Let us first consider select indices. A select index is typically made of two parts: a collection of values and a collection of pointers linking each value to all tuples sharing it. Assuming the indexed attribute varies on a domain, the index's collection of values can be saved since it exactly corresponds to the domain extension. The extra cost incurred by the index is then reduced to the pointers linking index values to tuples.

Let us go one step further and get these pointers almost for free. The idea is to store these *value-to-tuple* pointers in place of the *tuple-to-value* pointers within the tuples (i.e., pointers stored in the tuples to reference their attribute values in the domains). This yields to an index structure which makes a ring from the domain values to the tuples. Hence, we call it *ring index* (see Fig. 2a). However, the ring index can also be used to access the domain values from the tuples and thus serve as data storage model. Thus we call *ring storage (RS)* the storage of a domain-based attribute indexed by a ring. The index storage cost is reduced to its lowest bound, that is, one pointer per domain value, whatever the cardinality of the indexed relation. This important storage saving is obtained at the price of extra work for projecting a tuple to the corresponding attribute since retrieving the value of a ring stored attribute means traversing

on average half of the ring (i.e., up to reaching the domain value).

Join indices [40] can be treated in a similar way. A join predicate of the form ($R.a = S.b$) assumes that $R.a$ and $S.b$ vary on the same domain. Storing both $R.a$ and $S.b$ by means of rings leads to defining a join index. In this way, each domain value is linked by two separate rings to all tuples from R and S sharing the same join attribute value. However, most joins are performed on key attributes, $R.a$ being a primary key and $S.b$ being the foreign key referencing $R.a$. In our model, key attributes are not stored by domain but with FS. Nevertheless, since $R.a$ is the primary key of R, its extension forms precisely a domain, even if not stored outside of R. Since attributes $S.b$ take their values in $R.a$'s domain, they reference $R.a$ values by means of pointers. Thus, the domain-based storage model naturally implements for free a *unidirectional join index* from $S.b$ to $R.a$ (i.e., each S tuple is linked by a pointer to each R tuple matching with it). If traversals from $R.a$ to $S.b$ need to be optimized too, a *bi-directional join index* is required. This can be simply achieved by defining a ring index on $S.b$. Figure 2b shows the resulting situation where each R tuple is linked by a ring to all S tuples matching with it and vice versa. The cost of a bi-directional join index is restricted to a single pointer per R tuple, whatever the cardinality of S. Note that this situation resembles the well-known Codasyl model.

4.4 Storage cost evaluation

Our storage model combines FS, DS, and RS. Thus, the issue is to determine the best storage for each attribute. If the attributes need not be indexed, the choice is obviously between FS and DS. Otherwise, the choice is between RS and FS with a traditional index. Thus, we compare the storage cost for a single attribute, indexed or not, for each alternative. We introduce the following parameters:

- *CardRel*: cardinality of the relation holding the attribute.
- a: average length of the attribute (expressed in bytes).
- p: pointer size (3 bytes will be required to address "large" memory of future cards).
- S: selectivity factor of the attribute. S = *CardDom*/*CardRel*, where *CardDom* is the cardinality of the attribute domain extension (in all models). S measures the redundancy of the attribute (i.e., the same attribute value appears in $1/S$ tuples).

```
Cost(FS) =          CardRel*a          // attribute storage cost in
                                       // the relation
Cost(DS) =          CardRel*p          // attribute storage cost in
                                       // the relation
                  + S*CardRel*a        // values storage cost in
                                       // the domain

Cost(Indexed_FS) = Cost(FS)            // flat attribute storage cost
                  + S*CardRel*a        // value storage cost in the
                                       // index
                  + CardRel*p          // pointer storage cost in
                                       // the index

Cost(RS) =          Cost(DS)           // domain-based attribute
                                       // storage cost
                  + S*CardRel*p        // pointer storage cost in
                                       // the index
```

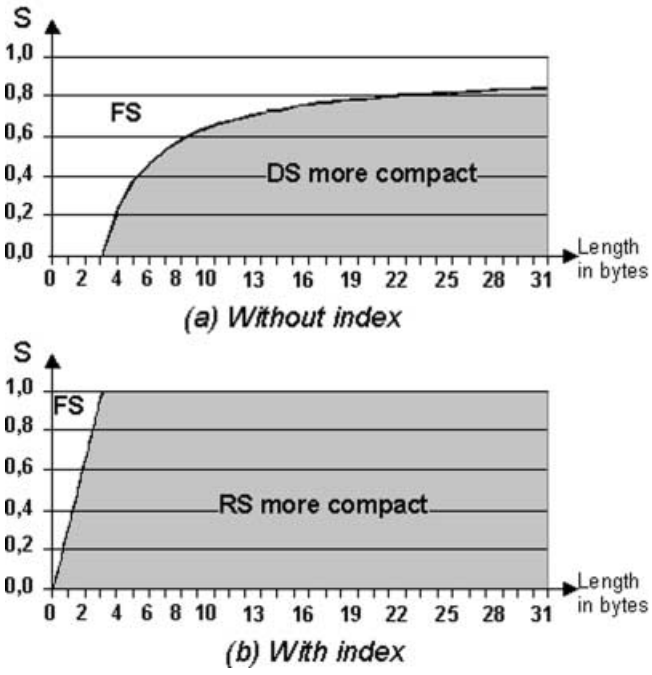


Fig. 3. Storage models' tradeoff

The cost equality between FS and DS gives: $S = (a-p)/a$. The cost equality between Indexed_FS and RS gives:

$$S = a/p$$

Figure 3a shows the different values of S and a for which FS and DS are equivalent. Thus, each curve divides the plan into a gain area for FS (above the curve) and a gain area for DS (under the curve). For values of a less than 3 (i.e., the size of a pointer), FS is obviously always more compact than DS. For higher values of a, DS becomes rapidly more compact than FS except for high values of S. For instance, considering $S = 0.5$, that is the same value is shared by only two tuples, DS outperforms FS for all a larger than 6 bytes. The higher$a$ and the lower S, the better DS. The benefit of DS is thus particularly important for enumerated type attributes. Figure 3b compares Indexed_FS with RS. The superiority of RS is obvious, except for 1- and 2-byte-long key attributes. Thus, Figs. 3a and 3b are guidelines for the database designer to decide how to store each attribute, by considering its size and selectivity.

5 Query processing

Traditional query processing strives to exploit large main memory for storing temporary data structures (e.g., hash tables) and intermediate results. When main memory is not large enough to hold some data, state-of-the-art algorithms (e.g., hybrid hash join [33]) resort to materialization on disk to avoid memory overflow. These algorithms cannot be used for a PicoDBMS because:

- Given the write rule and the lifetime of stable memory, writes in stable memory are proscribed, even for temporary materialization.

- Dedicating a specific RAM area does not help since we cannot estimate its size a priori. Making it small increases the risk of memory overflow, thereby leading to writes in stable memory. Making it large reduces the stable memory area, already limited in a smartcard (RAM rule). Moreover, even a large RAM area cannot guarantee that query execution will not produce memory overflow [9].
- State-of-the-art algorithms are quite sophisticated, which precludes their implementation in a PicoDBMS whose code must be simple, compact, and secure (compactness and security rules).

To solve this problem, we propose query processing techniques that do not use any working RAM area nor incur any writes in stable memory. In the following, we describe these techniques for simple and complex queries, including aggregation and remove duplicates. We show the effectiveness of our solution through a performance analysis.

5.1 Basic query execution without RAM

We consider the execution of *SPJ* (*Select/Project/Join*) queries. Query processing is classically done in two steps. The query optimizer first generates an "optimal" *query execution plan* (*QEP*). The QEP is then executed by the query engine which implements an *execution model* and uses a library of relational operators [17]. The optimizer can consider different shapes of QEP: *left-deep*, *right-deep* or *bushy trees* (see Fig. 4). In a left-deep tree, operators are executed sequentially and each intermediate result is materialized. On the contrary, right-deep trees execute operators in a pipeline fashion, thus avoiding intermediate result materialization. However, they require materializing in memory all left relations. Bushy trees offer opportunities to deal with the size of intermediate results and memory consumption [38].

In a PicoDBMS, the query optimizer should not consider any of these execution trees as they incur materialization. The solution is to only use pipelining with *extreme right-deep trees* where all the operators (including select) are pipelined. As left operands are always base relations, they are already materialized in stable memory, thus allowing us to execute a plan with no RAM consumption. Pipeline execution can be easily achieved using the well-known *Iterator Model* [17]. In this model, each operator is an *iterator* that supports three procedure calls: *open* to prepare an operator for producing an item, *next* to produce an item, and *close* to perform final clean-up. A *QEP* is activated starting at the root of the operator tree and progressing towards the leaves. The dataflow in the model is demand-driven: a child operator passes a tuple to its parent node in response to a *next* call from the parent.

Let us now detail how select, project, and join are performed. These operators can be executed either sequentially or with a ring index. Given the access rule, the use of indices seems always to be the right choice. However, extreme right-deep trees allow us to speed-up a single select on the first base relation (e.g., *Drug.type* in our example), but using a ring index on the other selected attributes (e.g., *Visit.date*) may slow down execution as the rings need to be traversed to retrieve their value. Project operators are pushed up to the tree since no materialization occurs. Note that the final project incurs

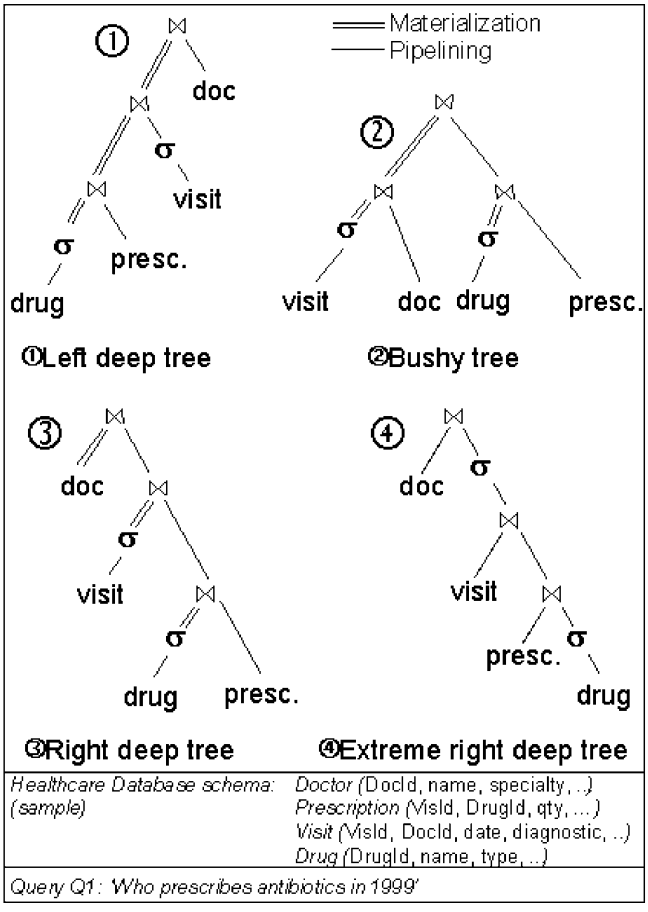


Fig. 4. Several execution trees for query Q1

an additional cost in case of ring attributes. Without indices, joining relations is done by a nested-loop algorithm since no other join technique can be applied without ad hoc structures (e.g., hash tables) and/or working area (e.g., sorting). The cost of indexed joins depends on the way indices are traversed. Consider the indexed join between *Doctor* ($n$tuples) and *Visit* (m tuples) on their key attribute. Assuming a unidirectional index, the join cost is proportional to $n * m$ starting with *Doctor* and to m starting with *Visit*. Assuming now a bi-directional index, the join cost becomes proportional to $n + m$ starting with *Doctor* and to $m^2/2n$ starting with *Visit* (retrieving the doctor associated to each visit incurs traversing half of a ring in average). In the latter case, a naïve nested loop join can be more efficient if the ring cardinality is greater than the target relation cardinality (i.e., when $m > n^2$). In that case, the database designer must clearly choose a unidirectional index between the two relations.

5.2 Complex query execution without RAM

We now consider the execution of aggregate, sort, and duplicate removal operators. At first glance, pipeline execution is not compatible with these operators which are classically performed on materialized intermediate results. Such materialization cannot occur either in the smartcard due to the RAM rule or in the terminal due to the security rule. Note that sorting can be done in the terminal since the output order of the

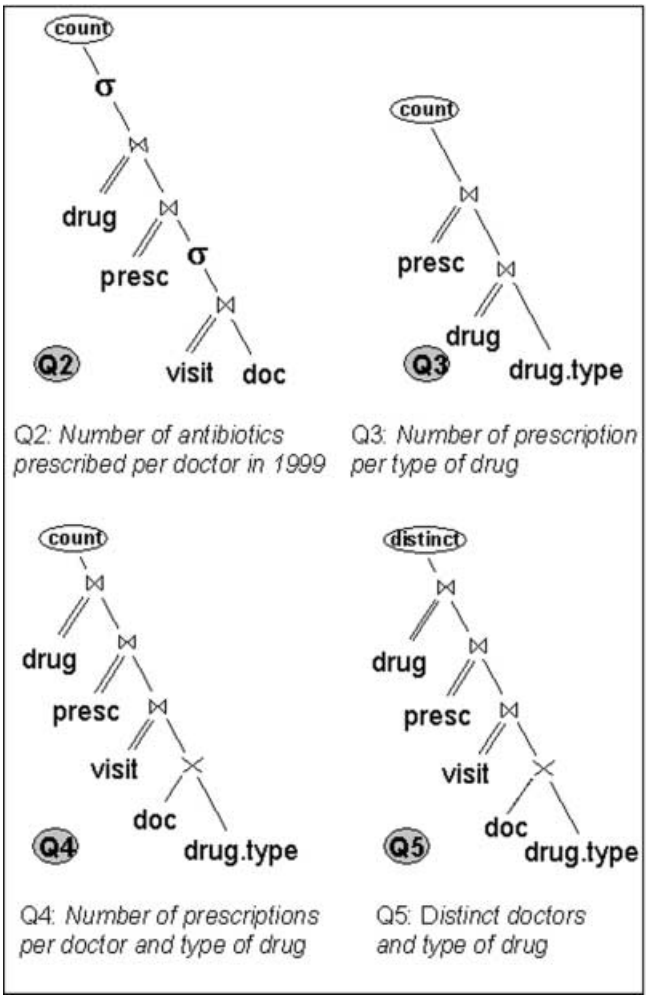


Fig. 5. Four 'complex' query execution plans

result tuples is not significant, i.e., depends on the DBMS algorithms.

We propose a solution to the above problem by exploiting two properties: (i) aggregate and duplicate removal can be done in pipeline if the incoming tuples are still grouped by distinct values; and (ii) pipeline operators are order-preserving since they consume (and produce) tuples in the arrival order. Thus, enforcing an adequate consumption order at the leaf of the execution tree allows pipelined aggregation and duplicate removal. For instance, the extreme right-deep tree of Fig. 4 delivers the tuples naturally grouped by *Drug.id*, thus allowing group queries on that attribute.

Let us now consider query Q2 of Fig. 5. As pictured, executing Q2 in pipeline requires rearranging the execution tree so that relation *Doctor* is explored first. Since *Doctor* contains distinct doctors, the tuples arriving to the *count* operator are naturally grouped by doctors.

The case of Q3 is harder. As the data must be grouped by *type of drugs* rather than by *Drug.id*, an additional join is required between relation *Drug* and domain *drug.type*. Domain values being unique, this join produces the tuples in the adequate order. If domain *Drug.type* does not exist, an operator must be introduced to sort relation *Drug* in pipeline. This can be done by performing n passes on *Drug* where n is the number of distinct values of *Drug.type*.

The case of Q4 is even trickier. The result must be grouped on two attributes (*Doctor.id* and *Drug.type*), introducing the need to start the tree with both relations! The solution is to insert a Cartesian product operator at the leaf of the tree in order to produce tuples ordered by *Doctor.id* and *Drug.type*. In this particular case, the query response time should be approximately n times greater than the same query without the 'group by' clause, where n is the number of distinct *types of drugs*.

Q5 retrieves the distinct couples of *doctor* and *type of prescribed drugs*. This query can be made similar to Q4 by expressing the distinct clause as an aggregate without function (i.e., the query "*select distinct* $a_1, \ldots, a_n$ *from* ... " is equivalent to "*select* $a_1, \ldots, a_n$ *from* ... *group by* $a_1, \ldots, a_n$"). The unique difference is that the computation for a given group, i.e., (*distinct result tuple*) can stop as soon as one tuple has been produced.

5.3 Query optimization

Heuristic optimization is attractive. However, well-known heuristics such as processing select and project first do not work here. Using extreme right-deep trees makes the former impractical and invalidates the latter. Heuristics for join ordering are even more risky considering our data structures. Conversely, there are many arguments for an exhaustive search of the best plan. First, the search space is limited since: (i) there is a single algorithm for each operator, depending on the existing indices; (ii) only extreme right-deep trees are considered; and (iii) typical queries will not involve many relations. Second, exhaustive search using depth-first algorithms do not consume any RAM. Finally, exhaustive algorithms are simple and compact (even if they iterate a lot). Under the assumption that query optimization is required in a PicoDBMS, the remarks above strongly argue in favor of an exhaustive search strategy.

5.4 Performance evaluation

Our proposed query engine can handle fairly complex queries, taking advantage of the read and access rules[5] while satisfying the compactness, write, RAM, and security rules. We now evaluate whether the PicoDBMS performance matches the smartcard application's requirements, that is, any query issued by the application can be performed in reasonable time (i.e., may not exceed the user's patience). Since the PicoDBMS code's simplicity is an important consideration to conform to the compactness and security rules, we must also evaluate which acceleration techniques (i.e., ring indices, query optimization) are really mandatory. For instance, an accelerator reducing the response time from 10 ms to 1 ms is useless in the smartcard context[6]. Thus, unlike traditional performance evaluation, our major concern is on absolute rather than relative performance.

[5] With traditional DBMS, such techniques will induce so many disk accesses that the system would thrash!

[6] With traditional DBMS, such acceleration can improve the transactional throughput.

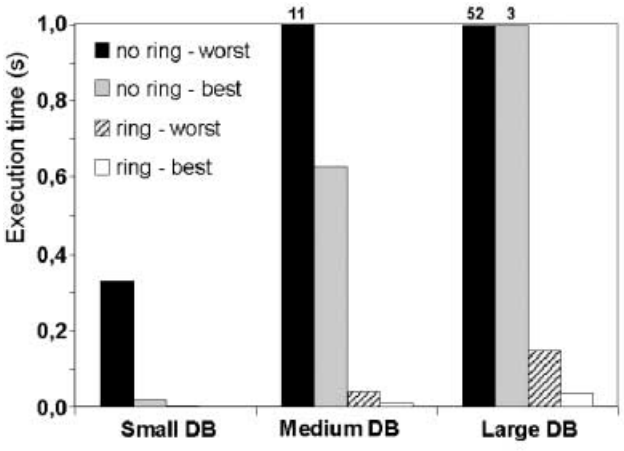


Fig. 6. Performance results for Q1

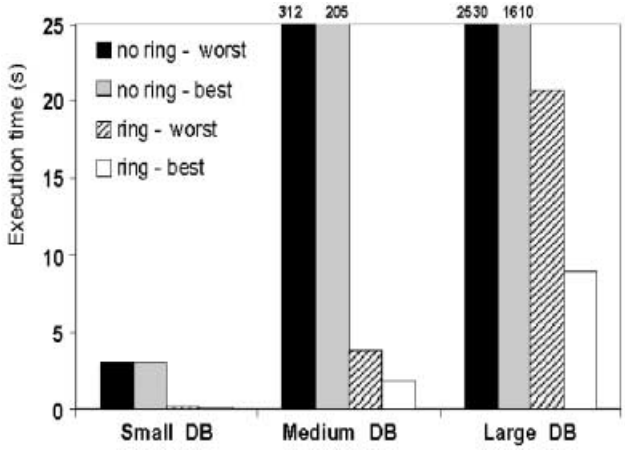


Fig. 7. Performance results for Q4

Evaluating absolute response time is complex in the smartcard environment because all platform parameters (e.g., processor speed, caching strategy, RAM, and EEPROM speed) strongly impact on the measurements[7]. Measuring the performance of our PicoDBMS on Bull's smartcard technology is attractive but introduces two problems. First, Bull's smartcards compatible with database applications are still prototypes [39]. Second, we are interested in providing the most general conclusions (i.e., as independent as possible of smartcard architectures). Therefore, we prefer to measure our query engine on two oldfashioned computers (a PC 486/25 Mhz and a Sun SparcStation 1+) which we felt roughly similar to forthcoming smartcard architectures. For each computer, we vary the system parameters (clock frequency, cache) and perform the experimentation tests. The performance ratios between all configurations were roughly constant (i.e., whatever the query), the slowest configuration (Intel 486 with no cache) performing eight times worse than the fastest (RISC with cache). In the following, we present response times for the slowest architecture to check the viability of our solutions in the worst environment.

