Authorization-Transparent Access Control for XML Under the Non-Truman Model

Yaron Kanza, Alberto O. Mendelzon, Renée J. Miller, and Zheng Zhang

Department of Computer Science,
University of Toronto,
Toronto, Canada
{yaron, mendel, miller, zhzhang}@cs.toronto.edu

Abstract. In authorization-transparent access control, user queries are formulated against the database schema rather than against authorization views that transform and hide data. The Truman and the Non-Truman are two approaches to authorization transparency where in a Truman model, queries that violate the access restrictions are modified transparently by the system to only reveal accessible data, while in a Non-Truman model, such queries are rejected. The advantage of a Non-Truman model is that the semantics of user queries is not changed by the access-control mechanism. This work presents an access-control mechanism for XML, under the Non-Truman model. Security policies are specified as parameterized rules formulated using XPath. The rules specify relationships between elements, that should be concealed from users. Hence, not only elements, but also edges and paths within an XML document, can be concealed. The access-control mechanism authorizes only *valid queries*, *i.e.*, queries that do not disclose the existence of concealed relationships. The additional expressive power, provided by these rules, over element-based authorization techniques is illustrated. The proposed access-control mechanism can either serve as a substitute for views or as a layer for verifying that specific relationships are concealed by a view.

1 Introduction

Access control is a fundamental part of database systems. The purpose of access control is to protect private or secret information from unauthorized users. Given the status of XML as a standard for storing and exchanging data, the need for XML access control has been recognized and has received a lot of attention [3, 4, 9, 11, 14].

When an access-control model is *authorization transparent*, users formulate their queries against the database schema rather than against authorization views that transform and hide data [21]. Rizvi *et al.* [22] present two basic approaches to access control in authorization-transparent systems. The first approach is referred to as the *Truman model* and the second as the *Non-Truman model* [22]. In the Truman model, an access control language (often a view language) is used for specifying what data is accessible to a user. User queries are modified by the system so that the answer includes only accessible data. Suppose Q is a user query, D is a database and D_u is the part of D that the user is permitted to access, then Q is modified to a safe query Q_s such that $Q_s(D) = Q(D_u)$.

Y. Ioannidis et al. (Eds.): EDBT 2006, LNCS 3896, pp. 222–239, 2006.
© Springer-Verlag Berlin Heidelberg 2006

Example 1. Consider a database that contains information on courses in a university. For each course, the system stores information about the students who are enrolled, and the grades that they have received. Suppose that a Truman access control model is used to specify that each student is permitted to see only her grades (not the grades of other students). If student Alice poses a query that asks for the highest grade received in one of the courses in which she is enrolled, say *Databases 101*, the system will modify the query to return the highest grade that Alice has received in *Databases 101*.

As Rizvi *et al.* [22] point out, using a Truman access-control model, the answers to queries may be misleading. A user may wrongly assume that an answer to a query is correct over the entire database. In our example, Alice may be misled into thinking she is the best in the class (after all, she asked for the highest grade over all students).

Misleading answers are prevented by the Non-Truman model, an alternative, authorization-transparent model. In the Non-Truman model, a query that violates access-control specifications is rejected, rather than modified. Only *valid* queries, *i.e.*, queries that do not violate the access specifications, are answered. Hence, query answers are always the result of applying the user query to the entire database. The Non-Truman model has the desirable property that the semantics of a query is independent of the access-control specification. In Example 1, for instance, if the system uses a Non-Truman access-control model, then the query of Alice will be rejected. Alice will only receive answers to queries that are valid with respect to the access-control policy.

In a Non-Truman model, a fundamental question is the definition of validity. Rizvi *et al.* [22] use a mechanism in which the accessible data is defined using views. Given a database D, a query Q is validated by checking whether it could be rewritten using only the authorized views V. The rewritten query needs to be equivalent to Q either for all possible database states (referred to as *unconditional equivalence* [22] since it is independent of the current database state D) or for only those database states D' for which $V(D) = V(D')$ (termed *conditional equivalence* [22]).

Certainly, such an approach is possible for XML as well. However, results on answering queries using views for small fragments of XML query languages are still emerging [28], and may be undecidable even for the relational model [22]. Furthermore, a view is a positive statement about what data is accessible and it is up to the designer of the view to decide what can be put in the view while still hiding the desired private data. Regardless of the form of the view or access control mechanism, we would like to be able to make statements about what information is *concealed* from a user. In our work, we will specifically consider what it means to conceal a relationship in an XML document.

Example 2. Consider an XML document D that contains information about departments and employees in a company. There is an edge from each department element d to an employee element e whenever e works in d. A company may have an access control policy that permits access to all employees and departments, but that restricts access to the `works-in` relationship. That is, a user should be able to ask queries about employees and departments, but the company may not wish to reveal who works in which department. Perhaps this information may reveal strategic information about the direction of the company.

