The Case for Access Control on XML Relationships

Béatrice Finance, Saïda Medjdoub, Philippe Pucheral

The Case for Access Control on XML Relationships

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

Saïda Medjdoub
&

Philippe Pucheral
**INRIA Rocquencourt
78153 Le Chesnay -France
<Firstname.Lastname@inria.fr>

Abstract

With the emergence of XML as the de facto standard to exchange and disseminate information, the problem of regulating access to XML documents has attracted a considerable attention in recent years. Existing models attach authorizations to nodes of an XML document but disregard relationships between them. However, ancestor and sibling relationships may reveal information as sensitive as the one carried out by the nodes themselves (e.g., classification). This paper advocates the integration of relationships as first class citizen in the access control models for XML and makes the following contributions. First, it characterizes important relationship authorizations and identifies the mechanisms required to translate them accurately in an authorized view of a source document. Second, it introduces a rule-based formulation for expressing these classes of relationship authorizations and defines an associated conflict resolution strategy. Third, it proposes tractable algorithms to support relationship authorizations. Rather than being yet-another XML access control model, the proposed approach allows a seamless integration of relationship authorizations in existing XML access control model.

Key words: Data confidentiality and privacy, XML access control, XML relationship, need-to-know.

1. Introduction

XML has become the de facto standard to describe, exchange and disseminate any kind of information among various partners and for various purposes. Meanwhile, safeguarding data confidentiality, privacy and intellectual property has become a primary concern for citizens, administrations and companies. This motivated several recent works on XML access control, tackling different facets of the problem. Discretionary [3,10,16,23], Role-Based [21,29,37] and Mandatory [9] access control models have been proposed in the context of XML. A particular attention has been paid on the granularity of the access control (from DTD to attribute instances) [3,10,16], on the performance of the algorithms implementing this control [9,26,38], on the distribution channel used to expose the information (pull, push, selective dissemination) [4,6,27,39] and on the tamper-resistance of the access control [7,25,34].

All these works have the commonality to focus the access control on the nodes of an XML document (elements and attributes). Ancestor and sibling relationships among nodes are not considered as legitimate targets of the access control. Roughly speaking, an access control policy is composed of a set of positive (resp. negative) authorization rules granting (resp. denying) a given subject access to some nodes of the document. These nodes are usually selected thanks to XPath expressions. The descendant relationship among nodes is simply exploited as a mean to propagate authorization rules down through the XML hierarchy. There are substantial differences among the models in the way conflicts among – potentially propagated – positive and negative rules are tackled. In [3,16], the complete subtree rooted at a forbidden node is forbidden. This constraint is relaxed in [10], allowing exceptions to a negative rule to be expressed. However, this leads to make visible the label (i.e., tag) of forbidden ancestor(s) in the path from the root to an authorized node. Replacing the node label by a dummy value has been proposed in [13, 17] to reduce information disclosure in such situation. These discrepancies among the models brings to light the difficulty to define accurately the view that should be delivered of the path leading to an authorized node.

More precisely, disregarding XML relationships in the expression of the access control leads to two important problems.

– *Classification disclosure*: the structure of an XML document often reveals a classification[1] (e.g., subtrees organized according to the medical services where patients are treated, activities or sales areas of companies, socio-economic categories of citizens or profiles of customers). Therefore, the membership of an authorized node to a given subtree conveys its classification. Whatever be the information hidden in the root node of that class to protect this sensitive information, it can be often inferred by simple statistical attacks (the cardinality of a class frequently reveals this class). In addition, disclosing the class membership for a single element discloses the membership for all, making any obfuscation mechanism preserving the class decomposition non-robust.
– *Uniform filiation:* the authorization rules expressed on ancestor nodes determines a common authorized view of the path leading to all their descendants. In other words, there is no way to deliver two different authorized views of the same ancestor for two of its descendants (e.g. one patient is willing to hide the medical service she is treated in, while another consents to disclose this information).

These two problems hurt the basic *need-to-know* and *consent* principles enacted in most directives and laws related to the safeguard of personal information, like the Federal Privacy Act in the US [30] and the Data Protection Directive in the EU [12]. The need-to-know principle limits access to information to those people who need strictly this information to carry out their duties. Clearly, classification disclosure hurts this principle each time the information contained in a given subtree (e.g., a personal folder) is self-content wrt a given purpose. The consent principle prohibits the disclosure of personal information without the explicit consent of the donor in a number of situations defined by the law. In our context, this means that the donor must be given some prerogative to control how her information (e.g., her medical folder) is exposed and made accessible in an XML document. This requirement contradicts a uniform filiation.

More than ever, there is a strong need to define access control models that help translating more accurately law principles into practice. To make a step forward in this direction, this paper advocates the integration of ancestor and sibling relationships as first class citizen in the access control models for XML. As mentioned above, relationships between nodes may reveal information as sensitive as the one carried out by the nodes themselves and hence, deserve to be protected as such. The objective is to provide means to control accurately the view that must be delivered of the path leading to any authorized node (i.e., subtree) in an XML document. More precisely, this paper makes the following contributions:

1. *Characterization of relationship authorizations*
   The first problem is to characterize which relationships need to be protected and to define the means by which they can be protected. We identify the relationship authorizations required to deal with the classification disclosure and uniform filiation problems. Then, we exhibit the mechanisms necessary to translate them accurately into the authorized view of a source document.
2. *Relationship-aware access control model*
   The second problem is to define a simple but comprehensive access control model encompassing nodes and relationships authorizations. We propose a rule-based formulation for expressing relationship authorizations and we define a conflict resolution strategy to manage the conflicts that may occur among them. Rather than being yet-another XML access control model, our approach allows a seamless integration of relationship authorizations in existing XML access control models.
3. *Relationship-aware access control algorithms*
   The last problem is to assess whether algorithms implementing a relationship-aware access control model can meet reasonable complexity and performance. We introduce the basic principles of these algorithms and discuss performance issues based on preliminary performance measurements.

The paper is organized as follows. Section 2 introduces a case study motivating the integration of XML relationships in the expression of access control policies. Section 3 characterizes the relationship authorizations required to deal with the classification disclosure and uniform filiation problems. Section 4 introduces a rule-based formulation for expressing relationship authorizations and discusses the upward compatibility of our approach with existing XML access control models. Section 5 focuses on implementation and performance issues. Section 6

[1] Classification is used here in its usual meaning (i.e., synonym of categorization).

presents related works. Finally, section 7 concludes and sketches important open issues.

2. Motivating example

We built our motivating example from requirements expressed by a real life medical application related to the treatment of AIDS disease. Figure 1 depicts the way the medical information of interest is structured. Organizing a safe sharing of medical folders among several parties (patients, physicians, pharmacists, medical labs, Medicare and insurance companies) having different duties and objectives is a rather challenging task. Specific laws are regulating access to medical records, like the well recognized Health Insurance Portability and Accountability Act (HIPAA) in the US[36]. However, the failure in translating law statements into convincing technology tools strongly curbs the deployment of Electronic Health Record (EHR) systems all around the world. An analysis of the benefits and shortcomings of HIPAA illustrates this situation quite well [31]. Tackling this issue is a tremendous challenge for the database community, considering the expected benefits of EHR systems in terms of quality of care and financial saving.