We generated three instances of a simplified healthcare database: the *small, medium, and large* databases containing, respectively, (10, 30, 50) doctors, (100, 500, 1,000) visits, (300, 2,000, 5,000) prescriptions, and (40, 120, 200) drugs. Although we tested several queries, we describe below only the two most significant. Query Q1, which contains three joins and two selects on *Visit* and *Drug* (with selectivities of 20% and 5%), is representative of medium-complexity queries. Query Q4, which performs an aggregate on two attributes and requires the introduction of a Cartesian product, is representative of complex queries. For each query, we measure the performance for all possible query execution plans, excluding those which induce additional Cartesian product, varying the storage choices (with and without select and join ring indices). Figures 6 and 7 show the results for both best and worst plans on databases built with or without join indices.

Considering SPJ queries, the PicoDBMS performance clearly matches the application's requirements as soon as join rings are used. Indeed, the performance with join rings is at most 146 ms for the largest database and with the worst execution plan. With small databases, all the acceleration techniques can be discarded, while with larger ones, join rings remain necessary to obtain good response time. In that case, the absolute gain (110 ms) between the best and the worst plan does not justify the use of a query optimizer.

The performance of aggregate queries is clearly the worst because they introduce a Cartesian product at the leaf of the execution tree. Join rings are useful for medium and large databases. With large databases, the optimizer turns out to be necessary since the worst execution plan with join rings achieves a rather long response time (20.6 s).

The influence of ring indices for selects (not shown) is insignificant. Depending on the selectivity, it can bring slight improvement or overhead on the results. Although it may achieve an important relative speed-up for the select itself, the absolute gain is not significant considering the small influence of select on the global query execution cost (which is not the case in disk-based DBMS). Select ring indices are, however, useful for queries with aggregates or duplicate removal, that can result in a join between a relation and the domain attribute. In that case, the select index plays the role of a join index, thereby generating a significant gain on large relations and large domains.

Thus, this performance evaluation shows that our approach is feasible and that join indices are mandatory in all cases while query optimization turns out to be useful only with large databases and complex queries.

6 Transaction management

Like any data server, a PicoDBMS must enforce the well-known transactional ACID properties [8] to guarantee the consistency of the local data it manages as well as be able to participate in distributed transactions. We discuss below these properties with respect to a PicoDBMS.

- *Atomicity*: *local atomicity* means that the set of actions performed by the PicoDBMS on a transaction's behalf is made persistent following the *all or nothing* scheme. *Global atomicity*: this means that all data servers – including the PicoDBMS – accessed by a distributed transaction

[7] With traditional DBMS, very slow disk access allows us to ignore finer parameters.

agree on the same transaction outcome (either commit or rollback). The distinguishing features of a PicoDBMS regarding atomicity are no demarcation between main memory and persistent storage, the dramatic cost of writes, and the fact that they cannot be deferred.

- *Consistency*: this property ensures that the actions performed by the PicoDBMS satisfy all integrity constraints defined on the local data. Considering that traditional integrity constraint management can be used, we do not discuss it any further.
- *Isolation*: this property guarantees the serializability of concurrent executions. A PicoDBMS manages personal data and is typically single-user[8]. Furthermore, smartcard operating systems do not even support multithreading. Therefore, isolation is useless here.
- *Durability*: durability means that committed updates are never lost whatever the situation (i.e., even in case of a media failure). Durability cannot be enforced locally by the PicoDBMS because the smartcard is more likely to be stolen, lost or destroyed than a traditional computer. Indeed, mobility and smallness play against safety. Consequently, durability must be enforced through the network. The major issue is then preserving the privacy of data while delegating the durability to an external agent.

The remainder of this section addresses local atomicity, global atomicity, and durability.

6.1 Local atomicity

There are basically two ways to perform updates in a DBMS. The updates are either performed on *shadow objects* that are atomically integrated in the database at commit time or done *in place* (i.e., the transaction updates the shared copy of the database objects) [8]. We discuss these two traditional models below.

- *Shadow update*: This model is rarely employed in disk-based DBMSs because it destroys data locality on disk and increases concurrent updates on the catalog. In a PicoDBMS, disk locality and concurrency are not a concern. This model has been shown to be convenient for smartcards equipped with a small Flash memory [25]. However, it is poorly adapted to pointer-based storage models like RS since the object location changes at every update. In addition, the cost incurred by shadowing grows with the memory size. Indeed, either the granularity of the shadow objects increases or the paths to be duplicated in the catalog become longer. In both cases, the writing cost – which is the dominant factor – increases.
- *Update in-place*: write-ahead logging (*WAL*) [8] is required in this model to undo the effects of an aborted transaction. Unfortunately, the relative cost of WAL is much higher in a PicoDBMS than in a traditional disk-based DBMS which uses buffering to minimize I/Os. In a smartcard, the log must be written for each update since each update becomes immediately persistent. This roughly doubles the cost of writing.

Despite its drawbacks, *update in-place* is better suited than *shadow update* for a PicoDBMS because it accommodates pointer-based storage models and its cost is insensitive to the rapid growth of stable memory capacity. We also propose two optimizations to *update in-place*:

- *Pointer-based logging*: traditional WAL logs the values of all modified data. RS allows a finer granularity by logging pointers in place of values. The smallest the log records, the cheapest the WAL. The logging process must consider two types of information:
- *Values*: in case of a tuple update, the log record must contain the tuple address and the old attribute values, that is a pointer for all RS stored attributes and a regular value for FS stored attributes. In case of a tuple insertion or deletion, assuming each tuple header contains a status bit (i.e., dead or alive), only the tuple address has to be logged in order to recover its state.
- *Rings*: tuple insertion, deletion, and update (of a ring attribute) modify the structure of each ring traversing the corresponding tuple t. Since a ring is a circular chain of pointers, recovering its state means recovering the *next* pointer of t's predecessor (let us call it t_{pred}). The information to restore in t_{pred}.*next* is either t's address if t has been updated or deleted, or *t.next* if t has been inserted. t's address already belongs to the log (see above) and *t.next* does not have to be logged since t's content still exists in stable storage at recovery time. The issue is how to identify t_{pred} at recovery time. Logging this information can be saved at the price of traversing the whole ring starting from t, until reaching t again. Thus, ring recovery comes for free in terms of logging.
- *Garbage-collecting values*: insertion and deletion of domain values (domain values are never modified) should be logged as any other updates. This overhead can be avoided by implementing a deferred garbage collector that destroys all domain values no longer referenced by any tuple. Garbage-collecting a domain amounts to execute an ad hoc semi-join operator between the domain and all relations varying on it which discards the domain values that do not match[9]. The benefit of this solution is threefold: (i) the lazy deletion of unreferenced values does not entail the storage model coherency; (ii) garbage-collecting domain values is required anyway by RS (even in the absence of transaction control); and (iii) a deferred garbage-collector can be implemented without reference counters, thereby saving storage space. The deferred garbage collector cannot work in the background since smartcards do not yet support multi-threading. The most pragmatic solution is to launch it manually when the card is nearly full. An alternative to this manual procedure is to execute the garbage collector automatically at each card connection on a very small subset of the database (so that its cost remains hidden to the user). Garbage-collecting the database in such an incremental way is straightforward since domain values are examined one after the other.

[8] Even if the data managed by the PicoDBMS are shared among multiple users (e.g., as in the healthcare application), the PicoDBMS serves a single user at a time.

[9] Unlike reachability algorithms that start from the persistent roots and need marking [6], the proposed garbage-collector starts from the persistent leaves (i.e., the domain values) and exploits them one after the other, in a pipelined fashion (thus, it conforms to the RAM rule).

The update in-place model along with pointer-based logging and deferred garbage-collector reduces logging cost to its lowest bound, that is, a tuple address for inserted and deleted tuples, and the values of updated attributes (again, a pointer for DS and RS stored attributes).

6.2 Global atomicity

Global atomicity is traditionally enforced by an *atomic commitment protocol* (*ACP*). The most well known and widely used ACP is 2PC [8]. While extensively studied [19] and standardized [21, 29, 41], 2PC suffers from the following weaknesses in our context:

- *Need for a standard prepared state*: any server must externalize the standard *Xa* interface [41] to participate to 2PC. Unfortunately, ISO defines a transactional interface for smartcards but it does not cover distributed transactions [24]. In addition, participating to 2PC requires building a local prepared state that consumes valuable resources.
- *Disconnection means aborting*: a smartcard can be extracted from its terminal or its mobile host (e.g., a cellular phone) can be temporarily unreachable during 2PC. A participant's disconnection leads 2PC to abort the transaction even if all its operations have been successfully executed.
- *Badly adapted to moving participants*: the 2PC incurs two message rounds to commit a transaction. Considering the high cost of wireless communication, the overhead is significant for mobile terminals equipped with a smartcard reader (e.g., PDA, cellular phones).

As its name indicates, 2PC has two phases: the *voting* phase and the *decision* phase. The voting phase is the means by which the coordinator checks whether or not the participants can locally guarantee the ACID properties of the distributed transaction. The decision is *commit* if all participants vote *yes* and *abort* otherwise. Thus, the voting phase introduces an uncertainty period at transaction termination that leads to the aforementioned drawbacks.

Variations of *one-phase commit* protocols (*1PC*) have been recently proposed [2, 4, 35]. As stated in [2], 1PC eliminates the voting phase of 2PC by enforcing the following properties on the participant's behavior: (1) all operations are acknowledged before the 1PC is launched; (2) there are no deferred integrity constraints; (3) all participants are ruled by a rigorous concurrency control scheduler; and (4) all updates are logged on stable storage before 1PC is launched. These assumptions guarantee, respectively, the A, C, I, D properties before the ACP is launched. Then, the ACP reduces to a single phase, that is broadcasting the coordinator's decision to all participants (this decision is commit if all transaction's operations have been successfully executed and abort otherwise). If a crash or a disconnection precludes a participant from conforming to this decision, the corresponding transaction branch is simply forward recovered (potentially at the next reconnection). While the assumptions on the participant's behavior seem constraining in the general case, they are quite acceptable in the smartcard context [10]. Property (1) is common to all ACPs and is enforced by the ISO7816 standard [22]; property (2) conforms to the fact that PicoDBMS have lighter capabilities than full-fledged DBMS; and property (3) is satisfied by definition since smartcards do not support parallel executions. Property (4) is discussed in Sect. 6.3.

Eliminating the voting phase of the ACP solves altogether the three aforementioned problems. However, one may wonder about the interoperability between transaction managers and data managers supporting different protocols (either 1PC or 2PC). We have shown in [1] that the participation of legacy (i.e., 2PC compliant) data managers in 1PC is straightforward. Conversely, the participation of 1PC compliant data managers (e.g., a smartcard) in the 2PC can be achieved by associating a *log agent* to each participant. The role of the log agent is twofold. First, it manages the data manager's part of the 1PC's coordinator log, forces it to stable storage during the 2PC prepare phase, and exploits it if the transaction branch needs to be forward-recovered. Second, it translates the 2PC interface into that of 1PC. The log agent can be located on the terminal, so that the benefit of 1PC is lost for the terminal but it is preserved for the smartcard.

6.3 Durability

Most 1PC protocols assume that the coordinator is in charge of logging all participants' updates before triggering the ACP (all these protocols belong to the coordinator log family). *Coordinator log* [35] and *implicit yes vote* [4] assume that the participants piggyback their log records on the acknowledgment messages of each operation while *coordinator logical log* [2] assumes that the coordinator logs all operations sent to each participant. In all cases, the durability of the distributed transaction relies on the coordinator log. Thus, 1PC is a means by which global atomicity and durability can be solved altogether, at the same price.

Two issues remain to be solved: (i) where to store the coordinator log; and (ii) how to preserve the security rule, that is, how to make the log content as secure as the data stored in the smartcard. Since the log must sustain any kind of failure, it must be stored on the network by a trustee server (e.g., a public organism, a central bank, the card issuer, etc.). If some transactions are executed in disconnected mode (e.g., on a mobile terminal), the durability will be effective only at the time the terminal reconnects to the network. Protecting the log content against attacks imposes encryption. The way encryption is performed depends on the model of logging. If the coordinator log is fed by the log records piggybacked by the participants, the smartcard can encrypt them with an algorithm based on a private key (e.g., DES [28]). Otherwise (i.e., if the *coordinator logical log* scheme is selected), the smartcard can provide the coordinator with a public key that will be used by the coordinator itself to encrypt its log [32].

6.4 Transaction cost evaluation

The goal of this section is to approximate the time required by a representative update transaction. The objective is to confirm whether or not the write performance of smartcards assumed in this paper is acceptable for database applications like health cards. To this end, we estimate the time required to create a tuple in a relation, including the creation of domain values,

the insertion of the tuple in the rings potentially defined on this relation and the log time. Let us introduce the following parameters, in addition to those already defined in Sect. 4.4:

- *nbAttFS*: number of FS stored attributes
- *nbAttDS:* number of DS stored attributes
- *nbAttRS*: number of RS stored attributes
- w: size of a word (4 bytes in a 32-bit card)
- t: time to write one word in stable storage (5 ms in the worst case)

```
Cost(insertTuple) =
   ([(nbAttFS*a + nbAttDS*p + nbAttRS*p)/w]   // ①
   + (nbAttRS + nbAttDS) * S * [a/w]          // ②
   + nbAttRS * [p/w]                          // ③
   + [p/w]                                    // ④
   ) * t                                      // ⑤
```

① Tuple size
② Domain values size. S ≈ probability to create a new domain value
③ Ring pointers to be updated
④ Log record size
⑤ Write time

Let us consider a representative transaction executed on the healthcare. This transaction inserts a new tuple in *Doctor* and *Visit* and five tuples in *Prescription* and *Drug*. This is somehow a worst case for this application in the sense that the visited doctor is a new one and prescribes five new drugs. The considered attribute distribution is as follows:

Doctor *(nbAttFS=3, nbAttDS=4, nbAttRS=0),*
Visit *(nbAttFS=2, nbAttDS=3, nbAttRS=2),*
Prescription *(nbAttFS=1, nbAttDS=1, nbAttRS=2),*
Drug *(nbAttFS=2, nbAttDS=4, nbAttRS=0).*

The average attribute length a is fixed to 10 bytes. Figure 8 plots the update transaction execution time depending on S ($S = 0$ means that all attribute values already exist in the domains, while $S = 1$ means that all these values need be inserted in the domains).

The figure is self-explanatory. Note that the logging cost represents less than 3% of the total cost. This simple analysis shows that the time expected for this kind of transaction (less than 1 s) is clearly compatible with the healthcare application's requirements.

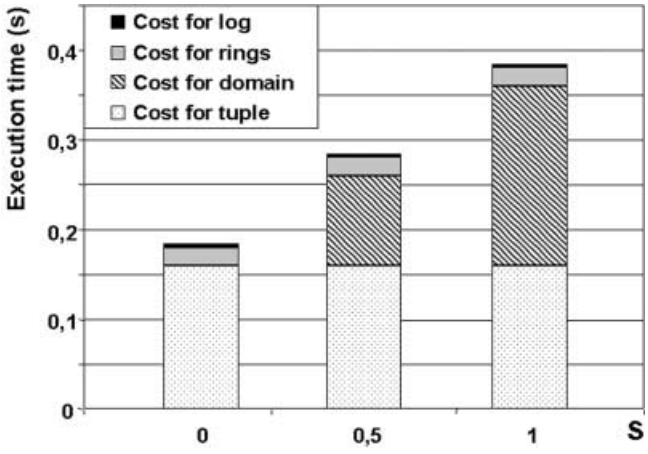


Fig. 8. Performance of a typical update transaction

7 Conclusion

As smartcards become more and more versatile, multi-application, and powerful, the need for database techniques arises. However, smartcards have severe hardware limitations which make traditional database technology irrelevant. The major problem is scaling down database techniques so they perform well under these limitations. In this paper, we addressed this problem and proposed the design of a PicoDBMS, concentrating on the components which require non-traditional techniques (storage manager, query manager, and transaction manager).

This paper makes several contributions. First, we analyzed the requirements for a PicoDBMS based on a healthcare application which is representative of personal folder applications and has strong database requirements. We showed that the minimal functionality should include select/project/join/aggregate, access right management, and views as well as transaction's atomicity and durability.

Second, we gave an in-depth analysis of the problem by considering the smartcard hardware trends. Based on this analysis, we assumed a smartcard with a reasonable stable memory of a few megabytes and a small RAM of some kilobytes, and we derived design rules for a PicoDBMS architecture.

Third, we proposed a new highly compact storage model that combines flat storage (FS), domain storage (DS), and ring storage (RS). Ring storage reduces the indexing cost to its lowest bound. Based on performance evaluation, we derived guidelines to decide the best way to store an attribute.

Fourth, we proposed query processing techniques which handle complex query plans with no RAM consumption. This is achieved by considering extreme right-deep trees which can pipeline all operators of the plan including aggregates. We also argued that, if query optimization is needed, the strategy should be exhaustive search. We measured the performance of our execution model with an implementation of our query engine on two old-fashioned computers which we configured to be similar to forthcoming smartcard architectures. We showed that the resulting performance matches the smartcard application's requirements.

Finally, we proposed techniques for transaction atomicity and durability. Local atomicity is achieved through update in-place with two optimizations which exploit the storage model: pointer-based logging and garbage collection of domain values. Global atomicity and durability are enforced by 1PC which is easily applicable in the smartcard context and more efficient than 2PC. We showed that the performance of typical update transactions is acceptable for representative applications like the health card.

This work is done in the context of a new project with Bull Smart Cards and Terminals. The next step is to port our PicoDBMS prototype on Bull's smartcard new technology, called *OverSoft* [12], and to assess its functionality and performance on real-world applications. To this end, a benchmark dedicated to PicoDBMS must be set up. We also plan to address open issues such as protected logging for durability, query execution on encrypted data (e.g., stored in an external Flash), and statistics maintenance on a population of cards.

