# Sécurisation matérielle du contrôle d'accès à des documents XML

Luc Bouganim, François Dang Ngoc, Philippe Pucheral

# Sécurisation matérielle du contrôle d'accès à des documents XML

**Luc Bouganim — François Dang Ngoc — Philippe Pucheral**

** I.N.R.I.A. Rocquencourt*
*Domaine de Voluceau*
*78153 Le Chesnay Cedex*
*{Prénom.Nom}@inria.fr*

RÉSUMÉ. *La baisse de confiance accordée aux serveurs traditionnels de bases de données et aux fournisseurs de services de bases de données (Database Service Providers), l'intérêt croissant pour la distribution sélective de données et la nécessité de protéger les enfants de contenus digitaux suspects sont différents facteurs qui amènent à migrer le contrôle d'accès du serveur vers les clients. Pour cela, plusieurs schémas de chiffrement peuvent être utilisés mais tous imposent un partage statique des données. L'apparition d'éléments sécurisés matériels et logiciels sur divers terminaux clients permet de proposer des schémas dynamiques de contrôle d'accès. Cet article présente une solution sécurisée et efficace pour évaluer sur le client des règles de contrôle d'accès sur des documents XML. Cet évaluateur tire parti d'un index approprié pour converger rapidement vers les parties autorisées d'un document arrivant éventuellement en flux sur le terminal client. Des mécanismes supplémentaires de sécurité garantissent que les données non autorisées ne sont jamais révélées lors du traitement et que le document d'entrée est protégé contre toute forme de modification illicite. L'approche est validée par des mesures réalisées sur des données synthétiques et réelles.*

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 mechanisms 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.*

MOTS-CLÉS : *contrôle d'accès XML, confidentialité des données, gestion ubiquitaire de données, carte à puce*

KEYWORDS: *XML access control, data confidentiality, ubiquitous data management, smart card*

## 1. Introduction

La gestion du contrôle d'accès, un des éléments fondamentaux des systèmes de bases de données, est traditionnellement effectuée sur les serveurs, à qui les utilisateurs accordent leur confiance. Cependant, cette situation évolue rapidement à cause de plusieurs facteurs : la méfiance vis à vis des hébergeurs de données ou *Database Service Providers* (DSP) concernant la préservation de la confidentialité des données [HIL02, BoP02], la vulnérabilité croissante des serveurs de bases de données face aux attaques externes et internes [FBI03], l'émergence de moyens décentralisés pour partager les données grâce aux bases de données pair à pair [NOT03] et aux systèmes de distribution basé sur des licences [XrM], et l'inquiétude croissante des parents et des enseignants vis à vis des contenus digitaux téléchargés sur Internet par des mineurs [PIC].

La conséquence commune de ces facteurs orthogonaux est la nécessité de faire migrer le contrôle d'accès des serveurs vers les clients. Du fait de l'absence intrinsèque de sécurité des terminaux clients, toutes les solutions de gestion du contrôle d'accès sur le client reposent sur le chiffrement de données. Les données sont stockées chiffrées sur le serveur et un client ne peut accéder qu'aux parties pour lesquelles il possède la clé de déchiffrement. Plusieurs améliorations de ce modèle ont été proposées dans des contextes très variés comme l'hébergement de données [HIL02], la sécurité des serveurs de bases de données [HeW01], la publication de données lucrative et non-lucrative [MiS03, BCF01, Med] et les bases de données multi-niveaux [AkT82, BZN01, RRN02]. Ces modèles diffèrent sur plusieurs points : le modèle d'accès aux données (interrogation vs. diffusion), le modèle de droit d'accès (DAC, RBAC, MAC), le schéma de chiffrement, le mécanisme de distribution des clés et la granularité de partage. Cependant, ces modèles 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) [ABM03], peuvent évoluer avec le temps (e.g., en fonction du traitement en cours du patient) et peuvent être sujet à des autorisations temporaires [KmS00]. 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 gérées localement [BoP02][1]. En ce qui concerne le contrôle parental, ni les gestionnaires de sites Web ni les fournisseurs d'accès Internet ne peuvent prédire la diversité des règles de contrôle d'accès que les parents, avec leurs sensibilités différentes, veulent voir appliquer à leurs enfants. Finalement, la diversité des modèles de publication (lucratif et non-lucratif) 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.

Parallèlement, les architectures matérielles et logicielles évoluent rapidement et intègrent des éléments de confiance dans les terminaux clients. Windows Media 9 [Med] est un exemple de solution logicielle qui sécurise du contenu numérique publié sur le PC et d'autres terminaux électroniques. Les tokens sécurisés et les cartes à puce reliés ou intégrés dans divers appareils (e.g., PC, PDA, téléphone portable, set-top-box) sont des solutions matérielles utilisées dans un nombre croissant d'applications (certification, authentification, vote électronique, paiement électronique, santé, gestion de droits digitaux, etc.). Finalement, TCPA [TCP] est une solution hybride où une puce sécurisée est utilisée pour certifier le logiciel installé sur une plate-forme donnée, l'empêchant ainsi d'être piraté[2]. Ainsi, les Secure Operating Environments (SOE) deviennent aujourd'hui une réalité sur les terminaux clients [Vin02]. Les SOE garantissent une grande résistance aux attaques, généralement sur des ressources limitées (e.g., une petite portion de mémoire stable et de RAM sont protégées pour garder secret des données comme les clés de chiffrement et des structures de données sensibles).

---

1. Dans [BoP02], nous avons identifié le besoin de séparer la gestion des droits d'accès du schéma de chiffrement. Nous avons également proposé une solution pour protéger une base de données hébergée sur un DSP des attaques internes conduites par l'administrateur de bases de données.

2. Les architectures comme TCPA font aujourd'hui l'objet d'une controverse. Notre objectif n'est pas d'alimenter ce débat. Cependant on peut constater que les architectures basées sur un client sécurisé se développent de plus en plus et les considérer pour concevoir des nouveaux modèles de sécurité, de nouveaux moyens pour protéger la confidentialité et la privacité des données est indéniablement un challenge important. Le vrai danger serait de laisser un seul acteur ou consortium décider d'un modèle unique de sécurité qui s'imposerait à tous.

L'objectif de cet article est d'exploiter ces nouveaux éléments de confiance pour établir de meilleures solutions de gestion du contrôle d'accès sur le client. 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. Les documents considérés sont des documents XML, le standard de fait pour l'échange de données. Les modèles d'autorisation proposés pour réguler l'accès aux documents XML utilisent des expressions XPath pour définir la portée de chacune des règles de contrôle d'accès [BCF01, GaB01, DDP02]. Dans ce contexte, nous définissons le problème attaqué dans cet article de la manière suivante :

– *Proposer une évaluation en flux efficace de règles de contrôle d'accès*

  L'intérêt du flux est double. Premièrement, l'évaluation doit s'adapter aux contraintes mémoire du SOE, interdisant de ce fait toute matérialisation (e.g., construire une représentation DOM du document). Deuxièmement, parmi les applications mentionnées précédemment, certaines, comme les applications vidéo, considérent des documents arrivant en flux. L'efficacité est quant à elle une préoccupation majeure et récurrente.

– *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 d'entrée, par exemple en substituant ou en modifiant des blocs chiffrés de ce document.

**Contributions**

Pour résoudre ce problème, cet article apporte les contributions suivantes :

1. *Evaluation en flux de règles de contrôle d'accès*
   Nous proposons un évaluateur en flux de règles de contrôle d'accès XML supportant un sous-ensemble conséquent du langage XPath. A première vue, évaluer un ensemble de règles de contrôle d'accès basé sur XPath et évaluer des requêtes XPath sur un document arrivant en flux paraissent des problèmes équivalents [DF03, GMO03, CFG02]. Cependant, les règles de contrôle d'accès ne sont pas indépendantes et peuvent générer des conflits ou devenir redondantes sur certaines parties du document. L'évaluateur proposé détecte de manière précise ces situations et en tire profit pour stopper activement les règles qui deviennent non pertinentes.

2. *Index de Saut*

Nous proposons une structure compacte d'indexation 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.

Combinées ensemble, ces deux contributions forment le noyau de notre solution de gestion du contrôle d'accès sur le client. Des mécanismes supplémentaires sont cependant requis pour garantir que les données non autorisées ne sont jamais révélées pendant le traitement et que le document d'entrée est protégé contre toute forme de modification illicite. Pour des raisons de concision, ces mécanismes sont mentionnés ci-dessous mais ne sont pas développés dans cet article. Le lecteur intéressé par ces aspects pourra se référer à [BDP04] :

- *Gestion des prédicats en attente*
  Les prédicats en attente (i.e., un prédicat P qui conditionne la délivrance d'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 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.