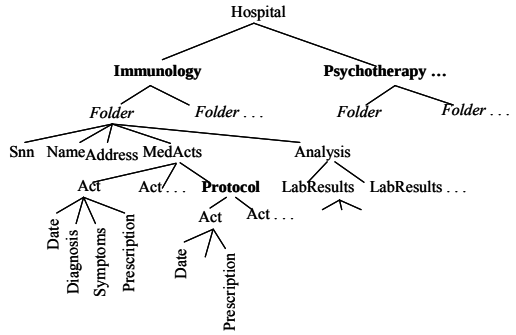

Figure 1: Medical Folders

Below are three examples of simple but important authorization rules that cannot be managed accurately with existing XML access control models.

- ***R1****: Hide to the hospital's directory application the name of the service where patients are treated, for those who didn't consent making this information public.*

As stated in HIPAA, the hospital directory is a rather sensitive information considering the inquiries made about patients by relatives, employers, media, police and members of religious groups. The effect of this authorization rule on the document pictured in Figure 1 should be to attach the `Folder` element of each patient of interest to a depersonalized medical service element (i.e., element with an anonymous label) while keeping the ancestor chain of the other folders unaffected. As pictured in Figure 2.a, this restructuration must be done in a way that prevents the inference of the initial classification.

XML access control models like [3,16] give the ability to hide a folder in the document. This does not match the objective since the presence of the patient in the hospital turns to be hidden as well, which is not the purpose of Rule R1. The model proposed in [10] allows defining a negative rule on a medical service and a positive one on folder. Unfortunately, the label of the medical service (the information to be protected in our context) will be disclosed. This problem is tackled in [13, 17], by replacing this label by a dummy value. However, whatever be the model, classification disclosure is not precluded and above all, the authorization rule applies uniformly to all folders, neglecting the patient's consent.

- ***R2****: Hide to pharmacists the fact that some drug prescriptions participate in a protocol (i.e., medical trial).*

The pharmacist must be aware of all prescriptions to check drug incompatibilities. However, giving him the knowledge that some drugs participate in a protocol discloses unnecessary information on the patient's disease and its stage. The expected effect of this authorization rule is to drop `Protocol` elements and attach `Act` elements as direct children of their `MedActs` ancestor, giving them a position similar to regular `Act` elements. Depersonalizing `Protocol` is useless in this case since that information would be obvious to infer (`Act` elements children of `MedActs` and those children of `Protocol` form two classes with their ancestor as a distinguishing factor).

Again, existing access control models give the ability either to hide the complete subtree rooted at a `Protocol` element or to depersonalize `Protocol` elements[2]. Both solutions hurt the need-to-know principle by disclosing too little or too much information.

- ***R3****: Hide to medical labs the correlation between medical and administrative information inside each folder.*

HIPAA stipulates that the patient consent is required for any disclosure related to marketing. In the following we assume that the medical information (`MedActs, Analysis`) is required

[2] As discussed in the related works, [13] offers a third alternative to this issue, under specific constraints on the document DTD.

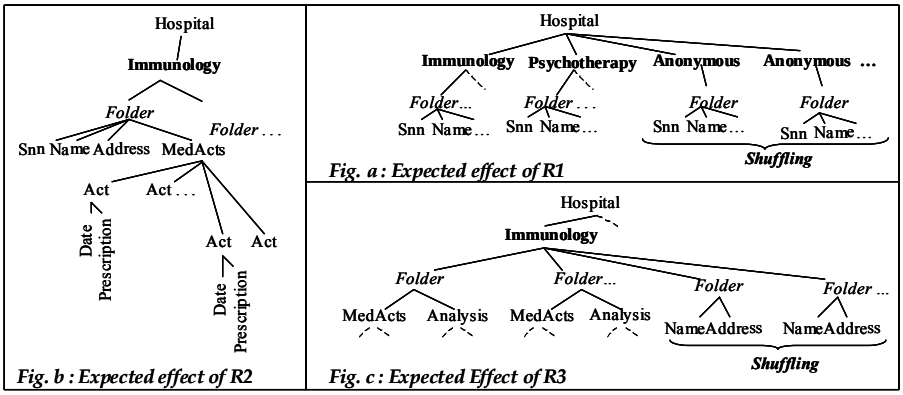

Figure 2: Authorized views.

wrt the need-to-know principle while the administrative information (`Name, Address`) is collected under the patient consent for marketing purpose (e.g., related to new medications). The expected effect of this authorization rule is to make both groups of information available while precluding the inference of their initial sibling relationship.

Using existing access control models would impose to define two separate access control policies for the same document and the same user (i.e., the medical lab). However, the conjunction of both authorized views must be precluded and the risk of inference mentioned above must be carefully tackled[3], a rather challenging issue [19].

These three situations demonstrate the limits of existing access control models and advocate the integration of explicit authorizations on ancestor and sibling relationships in the expression of access control policies. On the other hand, authorizations on nodes are well suited to define the fraction of each folder that must be disclosed in each situation. The three expected authorized views of the initial document, derived by a combination of nodes and relationship authorizations, are pictured in Figure 2.

Finally, one may wonder about the number of authorization rules that should be defined to capture the personal consent of each person. While consent is a personal matter, it is worth-noting that its dimensions (as enacted by the law) are quite reduced. Therefore, consent can be integrated in each folder by dedicated XML elements, allowing capturing the common consent of several people in a single set-oriented authorization rule. This point will be illustrated in Section 4.

[3] Note that a simple inference attack could be conducted by comparing the element ordering in both views.

3. Characterization of relationship authorizations

Existing XML access control models interpret an access control policy as a mapping between a source document (or `Source` for short) and an authorized view of this same document (or `View`) and rely on the assumption that `View` $\subseteq$ `Source`. More precisely, authorization rules select the subset of `Source` nodes that will participate in `View`. As a side effect, edges having one of their extremity node discarded by an authorization rule are in turn discarded from `View`. As our motivating example makes clear, considering relationship authorizations compels us to revisit this assumption since `View` may result from a more complex restructuration of `Source`. Typically, new paths and nodes may appear in `View` and the node ordering may be different from the `Source` one to prevent inference. In other words, `View` represents the total amount of information a grantee can gather by subsequent queries. In this section, we concentrate on the semantics of relationship authorizations and define for each of them its impact on the resulting `View`. We do not make any assumption on how `View` is actually defined (XQuery statement vs. system of rules) nor built (materialization vs. streaming), these two issues being discussed respectively in Section 4 and Section 5.

3.1 Notations

Let us now introduce the model of XML document we consider and the associated notations that will be used along the paper. An XML document d is defined by a tuple ($Label_d$, $Value_d$, N_d, r_d, E_d, ϕ_{label}, λ_{value}, φ_{order}), where:

– $Label_d$: is a set of element labels (also called tags) and attribute names of type string.