References

1. Abdallah M., Bobineau C., Guerraoui R., Pucheral P: Specification of the transaction service. Esprit project OpenDREAMS-II n° 25262, Deliverable n° R13, 1998
2. Abdallah M., Guerraoui R., Pucheral P.: One-phase commit: does it make sense? Int. Conf. on Parallel and Distributed Systems (ICPADS), 1998
3. Ammann A., Hanrahan M., Krishnamurthy R.: design of a memory resident DBMS. IEEE COMPCON, 1985
4. Al-Houmaily Y., Chrysanthis P.K.: Two-phase commit in gigabit-networked distributed databases. Int. Conf. on Parallel and Distributed Computing Systems (PDCS), 1995
5. Anderson R., Kuhn M.: Tamper resistance – a cautionary note. USENIX Workshop on Electronic Commerce, 1996
6. Amsaleg L., Franklin M.J., Gruber O.: Efficient incremental garbage collection for client-server object database systems. Int. Conf. on Very Large Databases (VLDB), 1995
7. Bobineau C., Bouganim L., Pucheral P., Valduriez P.: PicoDBMS: scaling down database techniques for the smartcard (Best Paper Award). Int. Conf. on Very Large Databases (VLDB), 2000
8. Bernstein P.A., Hadzilacos V., Goodman N.: Concurrency control and recovery in database systems. Addison-Wesley, Reading, Mass., USA,1987
9. Bouganim L., Kapitskaia O., Valduriez P.: Memory-adaptive scheduling for large query execution. Int. Conf. on Information and Knowledge Management (CIKM), 1998
10. Bobineau C., Pucheral P., Abdallah M.: A unilateral commit protocol for mobile and disconnected computing. Int. Conf. On Parallel and Distributed Computing Systems (PDCS), 2000
11. van Bommel F.A., Sembritzki J., Buettner H.-G.: Overview on healthcard projects and standards. Health Cards Int. Conf., 1999
12. Bull S.A.: Bull unveils iSimplify! the personal portable portal. Available at: http://www.bull.com:80/bull_news/
13. Carrasco L.C.: RDBMS's for Java cards? What a senseless idea! Available at: www.sqlmachine.com, 1999
14. DataQuest.: Chip card market and technology charge ahead. MSAM-WW-DP-9808, 1998
15. Dipert B.: FRAM: Ready to ditch niche? EDN Access Magazine, Cahners, London, 1997
16. Gemplus.: SIM Cards: From kilobytes to megabytes. Available at: www.gemplus.fr/about/pressroom/, 1999
17. Graefe G.: Query evaluation techniques for large databases. ACM Comput Surv, 25(2), 1993
18. Graefe G.: The new database imperatives. Int. Conf. on Data Engineering (ICDE), 1998
19. Gray J., Reuter A.: Transaction processing. Concepts and Techniques. Morgan Kaufmann, San Francisco, 1993
20. IBM Corporation.: DB2 Everywhere – administration and application programming guide. IBM Software Documentation, SC26-9675-00, 1999
21. International Standardization Organization (ISO).: Information technology - open systems interconnection - distributed transaction processing. ISO/IEC 10026, 1992
22. International Standardization Organization (ISO).: Integrated circuit(s) cards with contacts – part 3: electronic signal and transmission protocols. ISO/IEC 7816-3, 1997
23. International Standardization Organization (ISO).: Integrated circuit(s) cards with contacts – part 1: physical characteristics. ISO/IEC 7816-1, 1998
24. International Standardization Organization (ISO).: Integrated circuit(s) cards with contacts – part 7: interindustry commands for structured card query language (SCQL). ISO/IEC 7816-7, 1999
25. Lecomte S., Trane P.: Failure recovery using action log for smartcards transaction based system. IEEE Online Testing Workshop, 1997
26. Microsoft Corporation.: Windows for smartcards toolkit for visual basic 6.0. Available at: www.microsoft.com/windowsce/smartcard/, 2000
27. Missikov M., Scholl M.: Relational queries in a domain based DBMS. ACM SIGMOD Int. Conf. on Management of Data, 1983
28. National Institute of Standards and Technology.: Announcing the Data Encryption Standard (DES). FIPS PUB 46-2, 1993
29. Object Management Group.: Object transaction service. Document 94.8.4, OMG editor, 1994
30. Oracle Corporation.: Oracle 8i Lite - Oracle Lite SQL reference. Oracle documentation, A73270-01, 1999
31. Pucheral P., Thévenin J.M., Valduriez P.: Efficient main memory data management using the DBGraph storage model. Int. Conf. on Very Large Databases (VLDB), 1990
32. RSA Laboratories.: PKCS # 1: RSA Encryption Standard. RSA Laboratories Technical Note, v.1.5, 1993
33. Schneider D., DeWitt D.: A performance evaluation of four parallel join algorithms in a shared-nothing multiprocessor environment. ACM-SIGMOD Int. Conf., 1989
34. Schneier B., Shostack A.: Breaking up is hard to do: modeling security threats for smart cards. USENIX Symposium on Smart Cards, 1999
35. Stamos J., Cristian F.: A low-cost atomic commit protocol. IEEE Symposium on Reliable Distributed Systems, 1990
36. Sun Microsystems.: JavaCard 2.1 application programming interface specification. JavaSoft documentation, 1999
37. Sybase Inc.: Sybase adaptive server anywhere reference. CT75KNA, 1999
38. Shekita E., Young H., Tan K.L.: Multi-join optimization for symmetric multiprocessors. Int. Conf. on Very Large Data Bases (VLDB), 1993
39. Tual J.-P.: MASSC: a generic architecture for multiapplication smart cards. IEEE Micro J, N° 0272-1739/99, 1999
40. Valduriez P.: Join indices. ACM Trans. Database Syst, 12(2), 1987
41. X/Open.: Distributed transaction processing: reference model. X/Open Guide, Version 3. G307., X/Open Company Limited, 1996

Annexe B

Chip-Secured Data Access: Confidential Data on Untrusted Servers

Luc Bouganim, Philippe Pucheral

28th International Conference on Very Large Data Bases, VLDB'02,

Hong Kong, August 2002.

Chip-Secured Data Access: Confidential Data on Untrusted Servers

Luc Bouganim, Philippe Pucheral

PRISM Laboratory – 78035 Versailles – France
<Firstname.Lastname>@prism.uvsq.fr

Abstract

The democratization of ubiquitous computing (access data anywhere, anytime, anyhow), the increasing connection of corporate databases to the Internet and the today's natural resort to Web-hosting companies strongly emphasize the need for data confidentiality. Database servers arouse user's suspicion because no one can fully trust traditional security mechanisms against more and more frequent and malicious attacks and no one can be fully confident on an invisible DBA administering confidential data. This paper gives an in-depth analysis of existing security solutions and concludes on the intrinsic weakness of the traditional server-based approach to preserve data confidentiality. With this statement in mind, we propose a solution called C-SDA (Chip-Secured Data Access), which enforces data confidentiality and controls personal privileges thanks to a client-based security component acting as a mediator between a client and an encrypted database. This component is embedded in a smartcard to prevent any tampering to occur. This cooperation of hardware and software security components constitutes a strong guarantee against attacks threatening personal as well as business data.

1. Introduction

The rapid growth of ubiquitous computing impels mobile users to store personal data on the Web to increase its availability. In the same way, corporate databases are made more and more accessible to authorized employees over the Internet. Small businesses are prompted to delegate part of their information system to Web-hosting companies or Database Service Providers (DSP) that guarantee data resiliency, consistency and high availability [eCr02,CaB02,Qck02]. Customer information is also maintained on-line for the needs of e-commerce and e-business applications. Typically, Microsoft .NET Passport [Mic02] gathers customer information (identity, passwords, credit card numbers, profiling data) in an electronic wallet shared by all participating .NET Web sites. Consequently, the amount of sensitive information collected and shared in the marketplace is such that data confidentiality has become one of the major concerns of citizens, companies and public organizations, and constitutes a tremendous challenge for the database community.

Confidential data threatened by attackers is manifold: information related to the private life of individuals (e.g., agenda, address book, bookmarks, medical records, household expenses), credit card numbers, patents, business strategies, diplomatic or military secrets. Even ordinary data may become sensitive once grouped and well organized in databases. Customers have no other choice than trusting DSP's arguing that their systems are fully secured and their employees are beyond any suspicion. However, according to the "Computer Crime and Security Survey" published by the Computer Security Institute (CSI) and the FBI [FBI01], the theft of intellectual property due to database vulnerability costs American businesses $103 billion annually and 45% of the attacks are conducted by insiders.

Traditional database security policies rely on user authentication, communication encryption and server-enforced access controls [BPS96]. Unfortunately, these mechanisms are inoperative against most insider attacks and particularly against database administrator attacks. Several attempts have been made recently to strengthen server-based database security policies thanks to database encryption [Ora99, Mat00, HeW01].

This paper first characterizes the intrinsic limits of these server-based solutions with respect to the different types of attacks that can be conducted. With these limitations in mind, we state the dimensions of the *data confidentiality problem*.

While client-based security policies have been historically disregarded considering the vulnerability of client environments [Rus01], we argue that the emergence of smartcard secured client devices fundamentally changes the problem statement. Initially developed by Bull to secure the French banking system, smartcards have been

**Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002**

used successfully around the world in various applications managing secured data (such as banking, pay-TV or GSM subscriber identification, loyalty, healthcare, insurance, etc.). Unfortunately, smartcards suffer from intrinsic hardware constraints that confine their applicability in terms of data management to secure portable folders (e.g., healthcare folder) [ISO99, PBV01].

We capitalize on the security properties of the smartcard to devise a solution to the data confidentiality problem, named *Chip-Secured Data Access (C-SDA)*. C-SDA takes the form of a security software embedded in a smartcard. This software acts as an incorruptible mediator between a client and a server hosting an encrypted database. The confidence in C-SDA relies on the fact that data encryption, query evaluation and access right management are insulated in a smartcard and cannot be tampered by anyone, including the cardholder. Dedicated query evaluation techniques are proposed to tackle the strong smartcard hardware constraints. We show the conclusive benefit of associating software and hardware security to preserve data confidentiality.

The contribution of this paper is twofold. First, it clearly states the dimensions of the data confidentiality problem and explains to which extent existing security solutions fail in addressing some of these dimensions. Second, it proposes a novel database security model where confidentiality is delegated to a tamper-resistant client device. This model is being validated in the context of a BtoB project supported by the French ANVAR agency (Agence Nationale pour la VAlorisation de la Recherche). This project will give us the opportunity to assess the functionality and performance of C-SDA on a real world application.

This paper is organized as follows. Section 2 characterizes the attacks that can be conducted against confidential data, analyzes the strengths and weaknesses of existing secure database solutions and concludes with a precise formulation of the *data confidentiality problem*. Section 3 introduces the Chip-Secured Data Access approach and shows how it answers each dimension of the data confidentiality problem. Section 4 addresses query management issues. Section 5 concentrates on data encryption and access right management. Section 6 develops a complete scenario illustrating the behavior of C-SDA on a concrete example. Finally, section 7 concludes the paper and sketches future research directions.

2. Data confidentiality problem

In this section, we first introduce the distinction between data privacy and data confidentiality. Then, we characterize the class of attacks that are commonly directed against databases. We discuss afterwards how server-based and client-based approaches resist to these attacks. We conclude by a precise formulation of the *data confidentiality problem* addressed in this paper.

2.1. Data privacy vs. data confidentiality

This paper concentrates on a particular aspect of database security, that is *data confidentiality*. Data confidentiality refers to the ability to share sensitive data among a community of users while respecting the privileges granted by the data owner to each member of the community. Any user external to the community is assumed to have no privilege at all. A special case of data confidentiality is *data privacy*. Data privacy means that the data owned by an individual will never be disclosed to anyone else.

Privacy is easier to enforce than confidentiality since sharing is precluded. The simplest and most effective way to ensure data privacy is to encrypt the user's data thanks to a symmetric key algorithm (e.g., DES [NIS93]). The user being the unique holder of the cipher key, no one else can access the clear text form of the data. Several Storage Service Providers propose to manage encrypted backups for personal data [Sky02]. They guarantee that data is encrypted at all times from transmission of a customer's computer to their server and back and remains safe from unauthorized access even by their staff.

Data privacy solutions cover only a restricted range of applications considering that even private data is subject to sharing (e.g., patient's medical records are shared by doctors, customer's information is shared by e-commerce sites). Thus, the remainder of the paper focuses on the more general problem of data confidentiality and places much emphasis on access right management.

2.2. The attackers

In the light of the preceding section, we can identify three classes of attackers that can threaten data confidentiality:

- *Intruder:* a person who infiltrates a computer system and tries to extract valuable information from the database footprint on disk (DBMS access controls are bypassed).
- *Insider*: a person who belongs to a community of users properly identified by the computer system and the database server and who tries to get information exceeding her own access rights.
- *Administrator*: a person who has enough (usually all) privileges to administer a computer system (System Administrator) or a DBMS (Database Administrator or DBA). These privileges give her the opportunity to tamper the access right definition and to spy on the DBMS behavior.

An Intruder who usurps successfully the identity of an Insider or an Administrator will be considered as such in the rest of the paper.

2.3. Weaknesses of server-based security policies

Traditional database security policies rely on three well established principles [BPS96]: (1) user identification and authentication, that can be supported by mechanisms ranging from simple login/password methods up to smartcard or biometrics device-based methods;

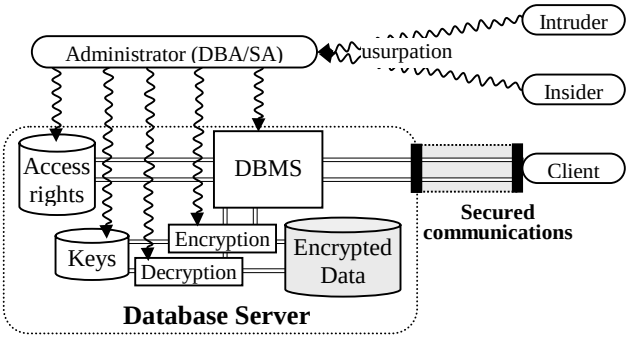


Figure 1: *Database Server approach*

(2) network encryption, that guarantees the confidentiality and the integrity of client/server communications; and (3) server-enforced access control and privilege management. Although these mechanisms are clearly required, they fail to answer all threats identified earlier for two obvious reasons. The first reason is that the confidence on the server never exceeds the confidence the user is ready to place in the DBA. This confidence may vary depending on the users, the Web-hosting companies or the countries but, as far as data confidentiality is concerned, this confidence is generally quite low. The second reason is the increasing number of commercial or institutional sites that are hacked, demonstrating the difficulty of making the hosting computing system secure enough to prevent any intrusion.

Recent attempts have been made to reinforce the server security by encrypting the database. Some commercial DBMSs provide encryption packages to this end [Ora00]. However, if encryption provides an effective answer to attacks conducted on the database footprint by an Intruder, it does not enforce data confidentiality on its own. Indeed, the server being still responsible for query execution and access right management, encryption makes just a bit more tedious the Administrator attacks. In these solutions, the management of cryptographic keys is under the application's responsibility and data is decrypted on the fly by the server at query evaluation time. Thanks to her privileges and to the DBMS auditing tools, the DBA can change the encryption package, can get the cryptographic keys, can modify the access right definition and can even snoop the memory to get the data while it is decrypted. Thus, as Oracle confesses, encryption is not the expected "armor plating" because the DBA (or an Intruder usurping her identity) has all privileges (see Figure 1).

Solutions complementary to database encryption have been recently investigated to guard the DBMS from the DBA. Protegrity [Mat00] introduces a clear distinction between the role of the DBA, administering the database resources, and the role of the SA (Security Administrator), administering user privileges, encryption keys and other related security parameters. This distinction is also made effective at the system level by separating the database server from the security server. The gain in confidence comes from the fact that an attack requires a conspiracy between DBA and SA. Anyway, one must keep in mind that data is still decrypted by the database server at query execution time. An alternative to this approach is to design a secure DBMS engine that restricts DBA privileges in order to make the aforementioned attacks inoperative [HeW01]. This raises the following question "can a DBA administrate a DBMS with restricted privileges?". Unfortunately, DBMS vendors answer today negatively. In addition, this solution suffers from the same security breach as Protegrity regarding data decryption on the server.

The proliferation of solutions to increase database security exemplifies the acuity of the problem. However, existing solutions fail in answering the data confidentiality requirements listed below:

Data confidentiality requirements

1. confidential data must be managed by an auto-administered DBMS to cast off the DBA privileges,
2. this DBMS must be hosted by an auto-administered computing system to cast off the system administrator privileges,
3. this computing system must constitute a Secure Operating Environment (SOE)[1] to cast off any Intruder action.

The traditional database server approach suffer from a strong handicap to meet these requirements because existing DBMSs, as well as the computing systems they rely on, are far too complex, first to be auto-administered and second to constitute a SOE. The first assumption is strengthen by the analysis done in [ChW00] which measures the distance separating current technologies from future self-tuning and zero-admin DBMSs[2]. The worrying numbers regularly published by the Computer Security Institute and the FBI on database vulnerability [FBI01] truly confirms the second assumption.

2.4. Client-based security policies

The weaknesses of the server-based approach to meet the data confidentiality requirements led us to devise client-based solutions. As a preliminary remark, let us notice that the solution presented in section 2.1 to enforce data privacy is typically client-based since the server does nothing but storing encrypted data. Unfortunately, these solutions do not support sharing. Enforcing data confidentiality in a client-based approach means delegating the sharing control to the client devices. However, client-based approaches have been historically disregarded considering that users have themselves the opportunity to hack the client system, and then the sharing control in our context, with total impunity [Rus01].

1 A Secure Operating Environment was defined by [HeW01] as an environment able to manipulate secret data without causing secret leak.

2 Although security is not the concern of [ChW00], the following sentence from the authors is eloquent "*tuning is a nightmare and auto-tuning is wishful thinking at this stage*".

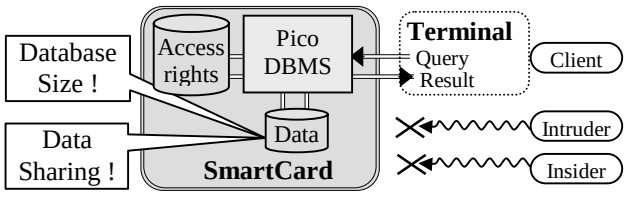


Figure 2: *PicoDBMS, a smartcard client-based approach*

The emergence of smartcard secured client equipment's drastically changes these conclusions. We illustrate the *smartcard-client-based* approach below through practical examples, and discuss to which extent they meet the data confidentiality requirements.

Smartcard is undoubtedly the most secure and cheap computing device today. The strength of smartcard applications regarding data confidentiality is threefold: (1) existing smartcard applications are simple enough to require zero-administration once downloaded in the card, (2) thanks to its hardware architecture making tampering extremely difficult [AnK96, ScS99], the smartcard is probably the best representative of SOE, (3) the high cost of an attack and its practical difficulty (holding the card) must be weighted up with its benefit (the data of a single user can be revealed). A common assumption is that a system can be considered secure if the cost of hacking it exceeds the value of the disclosed information. Conversely, the cost of security for the user is negligible considering the price of a smartcard (a few dollars).

Smartcards become more and more versatile thanks to the emergence of the JavaCard standard [Sun99] and to their increasing computing power. Thus, complex applications can now be downloaded and coexist in smartcards. Simple smartcard applications do not require administration because they are in some sense pre-administered (data schema, user and access rights are hard-coded). The side effect is the lack of extensibility. To circumvent this limitation, ISO has recently promoted a database approach for smartcards, named SCQL [ISO99], which allows for the dynamic declaration of data, users and access rights. Thus, smartcard embedded databases require administration but this task is handled by the cardholder (the data owner) instead of by a DBA, thereby preserving data confidentiality (see Figure 2). The problem of designing database engines dedicated to smartcards (called *smartcard DBMSs* in the sequel) has been extensively studied in [PBV01] and the feasibility of the approach has been recently demonstrated [ABB01]. While smartcard DBMSs pave the way for complex secured client-based applications, they suffer from a tiny storage capacity[3], which confines them to specific applications (typically secured portable folders).

Interesting attempts have been made to push away the smartcard storage limit. The first solution, due to the *WebCard* project [Van98], consists of storing in the smartcard URLs referencing huge, but unprotected, external data. *The Vault* [Big98] extends the WebCard approach by encrypting the documents referenced by URLs. Undoubtedly, the Vault meets the requirements of some applications but it does not constitute a solution from the database point of view. Indeed, the on-board database is seen as a catalog of large encrypted documents rather than as a regular database holding numerous fine-grain objects that can be shared and queried.

2.5. Problem definition

From the preceding discussions, we can identify the different dimensions of the *data confidentiality problem* addressed in this paper.

Data confidentiality problem

- *Privacy and confidentiality*: privacy of personal data and confidentiality of shared data must be guaranteed against attacks conducted by Intruders, Insiders and Administrators.
- *Storage capacity*: the system must not limit the volume nor the cardinality of the database.
- *Sharing capacity*: if required, any data may be shared among multiple authorized users.
- *Query capacity*: any data, whatever its granularity, may be queried through a predicate-based language (typically SQL).
- *Pertinence*: the system must guarantee an acceptable response time to each user, must be scalable and must be economically viable to meet the requirements of large public applications.