- *Vérification de l'intégrité sur des fragments de document aléatoires*
  Nous combinons des techniques de hachage (arbre de hachage de Merkle [Mer90]) et de chiffrement (Cipher Block Chaining ou CBC [Sch96]) pour vérifier l'intégrité du document en flux, malgré 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.

Cet article est organisé comme suit. La section 2 introduit le modèle de contrôle d'accès XML considéré ici et l'illustre sur un exemple motivant l'intérêt de l'approche. Les sections 3 et 4 détaillent les deux principales contributions mentionnées ci-dessus. La section 5 présente les résultats expérimentaux basés sur des jeux de données synthétiques et réels. Finalement, la section 6 conclut. Les travaux connexes sont mentionnés dans chaque section.

## 2. Modèle de contrôle d'accès

### Sémantique du modèle de contrôle d'accès

Plusieurs modèles d'autorisation ont été récemment proposés pour réguler l'accès à des documents XML. La plupart de ces modèles suivent le modèle discrétionnaire (DAC) [BCF01, GaB01, DDP02], même si les modèles RBAC et MAC ont aussi été considérés [Cha00, CAL02]. Nous introduisons ci-dessous un modèle simplifié de contrôle d'accès pour XML, inspiré du modèle de Bertino [BCF01] et de celui de Samarati [DDP02] qui partagent globalement les mêmes principes. Les subtilités de ces modèles sont ignorées pour des raisons de simplicité.

Dans ce modèle simplifié, les règles de contrôle d'accès (ou *règles 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 puissance d'expression du modèle de contrôle d'accès, et donc la granularité du partage, est directement liée au sous-ensemble de XPath supporté.

Dans cet article, nous considérons un sous-ensemble significatif de XPath désigné par $XP^{\{[],*,//\}}$ [MiS02]. Ce sous-ensemble, largement utilisé en pratique, englobe l'utilisation de tests sur les nœuds, de l'axe parent-enfant (/), de l'axe de descendance (//), de jokers (*) et de prédicats […]. Les attributs sont gérés dans ce modèle de manière similaire aux éléments et ne seront donc pas considérés explicitement dans la suite.

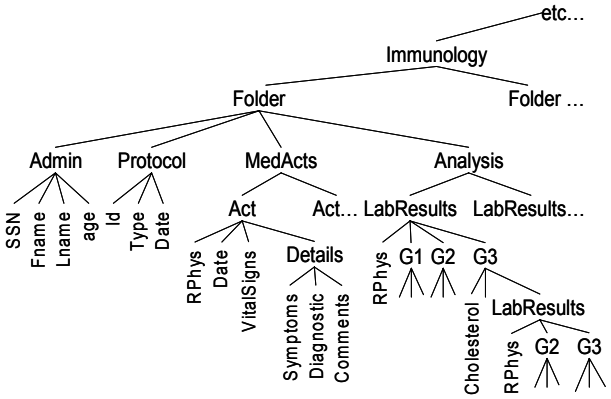
La propagation en cascade des règles est implicite dans ce modèle, ce qui signifie qu'une règle se propage d'un objet à tous ses descendants dans la hiérarchie XML. Etant donné ce mécanisme de propagation et le fait que plusieurs règles peuvent être définies pour un même utilisateur sur un même document, un principe de résolution de conflit est nécessaire. Les conflits sont résolus en utilisant deux politiques : *L'Interdiction-Est-Prioritaire* et *Le-Plus-Spécifique-Est-Prioritaire*. Considérons deux règles R1 et R2 de signe opposé. 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 (i.e., O1 est l'ancêtre de O2). Dans le premier cas, la politique *L'Interdiction-Est-Prioritaire* donne priorité à la règle négative. Dans le second cas, la politique *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). Finalement, un sujet se voyant accorder l'accès à un objet obtient aussi l'accès au chemin reliant la racine du document à cet objet (les noms des éléments interdits sur le chemin peuvent être remplacés par des valeurs factices). Cette règle *Structurelle* conserve ainsi la structure du document qui reste cohérente avec l'original.

L'ensemble des règles associé à un sujet et s'appliquant sur un document donné est appelé une *politique de contrôle d'accès*. Ces politiques définissent une vue autorisée de ce document, qui, suivant le contexte applicatif, peut être interrogée. Nous considérons que les requêtes sont exprimées avec le même sous-ensemble

XPath que celui considéré pour les règles d'accès, c'est à dire $XP^{\{[],*,//\}}$. 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.

**Exemple**

Nous utilisons un document XML représentant des dossiers médicaux pour illustrer la sémantique du modèle de contrôle d'accès. Cet exemple sera utilisé tout au long de l'article. Une partie de ce document est représenté dans la Figure 1, avec les politiques de contrôle d'accès associées à trois profils d'utilisateurs : les secrétaires, les docteurs et les chercheurs du domaine médical. La secrétaire a seulement accès aux sous-dossiers administratifs des patients. Le docteur peut accéder à tous les sous-dossiers administratifs ainsi qu'à tous les actes médicaux à l'exception des détails des actes des patients dont il n'a pas la charge. Finalement, le chercheur ne peut accéder qu'aux résultats de laboratoire et à l'âge des patients qui ont souscrit à un test de protocole de type G3, à condition que la mesure de l'élément Cholestérol n'excède pas 250mg/dL




| **Politique de contrôle d'accès du docteur** |
|---|
| D1: ⊕, //Folder/Admin<br>D2: ⊕, //MedActs[//RPhys = USER]<br>D3: ⊖, //Act[RPhys != USER]/Details<br>D4: ⊕, //Folder[MedActs//RPhys = USER]/Analysis |
| **Politique de contrôle d'accès du chercheur** |
| R1: ⊕, //Folder[Protocol]//Age<br>R2: ⊕, //Folder[Protocol/Type=G3//LabResults//G3<br>R3: ⊖, //G3[Cholesterol > 250]<br>Les règles 2 & 3 s'appliquent de manière similaire pour chacun des 10 groupes {G1, ..., G10} |

| Politique de contrôle d'accès de la secrétaire |
| --- |
| S1: ⊕, //Admin |

**Figure 1.** *Document XML Hôpital*

Les applications médicales illustrent bien le besoin d'avoir des règles d'accès dynamiques. Par exemple, un chercheur peut être autorisé à avoir accès à titre exceptionnel et cela pendant une durée limitée à une partie des dossiers médicaux dans le cas où le taux de cholestérol dépasse 300mg/dL (une situation plutôt rare). Un patient ayant souscrit à un protocole pour tester l'efficacité d'un nouveau traitement peut sortir du protocole à n'importe quel moment à cause par exemple d'une dégradation de son état de santé ou pour toute autre raison personnelle. Les modèles qui compilent les politiques de contrôle d'accès dans le schéma de chiffrement ne peuvent gérer cette dynamicité. Il est pourtant souvent nécessaire de chiffrer les données et de déléguer le contrôle d'accès sur le client : échange de données entre plusieurs équipes de recherche médicale d'une manière sécurisée en pair à pair, protection des données aussi bien contre les attaques externes que les attaques internes. Le dernier aspect est particulièrement important dans le domaine médical à cause du haut niveau de confidentialité requis par les données et du très haut niveau de décentralisation du système d'information (e.g., les petites cliniques et les médecins généralistes peuvent être amenés à déléguer la gestion de leur système d'information).

### Architecture cible

La Figure 2 donne une représentation abstraite de l'architecture cible pour les applications mentionnées dans l'introduction. Le contrôle d'accès étant évalué sur le client, le terminal client doit être rendu résistant à toute attaque grâce à un Secure Operating Environment (SOE). Comme mentionné dans l'introduction, le SOE peut reposer sur une solution logicielle ou matérielle ou un mélange des deux. Dans la suite de cet article, et cela jusqu'à la section évaluation, nous ne faisons aucune hypothèse sur le SOE, à l'exception des hypothèses traditionnelles : 1) le code exécuté par le SOE ne peut être corrompu, 2) le SOE possède au moins une petite quantité de stockage stable et sécurisée (pour stocker les données secrètes comme les clés de chiffrement), 3) Le SOE a au moins une petite quantité de mémoire de travail sécurisé (pour protéger les structures de données sensibles au moment du traitement).

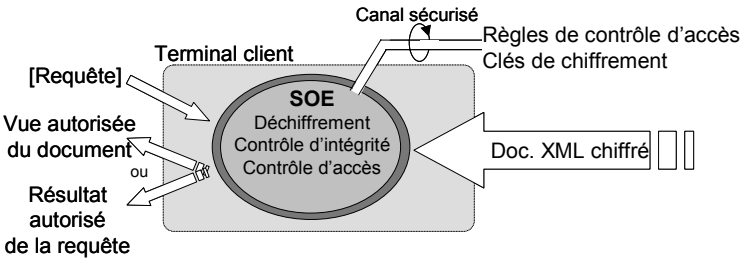


**Figure 2.** *Architecture cible abstraite*

Dans notre contexte, le SOE est en charge de déchiffrer le document d'entrée, de vérifier son intégrité et d'évaluer la politique de contrôle d'accès correspondant à un couple (document, sujet). La politique de contrôle d'accès ainsi que les clés requises pour déchiffrer le document peuvent être stockées de manière permanente dans le SOE, mises à jour ou téléchargées via un canal sécurisé provenant de sources diverses (tiers de confiance, serveur de sécurité, parent ou enseignant, etc.).