Information disclosure has been studied formally. Miklau and Suciu [20], define disclosure as exposing information that increases the probability that a user can guess concealed information. There are cases, however, where rejecting a query just because its answer decreases the user's uncertainty about the concealed data is too restrictive [29]. If we consider a set of relationships, it may be sufficient to ensure that a user cannot distinguish between the current document and other documents that differ from the current document only in the concealed relationships.

Intuitively, a query conceals a relationship if the query answer does not reveal the presence (or absence) of a relationship in the document. To understand our semantics, consider the following example.

Example 3. Considering again Example 2 where the relationship between departments and employees is secret. Consider a query Q_1 that looks for all the employees in the company, regardless of their department, and a query Q_2 that looks for the employees in a specific department d. The query Q_1 conceals the relationships between departments and employees, while Q_2 does not.

In this work, we propose a precise semantics for what it means to *conceal* a relationship. We propose a mechanism for testing whether an XPath query *conceals* a relationship or set of relationships. In particular, we can test whether a view, specified by an XPath query, conceals a relationship.

Our model controls access to relationships. This approach provides a finer granularity than restricting access to elements. On one hand, restricting access to an element is possible in our approach. This is done by concealing all the relationships (edges and paths) to that element. On the other hand, in our approach it is possible to conceal a relationship without restricting access to any of the elements in the document. Returning to our example, our mechanism will permit access to employees and departments while restricting only access to the set of works-in relationships.

The main contributions of our work are the following.

- The first authorization-transparent, Non-Truman access-control model for XML. Our mechanism is fine-grained and enforces access control at the level of ancestor-descendant relationships among elements.
- A new semantics for concealing relationships in an XML document, where a relationship is defined by an edge or a path in the document. Our semantics uses a variation of k-anonymity [25]. To specify relationships in our mechanism, we use rules, each containing a pair of XPath expressions.
- We define two forms of query validity. A query is *locally valid* for a document and a set of rules, if it conceals all the relationships that are specified by the rules. Queries may be executed only if they are locally valid. For documents conforming to a schema, we define a stronger form of validity. A query is *globally valid* for a set of rules and a schema if the query is locally valid for the rules and each document that conforms to the schema.
- Finally, we show that indeed valid queries do not reveal information about concealed edges.

2 Related Work

Many non-authorization-transparent access-control models for XML have been proposed. Damiani *et al.* [9, 10] presented a model where restricted elements are identified using labels. These restricted elements are pruned from the document before queries are posed. A similar mechanism was proposed by Bertino *et al.* [2, 3] where the restricted parts of an XML document are encrypted rather than pruned. Encrypting the unauthorized data has also been used in the access-control model of Miklau and Suciu [18]. In their model the access control specifications are defined using a language that extends XQuery. Fan *et al.* [11] specified security by extending the document DTD with annotations and publishing a modified DTD. In their model, queries are formulated over the modified DTD and are rewritten by the system to befit the original DTD. The optimization of secure queries has also been given some attention [6, 30].

Fundulaki and Marx [13] survey a number of approaches that permit access control to be specified on elements within a document. Restricting access to elements has also been used in XACML [15] and XACL [16], two proposed industrial standards. An alternative approach of hiding element relationships was proposed by Finance *et al.* [12], however, their model is not authorization transparent. Authorization-transparent models have been proposed, so far, only for the relational model [21, 23, 24].

In contrast, we present the first authorization-transparent, Non-Truman model for XML. Queries are posed on the original document, thus, we do not present a model for publishing secure data. In our model, users simply specify the element relationships that should be concealed. For defining concealment we use a variation of k-anonymity. Various aspects of k-anonymity were studied in the relational model [1, 17, 25, 29]. To our knowledge, our work is the first to apply k-anonymity to XML. In Section 4, we define precisely the relationship of our model with k-anonymity. Our main focus is to provide a test of query validity for ensuring that valid queries effectively conceal secure relationships. This is important since unlike the non-authorization-transparent approaches, in our model, queries are posed on the entire document.

3 Data Model

In this section, we introduce our data model. We assume that the reader is familiar with the notion of a rooted labeled directed graph. We present a rooted labeled directed graph G, over a set L of labels, by a 4-tuple $(V, E, r, \textit{label-of}_G)$, where V is a set of nodes, E is a set of edges, r is the root of G and $\textit{label-of}_G$ is a function that maps each node to an element of L.

Document. Let L be a finite set of labels and A be a finite set of atomic values. An *XML document* is a rooted labeled directed tree over L with values of A attached to atomic nodes (*i.e.*, to nodes that do not have outgoing edges). Formally, a document D is a 5-tuple $(X, E_D, \textit{root}_D, \textit{label-of}_D, \textit{value-of}_D)$, where the tuple $(X, E_D, \textit{root}_D, \textit{label-of}_D)$ is a rooted labeled directed tree over L, and $\textit{value-of}_D$ is a function that maps each atomic node to a value of A. The nodes in X are called *elements*. In order to simplify the model, we do not distinguish between elements and attributes and we assume that all the values on atomic nodes are of type PCDATA (*i.e.*, *String*).