3. C-SDA baseline

Before discussing the principles of Chip-Secured Data Access (C-SDA), we first analyze how smartcard client-based solutions answer each dimension of the data confidentiality problem:

- *Privacy and confidentiality:* enforced by the fact that the smartcard is a SOE hosting the data as well as the DBMS engine and that this DBMS is self or user-administered.
- *Storage capacity*: limited by the smartcard stable storage capacity.
- *Sharing capacity*: limited by the need to share physically the same card[4].
- *Query capacity:* depends on the power of the embedded database engine. While query capacity is limited to simple selection in the SCQL standard [ISO99], PicoDBMS [PBV01] demonstrates the feasibility of

[3] Existing smartcards provide around 128KB of EEPROM stable memory, while stable storage is rapidly growing, it will remain quite limited compared with traditional computers.

[4] Typically, a smartcard medical folder has vocation for being shared among multiple users (patient, doctors, pharmacists, …) but a single user is active at a time.

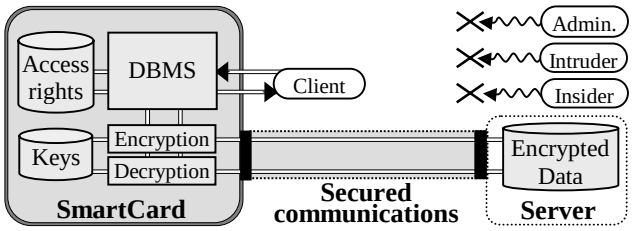


Figure 3: *C-SDA sphere of confidentiality*

embedding powerful query engines supporting selection, join, grouping and aggregate calculus.

- *Pertinence:* well suited in terms of performance (the smartcard DBMS is mono-user and works on a reduced set of data), of scalability (one smartcard per user) and of price (a few dollar per smartcard).

Given these statements, solving the data confidentiality problem sums up to bypass the storage and sharing limitations without hurting the other dimensions. The concept of server typically addresses the storage and sharing issues. Thus, let us consider to which extent the sphere of security provided by the smartcard could be extended to a remote server holding encrypted data. As discussed in section 2.3, the first security breach of the server-based approach comes from the fact that data is decrypted by the server at query execution time. Assuming that the DBMS query engine remains hosted by the smartcard, this eliminates the need to decrypt data on the server side. The second security breach of the server-based approach comes from the fact that access rights are enforced by the server and administered by an untrusted DBA. Let us assume that the DBMS access right manager remains hosted by the smartcard, the DBA (or an Intruder usurping her identity) is no longer able to abuse them.

Can we infer from the preceding assumptions that a server acting as an encrypted repository for a smartcard DBMS can integrate the smartcard's sphere of security (i.e., while keeping the level of confidence unchanged)? The answer is obviously 'no' since the server is not hosted by a SOE. Typically, an Intruder may conduct destructive or deny of service attacks on the server. However, privacy and confidentiality are preserved thanks to encryption[5].

In the same spirit, since the data that flows from the server to the smartcard DBMS is encrypted, can we infer that the communication channel is part of the smartcard's sphere of security? Again, the answer is 'no' since the communication channel may undergo several forms of attacks. At first sight, privacy and confidentiality are preserved anyway. However, an Insider may compare the encrypted data issued from the server with the query result that appears in plain text on its terminal. This may help her to conduct a *known plain text cryptanalysis* in order to deduce the encryption keys hosted by the smartcard. Thanks to these keys, the Insider may attempt to access data exceeding her own access rights. Indeed, the Insider may have the privilege to see the result of a query computed by the smartcard DBMS on data that is outside the scope of her privilege[6]. Consequently, re-encrypting the communication with a session key protocol (e.g., SSL) is necessary to enforce confidentiality[7].

The baseline of C-SDA is then to build a sphere of confidentiality encompassing the smartcard DBMS, the server and the communication channel linking them. The resulting functional architecture is pictured in Figure 3 and roughly works as follows. Each smartcard is equipped with a database engine managing access rights, query evaluation and encryption. When the user issues a query, the smartcard DBMS first checks the user's access rights and, in the positive case, gets the data from the server, decrypts it, executes the query and delivers the result to the terminal.

The server component of the C-SDA architecture is an answer to the storage and sharing dimensions of the data confidentiality problem. However, one may wonder about the impact of this answer on the other dimensions of the problem. The main question is whether the smartcard DBMS technology can conciliate complex queries, large volumes of data and performance, considering the inherent hardware constraints in the smartcard. The second question relates to data confidentiality and concerns the level of confidence that can be placed in data encryption (with respect to data hosted by the smartcard) and the granularity of sharing compatible with encryption. The next sections investigate these two issues.

4. Query Management

In order to evaluate the technical soundness of the C-SDA architecture in terms of query evaluation feasibility and efficiency, we first recall the smartcard characteristics that are relevant to this issue. Then, we propose a query evaluation principle that matches these smartcard characteristics whatever the volume of data involved in a query.

4.1. Smartcard characteristics

Current smartcards include in a monolithic chip, a 32 bits RISC processor at about 30 MIPS, memory modules (of about 96 KB of ROM, 4 KB of static RAM and 128 KB of EEPROM), a serial I/O channel (current bandwidth is around 9.6Kbps but the ISO standard allows up to 100Kbps) and security components preventing tampering [ISO98]. ROM is used to store the operating system, fixed data and standard routines. RAM is used to manage the execution stack of programs and to calculate results. EEPROM is used to store persistent information.

[5] The confidence that can be placed on data encryption itself will be more deeply discussed in section 5.

[6] For instance, a user may be authorized to consult the result of an aggregation without be aware of the elementary values from which this aggregation is computed.

[7] A side effect of an SSL-like protocol is to guarantee at the same time a mutual identification/authentification of the client and the server as well as the integrity of messages.

EEPROM has very fast read time (60-100 ns/word) comparable to RAM, but a dramatically slow write time (1 to 5 ms/word).

The main constraints of current smartcards are therefore: (i) the very limited storage capacity; (ii) the very slow write time in EEPROM and (iii) the extremely reduced size of the RAM. On the other hand, smartcards benefit from a very high security level and from a very powerful CPU with respect to the other resources. This makes the smartcard an asymmetric computing architecture which strongly differs from any other computing devices.

The current trends in hardware advances are on: (i) augmenting the CPU power to increase the speed of cipher algorithms, (ii) augmenting the capacity of the stable storage and (iii) augmenting the communication bandwidth between the chip and the card-reader[8]. More details on existing smartcard platforms and their evolution can be found in [Tua99, PBV01].

4.2. Query evaluation principle

A naive interpretation of the C-SDA architecture depicted in Figure 3 is to consider that the server acts as a persistent encrypted virtual memory which is accessed by the smartcard DBMS during query evaluation, any time a data item is requested for computation. Such an architecture would suffer from disastrous performance because it would incur a prohibitive communication cost (one call per data item) and I/O cost (traditional server optimizations become irrelevant). It may even happen that the same data be loaded several times from the server if the smartcard DBMS cannot keep enough local resources to cache it. Last but not least, the smartcard hardware constraints impose to design very specific query evaluation strategies. While ad-hoc strategies have been shown convenient in the context of small-embedded databases, their algorithm complexity renders them totally inappropriate for large databases [PBV01].

Thus, new query evaluation strategies that better exploit the computational resources available on the server and even on the terminal must be devised. This leads to split a query Q into a composition of the form $Q_s \circ Q_c \circ Q_t$, where Q_s, Q_c and Q_t denote respectively the sub-query evaluated on the server, the card and the terminal. The imbalance between the smartcard, the server and the terminal in terms of computing resources advocates pushing the biggest part of the computation down into Q_s and Q_t. However, the imbalance between these same components in terms of security leads to the following compromise:

- *Server subquery (Q_s):* the server must execute the largest part of the query as far as confidentiality is not compromised. That is, any predicate that can be evaluated on the encrypted form of the data must be pushed down to the server. To simplify things, we consider below that predicates based on an equality comparator {=, ≠} satisfy this condition[9]. In the sequel, we call these predicates *equi-predicates* in opposition to *inequi-predicates* based on inequality operators {>, ≥, <, ≤}.
- *Smartcard subquery (Q_c):* the smartcard DBMS is responsible for filtering the result of Q_s to evaluate all predicates that cannot be pushed down to Q_s and for computing aggregation functions if required. The terminal cannot participate to this evaluation because the data flow resulting from Q_s may go beyond the user's access rights.
- *Terminal subquery (Q_t):* due to the confidentiality consideration mentioned earlier, the terminal can only evaluate the part of the query related to the result presentation. Typically, it can handle the sort and the distinct operators, if requested by the user.

The challenge in decomposing Q into $Q_s \circ Q_c \circ Q_t$ is twofold. First, the global evaluation must meet the *pertinence* requirement in terms of performance and scalability. Second, Q_c must accommodate the smartcard's hardware constraints. Query evaluation on the smartcard precludes the generation of any intermediate results since: (i) the RAM capacity cannot accommodate them, (ii) RAM cannot overflow into EEPROM due to the dramatic cost of EEPROM writes and (iii) intermediate results cannot be externalized to the terminal without hurting confidentiality.

To explain how this challenge can be tackled, we will consider unnested SQL queries composed by the traditional Select, From, Where, Group by, Having and Order by clause and we will reason about them in terms of relational algebra. Let us first introduce some notations:

R, S, …U: relations involved in the query

R.a : attribute *a* from relation R

$\pi_{p,fp}$: projection operator, where *p* denotes the list of attributes to be projected and f_p denotes the list of aggregate functions to be computed before projection

χ: cartesian product operator

σ_q: selection operator, where *q* denotes the selection qualification: q is expressed in conjunctive normal form as follows: $C_1 \wedge C_2 \ldots \wedge C_n$, each condition C_i being of the form $(P_1 \vee P_2 \ldots \vee P_k)$, each predicate P_j being of the form (R.a θ value) or (R.a θ S.b), with $\theta \in \{=, \neq, >, \geq, <, \leq\}$.
C^q denotes the set $\{C_1, C_2 \ldots, C_n\}$ of conditions participating in q.
P^{Ci} denotes the set $\{P_1, P_2, P_k\}$ of predicates participating in C_i.

[8] These trends are partly explained by market perspectives on delivering multimedia objects (e.g., an mp3 flow) that can be decrypted on the fly by the card of a subscriber.

[9] This assumption means that any couple of data subject to comparison is encrypted with the same key. Data encryption is more deeply detailed in section 5.

γ_g: grouping operator, where *g* denotes the list of attributes on which the grouping applies

$\eta_{c, f\eta}$: having operator, where *c* denotes the having qualification and f_η the list of aggregate functions on which *c* applies

ϕ: presentation operators: sort, duplicate removal

E (resp. D): encryption (resp. decryption) operator

According to the operational semantics of SQL, an unnested query Q is equivalent to the following formula:

$Q = \phi(\pi_{p, fp}(\eta_{c, f\eta}(\gamma_g(\sigma_q(R\chi S\chi \ldots U)))))$

Under the assumption made about database encryption, that is: $\forall d_i,d_j$, $E(d_i) = E(d_j) \Leftrightarrow d_i = d_j$, we can infer that the largest part of Q that can be delegated to the server is:

$Q_s = \pi_{ps}(\gamma_g(\sigma_{qs}(R\chi S\chi \ldots U)))$, with

$C^{qs} \subseteq C^q$ and $C_i \in C^{qs} \Rightarrow \forall P_k \in P^{Ci}, \theta \in \{=, \neq\}$,

$ps = p \cup g \cup l_p \cup l_\eta \cup l_{qc}$, where

l_p is the list of attributes referenced by f_p

l_η is the list of attributes referenced by f_η

l_{qc} is the list of attributes referenced by the conditions $\in (C^q - C^{qs})$

This leads to define Q_c and Q_t as follows:

$Q_c = \pi_{p,fp}(\eta_{c,f\eta}(\sigma_{qc}(D(Q_s))))$, with $C^{qc} = C^q - C^{qs}$

$Q_t = \phi(Q_c)$

Roughly speaking, this means that equi-selection, equi-join and group by are computed on the server while inequi-predicates, aggregation and predicates over aggregate results have all to be evaluated on the smartcard. Figure 4 sketches the algorithm in charge of the evaluation of Q_c in the smartcard. This algorithm is self-explanatory. It consumes one tuple at a time from Q_s and requires a single buffer to cache the tuple of Q_c under construction. Note that if an aggregation is to be computed, the tuples of Q_s have already been grouped by the server and then do not need to be reordered in the smartcard. Thus, it clearly does not produce intermediate results and fulfills the second part of the decomposition challenge. As far as performance and scalability are concerned, two remarks have to be made. First, the cost incurred by the security mechanism (i.e., decryption) is spread over all users' smartcards instead of being concentrated on the server, thereby improving scalability. Second, the in-card computation is not CPU bound (powerful processor, low algorithm complexity) nor memory bound (one tuple at a time) but communication bandwidth bound. Let us remind that the communication channel between a smartcard and the card reader range from 9.6Kbps to 115Kbps maximum. The output-channel is not the limiting factor because it can deliver the resulting tuples at a reasonable rate (i.e., up to bandwidth/sizeof($\pi_{p,fp}$(result)). However, the input-channel may become the bottleneck if the ratio $|Q_c|/|Q_s|$ is low, because this ratio decreases in the same proportion the output rate. To illustrate the problem, let assume an inequi-join between relations R and S having a selectivity factor of 0.01. All tuples resulting from the cartesian product $R\chi S$ computed in Q_s will traverse the input-channel while only 1% of relevant tuples will traverse the output-channel. Optimization techniques are clearly required to handle this problem. This issue is addressed in section 6.2

5. Confidentiality and encryption

This section fixes a set of encryption rules required to answer accurately the data confidentiality problem. Then, it shows how the smartcard device can be exploited to increase the privacy and confidentiality of a reduced set of

```
∅ → result                                          // buffer caching the tuple of Qc being constructed
                                                    // result owns one attribute for each element in ps
0 → groupCard                                       // cardinality of a group
For each t ∈ Qs, do
    D(t) → t                                        // decrypt tuple t
    If σqc(t) = true, then                          // check selection qualification
        If g = ∅ then                               // no grouping
            send πp(t) to the terminal
        Else                                        // group by clause
            If πg(t) ≠ πg(result), then             // a new group is detected
                If ηc, fη(result) = true then       // check having qualification
                    send πp, fp(result) to the terminal
                ∅ → result
                0 → groupCard
            For each aggi ∈ (fp ∪ fη), do           // t belongs to the same group as result
                aggi(t) → result.aggi               // evaluate aggregation functions[1]
            groupCard + 1 → groupCard
send πp, fp(result) to the terminal                 // send the last tuple under construction
```

[1] Each aggregate function to be computed uses one attribute of *result* as a state variable. Assume the *avg* function is to be computed, *avg(t)* → *result.avg* sums up the current attribute value of *t* into *result.avg* while π_{avg}*(result)* divides this sum by the cardinality *groupCard* of the current group.

Figure 4: *Q_c in-card algorithm*

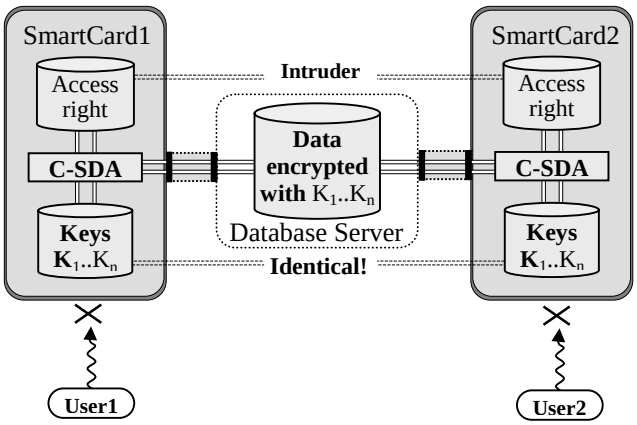


Figure 5: *Encryption keys versus access rights*

highly sensitive data. Finally, it addresses the management of access rights and concludes with a discussion on the limits of the solution.

5.1. Database encryption

From the beginning of the paper, we have considered implicitly that the whole database was encrypted. Obviously, only the confidential part of it needs to be encrypted. For the sake of simplicity, we will not discuss further the cohabitation between clear and encrypted data because it does not present a major technical difficulty. Thus, we concentrate in the sequel on the quality of the database encryption.

As stated in section 3, the level of confidence placed in C-SDA is strongly related to the confidence placed in the data encryption strategy. In our context, the following *data encryption rules* apply:

Key insulation rule: *encryption keys must remain confined in the smartcard.*
This rule is required to prevent any attack conducted by the DBA, an Intruder and even an Insider. Consequently, data encryption and decryption must be handled by the smartcard as well. Note that the cardholder herself has no way to access the encryption keys hosted by its own card. These keys remain under the exclusive control of the in-card C-SDA software.

Sharing rule: *encryption must remain orthogonal to access rights.*
As explained in section 2.1, encryption alone is sufficient to implement data privacy, assuming that each user encrypts her own data with a secrete key. Thus, encryption acts as a binary access right granting or revoking all privileges to the user depending on whether or not she knows the secrete key. On the contrary, data confidentiality requires sharing the same key(s) among a community of authorized users. Unfortunately, there is no bijection between encryption and access rights because these two mechanisms do not operate at the same level of granularity. Access rights are commonly attached to database views to share data at a very fine-grain level. The sharing is thus predicate-based. Achieving the same level of sharing with encryption alone would require defining as many encryption keys as possible SQL qualifications. Access rights can even be defined on virtual data (e.g., aggregate calculus) that obviously cannot be encrypted. Consequently, encryption rules must remain orthogonal to access right management. Assuming key K_i is used to encrypt data shared among multiple users, K_i must be hosted by the smartcard of each of these users but the key usage is restricted to the in-card C-SDA software that controls access rights (see Figure 5).

Computation rule: *encryption must preserve attribute equality comparisons.*
Encrypting the database on a tuple, column, or relation basis precludes any computation to occur on the server side without decrypting the data first. Thus, the encryption must be done on an attribute basis. In addition, as stated in section 4.2, the minimal assumption required to allow server computation without decryption is $\forall d_i,d_j$, $E(d_i) = E(d_j) \Leftrightarrow d_i = d_j$. Obviously, this assumption is required only for couple of data that may be subject to comparison. Fortunately, most block encryption algorithms (e.g., DES [NIS93]) satisfy this assumption.

Stronger assumptions on the encryption method might increase the range of computations that can be delegated to the server. *Privacy homomorphisms* (PH) introduced in [RAD78] allow to perform *some computation* on encrypted data. For instance, the PH proposed in [Dom97] preserves the basic four arithmetic operations, but equality predicates can no longer be checked. Order-preserving PH and more generally PH maintaining range information can also be devised but they drastically reduce the robustness of the encryption method [Sch96, Dom97].

Performance rule: *encryption must be symmetric and client-based.*
As stated in section 4.2, client-based encryption/decryption is the first guarantee of scalability. Moreover, considering the large volume of data to be encrypted/decrypted, we promote the use of symmetric encryption algorithms (e.g., DES) because they are more robust and much more efficient (three orders of magnitude faster) than asymmetric algorithms (e.g., RSA[RSA93]). The secure diffusion of secrete keys is the major problem of symmetric algorithms in traditional architectures. This problem is solved by nature in the C-SDA context, thanks to the smartcard device that provides a secure key hosting. Thus, keys are distributed among users along with smartcards.

Multi-key encryption rule: *encryption must exploit as much different keys as possible.*
Increasing the number of keys in the encryption process has two main advantages. First, it makes statistical attacks more difficult to conduct. Second, it reduces the amount of data that will be disclosed if the aforementioned attack succeeds. Different techniques can be envisioned to use multiple keys while respecting the computation rule. A

first solution is based on vertical fragmentation, that is encrypting with different keys the columns that will never participate to an equi-join (e.g., Person.name and Person.age). A second solution is based on horizontal partitioning, that is encrypting with different keys the attribute values of the same column thanks to a one-way hash function. For instance, *Key(h(a))* can be used as a parameter to encrypt the attribute value *a*, and the value $(h(a), E_{key(h(a))}(a))$ is stored in the database in place of *a*. Note that this solution respects the computation rule. Other techniques may be used but space limitations forbid their presentation in this paper.