## 3. Gestion en flux du contrôle d'accès

Bien que plusieurs modèles de contrôle d'accès XML aient été récemment proposés, peu de travaux portent sur la mise en pratique de ces modèles et à notre connaissance, aucun ne considère l'évaluation du contrôle d'accès en flux. 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. Beaucoup de travaux ont été réalisés sur ce problème dans le contexte de filtrage XML [DF03, GMO03, CFG02]. Ces études considèrent un grand nombre d'expressions XPath (généralement des dizaines de milliers). L'objectif principal est alors de trouver le sous-ensemble de requêtes XPath qui produisent une réponse pour un document donné (on ne s'intéresse pas aux parties du sous-document satisfaisant ces expressions XPath) et 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 [PfC03] qui propose une solution pour délivrer les parties du document qui satisfont une requête XPath unique. Bien que les règles d'accès soient aussi exprimées en XPath, 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 (voir Section 2) 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. Ce problème est rendu plus complexe par le fait que des règles sont évaluées de manière paresseuse à cause de la présence de prédicats en attente.
- *Optimisation du contrôle d'accès :* L'imbrication de la portée des règles associée 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.

### *3.1. Evaluation des règles d'accès*

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 déclenche les évènements *ouverture*, *valeur* et *fermeture* respectivement pour chaque ouverture, texte et fermeture de tag du document d'entrée.

Nous représentons chaque règle d'accès (i.e., expression XPath) par un automate non-déterministe à états finis (NFA) [HjU79]. La Figure 3.b représente les Automates de Règles d'Accès (*ARA*) correspondant à deux règles d'accès relativement simples exprimées sur le document XML abstrait. Cet exemple abstrait, utilisé ici à la place de l'exemple introduit à la section 2, nous donne l'opportunité d'étudier plusieurs situations (dont les plus complexes) sur un document simple. Dans la représentation de nos ARA, un cercle désigne un état et un double cercle un état final, les deux étant identifiés par un identifiant unique d'état *IdEtat*. Les arcs orientés représentent des transitions, qui sont franchies par les évènements *ouverture* satisfaisant l'étiquette de l'arc (nom d'élément ou *). Par conséquent, les arcs orientés représentent l'axe XPath parent-enfant (/) ou un joker, cela dépendant de l'étiquette. Pour modéliser l'axe de descendance (//), nous ajoutons une transition réflexive étiquetée avec * qui est satisfaite par n'importe quel évènement d'*ouverture.* Un ARA se décompose en un *chemin de navigation* et de manière optionnelle d'un ou de plusieurs *chemins de prédicats* (en gris sur la figure). Pour gérer l'ensemble des ARA représentant une politique de contrôle d'accès, nous introduisons les structures de données suivantes :

- *Jetons et Piles de Jetons :* Nous faisons la distinction entre les *jetons de navigation* (JN) et les *jetons de prédicats* (JP) suivant le chemin de l'ARA dans lequel ils sont impliqués. Pour modéliser la traversée d'un ARA par un jeton donné, nous créons en fait une *copie* du jeton à chaque fois qu'une transition est franchie et nous l'étiquetons avec l'identifiant *IdEtat* de l'état de destination. Les termes jeton et copie de jeton seront utilisés indifféremment dans la suite de l'article. La progression de la navigation dans tous les ARA est mémorisée grâce à une structure de pile appelée *Pile de Jetons*. Le sommet de la pile contient tous les jetons actifs JN et JP, i.e., les jetons qui peuvent franchir une nouvelle transition au prochain évènement d'*ouverture.* Les jetons créés par une transition qui a été franchie sont empilés. La pile est dépilée à chaque évènement de *fermeture*. Le but de la Pile de Jetons est double : permettre de faire un retour en arrière de manière direct dans tous les ARA et réduire le nombre de jetons à vérifier à chaque événement (seuls sont considérés les jetons actifs, au sommet de la pile).
- *Statut des règles et la Pile d'Autorisation:* Supposons pour le moment que les expressions de règles d'accès n'exploitent pas l'axe de descendance (pas de //).

Dans ce cas, une règle est dite *active* - signifiant que sa portée couvre le nœud courant ainsi que son sous-arbre – si tous les états finaux de son ARA contiennent un jeton. Une règle est dite *en attente* si l'état final du chemin de navigation contient un jeton alors que l'état final d'un chemin de prédicat n'a pas encore été atteint. La *Pile d'Autorisation* enregistre les jetons JN qui ont atteint l'état final de leur chemin de navigation, à une certaine profondeur dans le document. La portée d'une règle est limitée par la durée pendant laquelle son jeton JN reste dans la pile. Cette pile est utilisée pour résoudre les conflits entre les règles. Le statut d'une règle présent dans la pile peut prendre quatre valeurs différentes : *positive-active* (noté $\oplus$), *positive-en-attente* (noté $\oplus^?$), *négative-active* (noté $\ominus$), *négative-en attente* (noté $\ominus^?$). Par convention, l'élément en bas de la pile contient implicitement une règle *négative-active* qui matérialise ainsi une politique de contrôle d'accès fermée (i.e., par défaut, l'ensemble des objets auxquels l'utilisateur à accès est vide).

– *Matérialisation des instances de règles* : La prise en compte de l'axe de descendance (//) dans les expressions des règles d'accès complexifie le problème. En effet, les mêmes noms d'éléments peuvent être rencontrés à différentes profondeurs dans le même document, amenant plusieurs jetons à atteindre leur état final du chemin de navigation et des chemins de prédicats du même ARA sans toutefois avoir de liens entre eux[3]. Pour résoudre cette situation, nous annotons les copies de jetons de navigation et de prédicat avec la *profondeur* à laquelle le jeton de prédicat a été créé, matérialisant ainsi sa participation à la même *instance de règle*[4]. Ainsi, un jeton doit garder les informations suivantes : IdRegle (noté R, S, ...), le statut du jeton (de navigation ou de prédicat), IdEtat et Profondeur[5]. Par exemple $Rn2_2$ et $Rp4_2$ (aussi noté $2_2$, $4_2$ pour simplifier les figures) désignent les jetons de navigation et de prédicat créés dans l'ARA de la règle R au moment où l'élément b est rencontré à la profondeur 2 dans le document. Quand la transition entre les états 4 et 5 de l'ARA est déclenchée, une copie de jeton $Rp5_2$ est créée et représentera alors la progression du jeton original $Rp4_2$ dans l'ARA. Tous ces jetons sont relatifs à la même instance de règle puisqu'ils sont annotés avec la même profondeur. Une instance de règle est dite *active* ou *en-attente* selon les mêmes critères qu'énoncés précédemment en prenant en compte seulement les jetons relatifs à cette instance.

---

3. La complexité du problème a été soulignée dans [PfC03].

4. Pour illustrer cela, considérons la règle R et le sous-arbre droit du document présenté à la Figure 3. L'état final 5 du chemin de prédicat (exprimant //b[c]) peut être atteint par deux instances différentes de b, une localisée à la profondeur 2 et l'autre à la profondeur 3 dans le document alors que l'état final 3 du chemin de navigation (exprimant //b/d) peut être atteint seulement par le b se trouvant à la profondeur 3. Ainsi, une seule instance de règle est valide ici, matérialisé grâce aux copies de jetons de navigation et de prédicat annotés avec la même profondeur 3.

5. Si un même ARA contient des chemins de prédicats différents commençant à des niveaux différents du chemin de navigation, un jeton JN devra en plus enregistrer tous les jetons JP qui lui sont associés.

– *Ensemble de Prédicats :* Cet ensemble mémorise les jetons JP qui ont atteint leur état final dans le chemin de prédicat. Un jeton JP, représentant une instance de prédicat, est supprimé de cet ensemble au moment où la profondeur courante dans le document devient inférieure à sa propre profondeur.

Les structures de données basées sur une pile sont bien adaptées à la traversée d'un document hiérarchique. Cependant, nous avons besoin d'un accès direct à tous les niveaux de la pile pour mettre à jour les informations en attente et pour permettre de faire certaines optimisations décrites dans la suite de l'article. La Figure 3.c représente une exécution pas à pas basée sur ces structures de données. Cette exécution s'expliquant pratiquement d'elle-même, nous ne détaillons qu'un petit ensemble d'étapes.