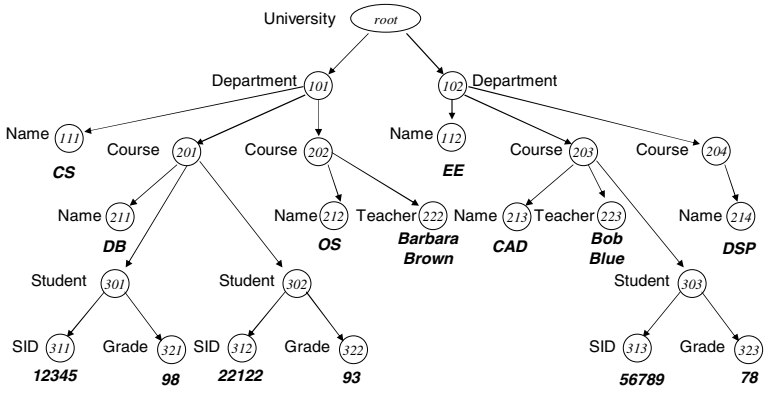


Fig. 1. A document that contains information on courses, students and grades in a university

Example 4. Figure 1 shows a document that contains information on courses, students and grades. Elements are represented by circles and are numbered, for easier reference. Atomic values appear below the atomic nodes and are written with a bold font.

XPath. In this work, we use XPath [8] for formulating queries and access control rules. XPath is a simple language for navigating in an XML document. XPath expressions are omnipresent in XML applications. In particular, XPath is part of both XSLT [7] and XQuery [5], the WWW-Consortium standards for querying and transforming XML.

In XPath there are thirteen types of axes that are used for navigating in a document. Our focus in this work is on the *child* axis (/) and the *descendant-or-self* axis (//) that are the most commonly used axes in XPath. Our model, however, can also be applied to queries that include the other axes.

4 Concealing Relationships

Before presenting our techniques, we first consider what it means to conceal a relationship. A *relationship* is a directed path between two elements. For example, in the university database shown in Figure 1, a student element is related to a grade element if there is an edge from the student element to the grade element.

A set of *relationships* is represented by a pair consisting of two sets of elements. For example, the pair (S, G), where S is the set of all elements labeled "Student" and G is the set of all elements labeled "Grade", represents the set of relationships between student and grades. Concealing the relationships (S, G) means that for every student s and grade g in the document, the user will not be able to infer (with certainty), from query answers, whether g is the grade for s. We will want this to be true for all authorized queries (*i.e.*, all *valid queries*). Note that we are concealing the presence or absence of relationships, so we are concealing whether any of the set of pairs in (S, G) exists in the document.

We also want to have some measure of the uncertainty that is gained by concealing relationships. Thus, we use a definition that is a variation of k-*anonymity* [25] applied to

relationships in an XML document. In the k-anonymity model, the goal is to provide a guarantee that each element cannot be distinguished from at least $k-1$ other elements. In our case, suppose that we want to conceal a set of relationships (A, B). Then, given the answer to a valid query and any element $b \in B$, the user will not be able to infer which element among some k sized subset of A is related to b. To make this more precise, we present a formal definition.

Definition 1 (k-Concealment). *Consider a set of valid queries $\mathcal{Q}$, a document D, and two sets A and B of elements in D. The relationships (A, B) are* k-concealed *if for every $b \in B$ there exist k elements $a_1, \ldots, a_k$ of A and k documents $D_1, \ldots, D_k$ over the element set of D, such that the following conditions hold for every $1 \leq i \leq k$.*

1. *In D_i, the element b is a descendant of a_i. Furthermore, b is not a descendant of any element among $a_1, \ldots, a_k$, except for a_i.*
2. *$Q(D) = Q(D_i)$, for every valid query $Q \in \mathcal{Q}$.*

Example 5. Consider a university document D, similar to the document in Figure 1, and the set (S, G) of relationships between students and grades. Suppose that (S, G) is k-concealed, and let $\mathcal{Q}$ be a set of authorized queries. Let g be some grade element in D. Then, there are k documents that provide the answer $Q(D)$ for every query Q in $\mathcal{Q}$ and in each one of these k documents, g is below a different student. That is, there is a set of k students such that from the information revealed by answers to queries in $\mathcal{Q}$, a user cannot tell which one among these k students received g.

We consider a relationship to be concealed as long as *some* uncertainty remains about the ancestor-descendant relationships. Thus, in the rest of this paper, we will use the phrase "concealing relationships" for 2-concealment.