5.2. Sensitive data

The persistent storage capacity of the smartcard introduces new alternatives to achieve data privacy and confidentiality. Basically, highly sensitive data may be stored in the smartcard instead of in the server, thereby making it ultimately robust against attacks. For instance, identification information could benefit from this property (e.g., name, social security number, …), so that the database in the server is depersonalized. This technique however introduces three issues: (i) how to integrate this sensitive data in the query evaluation process, (ii) how to guarantee its durability and (iii) how to share it if it is used by multiple users.

To make the integration of sensitive data in the query evaluation process as simple as possible, we propose to group sensitive data in *sensitive domains* (i.e., set of data items without duplicates) and to store indices referencing these domain values in place of the corresponding data in the server. This technique can be formally considered as a particular encryption method $E(data) \rightarrow domain\_index$ that is definitely unbreakable without the smartcard. The integration in the query evaluation process is straightforward since *E* satisfies the computation rule.

The complexity of enforcing sensitive data durability depends on whether a sensitive domain is static or dynamic. Static domain can simply be duplicated on any secure storage device (e.g., a backup smartcard). Dynamic domains are trickier to manage, especially if they are shared among multiple users. The solution is to leave an encrypted copy of the domain on a backup server (preferably distinct from the database server) and to synchronize this encrypted backup with the domain copy residing on a smartcard at each connection. One may wonder about the benefit of this method compared with leaving the data in their original form on the database server. The benefit is actually twofold. First, the database and the sensitive domains are located on two separate servers thereby increasing the complexity of attacks. Second, the backup copy of the domain doesn't need to participate in the query evaluation. Thus, it can be encrypted with stronger methods (e.g., a different key for each domain entry) since it is not affected by the computation rule.

5.3. Access Right Management

As stated in section 3, access right management must be embedded in the smartcard to prevent any DBA tampering. Since access rights are commonly defined on database views, the views have to be managed by the smartcard as well. This raises the problem of access rights and views evolution. If the smartcard is responsible for controlling access rights and views, their definitions have to be securely stored in a server accessible by all smartcards. Modeling the list of access right definitions and the list of database view definitions as two dynamic and shared sensitive domains brings a simple and accurate solution to this problem.

The crucial question regarding access rights is who is responsible for granting/revoking them. The common rule in database systems is that the owner of an object inherits this responsibility. In practice, the unlimited privileges of the DBA contradict this rule. Using C-SDA, the DBA conserves all her privileges, so that she can administer the database server but she has no way to break the data confidentiality, as long as she has no access to the user's smartcard. As a conclusion, a C-SDA user is definitely the unique holder of her data and she decides if she wants to exhibit them and to whom.

5.4. Limits of the solution

One may wonder whether this combination of hardware (the smartcard) and software (C-SDA) security components constitutes the ultimate protection against data confidentiality attacks. In this respect, we must state the limits of the solution.

First, an Intruder can infiltrate the user's terminal in order to snoop the query results that are presented in plain text to the user or to alter the query expression sent by the terminal to the smartcard before processing. By this way, the Intruder may try to execute a query selecting more data than expected by the user and snoop them. Anyway, such attack can reveal only data being in the user's access right scope. This threat cannot be avoided by any security architecture, unless the terminal is itself secure. To secure the terminal, both the screen and the keyboard must be part of the SOE, like in today's smartcard payment devices. This solution can be suitable for users willing only to consult their data but is inadequate as soon as computation is required on these data.

Second, an Intruder or an Administrator may try to tamper the database footprint on disk in the hope of decrypting unauthorized data thanks to the smartcard (e.g., by permuting columns encrypted with the same key). This attack can be beat off at the expense of introducing a checksum attribute in each tuple. Anyway, the scope of this attack is limited by the use of multiple encryption keys and by the fact that the attacker must be a cardholder.

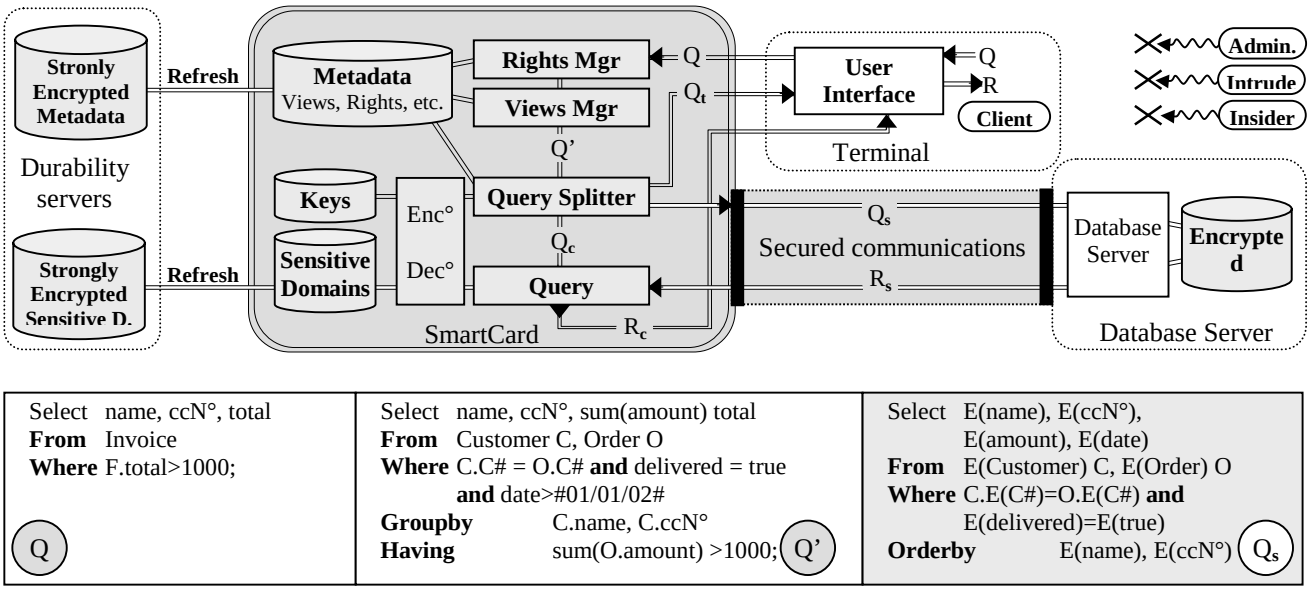


Figure 6: *C-SDA architecture and scenario*

6. C-SDA scenario

This section presents a complete C-SDA scenario illustrating the step by step evaluation of a simple query on a corporate database. Confidentiality and performance issues are discussed along the scenario unfolding.

6.1. Query Execution with C-SDA

Let us consider a business database application where the invoice department is willing to bill invoices having a total amount greater than 1000 US$. The privilege of the invoice department clerk is assumed to be restricted to the *select* operation on the view *Invoice*. This view calculates for each customer, the total amount of delivered orders since January 2002. This view prevents an untrusted clerk to access confidential order-lines. Figure 6 shows a query *Q*, issued by the clerk and expressed on the view Invoice, and the query *Q'*, resulting from the view resolution and expressed on the base relations Customer and Order. The execution of query *Q* comprises the following steps (see Figure 6).

1. *Metadata refreshing:* At connection time (i.e., when the user inserts her smartcard to the card reader), C-SDA contacts the durability server(s) in order to refresh its local copy of relation and view definitions, access right information's and sensitive data.
2. *Access Right checking and view resolution:* The *access right manager* checks that query *Q* involves only authorized relations and views. Then, the *view manager* merges *Q* with the view definition to produce *Q'*.
3. *Query splitting:* The *query splitter* splits query *Q'* into Q_s (step 4), Q_c (step 6) and Q_t (step 7) conforming to the decomposition principle detailed in section 4.2. Q_s is then rewritten in an "encrypted SQL form", that is relation names, attributes and constant are encrypted (encryption is denoted by *E()* in Figure 6). Note that the encrypted form of a well-formed SQL query is a well-formed SQL query.
4. Q_s *transmission and execution:* The encrypted query Q_s is sent to the database server using a secured communication protocol. Secured communication is mandatory to avoid any falsification of Q_s before transmission (which may permit a malicious user to access more data than granted). The database server optimizes and processes Q_s as any traditional query, without being aware of encryption. The query execution plan of Q_s is pictured in Figure 7.
5. Q_s *Result transmission:* The encrypted result R_s is sent back to the smartcard using a secured communication protocol. As explained in section 3, secured communication is mandatory here to avoid plaintext cryptanalysis on the terminal.
6. Q_c *execution:* The encrypted result R_s is processed in a pipelined fashion by C-SDA following the algorithm presented in Section 4.2. Figure 7 presents the query trees of Q_s and Q_c. As shown in the Figure, data decryption is pushed up to the query tree of Q_c as far as possible. This avoids decrypting all attributes of tuples that do not participate in the final result. For instance, the attribute *date* is first decrypted in order to check the predicate *date>01/01/02*. Tuples which survive this selection are further partially decrypted in order to compute the aggregation and to check the having clause. Finally, the qualified tuples are fully decrypted before being sent to the user. Assuming that ccN° (credit card number) is a sensitive domain, decryption of this attribute follows the principle described in section 5.2.
7. R_c *delivering and* Q_t *execution:* Finally, once decrypted, the tuples participating in the final result are sent to the terminal where the *distinct* and/or *sort* clauses potentially present in Q_t are applied.

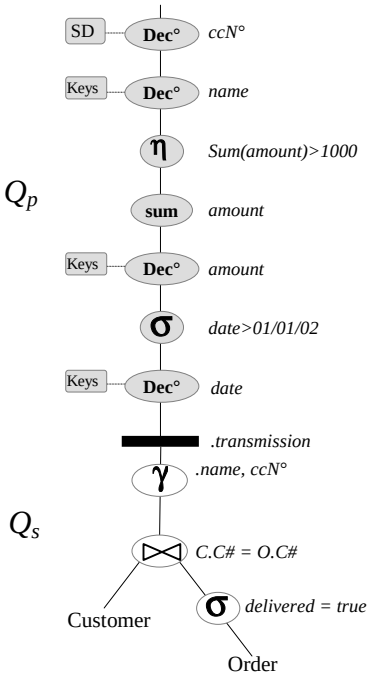


Figure 7: *Q_s and Q_c query tree*

6.2. Optimization Issues

The performance problem pointed out in section 4.2 is exemplified in this scenario. Assume that only 1% of orders satisfies the selection on *date*, 99% of the R_s tuples sent back to the smartcard are irrelevant, generating a bottleneck on the smartcard input-channel. In the following we sketch a solution alleviating this problem. Other optimizations of the C-SDA architecture can undoubtedly be devised but are out of the scope of this paper.

The solution proposed relies on a multi-stage cooperation between the smartcard and the server aiming at minimizing the flow of data traversing the smartcard input-channel. The intuition is to use the smartcard as a secured co-processor which can evaluate inequi-predicates on demand. The evaluation of each inequi-predicate is handled by a pre-processing query that takes as input a collection of encrypted values issued by the server, decrypts them, evaluates the inequi-predicate and sends back the matching values in their encrypted form to the server. On the server side, this result is integrated in the initial query thanks to a semi-join operator.

Let us illustrate the concept of pre-processing query on our scenario. The objective is to evaluate the inequi-predicate *date>01/01/02* on a data set smaller than R_s. Ideally, this predicate should be evaluated on the subset of Order tuples satisfying the selection *delivered = true*. This situation would be optimal in two respects. First, it would minimize the data flow traversing the smartcard input-channel. Second, it would minimize the cost of evaluating query Q_s on the server side by pushing up selections before joins in the regular way. Pre-processing is the way to achieve this goal. A pre-processing query PQ_s is first generated by the smartcard query splitter to get from the server the tuples resulting from π_{date} ($\sigma_{delivered=true}$ (Order)). Then, the smartcard query processor computes T = E(($\sigma_{date>01/01/02}$ (D(PQ_s)))), the content of which is stored in a temporary relation on the server side. Finally, the smartcard query splitter adds the semi-join predicate T.E(date) = Order.E(date) to the initial query Q_s and sends it to the server for computation (see Figure 8). This strategy applies as well to inequi-join predicates, and can be exploited iteratively for all inequi-predicates involved in the same query.

7. Conclusion and future prospects

The tremendous development of Internet applications prompts citizens and companies to put always more data accessible through the Web. Preserving data confidentiality in this context is becoming one of the most challenging issues for the database community.

This paper addresses this issue and makes the following contributions. First, it gives an in-depth analysis of the security solutions proposed in the database field and capitalizes on strengths and weaknesses of these approaches to clearly state the dimensions of the *data confidentiality problem*. Second, it proposes the *Chip-*

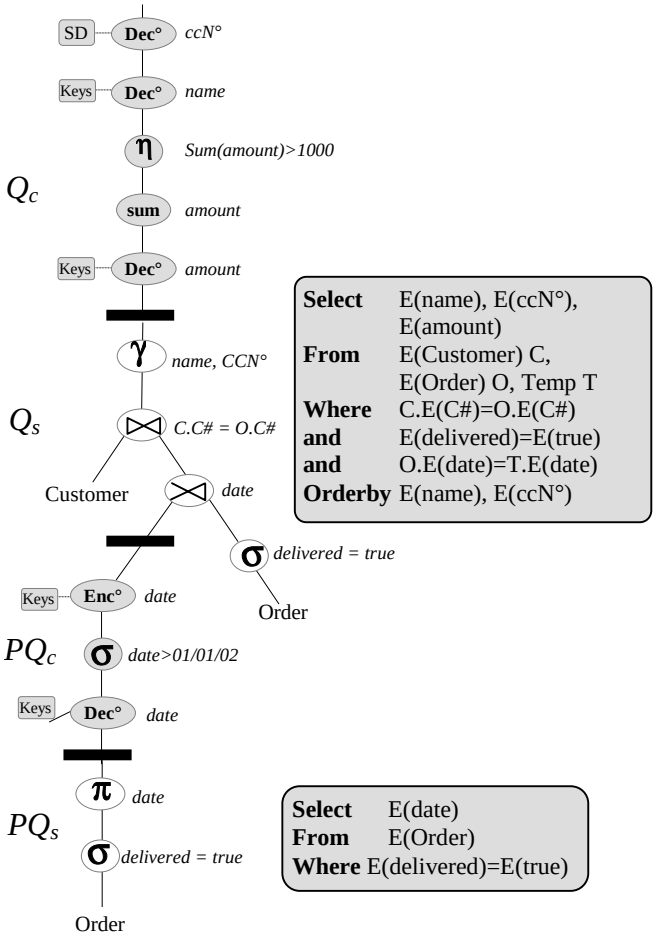


Figure 8: *Optimized query tree*

Secured Data Access (C-SDA) principle as a solution to this problem. The main idea underlying C-SDA is to insulate data encryption, query evaluation and access right management in a Secured Operating Environment (SOE). Third, query evaluation and optimization techniques are proposed to tackle the strong hardware constraints introduced by the most popular representative of SOE, the smartcard.

C-SDA is being validated in the context of a B2B project founded by the French ANVAR agency. This project, started in january 2002, aims at sharing an EDI database between business partners. Depending on the business model, this database can be hosted by a DSP or by one of the partner, but the data confidentiality requirements remain the same.

C-SDA has been devised in the context of smartcards because of its wide acceptance and its well-established technology. However, the C-SDA architecture can be adapted to other secured computing devices. For instance, the Dallas i-button [iBu02] provides a security level comparable to smartcards but benefits from a higher bandwidth with the terminal. Such technology could be exploited to alleviate the performance problem induced by inequi-predicates. The apparition of high-end secure coprocessor [Swe99] may, in the future, render viable tamper-resistant server-based solutions that are technically unfeasible today for performance and scalability reasons. In all situations, the interactions between the C-SDA software hosted by the secured device and the encrypted data store will remain the same, but with different technical tradeoffs.

Other important open issues concern the extension of C-SDA to more complex data models, query languages and client/server interactions. More generally, we believe that tamper-resistant devices will have an increasing influence on the way security solutions for information systems will be devised.

Acknowledgments

The authors wish to thank Philippe Bonnet and Ioana Manolescu for their helpful comments on this paper.

References

[ABB01] N. Anciaux, C. Bobineau, L. Bouganim, P. Pucheral, P. Valduriez, "PicoDBMS: Validation and Experience", *Int. Conf. on VLDB*, 2001.

[AnK96] R. Anderson, M. Kuhn, “Tamper Resistance – a Cautionary Note”, *USENIX Workshop on Electronic Commerce*, 1996.

[Big98] P. Biget “The Vault, an Architecture for Smartcards to Gain Infinite Memory”, Smart Card Research and Advanced Application Conference (CARDIS'98), 1998.

[Bla95] M. Blaze, “High-Bandwidth Encryption with Low-Bandwidth Smartcards”, AT&T Bell Labs, 1995. (ftp://ftp.research.att.com/dist/mab/ card cipher.ps)

[BPS96] A. Baraani, J. Pieprzyk, R. Safavi-Naini "Security In Databases: A Survey Study", 1996. citeseer.nj.nec.com/baraani-dastjerdi96security.html

[CaB02] The Caspio Bridge DSP. www.caspio.com/bridge.htm

[ChW00] S. Chaudhuri, G. Weikum, “Rethinking Database System Architecture: Towards a Self-tuning RISC-style Database System”, Int. Conf. on VLDB, 2000.

[Dom97] J. Domingo-Ferrer, “Multi-application smart cards and encrypted data processing”, Future Generation Computer Systems, (13), 1997.

[eCr02] The eCriteria DSP. www.ecriteria.net

[FBI01] Computer Security Institute, "CSI/FBI Computer Crime and Security Survey". www.gocsi.com/forms/fbi/pdf.html

[HeW01] J. He, M. Wang, “Cryptography and Relational Database Management Systems”, Int. Database and Engineering and Application Symposium, 2001.

[iBu02] The crypto iButton with Java - (http://www.ibutton.com/)

[ISO98] International Standardization Organization (ISO), *Integrated Circuit(s) Cards with Contacts – Part 1: Physical Characteristics*, ISO/IEC 7816-1, 1998.

[ISO99] International Standardization Organization (ISO), *Integrated Circuit(s) Cards with Contacts – Part 7: Interindustry Commands for Structured Card Query Language (SCQL)*, ISO/IEC 7816-7, 1999.

[Mat00] U. Mattsson, *Secure.Data Functional Overview*, Protegity Technical Paper TWP-0011, 2000. (http://www.protegrity.com/White_Papers.html)

[Mic02] The Microsoft.Net Passport. www.passport.com

[NIS93] National Institute of Standards and Technology, *Announcing the Data Encryption Standard (DES)*, FIPS PUB 46-2, 1993.

[NIS94] National Institute of Standards and Technology, Announcement of Weakness in the Secure Hash Standard, 1994.

[Ora99] Oracle Corp., *Database Security in Oracle8i*, 1999. otn.oracle.com/deploy/security/oracle8i

[Ora00] Oracle Corp., *Advanced Security Administrator Guide*, Release 8.1.7, 2000.

[PBV01] P. Pucheral, L. Bouganim, P. Valduriez, C. Bobineau, "PicoDBMS: Scaling down Database Techniques for the Smartcard", *VLDB Journal*, 10(2-3), 2001.

[Qck02] The Quickbase DSP. https://www.quickbase.com/

[RAD78] R. L. Rivest, L. Adleman and M. L. Dertouzos, “On Data Banks and Privacy Homomorphisms”, Foundations of Secure Computation. Academic Press, 1978.

[RSA93] RSA Laboratories, *PKCS #1: RSA Encryption Standard*, RSA Laboratories Technical Note, 1993.

[Rus01] Ryan Russel et al., *Hack Proofing Your Network*, Syngress Publishing, 2001.

[ScS99] B. Schneier, A. Shostack, “Breaking up is hard to do: Modeling Security Threats for Smart Cards”, *USENIX Symposium on Smart Cards*, 1999.

[Sch96] B. Schneier, *Applied Cryptography*, 2nd Edition, John Wiley & Sons, 1996.

[Sky02] SkyDesk : @Backup. www.backup.com/index.htm

[Swe99] S.W. Smith, S.H. Weingart, Building a High-Performance, Programmable, Secure Coprocessor, Computer Networks (31) - 1999

[Sun99] Sun Microsystems, *JavaCard 2.1 Application Programming Interface Specification*, JavaSoft documentation, 1999.