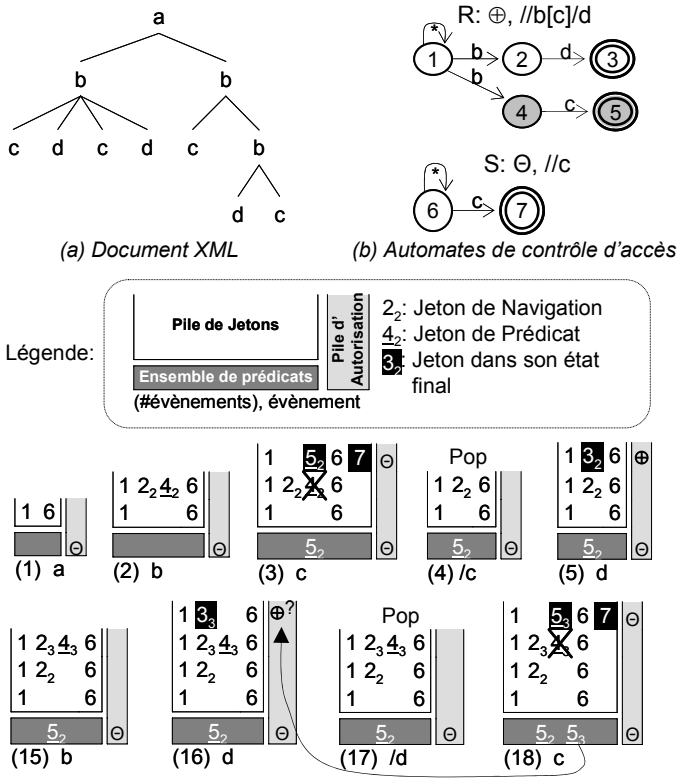



**Figure 3.** *Capture pas à pas d'une exécution*

- Etape 2 : L'évènement d'ouverture b génère deux jetons $Rn2_2$ et $Rp4_2$, participant à la même instance de règle.
- Etape 3 : L'ARA de la règle négative S a atteint son état final et une instance active de S est empilée dans la Pile d'Autorisation. L'autorisation courante reste

négative. Le jeton $Rp5_2$ est inséré dans l'ensemble de Prédicats. Le prédicat correspondant sera considéré comme vrai jusqu'à ce que la profondeur 2 de la Pile de Jetons soit dépilée (i.e., jusqu'à ce que l'événement /b soit déclenché à l'étape 9). Par conséquent, il n'y a pas besoin de continuer à évaluer ce prédicat dans ce sous-arbre et le jeton $Rp4_2$ peut être supprimé de la Pile de Jetons.

- Etape 5 : Une instance active de la règle positive R est empilée dans la Pile d'Autorisation. L'autorisation courante devient positive, permettant de délivrer l'élément d.
- Etape 16 : Une nouvelle instance de R représentée par le jeton $Rn3_3$ est empilée dans la Pile d'Autorisation. Cette instance est en attente car le jeton $Rp5_2$ présent dans l'Ensemble de Prédicats à l'étape 12 (évènement c) ne participe pas à la même instance de règle.
- Etape 18 : Le jeton $Rp5_3$ est inséré dans l'Ensemble des Prédicats, changeant ainsi le statut de l'instance de la règle qui lui est associé en *positive-active*.

### *3.2. Résolution de conflits*

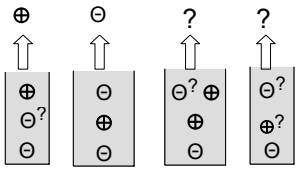
A partir des informations présentes dans la Pile d'Autorisation, l'issue du nœud courant peut être déterminée facilement. L'algorithme de résolution de conflits présenté dans la Figure 4 intègre la politique fermée de contrôle d'accès (ligne 1), les politiques *L'Interdiction-est-Prioritaire* (ligne 2) et *La-Plus-Spécifique-est-Prioritaire* (ligne 5 et 7) pour prendre une décision. Dans cet algorithme, PA désigne la Pile d'Autorisation et PA[i].StatutsRegles désigne l'ensemble des statuts de toutes les règles qui se sont enregistrées au niveau i de la pile. Dans le premier appel à l'algorithme récursif, le niveau correspond à celui du sommet de PA. La récursion matérialise le fait qu'une décision peut être atteinte même si les règles au sommet de la pile sont en attente, et que celle-ci dépend du statut des règles qui se trouvent à un niveau plus bas de la pile. Il faut noter cependant qu'une décision peut rester en attente si une règle en attente se trouvant au sommet de la pile est en conflit avec d'autres règles. Dans ce cas, le nœud courant doit être mis en tampon, en attendant de pouvoir être délivré par une condition. Ce problème est adressé et développé plus en détail dans [BDP04]. Le reste de cet algorithme se comprend de lui-même et des exemples de résolutions de conflits sont donnés dans la figure.

L'algorithme DecisionNoeud présenté ci-dessous considère seulement les règles d'accès. Les choses deviennent plus complexes si des requêtes sont également considérées. Les requêtes sont exprimées en XPath et sont traduites en un automate non-déterministe à états finis de la même manière que les règles d'accès. Cependant, une requête ne peut être vue comme une règle d'accès au moment de la résolution des conflits. La condition pour délivrer le nœud courant d'un document devient double : (1) la décision de délivrance doit être vraie et (2) ce nœud doit être qualifié par la requête. La première condition est le résultat de l'algorithme DecisionNoeud.

La seconde condition est satisfaite si la requête est *active*, c'est à dire si tous les états finaux de l'ARA de la requête contiennent un jeton, ce qui signifie que le nœud courant fait partie de la portée de la requête.

**DecisionNoeud(profondeur)** → Decision ∈ {⊕, ⊖,?}

```
1:  si profondeur = 0 alors retourne '⊖'
2:    sinon si  '⊖'∈ PA[profondeur].StatutsRegles alors retourne '⊖'
3:      sinon si '⊕' ∈ PA[profondeur].StatutsRegles et
4:            '⊖?' ∉ PA[profondeur].StatutsRegles alors retourne '⊕'
5:        sinon si DecisionNoeud(profondeur -1) = '⊖' et
6:            ∀t∈{'⊕?','⊕'} t∉ PA[profondeur].StatutsRegles alors retourne '⊖'
7:         sinon si DecisionNoeud(profondeur -1) = '⊕' et
8:                '⊖?' ∉ PA[profondeur] StatutsRegles alors retourne '⊕'
9:            sinon retourne '?'
```




*Exemples de résolution de conflits*

**Figure 4.** *Algorithme de résolution de conflits*

### *3.3 Optimisations*

La première optimisation qui peut être envisagée est de faire une analyse statique du système de règles qui composent la politique de contrôle d'accès. La propriété d'inclusion des règles peut être exploitée pour diminuer la complexité du système de règles. Notons $\subseteq$ la relation d'inclusion entre les règles R, S, …T. Si $S \subseteq R \wedge (R.Signe=S.Signe)$, l'élimination de S peut être envisagée. Mais cette élimination est à proscrire si par exemple, $\exists T / T \subseteq R \wedge (T.Signe \neq R.Signe) \wedge (S \subseteq T)$. De ce fait, les règles ne peuvent être examinées deux à deux et le problème revient à vérifier si un ordre partiel entre les règles peut être établi par rapport à la relation d'inclusion, e.g., $\{T_i, \ldots T_k\} \subset \{S_i, \ldots S_k\} \subseteq \{R_i, \ldots R_k\} \wedge \forall i, (R_i.Signe=S_i.Signe \wedge S_i.Signe \neq T_i.Signe) \Rightarrow \{S_i, \ldots S_k\}$ peut être éliminé. Notons que cette condition forte d'élimination est suffisante mais pas nécessaire. Par exemple, soient R et S deux règles positives définies respectivement par /a et /a/b[P1], et T une règle négative définie par /a/b[P2]/c. S peut être éliminée alors que $T \not\subset S$, car l'inclusion est valable pour tous les sous-arbres où les deux règles sont actives en même temps. Ce problème est particulièrement complexe sachant que le problème d'inclusion de requête est à lui seul un problème co-NP complet pour la classe de XPath considérée, c'est à dire $XP^{\{[],//,*\}}$ [MiS02]. Ce problème pourrait être étudié plus en détail sachant que des résultats favorables ont été trouvés pour des sous-classes de $XP^{\{[],//,*\}}$ [ACL01], mais ce travail est hors du contexte de cet article.