Given the definition of concealing relationships, we now turn to the logistics of specifying sets of relationships over XML documents. We will use pairs of XPath expressions for this purpose. Each pair will form an access-control rule. The two expressions will define a pair of sets, *i.e.*, a set of relationships that should be concealed.

5 Access Control Rules

Our approach to access control in XML documents is based on rules rather than views. While views are normally "positive" in the sense that they specify what the user is allowed to know, our rules are "negative" and specify what should be concealed from the user. Our access-control rules specify pairs of elements in the document and by this designate the relationships between these elements as being restricted. In this section, we first present the syntax of rules. Then, we explain why we use rules rather than views. We provide the semantics of rules in our model and define local and global validity. Finally, we briefly discuss the issue of testing validity.

5.1 The Syntax of Rules

Rules are formulated using XPath expressions. Each rule consists of two expressions specifying a set of ancestor and descendant elements. The rule specifies that the relationships between these two sets should be concealed.

Definition 2 (Rule). *An* access control rule *(or rule) is defined as:*

`for` $path_1$ `exclude` $path_2$

where $path_1$ *and* $path_2$ *are XPath expressions. The path* $path_2$ *is a relative XPath expression with respect to* $path_1$.

Example 6. Suppose that we want to prevent queries from disclosing information about what grades were given to which students. This restriction can be specified by the following rule: `for //Student exclude /Grade`.

Example 7. Suppose that in the CS department, relationships between students and grades and relationships between courses and grades should be concealed. To specify this restriction, two rules are used:

`for /Department[Name='CS']//Student exclude //Grade`, and
`for /Department[Name='CS']/Course exclude //Grade`.

In many scenarios, different users have different access permissions. For example, an institution could have a policy where a course teacher can have access to all the grades of the course while students can only see their own grades. To support this, the access control rules are parameterized. Parameterized variables are written with a preceding dollar sign. Common parameters include user ids, environment variables, time parameters, *etc.*

Example 8. Suppose that `$userid` is instantiated to be the current user identifier. Consider a policy where a student is permitted to see her own grades, but she should not see the student-grade relationships of other students. This policy is specified by the following rule:

`for //Student[not(SID=$userid)] exclude /Grade`.

Note that when `$userid` is null the comparison `SID=$userid` is false.

5.2 Rules Versus Views

We now explain why we use rules instead of views for XML access control in the Non-Truman model. The first reason is that there are many cases where using rules is simpler and requires a more succinct formulation than using views. The following example illustrates such a case.

Example 9. Suppose that we want to prevent users from knowing which student is enrolled in which course, but do not wish to conceal other information in the document. We can specify this using the rule: `for //Course exclude //Student`. If SID is identifying, we may also want to hide the relationship from course to a student's SID using the rule:

`for //Course exclude //SID`.

Note that these rules should not prevent evaluation of queries that "jump" over a restricted edge. For example, a query that returns the grades of a specific course does not violate the rules. Neither does a query that returns the grades of a specific student.

It is not easy to formulate an XQuery view that preserves all the relationships in the document except for the course-student and course-SID relationships. One example for such a view is a query Q_{cut} that reconstructs the whole document with the following changes. First, student elements should be moved, with all their content, to be below their department element. This cuts the relationship between students and courses but keeps the relationships between departments and students. Second, grade elements may be copied and pasted below their ancestor course elements. We need to duplicate grades, because we need grades to be related to both courses and students. Note that Q_{cut} would not work if in the original document, courses have an element named "Grade" as a child. It is cumbersome to formulate Q_{cut} in XQuery. Hence, in many cases, defining access-control policies by views is more error-prone than using our rules. We consider in this example XQuery, however, the same problem occurs also in other languages for defining "positive" views.

The second reason for choosing rules instead of views is that with views it is difficult to verify that what we want to conceal is indeed concealed.

Example 10. Consider two views. One view contains students and their grades. The second view contains courses and for each course the grades that were given in the course. Do these views really conceal all the relationships between courses and students? Apparently not. Suppose that there is a grade, say 78, that appears only once in the document. Then, knowing who received this grade and in which course this grade was given, it is possible to infer a relationship between a student and a course.

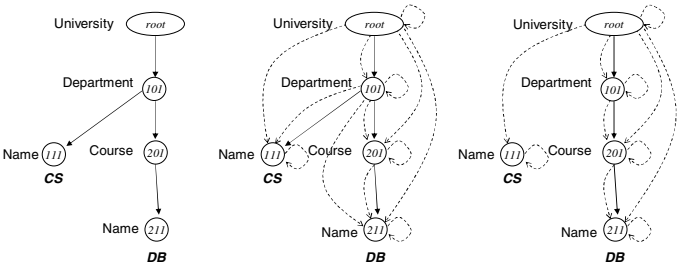
Later in this paper we will present the notion of a coherent set of rules and we will show that when using a coherent set of rules, we can guarantee that restricted relationships are indeed concealed.