[Tua99] J.-P. Tual, “MASSC: A Generic Architecture for Multiapplication Smart Cards”, *IEEE Micro Journal*, N° 0272-1739/99, 1999.

[Van98] J.J. Vandewalle, P. Biget, “Extended Memory Card”, European Multimedia Microprocessor Systems and Electronic Commerce Conf., 1998.

Annexe C

Client-Based Access Control Management for XML documents

Luc Bouganim, François Dang Ngoc, Philippe Pucheral

30th International Conference on Very Large Data Bases, VLDB'04

Toronto, September 2004

Client-Based Access Control Management for XML documents

Luc Bouganim[*] François Dang Ngoc[*,**] Philippe Pucheral[*,**]

[*]INRIA Rocquencourt
Domaine de Voluceau
78153 Le Chesnay - France
Firstname.Lastname@inria.fr

[**]PRiSM Laboratory
45, avenue des Etats-Unis
78035 Versailles - France
Firstname.Lastname@prism.uvsq.fr

Abstract

The erosion of trust put in traditional database servers and in Database Service Providers, the growing interest for different forms of data dissemination and the concern for protecting children from suspicious Internet content are different factors that lead to move the access control from servers to clients. Several encryption schemes can be used to serve this purpose but all suffer from a static way of sharing data. With the emergence of hardware and software security elements on client devices, more dynamic client-based access control schemes can be devised. This paper proposes an efficient client-based evaluator of access control rules for regulating access to XML documents. This evaluator takes benefit from a dedicated index to quickly converge towards the authorized parts of a – potentially streaming – document. Additional security mecanisms guarantee that prohibited data can never be disclosed during the processing and that the input document is protected from any form of tampering. Experiments on synthetic and real datasets demonstrate the effectiveness of the approach.

1. Introduction

Access control management is one of the foundation stone of database systems and is traditionally performed by the servers, the place where the trust is. This situation, however, is rapidly evolving due to very different factors: the suspicion about Database Service Providers (DSP) regarding data confidentiality preservation [HIL02, BoP02], the increasing vulnerability of database servers facing external and internal attacks [FBI03], the emergence of decentralized ways to share and process data thanks to peer-to-peer databases [NOT03] or license-based distribution systems [XrM] and the ever-increasing concern of parents and teachers to protect children by controlling and filtering out what they access on the Internet [PIC].

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

The common consequence of these orthogonal factors is to move access control from servers to clients. Due to the intrinsic untrustworthiness of client devices, all client-based access control solutions rely on data encryption. The data are kept encrypted at the server and a client is granted access to subparts of them according to the decryption keys in its possession. Sophisticated variations of this basic model have been designed in different context, such as DSP [HIL02], database server security [HeW01], non-profit and for-profit publishing [MiS03, BCF01, Med] and multilevel databases [AkT82, BZN01, RRN02]. These models differ in several ways: data access model (pulled vs. pushed), access right model (DAC, RBAC, MAC), encryption scheme, key delivery mechanism and granularity of sharing. However these models have in common to minimize the trust required on the client at the price of a static way of sharing data. Indeed, whatever the granularity of sharing, the dataset is split in subsets reflecting a current sharing situation, each encrypted with a different key, or composition of keys. Thus, access control rules intersections are precompiled by the encryption. Once the dataset is encrypted, changes in the access control rules definition may impact the subset boundaries, hence incurring a partial re-encryption of the dataset and a potential redistribution of keys.

Unfortunately, there are many situations where access control rules are user specific, dynamic and then difficult to predict. Let us consider a community of users (family, friends, research team) sharing data via a DSP or in a peer-to-peer fashion (agendas, address books, profiles, research experiments, working drafts, etc.). It is likely that the sharing policies change as the initial situation evolves (relationship between users, new partners, new projects with diverging interest, etc.). The exchange of medical information is traditionally ruled by strict sharing policies to protect the patient's privacy but these rules may suffer exceptions in particular situations (e.g., in case of emergency) [ABM03], may evolve over time (e.g., depending on the patient's treatment) and may be subject to provisional authorizations [KmS00]. In the same way, there is no particular reason for a corporate database hosted by a DSP to have more static access control rules than its home-administered counterpart [BoP02][1] . Regarding parental

[1] In [BoP02], we identified the need for separating the concern between encryption and access right management and we proposed a solution to protect a relational database server from internal attacks conducted by a

control, neither Web site nor Internet Service Provider can predict the diversity of access control rules that parents with different sensibility are willing to enforce. Finally, the diversity of publishing models (non-profit or lucrative) leads to the definition of sophisticated access control languages like XrML or ODRL [XrM, ODR]. The access control rules being more complex, the encrypted content and the licenses are managed through different channels, allowing different privileges to be exercised by different users on the same encrypted content.

In the meantime, software and hardware architectures are rapidly evolving to integrate elements of trust in client devices. Windows Media9 [Med] is an example of software solution securing published digital assets on PC and consumer electronics. Secure tokens and smart cards plugged or embedded into different devices (e.g., PC, PDA, cellular phone, set-top-box) are hardware solutions exploited in a growing variety of applications (certification, authentication, electronic voting, e-payment, healthcare, digital right management, etc.). Finally, TCPA [TCP] is a hybrid solution where a secured chip is used to certify the software's installed on a given platform, preventing them from hacking[2] . Thus, Secure Operating Environments (SOE) become a reality on client devices [Vin02]. SOE guarantee a high tamper-resistance, generally on limited resources (e.g., a small portion of stable storage and RAM is protected to preserve secrets like encryption keys and sensitive data structures).

The objective of this paper is to exploit these new elements of trust in order to devise smarter client-based access control managers. The goal pursued is being able to evaluate dynamic and personalized access control rules on a ciphered input document, with the benefit of dissociating access rights from encryption. The considered input documents are XML documents, the de-facto standard for data exchange. Authorization models proposed for regulating access to XML documents use XPath expressions to delineate the scope of each access control rule [BCF01, GaB01, DDP02]. Having this context in mind, the problem addressed in this paper can be stated as follows.

Problem statement

- *To propose an efficient streaming access control rules evaluator*
  The streaming requirement is twofold. First, the evaluator must adapt to the memory constrained SOE, thereby precluding materialization (e.g., building a DOM representation of the document). Second, some target applications mentioned above are likely to consume streaming documents. Efficiency is, as usual, an important concern.

Database Administrator.

[2] Architectures like TCPA are controversial today. Our objective is not to fuel this debate. But, clearly, secured client-based architectures are on the way and considering them to design new security models, new ways to protect data confidentiality and privacy is undoubtedly an important challenge. The real danger would be to leave a single actor or consortium decide about a unique security model that imposes to everyone.

- *To guarantee that prohibited information is never disclosed*
  The access control being realized on the client device, no clear-text data but the authorized ones must be made accessible to the untrusted part of this client device.
- *To protect the input document from any form of tampering*
  Under the assumption that the SOE is secure, the only way to mislead the access control rule evaluator is to tamper the input document, for example by substituting or modifying encrypted blocks.

Contributions

To tackle this problem, this paper makes the following contributions:

1. *Accurate streaming access control rules evaluator*
   We propose a streaming evaluator of XML access control rules, supporting a robust subset of the XPath language. At first glance, one may consider that evaluating a set of XPath-based access control rules and a set of XPath queries over a streaming document are equivalent problems [DF03, GMO03, CFG02]. However, access control rules are not independent. They may generate conflicts or become redundant on given parts of the document. The proposed evaluator detects accurately these situations and exploits them to stop eagerly rules becoming irrelevant.
2. *Skip index*
   We design a streaming and compact index structure allowing to quickly converge towards the authorized parts of the input document, while skipping the others, and to compute the intersection with a potential query expressed on this document (in a pull context). Indexing is of utmost importance considering the two limiting factors of the target architecture: the cost of decryption in the SOE and the cost of communication between the SOE, the client and the server. This second contribution complements the first one to match the performance objective.

Combined together, these two contributions form the core of our client-based XML access control solution. Additional mechanisms are however required to guarantee that prohibited data can never be disclosed during the processing and that the input document is protected from any form of tampering. For the sake of conciseness, these mechanisms are mentioned below but are not discussed further in the paper. The reader interested by these aspects is referred to [BDP04]:

- *Pending predicates management*
  Pending predicates (i.e., a predicate P conditioning the delivery of a subtree S but encountered after S while parsing the document) are difficult to manage in a streaming fashion. We propose a strategy to detect eagerly the pending parts of the document, to skip them at parsing time and to reassemble afterwards the relevant pending parts at the right place in the final

result. The way pending predicates are managed guarantees that prohibited data can never be disclosed on the client device.

- *Random integrity checking*
  We combine hashing (Merkle hash tree [Mer90]) and encryption (Cypher Block Chaining [Sch96]) techniques to make the integrity of the document verifiable in a streaming way, despite the forward and backward random accesses generated by the use of the skip index and by the management of pending predicates.

The paper is organized as follows. Section 2 introduces the XML access control model we consider and illustrates it on a motivating example. Sections 3 and 4 detail the two main contributions mentioned above. Section 5 presents experimental results based on both synthetic and real datasets. Section 6 concludes. Related works are addressed throughout each section.

2. Access control model

Access control model semantics

Several authorization models have been recently proposed for regulating access to XML documents. Most of these models follow the well-established Discretionary Access Control (DAC) model [BCF01, GaB01, DDP02], even though RBAC and MAC models have also been considered [Cha00, CAL02]. We introduce below a simplified access control model for XML, inspired by Bertino's model [BCF01] and Samarati's model [DDP02] that roughly share the same foundation. Subtleties of these models are ignored for the sake of simplicity.

In this simplified model, access control rules, or access rules for short, take the form of a 3-uple <*sign*, *subject*, *object*>. *Sign* denotes either a permission (positive rule) or a prohibition (negative rule) for the read operation. *Subject* is self-explanatory. *Object* corresponds to elements or subtrees in the XML document, identified by an XPath expression. The expressive power of the access control model, and then the granularity of sharing, is directly bounded by the supported subset of the XPath language. In this paper, we consider a rather robust subset of XPath denoted by $XP^{\{[],*,//\}}$ [MiS02]. This subset, widely used in practice, consists of node tests, the child axis (/), the descendant axis (//), wildcards (*) and predicates or branches […]. Attributes are handled in the model similarly to elements and are not further discussed.

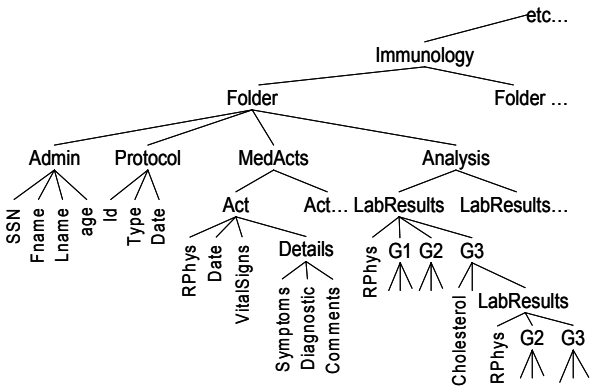
The cascading propagation of rules is implicit in the model, meaning that a rule propagates from an object to all its descendants in the XML hierarchy. Due to this propagation mechanism and to the multiplicity of rules for a same user, a conflict resolution principle is required. Conflicts are resolved using two policies: *Denial-Takes-Precedence* and *Most-Specific-Object-Takes-Precedence*. Let assume two rules R1 and R2 of opposite sign. These rules may conflict either because they are defined on the same object, or because they are defined respectively on two different objects O1 and O2, linked by an ancestor/descendant relationship (i.e., O1 is ancestor of O2). In the former situation, the *Denial-Takes-Precedence* policy favors the negative rule. In the latter situation, the *Most-Specific-Object-Takes-Precedence* policy favors the rule that applies directly to an object against the inherited one (i.e., R2 takes precedence over R1 on O2). Finally, if a subject is granted access to an object, the path from the document root to this object is granted too (names of denied elements in this path can be replaced by a dummy value). This *Structural* rule keeps the document structure consistent with respect to the original one.

The set of rules attached to a given subject on a given document is called an *access control policy*. This policy defines an authorized view of this document and, depending on the application context, this view may be queried. We consider that queries are expressed with the same XPath fragment as access rules, namely $XP^{\{[],*,//\}}$. Semantically, the result of a query is computed from the authorized view of the queried document (e.g., predicates cannot be expressed on denied elements even if these elements do not appear in the query result). However, access rules predicates can apply on any part of the initial document.

Motivating example

We use an XML document representing medical folders to illustrate the semantics of the access control model and to serve as motivating example. A sample of this document is pictured in Figure 1, along with the access control policies associated to three profiles of users: secretaries, doctors and medical researchers. A secretary is granted access only to the patient's administrative subfolders. A doctor is granted access to the patient's administrative subfolders and to all medical acts and analysis of her patients, except the details for acts she didn't carry out herself. Finally, a researcher is granted access only to the laboratory results and the age of patients who have subscribed to a protocol test of type G3, provided the measurement for the element Cholesterol does not exceed 250mg/dL.

Medical applications exemplify the need for dynamic access rules. For example, a researcher may be granted an exceptional and time-limited access to a fragment of all medical folders where the rate of Cholesterol exceeds 300mg/dL (a rather rare situation). A patient having subscribed to a protocol to test the effectiveness of a new treatment may revoke this protocol at any time due to a degradation of her state of health or for any other personal reasons. Models compiling access control policies in the data encryption cannot tackle this dynamicity. However, the reasons to encrypt the data and delegate the access control to the clients are manifold: exchanging data among medical research teams in a protected peer-to-peer fashion, protect the data from external attacks as well as from internal attacks. The latter aspect is particularly important in the medical domain due to the very high level of confidentiality attached to the data and to the very high level of decentralization of the information system (e.g., small clinics and general practitioners are prompted to subcontract the management of their information system).



Doctor access control policy
D1: ⊕, //Folder/Admin D2: ⊕, //MedActs[//RPhys = USER] D3: ⊖, //Act[RPhys != USER]/Details D4: ⊕, //Folder[MedActs//RPhys = USER]/Analysis
Researcher access control policy
R1: ⊕, //Folder[Protocol]//Age R2: ⊕, //Folder[Protocol/Type=G3//LabResults//G3 R3: ⊖, //G3[Cholesterol > 250] Rules 2 & 3 occur for each of the 10 groups {G1..G10}
Secretary access control policy
S1: ⊕, //Admin

Figure 1: *Hospital XML document*

Target architectures

Figure 2 pictures an abstract representation of the target architecture for the motivating example as well as for the applications mentioned in the introduction. The access control being evaluated on the client, the client device has to be made tamper resistant thanks to a Secure Operating Environment (SOE). As discussed in the introduction, this SOE can rely on software or hardware solutions or on a mix of them. In the sequel of this paper, and up to the performance evaluation section, we make no assumption on the SOE, except the traditional ones: 1) the code executed by the SOE cannot be corrupted, 2) the SOE has at least a small quantity of secure stable storage (to store secrets like encryption keys, 3) the SOE has at least a small quantity of secure working memory (to protect sensitive data structures at processing time). In our context, the SOE is in charge of decrypting the input document, checking its integrity and evaluating the access control policy corresponding to a given (document, subject) pair. This access control policy as well as the key(s) required to decrypt the document can be permanently hosted by the SOE, refreshed or downloaded via a secure channel from different sources (trusted third party, security server, parent or teacher, etc).

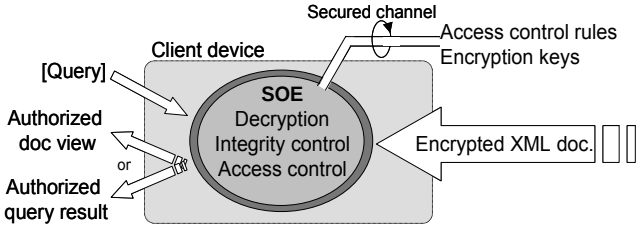


Figure 2: *Abstract target architecture*

3. Streaming the access control

While several access control models for XML have been proposed recently, few papers address the enforcement of these models and, to the best of our knowledge, no one considers access control in a streaming fashion. At first glance, streaming access control resembles the well-known problem of XPath processing on streaming documents. There is a large body of work on this latter problem in the context of XML filtering [DF03, GMO03, CFG02]. These studies consider a very large number of XPath expressions (typically tens of thousands). The primary goal here is to select the subset of queries matching a given document (the query result is not a concern) and the focus is on indexing and/or combining a large amount of queries. One of the first works addressing the precise evaluation of complex XPath expressions over streaming documents is due to [PfC03] which proposes a solution to deliver parts of a document matching a single XPath. While access rules are expressed in XPath, the nature of our problem differs significantly from the preceding ones. Indeed, the rule propagation principle along with its associated conflict resolution policies (see section 2) makes access rules not independent. The interference between rules introduces two new important issues:

– *Access rules evaluation:* for each node of the input document, the evaluator must be capable of determining the set of rules that applies to it and for each rule determining if it applies directly or is inherited. The nesting of the access rules scopes determines the authorization outcome for that node. This problem is made more complex by the fact that some rules are evaluated lazily due to pending predicates.

– *Access control optimization:* the nesting of rule scopes associated with the conflict resolution policies inhibits the effect of some rules. The rule evaluator must take advantage of this inhibition to suspend the evaluation of these rules and even to suspend the evaluation of all rules if a global decision can be reached for a given subtree.

3.1 Access rules evaluation

As streaming documents are considered, we make the assumption that the evaluator is fed by an event-based parser (e.g., SAX [SAX]) raising *open*, *value* and *close* events respectively for each opening, text and closing tag in the input document.

We represent each access rule (i.e., XPath expression) by a non-deterministic finite automaton (NFA) [HjU79]. Figure 3.b pictures the Access Rules Automata (*ARA*) corresponding to two rather simple access rules expressed on an abstract XML document. This abstract example, used in place of the motivating example introduced in Section 2, gives us the opportunity to study several situations (including the trickiest ones) on a simple document. In our ARA representation, a circle denotes a state and a double circle a final state, both identified by a unique *StateId*. Directed edges represent transitions,

triggered by *open* events matching the edge label (either an element name or *). Thus, directed edges represent the child (/) XPath axis or a wildcard depending on the label. To model the descendant axis (//), we add a self-transition with a label * matched by any *open* event. An ARA includes one *navigational path* and optionally one or several *predicate paths* (in grey in the figure). To manage the set of ARA representing a given access control policy, we introduce the following data structures:

- *Tokens and Token Stack:* we distinguish between *navigational tokens* (NT) and *predicate tokens* (PT) depending on the ARA path they are involved in. To model the traversal of an ARA by a given token, we actually create a token proxy each time a transition is triggered and we label it with the destination StateId. The terms token and token proxy are used interchangeably in the rest of the paper. The navigation progress in all ARA is memorized thanks to a unique stack-based data structure called *Token Stack*. The top of the stack contains all active NT and PT tokens, i.e. tokens that can trigger a new transition at the next incoming event. Tokens created by a triggered transition are pushed in the stack. The stack is popped at each close event. The goal of Token Stack is twofold: allowing a straightforward backtracking in all ARA and reducing the number of tokens to be checked at each event (only the active ones, at the top of the stack, are considered).
- *Rule status and Authorization Stack:* Let assume for the moment that access rule expressions do not exploit the descendant axis (no //). In this case, a rule is said to be *active*, – meaning that its scope covers the current node and its subtree – if all final states of its ARA contain a token. A rule is said *pending* if the final state of its navigational path contains a token while the final state of some predicate path has not yet been reached. The *Authorization Stack* registers the NT tokens having reached the final state of a navigational path, at a given depth in the document. The scope of the corresponding rule is bounded by the time the NT token remains in the stack. This stack is used to solve conflicts between rules. The status of a rule present in the stack can be fourfold: *positive-active* (denoted by $\oplus$), *positive-pending* (denoted by $\oplus^?$), *negative-active* (denoted by $\ominus$), *negative-pending* (denoted by $\ominus^?$). By convention, the bottom of the stack contains an implicit *negative-active* rule materializing a closed access control policy (i.e., by default, the set of objects the user is granted access to is empty).
- *Rule instances materialization:* Taking into account the descendant axis (//) in the access rules expressions makes things more complex to manage. Indeed, the same element names can be encountered at different depths in the same document, leading several tokens to reach the final state of a navigational path and predicate paths in the same ARA, without being related together[3]. To tackle this situation, we label navigational and predicate token proxies with the *depth* at which the original predicate token has been created, materializing their participation in the same *rule instance*[4].
- Consequently, a token (proxy) must hold the following information: RuleId (denoted by R, S, …), Navigational/Predicate status (denoted by n or p), StateId and Depth[5]. For example, $Rn2_2$ and $Rp4_2$ (also noted 2_2, 4_2 to simplify the figures) denotes the navigational and predicate tokens created in Rule R's ARA at the time element b is encountered at depth 2 in the document. If the transition between states 4 and 5 of this ARA is triggered, a token proxy $Rp5_2$ will be created and will represent the progress of the original token $Rp4_2$ in the ARA. All these tokens refer to the same rule instance since they are labeled by the same depth. A rule instance is said *active* or *pending* under the same condition as before, taking into account only the tokens related to this instance.
- *Predicate Set:* this set registers the PT tokens having reached the final state of a predicate path. A PT token, representing a predicate instance, is discarded from this set at the time the current depth in the document becomes less than its own depth.