Une deuxième forme d'optimisation est de suspendre dynamiquement l'évaluation des ARA qui deviennent non-pertinents ou inutiles dans un sous-arbre. Nous pouvons pour cela exploiter les différentes informations contenues dans la Pile de Jetons, la Pile d'Autorisation et l'Ensemble de Prédicats. La première optimisation est alors de suspendre l'évaluation d'un prédicat d'un sous-arbre aussitôt qu'une instance de ce prédicat a été évaluée à vrai dans le sous-arbre. Cette optimisation a été illustrée par l'étape 3 de la Figure 3.c. La seconde optimisation est d'évaluer dynamiquement la relation d'inclusion entre les règles actives et en attente et de tirer profit de la condition d'élimination décrite précédemment. La Pile d'Autorisation permet de détecter les situations où la condition s'applique localement : $(T \subset S \subseteq R) \wedge (R.Signe=S.Signe \wedge S.Signe \neq T.Signe)$, les niveaux de la pile reflétant la relation d'inclusion dans le sous-arbre courant. Ainsi S peut être inhibé dans ce sous-arbre. Même s'il est profitable de suspendre l'évaluation d'une règle, il faut garder à l'esprit que les deux facteurs limitants de notre architecture sont le coût de déchiffrement et de communication. Par conséquent le vrai challenge est d'être capable de prendre une décision pour l'ensemble d'un sous-arbre, une condition nécessaire pour détecter et sauter les sous-arbres non-autorisés afin de réduire les coûts de déchiffrement et de communication

Sans information supplémentaire sur le document d'entrée, une décision peut être prise pour l'ensemble d'un sous-arbre de racine *n* ssi : (1) l'algorithme DecisionNoeud peut retourner une décision D ($\oplus$ ou $\ominus$) pour *n* lui-même et (2) aucune règle R dont le signe est contraire à D ne peut devenir active dans le sous-arbre (ce qui signifie que tous ses états finaux, de son chemin de navigation et des chemins de prédicat, ne peuvent être atteints simultanément). Ces deux conditions sont compilées dans l'algorithme présenté à la Figure 5. Dans cet algorithme, PA représente la Pile d'Autorisation, PJ la Pile de Jetons, PJ[i].JN (resp. PJ[i].JP) l'ensemble des jetons JN (resp. JP) enregistrés au niveau *i* de la pile et Sommet qui correspond au sommet de la pile. De plus, j.InstRegle désigne l'instance de la règle associée à un jeton donné, Regle.Signe le signe de cette règle et Regle.Pred un booléen indiquant si cette règle inclut des prédicats dans sa définition.

Le bénéfice immédiat de cet algorithme est de pouvoir arrêter l'évaluation de jetons JN actifs et le bénéfice principal attendu est de pouvoir sauter des sous-arbres complets si la décision est $\ominus$. Notons cependant que seuls les jetons JN sont supprimés de la pile à la ligne 6. La raison est que les jetons JP actifs doivent être tout de même considérés, sinon des prédicats en attente pourraient rester en attente indéfiniment. En résumé, un sous-arbre de racine *n* peut être sauté ssi : (1) la décision pour *n* est $\ominus$, (2) L'algorithme DecisionSousArbre retourne $\ominus$ et (3) si aucun jeton JP ne se trouve au sommet de la Pile de Jetons (qui est alors vide).

```
DecisionSousArbre() → Decision ∈ {⊕, ⊖,?}
1:  D = DecisionNoeud(PA.sommet)
2:  si D = '?' alors retourne '?'
3:  si non (∃ jn ∈PJ[sommet].JN / jn.Regle.Signe ≠ D
4:          et (non jn.Regle.Pred
5:                  ou (∃ jp ∈PJ[sommet].JP / jp.InstRegle = jn.InstRegle))
```

6: alors PJ[sommet].JN = ∅; retourne (D)
7: sinon retourne '?'

**Figure 5.** *Décision sur un sous-arbre complet*

Malheureusement ces conditions sont rarement réunies simultanément, en particulier quand l'axe de descendance est utilisé dans les expressions de règles et de prédicat. La section suivante introduit une structure d'Index de Saut qui donne des informations utiles à propos du contenu à venir du document. Le but de cet index est de détecter à priori les règles et les prédicats qui deviendront non-pertinents, augmentant ainsi la probabilité de réunir les conditions précédemment mentionnées.

Quand les requêtes sont considérées, tout sous-arbre n'étant pas inclut dans la portée de la requête est candidat à un saut. Cette situation s'applique aussitôt que le jeton JN d'une requête (ou les jetons JN quand plusieurs instances d'une même requête peuvent co-exister) devient inactif (i.e., n'est plus un élément de PJ[top].JN). Ce jeton peut être supprimé de la Pile de Jetons mais les jetons JP doivent toutefois être considérés, encore une fois pour empêcher que les prédicats en attente restent en attente indéfiniment. Comme précédemment, le sous-arbre sera sauté si la Pile de Jeton devient vide.

## 4. Index de Saut

Cette section introduit une nouvelle forme de structure d'index, appelé *Index de Saut*, conçu pour détecter et sauter les fragments non-autorisés (selon la politique de contrôle d'accès) et les fragments non-pertinents (d'après une requête potentielle) d'un document XML, tout en satisfaisant les contraintes introduites par l'architecture cible (document chiffré en flux, peu de capacité mémoire du SOE).

La première contrainte que doit respecter l'index est la nécessité de le garder chiffré en dehors du SOE pour garantir qu'aucune information ne soit révélée. La deuxième contrainte (liée à la première et à la capacité du SOE) est que le SOE doit gérer l'index en flux, de la même manière que le document lui-même. Ces deux contraintes nous amènent à concevoir un index très compact (le surcoût induit par son déchiffrement et sa transmission ne doit pas excéder son propre bénéfice) et intégré dans le document d'une manière compatible avec le flux. Pour ces raisons nous nous concentrons sur l'indexation de la structure du document, en mettant de côté les index sur son contenu. Des résumés structurels [ABC04] ou un squelette XML [BKG03] pourraient être considérés comme des candidats pour cet index. Cependant, en dehors du fait qu'ils peuvent poser un problème de taille ou de lecture en flux, ces approches ne tiennent pas compte des irrégularités des documents XML (e.g., il est très possible que les dossiers médicaux diffèrent d'une instance à l'autre alors qu'ils partagent la même structure générale).

Dans la suite nous proposons un index structurel très compact, encodé récursivement dans le document XML pour permettre une gestion en flux. Une

conséquence intéressante du schéma d'indexation proposé est de permettre de compresser encore mieux les parties structurelles du document.

### *4.1 Schéma d'encodage de l'Index de Saut*

L'objectif principal de cet index est de détecter les règles et les requêtes qui ne peuvent s'appliquer à l'intérieur d'un sous-arbre, avec comme bénéfice attendu de pouvoir sauter le sous-arbre si les conditions mentionnées dans la Section 3.3 sont réunies. 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 d'élément (ou tags) qui sont présents dans chaque sous-arbre. Bien que ces méta-données 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. Nous proposons ci-dessous les structures de données qui encodent ces méta-données d'une manière très compacte. Ces structures de données sont illustrées dans la Figure 7.a sur un exemple abstrait de document XML.

- *Encoder l'ensemble des tags des descendants* : La taille du document d'entrée étant un problème, nous faisons l'hypothèse assez classique que la structure du document est compressée grâce à un dictionnaire de tags [ABC04, TpH02][6]. L'ensemble des tags qui sont présents dans le sous-arbre de racine *e*, appelé $TagsDesc_e$, peut être encodé sur un tableau de bits, appelé $TabTags_e$, de longueur $N_t$, où $N_t$ est le nombre d'entrées dans le dictionnaire de tags. Un encodage récursif peut réduire davantage la taille de ces méta-données. Notons *TagsDesc(e)* la fonction bijective qui associe $TabTags_e$ au dictionnaire de tags pour calculer $TagsDesc_e$. Nous pouvons diminuer la taille de l'index au prix d'une complexité de calcul plus importante en réduisant l'image de *TagsDesc(e)* en $TagsDesc_{parent(e)}$ à la place du dictionnaire de tags. La taille de la structure *TabTags* diminue ainsi au fur et à mesure que l'on descend dans la hiérarchie du document mais la fonction *TagsDesc()* devient alors récursive. Puisque le nombre d'éléments augmente généralement avec la taille du document, le gain est considérable. Pour faire la distinction entre les nœuds intermédiaires et les feuilles (qui n'ont pas besoin de la méta-donnée TabTags), un bit supplémentaire est ajouté à chaque nœud.
- *Encoder les tags d'éléments* : Dans une compression basée sur un dictionnaire, le tag de chaque élément *e* du document est remplacé par une référence à l'entrée correspondante dans le dictionnaire. $Log_2(N_t)$ bits sont nécessaires pour encoder cette référence. L'encodage récursif de l'ensemble des tags peut être exploité également pour compresser encore mieux l'encodage des tags eux-mêmes. En

6. Considérer la compression du contenu du document lui-même est hors du contexte de cet article. Cela étant, elle peut être utilisée conjointement avec notre proposition tant que le schéma de compression reste compatible avec les ressources du SOE.

utilisant ce schéma d'encodage, $Log_2(TagsDesc_{parent(e)})$ bits suffisent à encoder le tag d'un élément *e*.