The third reason for not using authorization views is that in the Non-Truman model, when using views, testing validity is defined as the problem of answering-queries-using-views. However, answering-queries-using-views is not always decidable and may have a very high time complexity [22]. Note that the problem of answering-queries-using-views is different from the simpler problem of answering queries posed on views.

5.3 Local Validity

In the Non-Truman model, queries are evaluated only if they pass a validity test. We now define local validity for queries, given a document and a set of rules. We start by providing some necessary definitions and notation. Our first definition, of a document expansion, is a tool to help us define (later) the set of documents that should be indistinguishable from a given document, when using valid queries.

Document Expansion. Let $D = (X, E_D, root_D, label\text{-}of_D, value\text{-}of_D)$ be a document. An *expansion* of D, denoted D'', is a labeled directed graph that is created by replacing E_D with a new set of edges E' called *child edges*. In addition, we add to D a second set E''_D of edges, called *descendant edges*. Hence, the expansion of D is a tuple $((X, E', root_D, label\text{-}of_D, value\text{-}of_D), E''_D)$, where E' is a set of child edges and E''_D is a set of descendant edges. Note that the expansion is not necessarily a tree and is not even required to be connected.



(a) Document D_1 (b) The transitive closure of D_1 (c) A document expansion

Fig. 2. A document D_1, the transitive closure of D_1 and a document expansion

To understand the role of the separate child and descendant edges, it is useful to consider one special expansion, the *transitive closure*, formed by adding to D a descendant edge between any two connected nodes in D.

Transitive Closure. The *transitive closure* of a document D, denoted as $\bar{D}$, is a document expansion where $E' = E_D$. The transitive closure is $(D, E^{//}_D)$, such that in $E^{//}_D$ there is an edge between every two nodes that are connected by a directed path in D. The direction of the edge is the same as the direction of the path. Also, $E^{//}_D$ contains an edge from every node to itself. Note that the original edge set of D is not being replaced. As an example, Figure 2(b) shows the transitive closure of the document in Figure 2(a). Child edges are drawn with solid lines and descendant edges with dashed lines.

The evaluation of an XPath expression over a document expansion is by *following* a child edge whenever a child axis occurs and following a descendant edge whenever a descendant-or-self axis occurs. We explain this in the following example.

Example 11. Consider the XPath query `//Department[Name='CS']//Course` over a document expansion $D^{//}$. This query returns course elements c that satisfy the following. There are a department element d and a descendant edge in $D^{//}$ from the root to d. There is an element n with label "Name", with value "CS" and there is a child edge in D from d to n. Finally, there is a descendant edge in $D^{//}$ from d to c. Note that to satisfy the `//` axis we require the existence of a descendant edge rather than the existence of a path between the relevant nodes.

It is easy to see that posing an XPath query Q on a document D is equivalent to evaluating Q over the transitive closure of D. However, when evaluating Q over a document expansion that is not the transitive closure of D, we may get an answer that is different from the answer to Q over D.

Pruning of a Document Expansion. Given a set R of access control rules, a *pruning* of a document expansion $D^{//}$ is a new document expansion, denoted $prune_R(D^{//})$, that is created by removing from $D^{//}$ all the edges (both child edges and descendant edges) that connect a *restricted* pair of nodes. By restricted pair, we mean two nodes whose relationship should be concealed according to R. For example, the pruning of the tran-

sitive closure of D_1 (Figure 2(b)) by the rule `for //Department exclude //Name` is depicted in Figure 2(c).

We represent a rule ρ of the form `for` x_1 `exclude` x_2 as a pair (x_1, x_2). By x_1x_2 we denote the XPath expression that is created by the concatenation of the expressions x_1 and x_2. In a document D, ρ specifies as restricted all the pairs (e_1, e_2) of elements of D such that $e_1 \in x_1(D)$ (*i.e.*, e_1 is in the answer to x_1 over D) and $e_2 \in x_1x_2(D)$. For example, the rule `for //Student exclude //Grade` specifies as restricted all the pairs of a student element and a grade of the student. A set of rules specify as restricted all the element pairs that are restricted according to at least one of the rules in the set.

Intuitively, given a rule $\rho = (x_1, x_2)$ we want to conceal whether (or not) there is a path between any two restricted elements. We can think of the existing paths in D as defining a subset P of $x_1(D) \times x_1x_2(D)$. We will define as valid only those queries that do not permit a user to distinguish whether D contains the subset P or another possible subset of $x_1(D) \times x_1x_2(D)$. This motivates the following definition.

Universe of Expansions. Consider a document D and a set of access control rules R. Let $\bar{D}$ be the transitive closure of D and let $prune_R(\bar{D})$ be the pruning of $\bar{D}$ using the rules of R. The *universe of expansions* (*universe*, for short) of D under the concealment of R, is the set of all document expansions D'' such that $prune_R(\bar{D}) = prune_R(D'')$. In other words, the universe contains all the document expansions that are created by adding to $prune_R(\bar{D})$ some edges that connect restricted pairs of nodes. We denote the universe of D by $\mathcal{U}_R(D)$.