Stack-based data structures are well adapted to the traversal of a hierarchical document. However, we need a direct access to any stack level to update pending information and to allow some optimizations detailed below. Figure 3.c represents an execution snapshot based on these data structures. This snapshot being almost self-explanatory, we detail only a small subset of steps.

- Step 2: the *open* event b generates two tokens $Rn2_2$ and $Rp4_2$, participating in the same rule instance.
- Step 3: the ARA of the negative rule S reaches its final state and an active instance of S is pushed in the Authorization Stack. The current authorization remains negative. Token $Rp5_2$ enters the Predicate Set. The corresponding predicate will be considered true until level 2 of the Token Stack is popped (i.e., until event /b is produced at step 9). Thus, there is no need to continue to evaluate this predicate in this subtree and token $Rp4_2$ can be discarded from the Token Stack.
- Step 5: An active instance of the positive rule R is pushed in the Authorization Stack. The current authorization becomes positive, allowing the delivery of element d.

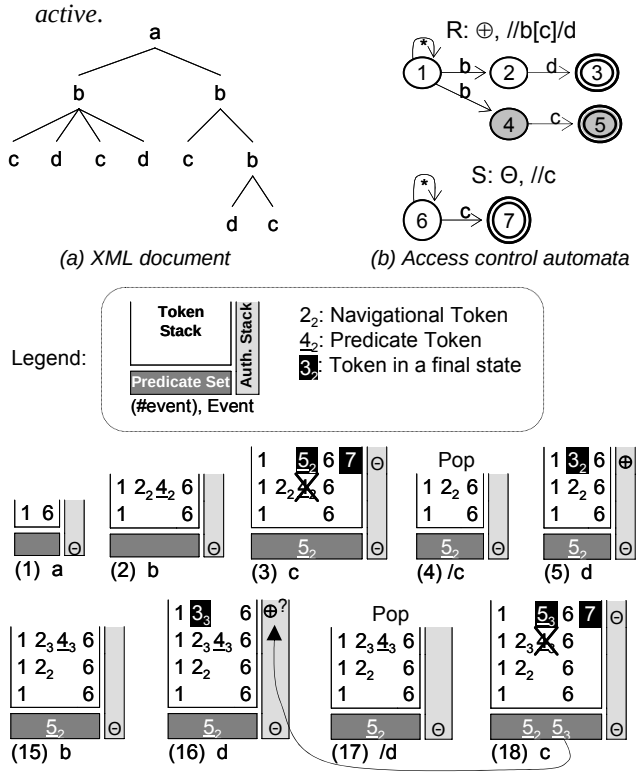
---

[3] The complexity of this problem has been highlighted in [PfC03].

[4] To illustrate this, let us consider the rule R and the right subtree of the document presented in Figure 3. The predicate path final state *5* (expressing //b[c]) can be reached from two different instances of b, respectively located at depth 2 and 3 in the document, while the navigational path final state *3* (expressing //b/d) can be reached only from b located at depth 3. Thus, a single rule instance is valid here, materialized by navigational and predicate tokens proxies labeled with the same depth 3.

[5] If a same ARA contains different predicate paths starting at different levels of the navigational path, a NT token will have in addition to register all PT tokens related to it.

– Step 16: A new instance of R is pushed in the Authorization Stack, represented by token $Rn3_3$. This instance is pending since the token $Rp5_2$ pushed in the Predicate Set at step 12 (event c) does not participate in the same rule instance.
– Step 18: Token $Rp5_3$ enters the Predicate Set, changing the status of the associated rule instance to *positive-active*.



(c) Snapshots of the stack structure

Figure 3: *Execution Snapshot*

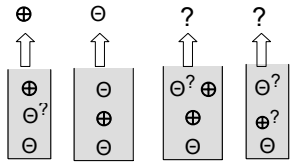
3.2 Conflict Resolution

From the information kept in the Authorization Stack, the outcome of the current document node can be easily determined. The conflict resolution algorithm presented in Figure 4 integrates the closed access control policy (line 1), the *Denial-Takes-Precedence* (line 2) and *Most-Specific-Object-Takes-Precedence* (lines 5 and 7) policies to reach a decision. In the algorithm, AS denotes the Authorization Stack and AS[i].RuleStatus denotes the set of status of all rules registered at level *i* in this stack. In the first call of this recursive algorithm, depth corresponds to the top of AS. Recursion captures the fact that a decision may be reached even if the rules at the top of the stack are pending, depending on the rule status found in the lower stack levels. Note, however, that the decision can remain pending if a pending rule at the top of the stack conflicts with other rules. In that case, the current node has to be buffered, waiting for a delivery condition. This issue is tackled in [BDP04]. The rest of the algorithm is self-explanatory and examples of conflict resolutions are given in the figure.

The DecideNode algorithm presented below considers only the access rules. Things are slightly more complex if queries are considered too. Queries are expressed in XPath and are translated in a non-deterministic finite automaton in a way similar to access rules. However, a query cannot be regarded as an access rule at conflict resolution time. The delivery condition for the current node of a document becomes twofold: (1) the delivery decision must be true and (2) the query must be interested in this node. The first condition is the outcome of the DecideNode algorithm. The second condition is matched if the query is *active*, that is if all final states of the query ARA contain a token, meaning that the current node is part of the query scope.

DecideNode(depth) → Decision ∈ {⊕, Θ,?}

```
1:  If depth = 0 then return 'Θ'
2:    elseif 'Θ'∈ AS[depth].RuleStatus then return 'Θ'
3:      elseif '⊕' ∈ AS[depth].RuleStatus and
4:            'Θ?' ∉ AS[depth].RuleStatus then return '⊕'
5:        elseif DecideNode(depth -1) = 'Θ' and
6:            ∀t∈{'⊕?','⊕'} t∉ AS[depth].RuleStatus then return 'Θ'
7:          elseif DecideNode(depth -1) = '⊕' and
8:                'Θ?' ∉ AS[depth] RuleStatus then return '⊕'
9:            else return '?'
```



Examples of conflict resolution

Figure 4: *Conflict resolution algorithm*

3.3 Optimization issues

The first optimization that can be devised is doing a static analysis of the system of rules composing an access control policy. Query containment property can be exploited to decrease the complexity of this system of rules. Let us denote by $\subseteq$ the containment relation between rules R, S …T. If $S \subseteq R \wedge (R.Sign=S.Sign)$, the elimination of S could be envisioned. However, this elimination is precluded if, for example, $\exists T / T \subseteq R \wedge (T.Sign \neq R.Sign) \wedge (S \subseteq T)$. Thus, rules cannot be pairwise examined and the problem turns to check whether some partial order among rules can be defined wrt. the containment relation, e.g., $\{T_i, \ldots T_k\} \subset \{S_i, \ldots S_k\} \subseteq \{R_i, \ldots R_k\} \wedge \forall i, (R_i.Sign=S_i.Sign \wedge S_i.Sign \neq T_i.Sign) \Rightarrow \{S_i, \ldots S_k\}$ can be eliminated. Note that this strong elimination condition is sufficient but not necessary. For instance, let R and S be two positive rules respectively expressed by /a and /a/b[P1] and T be a negative rule expressed by /a/b[P2]/c. S can still be eliminated while $T \not\subset S$, because the containment holds for each subtree where the two rules are active together. The problem is particularly complex considering that the query containment problem itself has been shown co-NP complete for the class of XPath expressions of interest, that is $XP^{\{[],//,*\}}$ [MiS02]. This issue

could be further investigated since more favorable results have been found for subclasses of $XP^{\{[],//,*\}}$ [ACL01], but this work is outside the scope of this paper.

A second form of optimization is to suspend dynamically the evaluation of ARA that become irrelevant or useless inside a subtree. The knowledge gathered in the Token Stack, Authorization Stack and Predicate Set can be exploited to this end. The first optimization is to suspend the evaluation of a predicate in a subtree as soon as an instance of this predicate has been evaluated to true in this subtree. This optimization has been illustrated by Step 3 of Figure 3.c. The second optimization is to evaluate dynamically the containment relation between active and pending rules and take benefit of the elimination condition mentioned above. From the Authorization Stack, we can detect situations where the following local condition holds: $(T \subset S \subseteq R) \wedge (R.Sign=S.Sign \wedge S.Sign \neq T.Sign)$, the stack levels reflecting the containment relation inside the current subtree. S can be inhibited in this subtree. If stopping the evaluation of some ARA is beneficial, one must keep in mind that the two limiting factors of our architecture are the decryption cost and the communication cost. Therefore, the real challenge is being able to take a common decision for complete subtrees, a necessary condition to detect and skip prohibited subtrees, thereby saving both decryption and communication costs.

Without any additional information on the input document, a common decision can be taken for a complete subtree rooted at node *n* iff: (1) the DecideNode algorithm can deliver a decision D (either $\oplus$ or $\ominus$) for *n* itself and (2) a rule R whose sign contradicts D cannot become active inside this subtree (meaning that all its final states, of navigational path and potential predicate paths, cannot be reached altogether). These two conditions are compiled in the algorithm presented in Figure 5. In this algorithm, AS denotes the Authorization Stack, TS the Token Stack, TS[i].NT (resp. TS[i].PT) the set of NT (resp. PT) tokens registered at level *i* in this stack and top is the level of the top of a stack. In addition, t.RuleInst denotes the rule instance associated with a given token, Rule.Sign the sign of this rule and Rule.Pred a boolean indicating if this rule includes predicates in its definition.

```
DecideSubtree() → Decision ∈ {⊕, ⊖,?}
1:  D = DecideNode(AS.top)
2:  if D = '?' then return '?'
3:  if not (∃ nt ∈TS[top].NT / nt.Rule.Sign ≠ D
4:          and (not nt.Rule.Pred
5:                  or (∃ pt ∈TS[top].PT / pt.RuleInst = nt.RuleInst))
6:  then TS[top].NT = ∅; return (D)
7:  else return '?'
```

Figure 5: *Decision on a complete subtree*

The immediate benefit of this algorithm is to stop the evaluation for any active NT tokens and the main expected benefit is to skip the complete subtree if this decision is $\ominus$. Note however that only NT tokens are removed from the stack at line 6. The reason for this is that active PT tokens must still be considered, otherwise pending predicates could remain pending forever. As a conclusion, a subtree rooted at *n* can be actually skipped iff: (1) the decision for *n* is $\ominus$, (2) the DecideSubtree algorithm decides $\ominus$ and (3) there are no PT token at the top of the Token Stack (which turns to be empty). Unfortunately, these conditions are rarely met together, especially when the descendant axis appears in the expression of rules and predicates. The next section introduces a Skip index structure that gives useful information about the forthcoming content of the input document. The goal of this index is to detect a priori rules and predicates that will become irrelevant, thereby increasing the probability to meet the aforementioned conditions.

When queries are considered, any subtree not contained in the query scope is candidate to a skip. This situation holds as soon as the NT token of the query (or NT tokens when several instances of the same query can co-exist) becomes inactive (i.e., is no longer element of TS[top].NT). This token can be removed from the Token Stack but potential PT tokens related to the query must still be considered, again to prevent pending predicate to remain pending forever. As before, the subtree will be actually skipped if the Token Stack becomes empty.

4. Skip index

This section introduces a new form of indexation structure, called *Skip Index*, designed to detect and skip the unauthorized fragments (wrt. an access control policy) and the irrelevant fragments (wrt. a potential query) of an XML document, while satisfying the constraints introduced by the target architecture (streaming encrypted document, scarce SOE storage capacity).

The first distinguishing feature of the required index is the necessity to keep it encrypted outside of the SOE to guarantee the absence of information disclosure. The second distinguishing feature (related to the first one and to the SOE storage capacity) is that the SOE must manage the index in a streaming fashion, similarly to the document itself. These two features lead to design a very compact index (its decryption and transmission overhead must not exceed its own benefit), embedded in the document in a way compatible with streaming. For these reasons, we concentrate on indexing the structure of the document, pushing aside the indexation of its content. Structural summaries [ABC04] or XML skeleton [BGK03] could be considered as candidate for this index. Beside the fact that they may conflict with the size and streaming requirements, these approaches do not capture the irregularity of XML documents (e.g., medical folders are likely to differ from one instance to another while sharing the same general structure).

In the following, we propose a highly compact structural index, encoded recursively into the XML document to allow streaming. An interesting side effect of the proposed indexation scheme is to provide new means to further compress the structural part of the document.

4.1 Skip Index encoding scheme

The primary objective of the index is to detect rules and queries that cannot apply inside a given subtree, with the expected benefit to skip this subtree if the conditions stated in section 3.3 are met. Keeping the compactness requirement in mind, the minimal structural information required to achieve this goal is the set of element tags, or tags for short, that appear in each subtree. While this metadata does not capture the tags nesting, it reveals oneself as a very effective way to filter out irrelevant XPath expressions. We propose below data structures encoding this metadata in a highly compact way. These data structures are illustrated in Figure 7.a on an abstract XML document.

- *Encoding the set of descendant tags*: The size of the input document being a concern, we make the rather classic assumption that the document structure is compressed thanks to a dictionary of tags [ABC04, TpH02][6]. The set of tags that appear in the subtree rooted by an element *e*, named $DescTag_e$, can be encoded by a bit array, named $TagArray_e$, of length N_t, where N_t is the number of entries of the tag dictionary. A recursive encoding can further reduce the size of this metadata. Let us call *DescTag(e)* the bijective function that maps $TagArray_e$ into the tag dictionary to compute $DescTag_e$. We can trade storage overhead for computation complexity by reducing the image of *DescTag(e)* to $\text{DescTag}_{\text{parent(e)}}$ in place of the tag dictionary. The length of the *TagArray* structure decreases while descending into the document hierachy at the price of making the *DescTag()* function recursive. Since the number of element generally increases with the depth of the document, the gain is substantial. To distinguish between intermediate nodes and leaves (that do not need the TagArray metadata), an additional bit is added to each node.
- *Encoding the element tags*: In a dictionary-based compression, the tag of each element *e* in the document is replaced by a reference to the corresponding entry in the dictionary. $\text{Log}_2(N_t)$ bits are necessary to encode this reference. The recursive encoding of the set of descendant tags can be exploited as well to compress further the encoding of tags themselves. Using this scheme, $\text{Log}_2(\text{DescTag}_{\text{parent(e)}})$ bits suffice to encode the tag of an element *e*.
- *Encoding the size of a subtree:* Encoding the size of each subtree is mandatory to implement the skip operation. At first glance, $\log_2(\text{size(document)})$ bits are necessary to encode $SubtreeSize_e$, the size of the subtree rooted by an element *e*. Again, a recursive scheme allows to reduce the encoding of this size to $\log_2(SubtreeSize_{\text{parent(e)}})$ bits. Storing the SubtreeSize for each element makes closing tags unnecessary.
- *Decoding the document structure*: The decoding of the document structure must be done by the SOE, efficiently, in a streaming fashion and without consuming much memory. To this end, the SOE stores the tag dictionary and uses an internal *SkipStack* to record the DescTag and SubtreeSize of the current element. When decoding an element *e*, $\text{DescTag}_{\text{parent(e)}}$ and $\text{SubtreeSize}_{\text{parent(e)}}$ are retrieved from this stack and used to decode in turn $TagArray_e$, $SubtreeSize_e$ and the encoded tag of *e*.
- *Updating the document:* In the worst case, updating an element *e* induces an update of the SubtreeSize, the TagArray and the encoded tag of each *e* ancestors and of their direct children. In the best case, only the SubtreeSize of *e* ancestors need be updated. The worst case occurs in two rather infrequent situations. The SubtreeSize of *e* ancestor's children have to be updated if the size of *e* father grows (resp. shrinks) and jumps a power of 2. The TagArray and the encoded tag of *e* ancestor's children have to be updated if the update of *e* generates an insertion or deletion in the tag dictionary.

4.2 Skip index usage

As said before, the primary objective of the Skip index is to detect rules and queries that cannot apply inside a given subtree. This means that any active token that cannot reach a final state in its ARA can be removed from the top of the Token Stack. Let us call *RemainingLabels(t)* the function that determines the set of transition labels encountered in the path separating the current state of a token *t* from the final state of its ARA, and let us call *e* the current element in the document. A token *t*, either navigational or predicate, will be unable to reach a final state in its ARA if $\text{RemainingLabels(t)} \not\subset \text{DescTag}_e$. Note that this condition is sufficient but not necessary since the Skip index does not capture the element tags nesting.

SkipSubtree () → Decision ∈ {true,false}

1: For each token t ∈TS[top].NT ∪ TS[top].PT
2: if RemainingLabels(t) ⊄ DescTag_e then remove t from TS[top]
3: if DecideSubTree() ∈ {'⊖', '?'} and (TS[top].NT = ∅) and
4: (TS[top].PT = ∅) then return true
5:else return false

Figure 6: *Skipping decision*

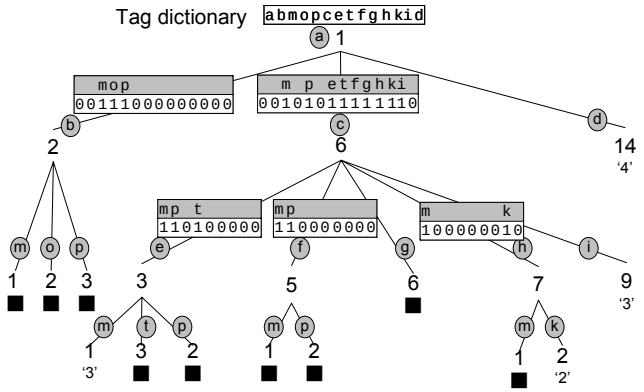
Once this token filtering has been done, the probability for the DecideSubtree algorithm to reach a global decision about the subtree rooted by the current element *e* is greatly increased since many irrelevant rules have been filtered. If this decision is negative (⊖) or pending (?), a skip of the subtree can be envisioned. This skip is actually possible if there are no more active tokens, either navigational or predicate, at the top of the Token Stack. The algorithm SkipSubtree given in Figure 6 decides whether the skip is possible or not. Let us remark that this algorithm should be triggered both on open and close events. Indeed, each

[6] Considering the compression of the document content itself is out of the scope of this paper. Anyway, value compression does not interfere with our proposal as far as the compression scheme remains compatible with the SOE resources.

element may change the decision delivered by the algorithm DecideNode, then DecideSubtree and finally SkipSubtree with the benefit of being able to skip a bigger subtree at the next step.

Figure 7 shows an illustrative XML document and its encoding, a set of access rules and the skips done while analyzing the document. The information in grey is presented to ease the understanding of the indexing scheme but is not stored in the document.