- $Value_d$: is a set of attribute/element values of type string.
- $N_d = N^e_d \cup N^a_d$ is a set of nodes representing elements and attributes, respectively. Each $n \in N^e_d$ has a required *element label* $\in Label_d$ and an optional *element value* $\in Value_d$, whereas each $n \in N^a_d$ has a required *attribute label* $\in Label_d$ and a required *attribute value* $\in Value_d$.
- r_d : is a particular node representing the document root.
- E_d : $E_d \subset (N_d \cup r_d) \times N_d$ is a set of edges, where each $e \in E_d$ represents an element-subelement or element-attribute relationship.
- ϕ_{label} : $N_d \rightarrow Label_d$ is the node labeling function.
- λ_{value} : $N_d \rightarrow Value_d$ is the node valuation function.
- φ_{order} : $N_d \rightarrow Integer$ is the node ordering function, reflecting a preorder traversal of the tree.

In addition, *Anc(n)*, *Child(n), Desc(n)* and *Sibling(n)* denote respectively the set of ancestors, childs, descendants and siblings of a given node $n \in N_d$ and *Parent(n)* denotes its parent node. *Path(n1,n2)* denotes a path from node *n1* to node *n2*. According to this model, an XML document is modeled as a labeled graph where nodes represent elements and attributes, and edges relationships between them. If a node does not have any parent, it is implicitly linked with the document root.

3.2 Cloning and Shuffling mechanisms

Taking into account user's consent in access control models imposes to generate in `View` different replicas of the same `Source` nodes and paths. Basically, replicating a `Source` node *n1* is required each time two of its authorized descendants *n2* and *n3* must be reachable in `View` by a path delivering two conflicting visions of *n* to conform to the semantics of a given authorization rule. Rule R1 of our motivating example illustrates this point. Since an XML document is a tree, every node participating in the common subpath Path(n1,Parent(n2))∩Path(n1,Parent (n3)) has in turn to be replicated. *Cloning* is the principle by which `Source` elements and paths are replicated in `View`. Note that leaf nodes of a source document (terminal elements and attributes) are never subject to cloning. In the following, we use the term *original* and *clone(s)* to distinguish in `View` between the genuine image of a `Source` element or path, and the element(s) or path(s) resulting from a cloning operation.

Element Cloning

We denote by $\tilde{n}_i \in N^e_{View}$ the i^{th} clone of node $n \in N^e_{Source}$. Subscripts are used when the different clones of a same node have to be distinguished and are omitted otherwise. The label, value and order of a clone are defined as follows:

$\phi_{label}(\tilde{n}) \in \{\phi_{label}(n), \text{“anonymous”}\}$[4],
$\lambda_{value}(\tilde{n}) = \Delta$, where Δ denotes the empty string,
$\varphi_{order}(\tilde{n}) = Shuffle(Sibling(\tilde{n}))$.

Where *Shuffle* defines a random order among the clones being sibling of a same node. The necessity of shuffling is explained afterward.

Path Cloning

Let u be a path (n1,n2…nk) / n1,n2…nk $\in N^e_{Source}$ and (n1,n2)…(nk-1,nk) $\in E_{Source}$, $\tilde{u}$ denotes the clone of u and is defined by ($\tilde{n}$1,$\tilde{n}$2…$\tilde{n}$k) / $\tilde{n}$1,$\tilde{n}$2…$\tilde{n}$k $\in N^e_{View}$ and ($\tilde{n}$1,$\tilde{n}$2)…($\tilde{n}$k-1,$\tilde{n}$k) $\in E_{View}$.

The ordering of clones in `View` has to be carefully managed to avoid basic inference. To illustrate this, let us consider Rule R3 of our motivating example and assume that the `View` ordering is such that all instances of the two groups (`MedActs`, `Analysis`) and (`Name`, `Address`) keep the same relative order as in `Source`. In this case, their initial sibling relationship, which should be obfuscated by the cloning mechanism, is patently disclosed by the element ordering (i.e., the i^{th} instance of `MedActs`, `Analysis`) corresponds to the i^{th} instance of (`Name`, `Address`)). A similar problem exists with Rule R1 if the clones of a medical service element are placed in close proximity to their original (e.g., direct right or left sibling). Thus, cloning does not make sense without *node shuffling* (see Figure 2.a and 2.c).

Node shuffling

Node shuffling is a recursive process that applies at each node of `View` containing clone children. All clones, children of a given node, are shuffled together to prevent ordering-based inference. For a given node, the clone children are grouped after the original ones (by convention), and then shuffled. The relative order of the original children must however be preserved in `View` since node ordering is significant in XML. The ordering function φ_{order} must therefore satisfy the two following properties:

[4] By default, a clone inherits the label of its source counterpart, except if the anonymous value is explicitely selected.

- $\forall ni,nj \in N_{Source}$, $\forall ni',nj' \in N_{View}$ / ni=ni' and nj=nj', $\varphi_{order}(ni) < \varphi_{order}(nj) \Rightarrow \varphi_{order}(ni') < \varphi_{order}(nj')$
- $\forall \tilde{n} \in N_{View}, \varphi_{order}(\tilde{n})$=random(]max($\varphi_{order}$(Parent($\tilde{n}$)), φ_{order}(iops($\tilde{n}$))), φ_{order}(ifol(Parent($\tilde{n}$)))[),

where iops and ifol denote respectively the immediate original preceding sibling and immediate following of a given node, where preceding sibling and following conform to the XPath semantics.

3.3 Authorizations on ancestor relationships

Relationship authorizations are introduced to tackle the two problems identified in the introduction and exemplified in the motivating example, namely classification disclosure and uniform filiation.

Let us first consider classification disclosure. The objective is to mask the membership of a given descendant node *n* to a given class rooted at an ancestor node *a*[5]. In this respect, two situations have to be carefully distinguished. The information to be protected can be either: (1) to which class the node *n* belongs to, or (2) the fact that *n* is actually classified.

In the former case, the information to be hidden is the identification of the class *n* belongs to. This information is carried out by the ancestor node *a*, either by its label, by one of its attribute or by one of its sub-element. Cloning the ancestor node *a* and the path leading to *n* is a mean to hide this information. Indeed, attributes and sub-elements do not participate in the cloning operation and *a*'s label can be made anonymous if it actually conveys the class identification. This leads to a first class of authorization called *ancestor depersonalization*.

In the latter case, the information to be hidden is the presence of *a* itself, in the path leading to *n*. This situation occurs each time a class membership reveals an exception to a general situation (e.g., Rule R2 of our motivating example). Cloning the path from the ancestor node *a* to *n* allows discarding *a* from that path. This leads to a second class of authorization called *path reduction*.

Regarding now the uniform filiation problem, the objective is to allow descendant nodes expressing potentially different requirements in terms of classification disclosure. Hence, both problems have to be considered jointly. Cloning provides a uniform solution to them. Indeed, cloning the path linking an ancestor to one of its descendant gives the opportunity to personalize the authorized view of each ancestor relationship.

[5] In the sequel, class membership and subtree membership will be considered as synonym.

The semantics of ancestor depersonalization and of path reduction authorizations is defined as follows in terms of their expected effects on `View`. Let $n \in N_{Source}$. Let a, $u \in N^e_{Source}$ / $a \in Anc(n)$ and u = Parent(n).