- *Encoder la taille d'un sous-arbre:* Encoder la taille de chaque sous-arbre est nécessaire pour implémenter l'opération de saut. A première vue, $Log_2$(taille(document)) bits sont requis pour encoder $TailleSousArbre_e$, la taille du sous-arbre de racine *e*. De la même manière, un schéma récursif permet de réduire l'encodage de la taille à $Log_2(TailleSousArbre_{parent(e)})$ bits. De plus, en stockant la taille du sous-arbre pour chaque élément, les tags de fermeture deviennent inutiles.
- *Décoder la structure du document* : Le décodage de la structure du document doit être fait par le SOE de manière efficace, en flux et sans consommer trop de mémoire. Pour cela, le SOE stocke le dictionnaire de tags et utilise une pile interne *PileSaut* pour enregistrer le TagsDesc et le TailleSousArbre de l'élément courant. Quand on décode un élément *e*, $TagsDesc_{parent(e)}$ et $TailleSousArbre_{parent(e)}$ sont récupérés de la pile et utilisés pour décoder à leur tour $TabTags_e$, $TailleSousArbre_e$ et le tag encodé de *e*.
- *Mettre à jour le document :* Dans le pire des cas, mettre à jour un élément *e* résulte en la mise à jour de TailleSousArbre, de TabTags et de l'encodage du tag de chaque ancêtre de l'élément *e* et de leurs fils directs. Dans le meilleur des cas, seulement le TailleSousArbre des ancêtres de *e* ont besoin d'être mis à jour. Le pire cas arrive dans deux situations assez peu courante. L'information TailleSousArbre des fils des ancêtres de *e* doit être mis à jour si la taille du père de *e* augmente (resp. diminue) et saute une puissance de 2. L'information TabTags et l'encodage du tag des fils des ancêtres de *e* doivent être mis à jour si la mise à jour de *e* génère une insertion ou une suppression dans le dictionnaire de tags.

### *4.2 Utilisation de l'Index de Saut*

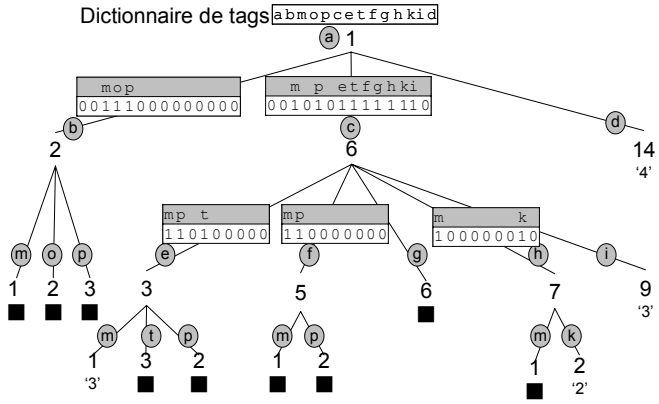
Comme mentionné précédemment, l'objectif principal de l'Index de Saut est de détecter les règles et les requêtes qui ne peuvent s'appliquer à l'intérieur d'un sous-arbre. Cela signifie que tous les jetons actifs qui ne peuvent atteindre un état final dans leurs ARA peuvent être supprimés du sommet de la Pile de Jetons. Soit *LabelsRestants(j)* la fonction qui détermine l'ensemble des tags de transitions rencontrés sur le chemin séparant l'état courant d'un jeton *j* de leur état final sur son ARA, et soit *e* l'élément courant dans le document. Un jeton *j* , de navigation ou de prédicat, ne pourra atteindre un état final de son ARA que si LabelsRestants(j) $\not\subset$ $TagsDesc_e$. Remarquons que cette condition est suffisante mais pas nécessaire puisque l'Index de Saut ne prend pas en compte l'imbrication des tags.

**SautSousArbre () → Decision ∈ {vrai, faux}**

1: pour chaque jeton j ∈PJ[sommet].JN ∪ PJ[sommet].JP
2: Si LabelsRestants(j) ⊄ $TagsDesc_e$ alors supprimer j de PJ[sommet]
3: si DecisionSousArbre () ∈ {'⊖', '?'} et (PJ[sommet].JN = ∅) et
4: (PJ[sommet].JP = ∅) alors retourne vrai
5:sinon retourne faux

**Figure 6.** *Décision de saut*

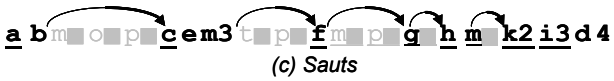
Une fois que ce filtrage de jetons a été fait, la probabilité que l'algorithme DecisionSousArbre atteigne une décision pour le sous-arbre dont la racine est l'élément courant *e* est beaucoup plus grande puisque plusieurs règles non-pertinentes ont été filtrées. Si cette décision est négative (⊖) ou en attente (?), le saut du sous-arbre peut être envisagé. Ce saut est en fait possible s'il n'y a plus de jetons actifs, de navigation ou de prédicat au sommet de la Pile de Jetons. L'algorithme SautSousArbre donné en Figure 6 décide si le saut est possible ou non. Remarquons que cet algorithme pourra être déclenché lors des événements d'ouverture et de fermeture de tags. En effet, chaque élément peut changer la décision retournée par l'algorithme DecisionNoeud, et donc DecisionSousArbre et finalement SautSousArbre avec comme bénéfice de pouvoir sauter un plus grand sous-arbre à l'étape suivante. La Figure 7 montre un exemple illustratif de document XML, son encodage, un ensemble de règles d'accès et les sauts réalisés lors de l'analyse du document. Les informations en gris sont présentées pour simplifier la compréhension du schéma d'indexation mais ne sont pas stockées dans le document.




*(a) Encodage du document XML*

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

*(b) Règles de contrôle d'accès*



*(c) Sauts*

**Figure 7.** *Exemple d'index de saut*

Considérons l'analyse du document (pour des raisons de clarté, nous utiliserons dans la suite le tag de l'élément réel au lieu de son encodage). Au moment où l'élément b (du sous-arbre le plus à gauche) est rencontré, toutes les règles actives sont stoppées grâce à $TabTags_b$ et le sous-arbre complet peut ainsi être sauté (la décision est $\ominus$ à cause de la politique de contrôle d'accès fermée). Quand l'élément c est atteint, la règle R devient en attente. Cependant, l'analyse du sous-arbre continue puisque $TabTags_c$ ne permet pas de filtrer davantage. Quand l'élément e est atteint, $TabTags_e$ filtre les règles R, T et U. La règle R devient négative-active quand la valeur '3' est rencontrée dans l'élément m.

A l'événement de fermeture de tag, l'algorithme SautSousArbre décide de sauter le sous-arbre de racine e. Cette situation illustre bien le bénéfice de déclencher SautSousArbre à chaque événement d'ouverture et de fermeture de tags. L'analyse continue en suivant le même principe et aboutit à délivrer les éléments soulignés dans la Figure 7.c

## 5. Résultats expérimentaux

Cette section présente les résultats expérimentaux obtenus à partir de jeux de données synthétiques et réels. Nous donnons d'abord les détails sur la plate-forme d'expérimentation. Nous analysons ensuite le surcoût du stockage induit par l'Index de Saut et le comparons avec des variantes possibles. Puis nous analysons les performances de la gestion du contrôle d'accès et de l'évaluation de requêtes. Finalement, les performances globales de la solution proposée sont évaluées sur quatre jeux de données ayant chacune des caractéristiques différentes.

### Plate-forme d'expérimentation

L'architecture cible abstraite présentée dans la Section 2 peut être instanciée de plusieurs manières différentes. Dans ces expérimentations, nous considérons que le SOE est intégré dans une plate-forme avancée de carte à puce. Bien que les cartes à puce existantes soient déjà puissantes (processeur 32 bits cadencé à 30Mhz, 4 KO de RAM, 128KO d'EEPROM), elles sont trop limitées pour supporter notre architecture, plus particulièrement en terme de bande passante de communication (9.6Kbps). Notre partenaire industriel Axalto (la filière carte à puce de Schlumberger), a annoncé pour la fin de l'année la sortie de cartes à puce équipées d'un processeur 32 bits cadencé à 40Mhz, 8KO de RAM, 1MO de Flash et supportant un protocole USB à 1MBps. Axalto nous a fourni un simulateur permettant de faire des mesures au cycle près pour la carte à puce à venir. Notre prototype a été développé en C et a été évalué en utilisant ce simulateur. La

précision au cycle près garantit une prédiction exacte des performances qui seront obtenues avec la plate-forme matérielle cible.