Definition 3 (Local Validity). *Given a document D and a set of rules R, a query Q is* locally valid *if $Q(D) = Q(D'')$ for any document expansion D'' in the universe $\mathcal{U}_R(D)$.*

We now explain why we need to consider, in Definition 3, all the document expansions in the universe $\mathcal{U}_R(D)$ instead of applying a simpler test, called *pseudo-validity*, where we just consider the single document expansion $prune_R(\bar{D})$ (the pruning of the transitive closure of D), *i.e.*, the document expansion that contains only edges between non-restricted pairs.

A query Q is pseudo-valid if $Q(D) = Q(prune_R(\bar{D}))$. By Definition 3, the condition of pseudo-validity is necessary, but not sufficient for Q to be locally valid. The following example demonstrates a situation where secure information may be leaked due to authorizing pseudo-valid queries.

Example 12. Consider the university document D of Figure 1 and the rule ρ in Example 6 that conceals relationships between students and grades. Suppose we authorize pseudo-valid queries such as Q_i : `//Student[SID='12345' and Grade=`i`]`, for $i = 0, 1, \ldots, 100$. In all the 100 cases where $i \neq 98$, the query will be authorized and return an empty result. For $i = 98$ (*i.e.*, the grade of the student in the DB course), the query will not be authorized. This reveals the grade of a student in some course.

Such information leakage does not occur when only locally valid queries are authorized. To see why this is true, consider the document expansion D'' constructed as follows. Let D'' be the result of removing two edges and adding two new edges to the transitive closure $\bar{D}$. The removed edges are the two Student-Grade edges that connect Node 301 to 321 and Node 302 to 322. The two added edges are Student-Grade

edges that connect Node 301 to 322 and Node 302 to 321. All these added and removed edges are Student-Grade edges and thus, are removed in a pruning w.r.t. ρ. That is, $prune_\rho(\bar{D}) = prune_\rho(D'')$. Yet, evaluating Q_{93} w.r.t. D'' provides a different answer from the answer to Q_{93} over D. Thus, Q_{93} is not valid. Q_{78} is also not valid by a similar construction. All the three queries Q_{78}, Q_{93} and Q_{98} are rejected. Thus, a user could only tell that the grade of Student '12345' is one of the grades 78, 93, 98; however, this is what she could have learned from the result of the valid query `//Grade`.

The definition of local validity has a number of properties that are important in practice. For example, if two documents are equal (that is, isomorphic) except for their restricted edges, then a locally valid query will not be able to distinguish between them.

Proposition 1. *Consider a set of rules R and let D_1 and D_2 be two documents such that $prune_R(\bar{D}_1) = prune_R(\bar{D}_2)$. If a query Q is locally valid w.r.t. D_1 and R then Q is also locally valid w.r.t. D_2 and R. Furthermore, $Q(D_1) = Q(D_2)$.*

5.4 Global Validity

For documents conforming to a schema, we define a more restrictive form of validity called *global validity*. First, we formally define the notion of a schema.

Schema. In our model, a *schema* is represented as a rooted labeled directed graph. Our schema representation is a simplification of common XML schema-definition languages such as DTD [26] and XSchema [27]. A schema can be used to provide a succinct description of a document structure, or as a constraint on the structure of documents in a repository. Formally, a schema S, over a finite set of labels L, is a rooted labeled directed graph $(Names_S, E_S, root_S, label\text{-}of_S)$ over L, where the nodes are uniquely labeled. A document *conforms* to a schema if there exists a *homomorphism* from the graph of the document to the schema. An example of a schema is given in Figure 3(c). The document in Figure 1 conforms to this schema.

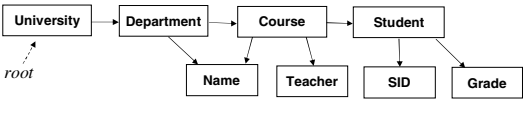


Fig. 3. A university schema

Definition 4 (Global Validity). *A query Q is globally valid for a set of rules R and a schema S, if, given R, Q is locally valid for every document D that conforms to S.*

Example 13. Let R contain the rule given in Example 6. This rule rejects queries that use the relationship between students and grades. Suppose a query Q that is asking for the grades of the student with id '00000' (*i.e.*, `//Student[SID='00000']//Grade`) is posed on the document in Figure 1. If there was a student with id '00000' in the document, then the query would not be considered locally valid and would not be authorized. Since there is no student with id '00000', there is no edge to prune and the

query is locally valid. Note that the query does not reveal the grade of any existing student. Although Q is locally valid, it is not globally valid if we consider the schema S shown in Figure 3. It is possible to construct a document D' that conforms to S and contains a student with id '00000'. Hence, the query will not be locally valid for D' and R. Thus, Q is not globally valid for schema S.