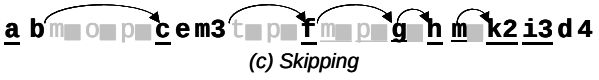
Let us consider the document analysis (for clarity, we use below the real element tags instead of their encoding). At the time element b (leftmost subtree) is reached, all the active rules are stopped thanks to $TagArray_b$ and the complete subtree can be skipped (the decision is $\ominus$ due to the closed access control policy). When element c is reached, Rule R becomes pending. However, the analysis of the subtree continues since $TagArray_c$ does not allow more filtering. When element e is reached, $TagArray_e$ filters out rules R, T and U. Rule S becomes negative-active when the value '3' is encountered below element m. On the closing event, SkipSubtree decides to skip the e subtree. This situation illustrate the benefit to trigger the SkipSubtree at each opening and closing events. The analysis continues following the same principle and leads to deliver the elements underlined in Figure 7.c.



(a) Encoded XML document

R:⊕, / a [d = 4] / c
S:⊖, // c / e[m=3]
T:⊕, // c [// i = 3] // f
U:⊕ , // h [k = 2]

(b) Access Control Rules



(c) Skipping

Figure 7: *Skip Index example*

5. Experimental results

This section presents experimental results obtained from both synthetic and real datasets. We first give details about the experimentation platform. Then, we analyze the storage overhead incurred by the Skip index and compare it with possible variants. Next, we study the performance of access control management and query evaluation. Finally, the global performance of the proposed solution is assessed on four datasets that exhibit different characteristics.

Experimentation platform

The abstract target architecture presented in Section 2 can be instantiated in many different ways. In this experimentation, we consider that the SOE is embedded in an advanced smart card platform. While existing smart cards are already powerful (32 bits CPU running at 30Mhz, 4 KB of RAM, 128KB of EEPROM), they are still too limited to support our architecture, especially in terms of communication bandwidth (9.6Kbps). Our industrial partner, Axalto (the Schlumberger's smart card subsidiary), announces by the end of this year a more powerful smart card equipped with a 32 bits CPU running at 40Mhz, 8KB of RAM, 1MB of Flash and supporting an USB protocol at 1MBps. Axalto provided us with a hardware cycle-accurate simulator for this forthcoming smart card. Our prototype has been developed in C and has been measured using this simulator. Cycle-accuracy guarantees an exact prediction of the performance that will be obtained with the target hardware platform.

As this section will make clear, our solution is strongly bounded by the decryption and the communication costs. The numbers given in Table 1 allow projecting the performance results given in this section on different target architectures. The number given for the smart card communication bandwidth corresponds to a worst case where each data entering the SOE takes part in the result. The decryption cost corresponds to the 3DES algorithm, hardwired in the smart card (line 1) and measured on a PC at 1Ghz (lines 2 and 3).

Context	Communication	Decryption
Hardware based (e.g., future smartcards)	0.5 MB/s	**0.15 MB/s**
Software based - Internet connection	**0.1 MB/s**	1.2 MB/s
Software based - LAN connection	10 MB/s	**1.2 MB/s**

Table 1: *Communication and decryption costs*

In the experiment, we consider three real datasets: *WSU* corresponding to university courses, *Sigmod records* containing index of articles and *Tree Bank* containing English sentences tagged with parts of speech [UWX]. In addition, we generate a synthetic content for the Hospital document depicted in Section 2 (real datasets are very difficult to obtain in this area), thanks to the ToXgene generator [ToX]. The characteristics of interest of these documents are summarized in Table 2.

	WSU	Sigmod	Treebank	Hospital
Size	1.3 MB	350KB	59MB	3.6 MB
Text size	210KB	146KB	33MB	2,1 MB
Maximum depth	4	6	36	8
Average depth	3.1	5.1	7.8	6.8
# distinct tags	20	11	250	89
# text nodes	48820	8383	1391845	98310
# elements	74557	11526	2437666	117795

Table 2: *Documents characteristics*

Index storage overhead

The Skip index is an aggregation of three techniques for encoding respectively tags, lists of descendant tags and subtree sizes. Variants of the Skip index could be devised by combining these techniques differently (e.g., encoding the tags and the subtree sizes without encoding the lists of descendant tags makes sense). Thus, to evaluate the overhead ascribed to each of these metadata, we compare the following techniques. NC corresponds to the original Non Compressed document. TC is a rather classic Tag Compression method and will serve as reference. In TC, each tag is encoded by a number expressed with log_2*(#distinct tags)* bits. We denote by TCS (Tag Compressed and Subtree size) the method storing the subtree size to allow subtrees to be skipped. The subtree size is encoded with log_2*(compressed document size)* bits. In TCS, the closing tag is useless and can be removed. TCSB complements TCS with a bitmap of descendant tags encoded with *#dictinct tags* bits for each element. Finally, TCSBR is the recursive variant of TCSB and corresponds actually to the Skip Index detailed in Section 4. In all these methods, the metadata need be aligned on a byte frontier. Figure 8 compares these five methods on the datasets introduced formerly. These datasets having different characteristics, the Y-axis is expressed in terms of the ratio *structure/(text length)*.

Clearly, TC drastically reduces the size of the structure in all datasets. Adding the subtree size to nodes (TCS) increases the structure size by 50%, up to 150% (big documents require an encoding of about 5 bytes for both the subtree size and the tag element while smaller documents need only 3 bytes). The bitmap of descendant tags (TCSB) is even more expensive, especially in the case of the Bank document which contains 250 distinct tags. TCSBR drastically reduces this overhead and brings back the size of the structure near the TC one. The reason is that the subtree size generally decreases rapidly, as well as the number of distinct tags inside each subtree. For the Sigmod document, TCSBR becomes even more compact than TC.

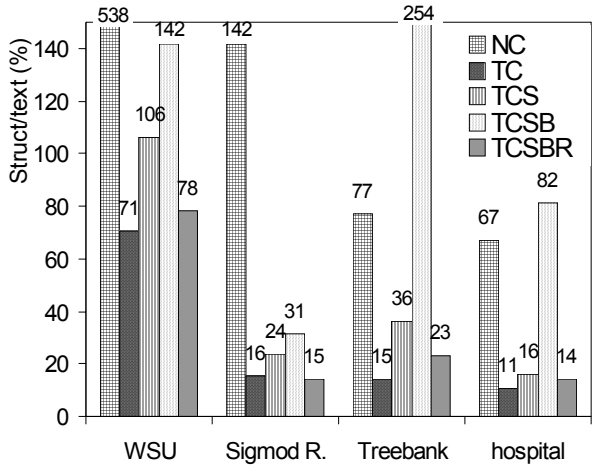


Figure 8: *Index storage overhead*

Access control overhead

To assess the efficiency of our strategy (based on TCSBR), we compare it with: (i) a Brute-Force strategy (BF) filtering the document without any index and (ii) a time lower bound LWB. LWB cannot be reached by any practical strategy. It corresponds to the time required by an oracle to read only the authorized fragments of a document and decrypt it. Obviously, a genuine oracle will be able to predict the outcome of all predicates – pending or not – without checking them and to guess where the relevant data are in the document.

Figure 9 shows the execution time required to evaluate the authorized view of the three profiles (Secretary, Doctor and Researcher) introduced in Section 2 on the Hospital document. Integrity checking is not taken into account here. The size of the compressed document is 2.5MB and the evaluation of the authorized view returns 135KB for the Secretary, 575KB for the Doctor and 95 KB for the Researcher. In order to compare the three profiles despite this size discrepancy, the Y-axis represents the ratio between each execution time and its respective LWB. The real execution time in seconds is mentioned on each histogram. To measure the impact of a rather complex access control policy, we consider that the Researcher is granted access to 10 medical protocols instead of a single one, each expressed by one positive and one negative rule, as in Section 2.

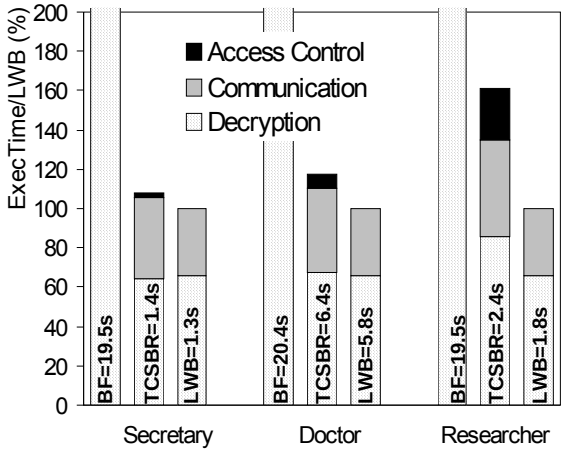


Figure 9: Access control overhead

The conclusions that can be drawn from this figure are threefold. First, the Brute-Force strategy exhibits dramatic performance, explained by the fact that the smart card has to read and decrypt the whole document in order to analyze it. Second, the performance of our TCSBR strategy is generally very close to the LWB (let us recall that LWB is a theoretical and unreachable lower bound), exemplifying the importance of minimizing the input flow entering the SOE. The more important overhead noticed for the Researcher profile compared to LWB is due to the predicate expressed on the protocol element that can

remain pending until the end of each folder. Indeed, if this predicate is evaluated to false, the access rule evaluator will continue – needlessly in the current case – to look at another instance of this predicate. Third, the cost of access control (from 2% to 15%) is largely dominated by the decryption cost (from 53% to 60%) and by the communication cost (from 30% to 38%). The cost of access control is determined by the number of active tokens that are to be managed at the same time. This number depends on the number of ARA in the access control policy and the number of descendant transitions (//) and predicates inside each ARA. This explain the larger cost of evaluating the Researcher access control policy.

Impact of queries

To measure accurately the impact of a query in the global performance, we consider the query //Folder[//Age>v] (v allows us to vary the query selectivity), executed over five different views built from the preceding profiles and corresponding to: a secretary (S), a part-time doctor (PTD) having in charge few patients, a full-time doctor (FTD) having in charge many patients, a junior researcher (JR) being granted access to few analysis results and a senior researcher (SR) being granted access to several analysis results. Figure 10 plots the query execution time (including the access control) as a function of the query result size. The execution time decreases linearly as the query and view selectivity's increase, showing the accuracy of TCSBR. Even if the query result is empty, the execution time is not null since parts of the document have to be analysed before being skipped. The parts of the document that need be analysed depends on the view and on the query. The embedded figure shows the same linearity for larger values of the query result size.

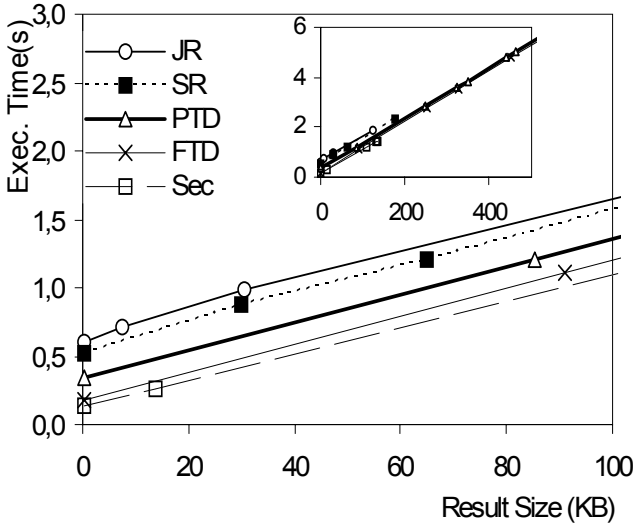


Figure 10: *Impact of queries*

Performance on real datasets

To assess the robustness of our approach when different document structures are faced, we measured the performance of our prototype on the three real datasets WSU, Sigmod and Bank. For these documents we generated random access rules (including // and predicates). Each document exhibits interesting characteristics. The Sigmod document is well-structured, non-recursive, of medium depth and the generated access control policy was simple and not much selective (50% of the document was returned). The WSU document is rather flat and contains a large amount of very small elements (its structure represents 78% of the document size after TCSBR indexation). The Bank document is very large, contains a large amount of tags that appear recursively in the document and the generated access control policy was complex (8 rules). Figure 11 reports the results. We added in the figure the measures obtained with the Hospital document to serve as a basis for comparisons. The figure plots the execution time in terms of throughput for our method and for LWB, both with and without integrity checking. Although integrity checking is not discussed in this paper (see [BDP04] for details), taking its cost into account is mandatory to fully assess our solution. We show that our method tackles well very different situations and produces a throughput ranging from 55KBps to 85KBps depending on the document and the access control policy. These preliminary results as encouraging when compared with xDSL Internet bandwidth available nowadays (ranging from 16KBps to 128KBps).

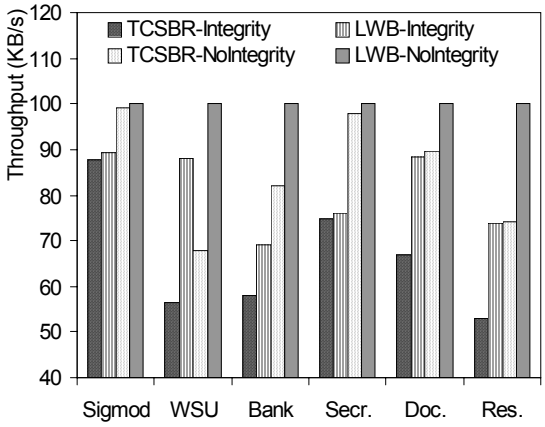


Figure 11: *Performance on real datasets*

6. Conclusion

Important factors motivate today the access control to be delegated to client devices. By compiling the access control policies into the data encryption, existing client-based access control solutions minimize the trust required on the client at the price of a rather static way of sharing data. Our objective is to take advantage of new elements of trust in client devices to propose a client-based access control manager capable of evaluating dynamic access rules on a ciphered XML document.

The contribution of this paper is twofold. First, we proposed a streaming evaluator of access control rules supporting a rather robust fragment of the XPath language. To the best of our knowledge, this is the first paper dealing with XML access control in a streaming fashion. Second, we designed a streaming index structure allowing skipping the irrelevant parts of the input document, with respect to the access control policy and to a potential query. This index is essential to circumvent the inherent bottlenecks of the target architecture, namely the decryption cost and the communication cost. Combined together, these two mechanisms form the core of our client-based XML access control solution. Pending predicate management and random integrity checking complement this solution [BDP04].

Our experimental results have been obtained from a C prototype running on a hardware cycle-accurate smart card simulator provided by Axalto. The global throughput measured is around 70KBps and the relative cost of the access control is less than 20% of the total cost. These first measurements are promising and demonstrate the applicability of the solution. A JavaCard prototype is currently developed and will be submitted to the e-gate'04 software contest organized by SUN and Axalto.

Open issues concern the better use of query containment techniques to improve the optimization before and during the access rules evaluation as well as the definition of more accurate streaming indexation techniques. More generally, client-based security solutions deserve a special attention for the new research perspectives they broaden and for their foreseeable impact on a growing scale of applications.

Acknowledgments

Special thanks are due to Anaenza Maresca, physician at the Tenon hospital (Paris), for her contribution to the definition of the motivating example, inspired by a real-life experience.

References

[ABC04] A. Arion, A. Bonifati, G. Costa, S. D'Aguanno, I. Manolescu, A. Puglies, "Efficient Query Evaluation over Compressed Data", EDBT, 2004.

[ABM03] A. El Kalam, S. Benferhat, A. Miege, R. Baida, F. Cuppens, C. Saurel, P. Balbiani, Y. Deswarte, G. Trouessin, "Organization based access control", IEEE 4th International Workshop on Policies for Distributed Systems and Networks, 2003.

[AkT82] S. Akl and P. Taylor, "Cryptographic solution to a problem of access control in a hierarchy". ACM TOCS, 1983.

[ACL01] S. Amer-Yahia, S. Cho, L. Lakshmanan, and D. Srivastava, "Minimization of tree pattern queries", ACM SIGMOD, 2001.

[BCF00] E.Bertino, S.Castano, E.Ferrari, M.Mesiti, "Specifying and Enforcing Access Control Policies for XML Document Sources", WWW Journal, vol.3, n.3, 2000.

[BCF01] E. Bertino, S. Castano, E. Ferrari, "Securing XML documents with Author-X", IEEE Internet Computing, 2001.

[BDP04] L. Bouganim, F. Dang Ngoc, P. Pucheral, "Client-Based Access Control Management for XML Documents", INRIA internal report, june 2004. www-smis.inria.fr/~bouganim/Publis/BDP04.pdf

[BGK03] P. Buneman, M. Grobe, C. Koch, "Path Queries on Compressed XML", VLDB, 2003

[BoP02] L. Bouganim, P. Pucheral, "Chip-Secured Data Access: Confidential Data on Untrusted Servers", VLDB, 2002.

[BZN01] J.-C. Birget, X. Zou, G. Noubir, B. Ramamurthy, "Hierarchy-Based Access Control in Distributed Environments", IEEE ICC, 2001.

[CAL02] S. Cho, S. Amer-Yahia, L. Lakshmanan, D. Srivastava, "Optimizing the secure evaluation of twig queries", VLDB, 2002.

[CFG02] C Chan, P. Felber, M. Garofalakis, R. Rastogi, "Efficient Filtering of XML Documents with Xpath Expressions", ICDE, 2002.

[Cha00] R.Chandramouli, "Application of XML Tools for Enterprise-Wide RBAC Implementation Tasks", 5th ACM workshop on Role-based Access Control, 2000.

[DDP02] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, P. Samarati, "A Fine-Grained Access Control System for XML Documents", ACM TISSEC, vol. 5, n. 2, 2002.

[DF03] Y. Diao, M. Franklin, "High-Performance XML Filtering: An Overview of YFilter", ICDE, 2003.

[FBI03] Computer Security Institute, "CSI/FBI Computer Crime and Security Survey", www.gocsi.com/forms/fbi/pdf.html

[GaB01] A. Gabillon and E. Bruno, "Regulating access to XML documents. IFIP Working Conference on Database and Application Security, 2001.

[GMO03] T. Green, G. Micklau, M. Onizuka, D. Suciu, "Processing XML streams with Deterministic Automata", ICDT, 2003.

[HeW01] J. He, M. Wang, "Cryptography and Relational Database Management Systems", IDEAS, 2001.

[HIL02] H. Hacigumus, B. Iyer, C. Li, S. Mehrotra, "Executing SQL over encrypted data in the database-service-provider model", ACM SIGMOD, 2002.

[HjU79] J. Hopcroft, J. Ullman, "Introduction to Automata Theory, Languages and Computation", Addison-Wesley, 1979.

[KmS00] M. Kudo, S. Hada, "XML document security based on provisional authorization", ACM CCS, 2000.

[Med] Windows Microsoft Windows Media 9, http://www.microsoft.com/windows/windowsmedia/.

[Mer90] R. Merkle, "A Certified Digital Signature", Advances in Cryptology--Crypto'89, 1989.

[MiS02] G. Miklau and D. Suciu, "Containment and equivalence for an XPath fragment", ACM PODS, 2002.

[MiS03] G. Micklau, D. Suciu, "Controlling Access to Published Data Using Cryptography", VLDB, 2003.

[NOT03] W. Ng, B. Ooi, K. Tan, A. Zhou, "Peerdb: A p2p-based system for distributed data sharing", ICDE, 2003.

[ODR] The Open Digital Rights Language Initiative, http://odrl.net/.

[PfC03] F. Peng, S. Chawathe, "XPath Queries on Streaming Data", ACM SIGMOD, 2003.

[PIC] W3C consortium, "PICS: Platform for Internet Content Selection", http://www.w3.org/PICS.

[RRN02] I. Ray, I. Ray, N. Narasimhamurthi, "A Cryptographic Solution to Implement Access Control in a Hierarchy and More", ACM SACMAT, 2002.

[SAX] Simple API for XML, http://www.saxproject.org/.

[Sch96] B. Schneier, "Applied Cryptography", 2nd Edition, John Wiley & Sons, 1996.

[TCP] Trusted Computing Platform Alliance, http://www.trustedcomputing.org/.

[ToX] ToXgene - the ToX XML Data Generator, http://www.cs.toronto.edu/tox/toxgene/.

[TpH02] P. Tolani, J. Haritsa, "XGRIND: A Query-Friendly XML Compressor", ICDE, 2002.

[UWX] UW XML Data Repository, www.cs.washington.edu/research/xmldatasets/.

[Vin02] R. Vingralek, "GnatDb: A Small-Footprint, Secure Database System", VLDB, 2002.

[XrM] XrML eXtendible rights Markup Language, www.xrml.org/