Ancestor depersonalization

- If a=u then Element_Cloning(a) else Path_Cloning(a,u);
- Parent($\tilde{a}$)←Parent(a);
- Parent(n) ← $\tilde{u}$; // if a=u then $\tilde{a} = \tilde{u}$

Path reduction

- If a=u then Parent(n) ← Parent(a) else {
  Let $e \in N^e_{Source}$ / e = Child(a) ∩ Anc(n)
- if e=u then Element_Cloning(e) else Path_Cloning(e,u);
  - Parent($\tilde{e}$)←Parent(a);
  - Parent(n) ← $\tilde{u}$; // if e=u then $\tilde{e} = \tilde{u}$ }

Notice that *n* is always attached along with its potential subtree to the extremity of the cloned path. At this point, it is interesting to note that node authorizations in existing models follow a top-down approach in the sense that node authorizations usually propagate down through the XML hierarchy. Conversely, ancestor depersonalization and path reduction authorizations follow a bottom-up approach by impacting the relationship between descendant nodes and their respective ancestors.

3.4 Effect on sibling relationships

While masking the membership of a given node to a given class, care should be taken about the correlation between that node and its initial siblings. The semantics introduced in the preceding section for relationship authorizations implicitly break down sibling relationships. This side effect may hurt the need-to-know principle in a number of situations. Conversely, implicitly preserving all sibling relationships while disconnecting a node from a class resurrects the uniform filiation problem and hence may hurt the consent principle. Therefore, there is a need for a more selective sibling decorrelation to cope with situations where a node has to retain its relationships with a group of siblings while being disconnected from its initial class.

Sibling Selection on Ancestor Relationships

Let $n \in N_{Source}$ and *n'* its image in N_{View}. Let g1, g2 $\subset N_{Source}$ / $\forall$e1, e2 $\in$ g1$\cup$g2, Parent(e1) = Parent(e2) = Parent(n). Keeping the sibling relationship between *g1*'s elements and *n* has the following impact on the document `View`:

- $\forall$e1 $\in$ g1, Parent(e1') ← Parent (n') : attach *g1*'s elements and *n* to the same parent in `View`,

while g2's elements remain attached to the original parent in `View`.

As a conclusion, dealing with classification disclosure and uniform filiation introduces a two-dimensional problem that must be tackled on a node basis: (1) how the relationships between a given node and its ancestors must be mapped in `View`; (2) which sibling relationships have to be preserved for that node in `View`. How relationship authorizations are actually declared, how they interact with node authorizations and how conflicts among rules are managed is the topic of the next section.

4. Relationship-aware access control model

4.1 Preliminaries

An authorized view can be seen as the result of a global selection and restructuration process applied to the source document. At first glance, a general-purpose language like XQuery could be considered as the appropriate mean to define this process. This solution is actually possible since XQuery is a Turing complete language. However, it has been shown that even simple XML document restructurations, such as deleting a subtree or an element, can result in rather complex XQuery expressions [8]. Such expressions resemble more iterative programs than declarative statements. Implementing an access control policy by an XQuery program would strongly entail its readability and manageability, and its soundness will be difficult to assess.

By contrast, an access control model is expected to exhibit the following properties: (i) *expressiveness* (a robust subset of practical situations must be tackled), (ii) *conciseness* (a given policy must be expressed with as few rules as possible), (iii) *soundness* (the resulting `View` must translate accurately the semantics of a given policy) and (iv) *manageability* (a policy must be human understandable and its evolution must be easy).

To meet these requirements, existing XML access control models (some of them being well recognized today [3,10]) express a policy as a set of authorization rules. A rule engine is responsible for implementing the conflict resolution among rules and for translating `Source` into `View` according to these rules. Due to its simplicity and expressiveness, XPath is usually the language elected to identify the target nodes of each rule. We will follow in existing models' footstep to express relationship authorizations.

In this section, we first introduce a reference model for expressing node authorizations that captures the common foundation of existing XML access control models. Then, we propose an extension to this reference model that supports relationship authorizations while preserving the four properties mentioned above. Thereby, rather than proposing yet-another access control model for XML, we show that the proposed approach allows a seamless integration of relationship authorizations in existing access control models.

4.2 Reference model for node authorizations

While existing XML access control models introduce subtleties on the way node authorizations propagate down through the hierarchy and conflicts are solved, they share strong commonalities. Basically, an authorization rule takes the form of a tuple <*Subject*, *Object*, *Operation*, *Sign*>. Depending on the models, *Subject* can take many forms (a user, a group of users, a role, an application, an IP address, etc). *Object* characterizes the part of the XML document targeted by the rule. *Operation* denotes the operation (read, update, delete, append) the Subject may perform on the Object. Finally, *Sign* denotes either a permission (grant rule) or a prohibition (deny rule) for that operation. In the sequel, we do not make any assumption on the way subjects are managed and, since the focus is on data confidentiality, read is the only operation of interest. As a consequence, we consider a simplified form of node authorization rule. This allows us concentrating on the fundamental issue addressed in this paper, that is to say how the objects to be controlled are characterized.

Node authorization rule

A node authorization rule NA is defined by a tuple: <*Subject*, *Objects*, *Sign*> where:

- *Subject* is an abstract entity,
- $Objects \subseteq N_{Source}$,
- $Sign \in \{+, -\}$.

Objects correspond to attributes and elements of the source 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. For the sake of generality, we consider a well recognized subset of XPath denoted by $XP^{\{[],*,//\}}$[24]. This subset consists of node tests, the child axis (/), the descendant axis (//), wildcards (*) and predicates ([]).

By allowing positive and negative authorizations, *Sign* provides a simple and effective way to specify authorization applicable to sets of objects with support for exception [22]. To match the well accepted *least privilege* principle, we consider a closed policy, meaning that an implicit negative authorization applies to the whole document. In other words, the access to every object that is not explicitly authorized is forbidden.

We assume that both positive and negative authorizations propagate implicitly down through the XML hierarchy. This mode of propagation corresponds to the *cascading* option present in well-known models [4,10,16,26]. Conflicts between direct and/or propagated rules are managed as follows. Let us assume two rules R1 and R2 of opposite sign. These rules may conflict because they are defined either on the same node, or on two different nodes *n1* and *n2*, linked by an ancestor relationship (i.e., $n1 \in Anc(n2)$). In the former situation, the *Denial-Takes-Precedence* policy favors the negative rule according to the least privilege principle. In the latter situation, the *Most-Specific-Object-Takes-Precedence* policy favors the rule that applies directly to a node against the inherited one (i.e., R2 takes precedence over R1 on *n2*). In other words, authorizations propagate until overridden by an opposite authorization on a descendant node.

4.3 Relationship authorization rules

The challenge is to complement this reference model with relationship authorizations without hurting conciseness, soundness and manageability. To this end, we propose a unified statement allowing declaring the relationship authorizations introduced in Section 3.

Relationship authorization rule

A relationship authorization rule RA is defined by a tuple <*Subject*, *Objects*>, where *Objects* is in turn defined by a 4-tuples: <*Anc, Desc, Path-visibility, Sibling*>

- *Anc* and *Desc* characterize the relationship(s) to be protected among a (set of) descendant(s) and one of its (their) ancestor. Anc and Desc are the common denominator of all relationship authorizations. They are both defined as XP{[],*,//} path expressions.