In some cases, global validity could be too restrictive; however, it does have some advantages over local validity. Suppose that there is a collection of documents and all the documents conform to the same schema. In this case, if a query is globally valid, then we do not need to check the validity of the query over each document. Furthermore, after a document is updated, if the new document still conforms to the schema, we do not need to revalidate queries.

5.5 Testing Validity

Due to lack of space, presenting algorithms for efficient validity testing is beyond the scope of this paper. However, it is important to notice that our model has the following advantages. First, local validity is always decidable. This is because for any document D and a set of rules R, the universe of expansions $\mathcal{U}_R(D)$ is finite. Secondly, for large classes of queries, local validity can be tested efficiently. Thirdly, there are important cases where global validity can be tested efficiently.

We now discuss one important case where testing local validity can be done efficiently. Consider XPath expressions that do not contain the logical operator `not`. Such queries are *monotone*. A query Q is monotone, if for every two document expansions $D''_1 \subseteq D''_2$ holds $Q(D''_1) \subseteq Q(D''_2)$. For testing local validity of a monotone query, it is sufficient to compute the query over two specific document expansions and compare the answers. Given a document D and a set of rules R, the document expansions on which the query should be computed are the following two. First, $\mathit{prune}_R(\bar{D})$. Second, the document expansion that is created from $\mathit{prune}_R(\bar{D})$ when connecting every restricted pair of elements, by both a child edge and a descendant edge.

6 A Coherent Set of Rules

Our goal is allowing users to conceal element relationships and let them be sure that what they want to conceal is truly concealed. Unfortunately, it is impossible to guarantee concealment for any arbitrary set of relationships. Sometimes, it is possible to infer a concealed relationship from the relationships that are not concealed. In this section, we characterize sets of rules whose designated relationships are indeed concealed.

We say that a set of rules is *coherent* if it is impossible to infer any concealed relationship from the relationships that are not pruned by the rules. Before providing the formal definition for a coherent set of rules, we give an example of two cases where a relationship can be inferred from a pair of non-concealed relationships.

Example 14. Suppose that in the university document it is known that the CAD course (Node 203) is given in the EE department (Node 102) and student 56789 (Node 303) is registered in the CAD course. In this case, the relationship between Node 102 and

Node 303 can be derived from the other two relationships, thus, there is no point in concealing it alone.

Suppose that the following is known. Student 12345 (Node 301) studies in the CS department (Node 101) and she is registered in the DB course (Node 211). Knowing that the document is a tree allows a user to infer that the DB course is given in the CS department (*i.e.*, Node 201 and Node 301 are related).

We now define when a set of rules is coherent. Consider a document D and a set of rules R. The set R has an *incomplete concealment* in a document D if one of the following two cases occurs. *(1)* Lack of transitivity: D has three elements e_1, e_2 and e_3 such that $\mathit{prune}_R(\bar{D})$ has an edge from e_1 to e_2 and an edge from e_2 to e_3, but $\mathit{prune}_R(\bar{D})$ does not have an edge from e_1 to e_3. *(2)* Lack of reverse transitivity: there are three elements e_1, e_2 and e_3 in D, such that $\mathit{prune}_R(\bar{D})$ has an edge from e_1 to e_3 and an edge from e_2 to e_3; however, $\mathit{prune}_R(\bar{D})$ does not have an edge from e_1 to e_2.

Definition 5 (A Coherent Set of Rules). *Given a document D, a set of rules R is* coherent *if an incomplete concealment does not occur in D. Given a schema S, a set R is coherent if R is coherent for every document that conforms to S.*

6.1 Coherence for Documents

There is a simple and efficient test for verifying that a set of rules R is coherent for a document D. The test starts by computing the pruning of the transitive closure of D according to R, and considering the edge set of $\mathit{prune}_R(\bar{D})$ as a relation r. There is a lack of transitivity if and only if the algebraic expression $\pi_{\$1,\$4}(r \bowtie_{\$2=\$1} r) - r$ is not empty. There is a lack of reverse transitivity if and only if the algebraic expression $\pi_{\$1,\$3}(r \bowtie_{\$2=\$2} r) - r$ is not empty.

Next, we provide intuitive conditions for constructing coherent sets of rules. Our conditions consider how relationships specified by different rules are related. We say that an edge (e_1, e_2) in a transitive closure $\bar{D}$ *encapsulates* an edge (e'_1, e'_2) if there is a path ϕ in $\bar{D}$ that goes through the four nodes e_1, e_2, e'_1, e'_2, and one of the following three cases holds: *(1)* e_1 appears on ϕ before e'_1, and e'_2 appears before e_2. *(2)* $e_1 = e'_1$, and e'_2 appears on ϕ before e_2. *(3)* e_1 appears on ϕ before e'_1, and $e_2 = e'_2$. The following is a necessary condition for the coherency of a set of rules.