Comme cette section le montrera, notre solution est principalement limitée par les coûts de déchiffrement et de communication. Les chiffres donnés dans la table 1 nous permettent de faire des projections sur les performances obtenues dans cette section pour des architectures cibles différentes. Les chiffres donnés pour le débit de communication correspondent au cas le plus défavorable où chaque donnée lue par le SOE prend part dans le résultat. Le coût de déchiffrement correspond à celui de l'algorithme 3DES, implanté dans un module matériel de la carte à puce (ligne 1) et mesuré sur un PC cadencé à 1Ghz (ligne 2 et 3).

| Contexte | Communication | Déchiffrement |
|---|---|---|
| Matériel (e.g., cartes à puce futures) | 0.5 MO/s | **0.15 MO/s** |
| Logiciel – Connexion Internet | **0.1 MO/s** | 1.2 MO/s |
| Logiciel – Connexion dans un réseau local | 10 MO/s | **1.2 MO/s** |

**Table 1.** *Coûts de communication et de déchiffrement*

| | **WSU** | **Sigmod** | **Treebank** | **Hôpital** |
|---|---|---|---|---|
| Taille | 1.3 MO | 350KO | 59MO | 3.6 MO |
| Taille du texte | 210KO | 146KO | 33MO | 2,1 MO |
| Profondeur maximale | 4 | 6 | 36 | 8 |
| Profondeur moyenne | 3.1 | 5.1 | 7.8 | 6.8 |
| # tags distincts | 20 | 11 | 250 | 89 |
| # nœuds texte | 48820 | 8383 | 1391845 | 98310 |
| # éléments | 74557 | 11526 | 2437666 | 117795 |

**Table 2.** *Caractéristiques des documents*

Dans cette expérience, nous considérons trois jeux de données réels : *WSU* correspondant à des cours d'université, *Sigmod records* qui contient un index d'articles et *Treebank* qui contient des phrases annotées avec des parties de discours [UWX]. De plus, nous avons généré des documents synthétiques pour l'exemple de l'Hôpital décrit dans la Section 2 (les jeux de données réels sont difficiles à obtenir dans ce domaine), grâce au générateur ToXgene [ToX]. Les caractéristiques principales de ces documents sont résumées dans la Table 2.

### Surcoût de stockage de l'Index de Saut

L'Index de Saut est une compilation de trois techniques pour encoder respectivement les tags, les listes des tags des descendants et les tailles des sous-arbres. Des variantes de l'Index de Saut peuvent être envisagées en combinant ces techniques différemment (e.g., encoder les tags et les tailles du sous-arbres sans encoder les listes des tags des descendants).

Ainsi, pour évaluer le surcoût associé à chacune de ces méta-données, nous comparons les techniques suivantes. NC correspond au document original non compressé. TC correspond à une compression classique de tags et servira de référence. Dans TC, chaque tag est encodé par un nombre exprimé sur $Log_2(\#tags\ distincts)$ bits. Nous désignons par TCS (compression des tag et taille des sous-arbres) la méthode qui stocke la taille des sous-arbres pour permettre de sauter les sous-arbres. La taille du sous-arbre est encodée sur $Log_2(taille\ du\ document\ compressé)$ bits. Dans TCS, le tag fermant est inutile et peut être supprimé. TCSB complémente TCS avec un bitmap des tags des descendants encodé sur *#tag distincts* bits pour chaque élément. Finalement, TCSBR est la variante récursive de TCSB et correspond en fait à l'Index de Saut décrit dans la Section 4. Dans toutes ces méthodes, les méta-données doivent être alignées sur une frontière d'octet. La figure 7 compare ces cinq méthodes sur les jeux de données réels introduits précédemment. Ces jeux de données ayant des caractéristiques différentes, l'axe des ordonnées est exprimé en terme de taux *structure/(taille du texte)*.

Clairement, TC réduit de manière importante la taille de la structure de tous les jeux de données. En ajoutant la taille des sous-arbres aux noeud (TCS), la taille de la structure passe de 50% à 150% (les gros documents nécessitent un encodage sur 5 octets pour la taille des sous-arbres et le tag de l'élément alors que de plus petits documents nécessitent seulement 3 octets). Le bitmap des tags des descendants (TCSB) a un coût encore plus élevé, plus particulièrement dans le cas du document Treebank qui contient 250 tags distincts. TCSBR réduit de manière importante le surcoût de stockage en le ramenant à une valeur proche de celle obtenue avec TC. Cela s'explique par le fait que la taille de la structure diminue rapidement, tout comme le nombre de tags distincts dans chaque sous-arbre. Pour le document Sigmod, TCSBR offre une meilleure compacité que TC.

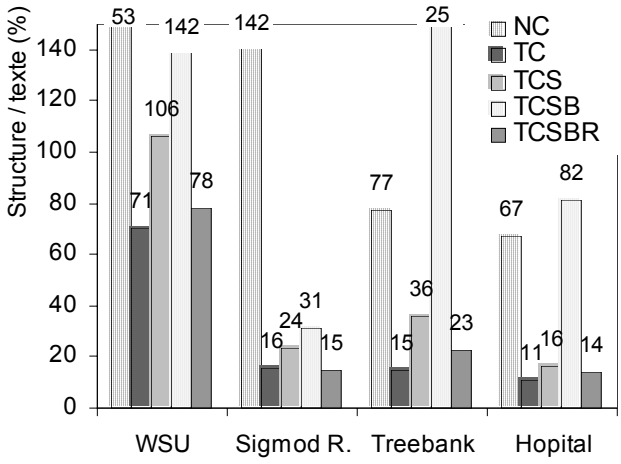


**Figure 8.** *Surcoût du stockage de l'index*

**Surcoût du contrôle d'accès**

Pour évaluer l'efficacité de notre stratégie (basé sur TCSBR), nous le comparons avec : (i) une stratégie force brute (BF) qui filtre le document sans utilisation d'aucun index et (ii) une borne inférieure (LWB). Les performances de LWB ne peuvent être atteintes par aucune stratégie et elles correspondent au temps que mettrait un oracle pour ne lire que les fragments autorisés du document et les déchiffrer. Evidemment, cet oracle doit pouvoir prédire l'issue de tous les prédicats – en attente ou non – sans avoir à les vérifier et deviner où sont les données pertinentes dans le document.

La Figure 9 montre le temps d'exécution requis pour évaluer la vue autorisée des trois profils (Secrétaire, Docteur et Chercheur) introduits dans la Section 2 sur le document Hôpital. La vérification de l'intégrité n'est pas prise en compte ici. La taille du document compressé est de 2,5MO et l'évaluation de la vue autorisée retourne 135KO pour la Secrétaire, 575KO pour le Docteur et 95KO pour le Chercheur. Pour pouvoir comparer ces trois profils malgré la différence de taille des résultats, l'axe des ordonnées représente le rapport entre le temps d'exécution et sa borne inférieure LWB respective. Le temps d'exécution réel est reporté dans chacun des histogrammes. Pour mesurer l'impact d'une politique de contrôle d'accès relativement complexe, nous considérons que le Chercheur est autorisé à accéder à 10 protocoles médicaux au lieu d'un, chacun d'eux étant exprimé par une règle positive et une règle négative comme décrit dans la Section 2.

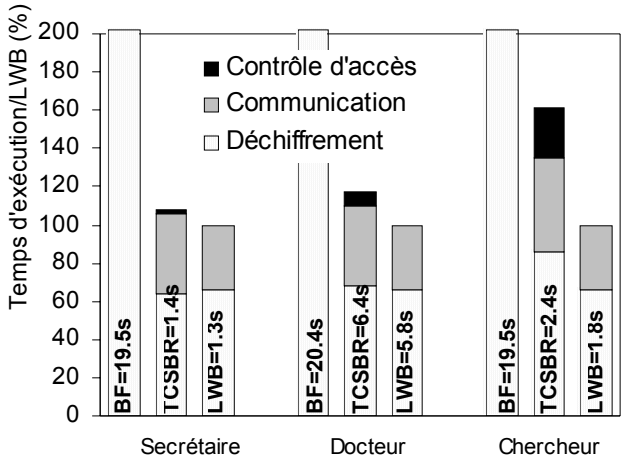



**Figure 9.** *Surcoût du contrôle d'accès*

Les conclusions qui peuvent être tirées de cette figure sont triples. Premièrement la stratégie force brute (BF) donne des performances désastreuses, ce qui s'explique par le fait que la carte doit lire et déchiffrer tout le document pour pouvoir l'analyser. Deuxièmement, la performance de notre stratégie TCSBR est en général très proche de LWB (rappelons-nous que LWB est une borne inférieure théorique et non-atteignable), ce qui met en avant l'importance de minimiser le flux entrant dans le SOE. La différence la plus importante entre ces deux stratégies est observée pour le profil du chercheur à cause des prédicats exprimés sur l'élément protocol qui peut rester en attente jusqu'à la fin de chaque élément folder. En effet, si ce prédicat est évalué à faux, l'évaluateur continuera – sans nécessité dans notre cas – à chercher une autre instance de ce prédicat. Troisièmement, le coût du contrôle d'accès (de 2% à 15%) est largement dominé par le coût de déchiffrement (de 53% à 60%) et par le coût de communication (de 30% à 38%). Le coût du contrôle d'accès est déterminé par le nombre de jetons actifs qui doivent être gérés en même temps. Ce nombre dépend du nombre d'ARA dans la politique de contrôle d'accès, du nombre de transition de descendance (//) et des prédicats présents dans chaque ARA. Cela explique un coût plus important pour la politique de contrôle d'accès du chercheur.