- *Path-visibility* characterizes the vision of the path *u* linking each descendant node to its ancestor. For each node *n* participating in *u*, Path-visibility states whether the node is preserved or not in the path clone $\tilde{u}$ and, in the positive case, whether *n*'s label is preserved or not in $\tilde{n}$.
- Implicitly, hiding an ancestor relationship hides the relationship between a descendant node and its siblings. To allow for a selective sibling decorrelation, *Sibling* characterizes the list of siblings a descendant must keep its relationships with.

The RA definition deserves two important remarks. First, regarding conciseness and manageability, RA captures gracefully and in a rather simple way the different forms of relationship authorizations. By defining Anc and Desc as XPath expressions, it allows to sump up $|Desc|$ ancestor relationships in a single statement. Second, unlike NA, RA does not integrate a Sign parameter. The reason for this is that RA characterizes only negative authorizations. The global semantics of the model is as follows. NA rules are defined according to a closed policy and deliver an authorized view $\texttt{View}' \subseteq \texttt{Source}$ in the usual way (i.e., edges having one of their extremity node discarded by a NA rule are in turn discarded from $\texttt{View}'$). RA rules are defined on $\texttt{View}'$ according to an open policy and deliver the final authorized view $\texttt{View}$. Consequently, if no RA rule is defined, the semantics of the model complies with the one of the existing XML access control models. Hence, a seamless integration of relationship authorizations in these models can be reached.

Ancestors and Descendants

For a RA rule to be consistent, condition $Desc \subseteq Anc$ must be enforced, where $\subseteq$ denotes the containment relation between XPath expressions. Unfortunately, the containment problem has been shown co-NP complete for the class of $XP^{\{[],//,*\}}$ expressions [24]. To avoid consistency checking, Desc is defined as a relative expression with respect to Anc. Thus, Anc determines a set of path origins while Anc/Desc determines a set of path extremities.

Path visibility and Sibling

Table 1 (resp. Table 2) summarizes the possible choices for the Path-visibility (resp. Sibling) parameter along with their associated semantics. The first row of each table gives an extensive syntax for the corresponding parameter while the next rows propose shortcuts to express a monotonic policy along the path. Figure 3 and Figure 4 illustrate the effects of these parameters.

Path-visibility	Semantics of Path visibility
[label$_1$?,.,label$_n$?]	gives the list of nodes to be discarded (?=†) or depersonalized (?=Φ).
[]	all nodes are kept on the path (i.e., all nodes are cloned) and their original label is inherited. This option is the default one.
[Φ]	all nodes are kept on the path and are depersonalized (i.e., the label of their respective clone is set to ”anonymous”).
[†]	All nodes are discarded from the path.

Table 1. Path-visibility semantics

Sibling	Semantics of Sibling
[label$_1$,… label$_n$]	Nodes those label belongs to this list must keep their sibling relationship with the descendant node of interest.
[⊥]	The descendant node is disconnected from all its siblings. This is the default option.
[ψ]	The descendant node preserves its sibling relationships with all siblings targeted by the same authorization rule as him.
[≡]	The descendant node preserves all the sibling relationships it is involved in.

Table 2. Sibling semantics

Figure 5 illustrates the use of a relationship-aware access control model for expressing the access control rules introduced in our motivating example. Each of these rules actually mix NA and RA rules. Some NA and RA rules reference the user's consent. We do the assumption that the user's consent is materialized by a `Consent` element present in each folder. The `Consent` element is in turn composed of subelements (e.g., `directory`, `marketing`) expressing each dimension of this user's consent. For expressing R1, three NA rules are required. NA1, NA2 and NA3 capture the information strictly required by the hospital's directory group to accomplish their duty (typically, `MedActs` and `Analysis` are withdrawn). RA1 depersonalizes ([Φ]) the medical service ancestor (/* targets all medical service elements) of each folder owned by a patient who didn't consent disclosing that information and disconnects that folder from its siblings ([⊥]). For rule R2, RA2 alone expresses a path reduction discarding the parent `Protocol` ([†]) of `Act` elements. For expressing R3, two NA rules deny to the medical lab access to the name and address of patients who didn't consent disclosing this information for marketing purpose. For patients giving their consent, RA3 precludes the inference between the identification information (`Name`, `Address`) and the rest of the folder.

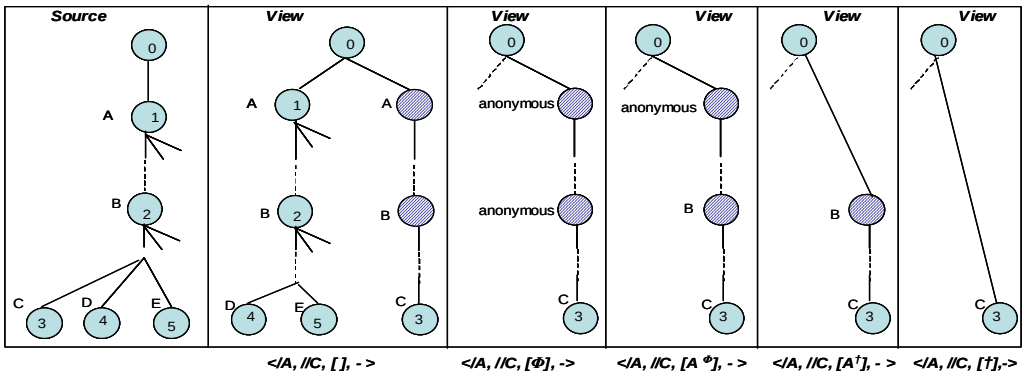

Figure 3: Path-Visibility example.

Figure 4: Selective sibling example.

***Rule R1*:**
NA1: < DirectoryGroup , /Hospital, + >
NA2: < DirectoryGroup, //MedActs, - >
NA3: < DirectoryGroup, //Analysis, - >
RA1: < DirectoryGroup, /*,
/Folder[./Consent/Directory
/Service= 'no visible'], [Φ], [⊥] >

Rule R2:
RA2: < Pharmacist, //MedActs/Protocol, /Act, [†],[⊥]>

Rule R3:
NA5: < Medical lab, //Folder[./Consent/Marketing/
PersonalInfo='no visible']/name, - >
NA6: < Medical lab, //Folder[./Consent/Marketing/
PersonalInfo ='no visible ']/Address, - >
RA3: < Medical lab, //Folder, /Name, [], [Address]>

Figure 5 : Motivating example's NA & RA rules

4.4 Conflict resolution

Three classes of conflicts have to be tackled in a relationship-aware model: conflicts among NA rules, conflict between NA and RA rules and conflicts among RA rules. The resolution of conflicts among NA rules is directly inherited from existing XML access control models and is detailed in Section 4.2. Conflicts between NA and RA rules are avoided by construction since RA rules are defined on the view produced by the evaluation of NA rules, according to an open policy. In other words, NA rules always take precedence on RA rules. This section is thus devoted to the management of the third class of conflicts, namely conflicts among RA rules.

Again, three classes of conflicts among RA rules have to be distinguished: conflicts on Desc (different rules targeting the same descendant node), on Path-visibility (different rules targeting the same ancestor-descendant relationship with different Path-visibility options) and on Sibling (different rules targeting the same ancestor-descendant relationship with different Sibling decorrelations).

Conflicts on Desc

Conflicts on Desc arise when two rules RA1 and RA2 are such that RA1.Anc⊆RA2.Anc and RA1.Desc=RA2.Desc, here ⊆ denotes the inclusion of XPath expressions[6].

[6] As stated in Section 4.3, the containment problem has been shown co-NP complete for the class of $XP^{\{[],//,*\}}$ expressions we are considering [24]. While a static evaluation of the containment property is thus precluded, its dynamic evaluation is rather straighforward. Static simplifications of the rule system could be envisioned anyway by considering a smaller (though powerful) subset of XPath [24]. This point is clearly outside the scope of this paper.

Independently of the value of the Path-visibility and Sibling parameters, rule RA1 states that a new path has to be created from any pair of nodes (Parent(n1),n2) such that n1∈RA1.Anc and n2∈RA1.Desc, thereby hiding the ancestor relationship between *n1* and *n2*. Similarly, rule RA2 states that a new path has to be created from any pair of nodes (Parent(n3),n2) such that n3∈RA2.Anc and n2∈RA2.Desc. Since RA1.Anc⊆RA2.Anc, Parent(n3)∈Anc(Parent(n1)). Thus, hiding the ancestor relationship between *n3* and *n2* in turn hides the one between *n1* and *n3*, due to the transitivity of the ancestor relationship. In other words, RA1 is subsumed by RA2.

Conflicts on Path-Visibility

Path-visibility conflicts arise when two different rules, targeting the same ancestor-descendant relationship (i.e., RA1.Anc=RA2.Anc and RA1.Desc=RA2.Desc), exhibit two different Path-visibility. Table 3 summarizes all possible combinations of Path-visibility for these two rules and their associated conflict resolution. The proposed conflict resolution being commutative, operand1 and operand2 represent either the Path-visibility of RA1 or RA2. While solving conflicts, the decision is always taken based on the least privilege principle.

Operand1	Operand2	Conflict Resolution
[ ]	∀	Operand2
[†]	∀	[†]
[Φ]	[Φ]	[Φ]
[Φ]	[$label_1$?,…, $label_n$?]	∀ node ∈ Path(anc,desc) if ϕ_{label} (node) ∉ [$label_1$?,..,$label_n$?] then [$label_1$?,.,$label_n$?] U [ϕ_{label} (node)$^{\Phi}$]
[$label_1$?,..., $label_n$?]	[$label'_1$?,…, $label'_n$?]	L = [$label_1$?,…,$label_n$?] U [$label'_1$?,…,$label'_n$?] ∀$label_i$∈ L, $label_i^{\dagger}$∈ L and $label_i^{\Phi}$ ∈ L ⇒ L – [$label_i^{\Phi}$]

Table 3. Path-visibility conflict resolution

If one rule does not impose any restriction on the path, the Path-visibility of the second rule always takes precedence (row 1). If one rule discards all nodes from the path, its Path-visibility always takes precedence (row 2). Row 3 is self-explanatory. If one rule depersonalizes all nodes on the path while the other rule selects a list of labels, this list of labels has to be expanded with missing depersonalized labels (row 4). Finally, if each rule selects a list of labels, the union of the two lists has to be computed (row 5). If the same label is present on both lists, $label_i^{\dagger}$ takes precedence over $label_i^{\Phi}$.

Conflicts on Sibling

Sibling conflicts arise when two different rules, targeting the same ancestor-descendant relationship, exhibit two different sibling decorrelations. Table 4 summarizes all possible combinations of Sibling for these two rules and their associated conflict resolution. Again, the proposed conflict resolution being commutative, operand1 and operand2 represent either the Sibling parameter of RA1 or RA2 and the conflict resolution is done in accordance with the least privilege principle.

Operand1	Operand2	Conflict Resolution
[⊥]	∀	[⊥]
[≡]	∀	Operand2
[ψ]	[$label_1$, … $label_n$]	[ψ] if ϕ_{label} (desc) ∈[$label_1$, … $label_n$], [⊥] otherwise
[$label_1$, … $label_n$]	[$label'_1$, … $label'_n$]	[$label_1$, … $label_n$] ∩ [$label'_1$, … $label'_n$] = A, [⊥] if A= ∅

Table 4. Sibling conflict resolution

Row 1 states that [⊥] takes always precedence over [ψ], [$label_1$,…,$label_n$] and [≡], since it is the most restrictive policy (the descendant is disconnected from all its siblings). In the same way, [ψ] and [$label_1$,…,$label_n$] takes always precedence over [≡], since [≡] is the less restrictive policy (row 2). Conflicts between [ψ] and [$label_1$,…,$label_n$] (row 3) and between two lists of labels (row 4) can be treated uniformly, considering that [ψ] is nothing but the singleton [ϕ_{label} (desc)]. Solving the conflict sums up to compute the intersection between both lists of labels. If this intersection turns to be empty, [⊥] becomes the final decision.

Note that sibling conflicts may happen indirectly if a same node is selected to participate in different groups of siblings (e.g., RA1:<-,/B,-,[A]> and RA2:<-,/C,-,[A]>). The intuition may lead to duplicate A elements in order to integrate them at the same time in B's and C's siblings. However, the presence of the same A element(s) in both sets may allow some inference regarding the relationship between B and C elements. For this reason, the conflict is solved by enforcing [⊥] as the Sibling parameter of the conflicting rules.

5. Implementation issues

An important question is whether tractable algorithms can be devised to implement relationship-aware access control models. To this end, we study how existing algorithms supporting node authorizations can be extended to cope with relationship authorizations. Two classes of algorithms are considered depending on whether the access control policy applies on a materialized document [2,10,16] or on a streaming document [7]. We give preliminary performance measurements for the first class of algorithms, which is the most popular one and for which one public domain algorithm can serve as reference [11]. The objective is to assess whether the relative cost incurred by the management of relationship authorizations remains acceptable. A more comprehensive performance analysis including algorithmic optimizations is part of our future work but is outside the scope of this paper.

Let us first consider the algorithm implemented by Damiani et al and put in the public domain [11]. This algorithm evaluates an access control policy composed by node authorizations and implements the foundation of the abstract model introduced in Section 4.2. Roughly speaking, this algorithm works as follows. First, the *building phase* of the algorithm parses the `Source` document and builds a DOM representation thanks to the Apache's Xalan tool. Main memory data structures are allocated at this stage to annotate later on the document nodes with authorization rules. Second, the *tree labeling phase* evaluates the NA rules related to a given subject. Each time a node is targeted by the XPath expression associated to a NA rule, the node is annotated with the rule. Third, the *conflict resolution phase* resolves potential conflicts among rules targeting the same node and propagates the final decision about the outcome of each node to its subtree. Finally, the *pruning phase* discards every node annotated negatively. Augmenting this algorithm with the management of RA rules is rather straightforward. The building phase remains unchanged. The tree labeling phase evaluates RA rules in parallel with NA rules and each descendant node targeted by the XPath expression (i.e., Anc/Desc) associated to a RA rule is annotated with the rule. The conflict resolution phase is extended with the management of potential conflicts among RA rules targeting the same node, according to the policy discussed in Section 4.4. No propagation is required by RA rules. The pruning phase remains unchanged. A final *reconstruction phase* is however mandatory to perform the cloning and shuffling operations.