**Impact des requêtes sur les performances**

Pour mesurer de manière précise l'impact des requêtes sur les performances globales, nous considérons la requête //Folder[//Age>v] (où v nous permet de faire varier la sélectivité de la requête) qui est exécutée sur cinq vues différentes construites à partir des profils correspondant à : la secrétaire (S), un docteur à temps partiel (PTD) qui a à sa charge quelques patients, un docteur à temps plein (FTD) qui a à sa charge un grand nombre de patients, un chercheur junior (JR) ayant accès à quelques résultats d'analyses et un chercheur senior (SR) qui aura accès à un grand nombre de résultats d'analyse.

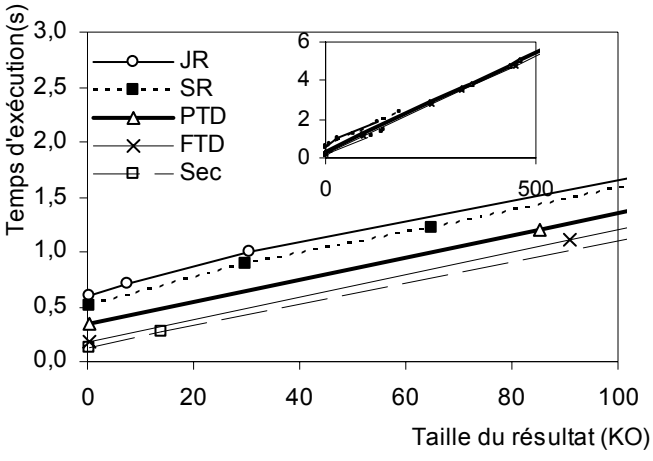



**Figure 10.** *Impact des requêtes*

La Figure 10 représente le temps d'exécution de la requête (incluant celui du contrôle d'accès) comme une fonction de la taille du résultat de la requête. Le temps d'exécution diminue linéairement plus la sélectivité de la requête et de la vue augmentent, ce qui montre la précision de TCSBR. Même si le résultat de la requête est vide, le temps d'exécution n'est pas nul puisque des parties du document doivent être analysées avant d'être sautées. Les parties du document qui nécessitent d'être analysées dépendent de la vue et de la requête. La figure en vignette montre la même linéarité pour des tailles plus importantes du résultat de la requête.

**Performance sur des jeux de données réels**

Pour évaluer la robustesse de notre approche et montrer qu'elle supporte des structures de documents très différentes, nous mesurons les performances de notre prototype sur trois jeux de données réels WSU, Sigmod et Treebank. Pour ces documents nous générons des règles d'accès aléatoires (incluant des // et des prédicats). Chaque document possède des caractéristiques différentes intéressantes. Le document Sigmod est bien structuré, non récursif, de profondeur moyenne et la politique de contrôle d'accès générée est simple et peu sélective (50% du document est renvoyé). Le document WSU est peu profond et contient une grande quantité de très petits éléments (sa structure représente 78% de la taille du document après indexation par TCSBR). Le document Treebank est de très grande taille, contient un nombre important de tags qui apparaissent récursivement dans le document et la politique de contrôle d'accès générée est complexe (8 règles). La Figure 11 reporte ces résultats. Nous avons ajouté dans la figure les mesures obtenues avec le document Hôpital comme base de comparaison. La figure représente les temps d'exécution en terme de débit pour notre méthode et pour la borne inférieure LWB, chacune d'elle avec et sans vérification de l'intégrité. Bien que la vérification de l'intégrité ne soit pas traitée dans cet article (voir [BDP04] pour plus de détails), prendre son coût en compte est nécessaire pour évaluer complètement notre solution.

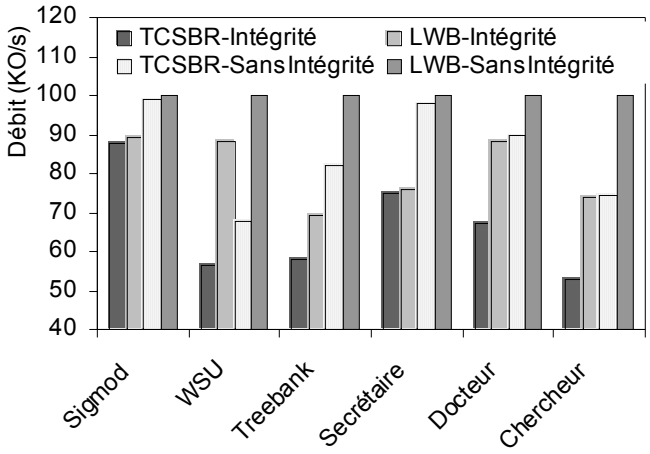



**Figure 11.** *Performance sur des jeux de données réels*

Nous montrons que notre méthode gère bien les différentes situations et produit un débit allant de 55KO/s à 85KO/s dépendant du document et de la politique de contrôle d'accès considérés. Ces résultats préliminaires sont encourageants comparés aux bandes passantes Internet obtenues actuellement avec de l'xDSL (de 16KO/s à 128 KO/s).

## 6. Conclusion

Plusieurs facteurs importants motivent aujourd'hui la délégation du contrôle d'accès aux terminaux clients. En compilant les politiques de contrôle d'accès dans le schéma de chiffrement, les solutions existantes de gestion du contrôle d'accès sur le client minimisent la confiance requise sur le client au prix d'un partage statique des données. Notre objectif est de tirer profit de nouveaux éléments de confiance présents sur les terminaux clients pour proposer une gestion du contrôle d'accès capable d'évaluer des règles d'accès dynamiques sur un document XML chiffré.

La contribution de cet article est double. Premièrement, nous avons proposé une évaluation en flux de règles de contrôle d'accès supportant un sous-ensemble conséquent du langage XPath. A notre connaissance, il s'agit de la première étude proposant une gestion du contrôle d'accès XML en flux. Deuxièmement, nous avons conçu une structure d'index en flux permettant de sauter les parties non pertinentes du document d'entrée, en fonction de la politique de contrôle d'accès et d'une requête éventuelle. Cet index est essentiel pour réduire le goulot d'étranglement inhérent à l'architecture cible, à savoir le coût de déchiffrement et de communication. Utilisés conjointement, ces deux mécanismes forment le noyau de

notre solution. La gestion des prédicats en attente et la vérification de l'intégrité sur des fragments aléatoires complémentent cette solution [BDP04].

Nos résultats expérimentaux ont été obtenus à partir d'un prototype écrit en C et tournant sur un simulateur matériel de carte à puce fourni par Axalto permettant de faire des mesures au cycle CPU près. Le débit global observé est d'environ 70KO/s et le coût relatif moyen du contrôle d'accès est inférieur à 20% du coût total. Ces premières mesures sont prometteuses et démontrent la viabilité de notre solution. Un prototype Javacard a été développé et a obtenu la médaille d'argent du concours logiciel e-gate Open 2004 organisé par SUN, Axalto et ST Microelectronics, montrant ainsi l'intérêt du monde industriel pour les solutions basées sur un élément sécurisé.

Les problèmes ouverts concernent une meilleure utilisation des techniques d'inclusion de requêtes afin de raffiner les optimisations avant et pendant l'évaluation des règles d'accès ainsi que la définition de techniques plus précises d'indexation en flux. De manière plus générale, les solutions de sécurité basées sur le client méritent une attention toute particulière du fait des nouvelles perspectives de recherche qu'elles offrent et de leur impact prévisible sur un nombre croissant d'applications.

Remerciements

Nous tenons à remercier particulièrement Anaenza Maresca, médecin à l'hôpital Tenon (Paris), pour sa contribution à la définition de l'exemple médical.

## 7. Bibliographie

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

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

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

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

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

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

[BDP04] Bouganim L., Dang Ngoc F., Pucheral P., *Client-Based Access Control Management for XML Documents*, INRIA internal report RR-582, June 2004. http://www.inria.fr/rrrt/rr-5282.html

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

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

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

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

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

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

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

[DF03] Diao Y., Franklin M., *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] Gabillon A., Bruno E., *Regulating access to XML documents*. IFIP Working Conference on Database and Application Security, 2001.

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

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

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

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

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

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

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

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

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

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

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

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

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

[RRN02] Ray I., Ray I., Narasimhamurthi N., *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] Schneier B., *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] Tolani P., Haritsa J., *XGRIND: A Query-Friendly XML Compressor*, ICDE, 2002.

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

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

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