The extension described above has been implemented in Java, making Damiani's algorithm relationship-aware at a minimal software development cost. Preliminary performance measurements have been conducted on this prototype on a PC equipped with a 3Ghz CPU and

1GB of RAM. Damiani's algorithm, as well as our extension, makes sense only if the DOM representation of the `Source` document fits in memory. Considering the high expansion factor between a `Source` document and its DOM representation (extended with annotation structures), we used synthetic datasets with a large population of small nodes, the node cardinality being the decisive factor in our context. We used ToXgene [35] to generate a collection of documents ranging from 10 K nodes to 150 K nodes and sharing the structure of the medical motivating example depicted in Figure 1. We conducted a first set of experiments varying the number of rules participating in an access control policy, the mix of NA and RA rules, the complexity of these rules (absolute vs. relative XPath expressions, predicates, number of targeted nodes) and the number of conflicts among rules. The meaning of these rules was not a concern at this stage. The impact of relationship authorizations on the performance of each algorithm's phase, as observed in these experiments, deserves the following remarks.

The building and pruning phases are common to the initial algorithm and its extension. Not surprisingly, the performance of these two phases evolved linearly with the document size and can thus be expressed in terms of ratios (building = 4.4ms/Knode, pruning = 0.4ms/Knode).

The cost of the labeling phase appeared to be more related to the number of rules participating in the access control policy than to their own complexity. This cost turned out to be the dominant factor with policies involving more than 8 rules. Different results could probably be obtained with different XPath engines but these variations would impact NA and RA rules exactly the same way, making this issue irrelevant for this study.

The conflict resolution phase is strongly dominated by the traversal of the DOM tree and the associated data structure containing annotations, making its cost again linear (1,07ms/Knode). Consequently, while conflicts among RA rules are semantically more complex to solve than conflicts among NA rules, the difference in terms of performance turned out to be non significant, increasing the total conflict resolution cost by less than 5%.

Finally, the reconstruction phase depends on the document size after pruning and on the number of nodes (i.e. Desc) that are targeted by RA rules and that need to be cloned. For example, the cost of depersonalizing the medical service ancestor of all Prescriptions (3640 elements) in a 150Knode document amounts to 840ms. Note that this rule has no other interest than generating a high number of clones (14560). There is no significant difference on the reconstruction cost between ancestor depersonalization and path reduction.

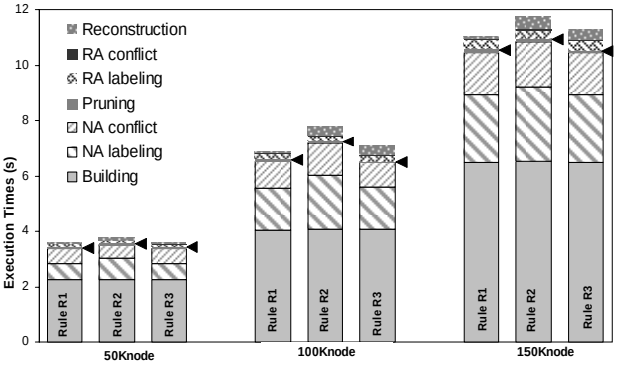

Figure 6: Impact of RA on the motivating example.

Figure 6 delivers a different view of the measurements. It pictures the total cost of each algorithm's phase and the extra cost incurred by relationship authorizations to implement the access control policies of our motivating example. Rules R1, R2 and R3, being actually a mix of NA and RA rules (see Figure 5) are evaluated on documents ranging from 50Knode to 150Knode. To facilitate the readability of the figure, a cursor (on the right side of each histogram) separates the cost incurred by the evaluation of NA rules (below the cursor) from the extra cost incurred by RA rules (above the cursor). The NA total cost corresponds to (building + NA labeling + NA conflict + pruning) while the RA extra cost corresponds to (RA labeling + RA conflict + reconstruction). The figure is almost self-explanatory. The total cost is here dominated by the building phase, the most complex policy containing 5 rules. Regarding the reconstruction phase, Rule R2 incurs the highest cost. The cloning cost explains this behavior. Indeed, R1 and R3 target the same number of Desc nodes (respectively 120, 240 and 360 for a 50Knode, 100Knode and 150Knode document) since there is a single `Name` element in each `folder`. Rule R2 targets three times more elements since each `Protocol` (one per `folder`) owns in the average three `Act` children. The shuffling cost itself is insignificant, shuffling being nothing but a random function call. As a final remark, the relationship authorizations come in this example at a rather low cost (roughly 10% of the total cost).

However, these preliminary conclusions have to be put in perspective with the fact that all measurements have been performed in memory. To

evaluate access control policy on huge documents, a streaming strategy must be investigated. An algorithm based on non-deterministic finite automata (NFA) has been recently proposed to tackle this issue [7]. This algorithm implements a traditional access control model, similar to the abstract model introduced in Section 4.2.

In this approach, NA rules are translated into NFA and a global authorization rules evaluator determines the authorization outcome for a current node according to the current state of all NFA. Extending the approach with relationship authorizations seems natural since the Anc and Desc XPath expressions of each RA rule can be translated in NFA, in the same way as NA rules. Once the outcome of a node is determined, the stack materializing the current states of each NFA contains enough information to deliver the path from the root to the current node. This information is sufficient to implement ancestor depersonalization and path reduction. However, the reconstruction phase relies on shuffling, a mechanism hurting data streaming. Indeed, the nodes to be shuffled have to be buffered until the subtree rooted at their common ancestor has been totally explored. Note however that buffering is required even in the case of node authorizations to manage pending predicates (i.e. a predicate P conditioning the delivery of a subtree S but encountered after S while parsing the document). This issue has been tackled in [7] and similar techniques seem to be applicable in our context. Making streaming access control algorithms relationship-aware is part of our future work.

6. Related works

As stated in the introduction, the problem of regulating access to XML documents has attracted a considerable attention from the database community in recent years.

Most discretionary access control (DAC) models for XML [3,4,10,16] share the same foundation while proposing different interpretations or options to define subjects, to propagate rules down through the hierarchy and to solve conflicts among rules. [26] introduces static analysis rather than run-time checks for this class of models. [23] proposes a provisional access control model that allows for more sophisticated controls (e.g., an access can be granted provided it is recorded in an audit trail). All these models focus on node authorizations and then fail in answering the requirements introduced in our motivating example. In addition, and as mentioned in the introduction, expressing an exception to a negative rule may lead to tricky situations that are difficult to manage accurately without relationship authorization. While this situation is even not mentioned in certain proposals, some models preclude it [3,16,26] while some others disclose the label of forbidden ancestors [10,23] or disclose at least the presence of these ancestors by replacing their label by a dummy value [13,17]. In addition to dummy labels, [13] provides a new answer to this problem by transforming the view of a given path thanks to an XPath expression (executed at traversal time), under the assumption that the resulting view remains compliant with the DTD provided to the subject.

Role-based access control (RBAC) models for XML [21,37] suffer from the same limitation wrt access control on XML relationships. [29] allows for relationships among XML documents, considering each XML document as an entity playing a certain role, semantically linked with other entities. The problem is clearly different from the one addressed in our paper. Finally, mandatory access control (MAC) models for XML deserve an interesting remark regarding the management of XML relationships. MAC models are generally monotonic, meaning that sensitivity level (classification in the MAC vocable) assigned to objects increase along the hierarchy (i.e., descendants have a higher sensitivity level than their respective ancestor). Therefore, the situation where a given subject is granted access to a descendant of a prohibited node can simply not be expressed. [9] introduces a non-monotonic MAC model that circumvents this limitation. However, the focus of this work is on optimizing the secure evaluation of XPath twig queries. Query rewriting techniques are used to append additional security check predicates on sensitivity levels to the original twig query. Therefore, the model can grant access to a descendant of a forbidden node but no visibility can be given of the path from the document root to this node.

The protection of relationships has been considered in other contexts. Access control models for object-oriented databases[7] handle relationship authorizations but with a focus on instantiation and inheritance [28]. [5] deals with the identification of sensitive associations in a

[7] Actually, basic concepts introduced in XML access control models, such as positive and negative authorizations, authorization propagation on hierarchies and conflict resolution policies, are directly inherited from access control models for object-oriented databases [32].

relational context but for release control rather than access control. Finally, [19] introduces the concept of association security objects in the RDF context. An history of query results is built and serves for the detection of forbidden correlations among data.

Therefore, we are not aware of other works trying to tackle the classification disclosure and uniform filiation problems in XML. In addition, due to the lack of explicit ancestor relationship authorizations, granting access to a descendant of a prohibited node is either impossible or source of important difficulties in existing models.

7. Conclusion

Several access control models have been proposed so far for regulating access to XML documents. By focusing on node authorizations and disregarding relationship authorizations, these models fail in answering the classification disclosure and uniform filiation problems. The consequence is the violation of the need-to-know and consent principles (two basic principles of laws related to the safeguard of personal information) in a number of situations.

To tackle this important issue, we advocate the integration of ancestor and sibling relationships as first class citizen in the access control models for XML. More precisely, this paper makes the following contributions. First, it characterizes two classes of relationship authorizations (ancestor depersonalization and path reduction) taking into account the sibling relationship dimension. Two mechanisms (cloning and shuffling) are introduced to translate accurately these authorizations into an authorized view of a source document. Second, it proposes a rule-based formulation for these classes of relationship authorizations allowing their seamless integration in existing XML access control models. This makes these models (some of them being well recognized today) relationship-aware. Third, it shows that tractable algorithms can be devised to support relationship authorizations and provides preliminary performance measurements.

The next step in our research agenda is to study streaming algorithms supporting relationship authorizations. Streaming access control is actually a major issue with respect to performance (management of large documents) and support of new forms of data delivery (selective data dissemination).

8. References

[1] Anderson, R.J. *A Security Policy Model for Clinical Infor-mation Systems*. IEEE Symp. on Security and Privacy, 1996.

[2] Bertino, E., Braum, M., Castano, S., Ferrari, E., Mesiti, M. *AuthorX: A Java-Based System for XML Data Protection*, IFIP Working conference on Databases Security, 2000.

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

[4] Bertino, E., Ferrari, E. *Secure and Selective Dissemination of XML Documents*. ACM TISSEC 5(2), 2002.

[5] Bettini, C., Wang, X.S., Jajodia, S. *Identifying Sensitive Associations in Databases for Release Control.* In Proc. Of SDM:VLDB Workshop. 2004.

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

[7] Bouganim, L., Dang-Ngoc, F., Pucheral, P. *Client-Based Access Control Management for XML Documents*, VLDB, 2004.

[8] Bruno, E. , Le Maitre, J. , Murisasco, E. *Extending XQuery with transformation operators*. ACM Symposium on Document Engineering 2003: 1-8.

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

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

[11] Damiani, E., De Capitani di Vimercati, S., Paraboschi, S., Samarati, P. *Fine-Grained XML Access Control Prototype,* http://seclab.dti.unimi.it/~xml-sec/

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

[13] Fan, W. , Chan, C.Y. , Garofalakis, M. *Secure XML Querying with Security Views*, SIGMOD, 2004.

[14] Fernandez, E.B., Gudes, E., and Song, H. *A Model for Evaluation and Administration of Security in Object-Oriented Databases*, IEEE TKDE 6(2),1994.

[15] Fernandez, E. , Summers, R., Wood, C. *Database security and integrity*. Addison-Wesley, Reading, Mass., 1981.

[16] Gabillon, A., Bruno, E. *Regulating access to XML documents*. IFIP Conf. on Database and Application Security, 2001.

[17] Gabillon, A. *An Authorization Model for XML DataBases*. ACM Workshop on Secure Web Services. 2004.

[18] Gostin, L.O., Lazzarini, Z., Flaherty, K.M. *Legislative Survey of State Confidentiality Laws,* Final Report Presented to: The U.S. Centers for Disease Control and Prevention; http://www.epic.org/privacy/medical/cdc_survey.html, 1997.

[19] Gowadia, V., Farkas, C. *RDF Metadata for XML Access Control.* ACM Workshop on XML Security. 2003.

[20] Gudes, E., Song, H., and Fernandez, E.B. *Evaluation of negative and predicate-based authorization in object-oriented databases*, IFIP Workshop on Database Security, 1990.

[21] Hitchens, M., Varadharajan, V. *RBAC for XML Document Stores*, ICICS, 2001.

[22] Jajodia, S., Samarati, P., Sapino, M., Subrahmanian, V., *Flexible support for multiple access control policies*, ACM TODS, 26(2), 2001.

[23]Kudo, M., Hada, S. *XML Document Security based on Provisional Authorization*, ACM CCS, 2000.

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

[25] Miklau, G., Suciu, D. *Cryptographically Enforced Conditional Access for XML*, WebDB, 2002.

[26] Murata M., Tozawa A., Kudo M., *XML Access Control Using Static Analysis*, ACM CCS, 2003.

[27]OASIS standard, *eXtensible Access Control Markup Language*, http://www.oasis-open.org/committees/xacml, 2003.

[28] Olivier, M., Von Solms, S. *A Taxonomy for Object-Oriented Secure Databases*, ACM TODS, 19(1), 1994.

[29] Parmar, V., Shi, H. XML Access Control for Semantically Related XML Documents, HICSS, 2003.

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

[31] Privacy Rights Clearinghouse/UCAN, *HIPAA Basics: Medical Privacy in the Electronic Age*, 2003. http://www.privacyrights.org/fs/fs8a-hipaa.htm#1

[32] Rabitti, F., Bertino, E., Kim, W., and Woelk, D. *A model of the authorization for next-generation databases systems*, ACM Trans Database Syst 16(1), 1991.

[33] Rabitti, F., Woelk, D., Ki, W. *A model of authorization for object-oriented and semantic databases*, EDBT, 1988.

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

[35] ToXgene – The ToX XML data generator, http://www.cs.toronto.edu/tox/toxgene/

[36] United States Department of Health and Human Services, "HIPAA : Health Insurance Portability and Accountability Act", Public Law 104-191, 104th Congress, 1996. http://www.hhs.gov/ocr/hipaa/

[37] Wang, J., Osborn, S.L. *A Role-Based Approach to Access Control for XML Databases*, ACM SACMAT, 2004.

[38] Wang, Y., Tan, K.L. *A Scalable XML Access Control System*, WWW Conference (poster), 2001.

[39] XrML eXtendible rights Markup Language, http://www.xrml.org/