Proposition 2. *Given a document D, if a set of rules R is coherent, then the following condition holds. For every descendant edge (e_1, e_2) in $\bar{D}$, which is removed in the pruning of $\bar{D}$ by R, there is an edge (e'_1, e'_2) in $\bar{D}$ such that (e'_1, e'_2) is encapsulated by (e_1, e_2) and (e'_1, e'_2) is also removed in the pruning of $\bar{D}$.*

Consider two edges (e_1, e_2) and (e_1, e'_2) that are outgoing edges of the same node. We say that these two edges are *parallel* in $\bar{D}$ if either there is a path from e_2 to e'_2 or vice-versa. That is, these two edges do not lead to two disjointed parts of $\bar{D}$. We will use this definition in the next proposition to provide a sufficient condition for coherency.

Proposition 3. *Let R be a set of rules and D be a document. If the following condition holds, then R is coherent w.r.t D. For every edge (e_1, e_2) that is removed in the pruning of $\bar{D}$ w.r.t. R, all the edges (e_1, e'_2) that are parallel to (e_1, e_2) are also removed in the pruning of $\bar{D}$.*

6.2 Coherence for Schemas

Given a schema, we can generalize, using containment of XPath expressions, the condition presented in Proposition 3 for coherency. However, testing containment of XPath has high complexity [19]. Hence, the test is inefficient and we do not present it here. Yet, a special case of the generalized condition is that for every document D that conforms to the schema, for each element e in D, either all the outgoing edges of e in $\bar{D}$ are removed in the pruning or none of them is removed.

A simple way to satisfy this condition is to allow only rules that have one of the following two forms: `for` *path* `exclude //*` or `for` *path* `exclude /`*label* `[`*condition*`]//*`, where *path* can be any XPath expression, *label* can be any label and *condition* can be any XPath condition.

Example 15. Consider the schema S in Figure 3. Suppose that we want to conceal in courses all the information on students. We can apply the following two rules. The rule `for //Course exclude /Student` and the rule `for //Course exclude /Student//*`. These rules are coherent w.r.t. the schema S.

7 Effectiveness of Concealment

In this section, we prove the effectiveness of a coherent set of rules in concealing relationships. The presence of a schema and the fact that documents are trees impose limitations on the relationships that we are able to conceal. These limitations will be discussed in the first part of this section.

7.1 The Singleton-Source Disclosure

A *singleton-source disclosure* occurs when a user can infer that two elements e_1 and e_2 are related, from the following two pieces of information. *(1)* The path from the root to e_2 must go through an element of *type* T. *(2)* The only element in the document of type T is e_1. The problem is illustrated by the following two examples.

Example 16. Consider a university document that conforms to the schema in Figure 3 and that contains only a single department element. Consider the rule

```
for //Department exclude /Course
```

which presumably conceals the relationships between departments and courses. A user that is familiar with the schema of the document and knows that the document contains only a single department can infer that every course element in the document is below the only department element.

Example 17. Consider the document in Figure 1 and the rule

```
for //Department[Name='CS'] exclude /Course
```

Suppose that Q_1 is a query that looks for all the courses in the document and Q_2 is a query that looks for all the courses in departments other than "CS". Both queries are locally valid w.r.t. the document and the rule. By applying set difference to the answers to Q_1 and to Q_2, it is possible to infer the set of courses in the "CS" department.

We use k-concealment (Definition 1) to define a singleton-source disclosure.

Definition 6 (Singleton-Source Disclosure). *Consider a document D and a set of rules R. A singleton-source disclosure occurs when there is a rule $\rho = (x_1, x_2)$ in R such that the set of relationships $(x_1(D), x_1x_2(D))$ is not 2-concealed.*

7.2 Verifying k-Concealment for a Coherent Set of Rules

We will describe now an algorithm that given a document D and a coherent set of rules R, tests if a singleton-source disclosure occurs. Essentially, the algorithm computes, for each rule $\rho = (x_1, x_2)$ in R, the maximal k for which the relationship $(x_1(D), x_1x_2(D))$ is k-concealed. If $k > 1$ for all the rules of R, then a singleton-source disclosure does not occur. Otherwise, a singleton-source disclosure does occur. Before presenting the algorithm that computes k, we provide an example that illustrates the algorithm.

Example 18. Consider the university document D_u in Figure 1 and a coherent set of rules R. Suppose R contains a rule ρ that hides the relationships between courses and students. Also, we assume that R does not hide the relationships between departments and students. We now discuss the computation of k for ρ.

There are three students and four courses that we need to consider. For each student s, we need to count the number of courses c for which s might be related to c. The element s might be related to c if, and only if, there exists a document D_{sc} for which the following conditions hold. *(1)* In D_{sc}, the element s is a descendant of the element c. *(2)* For every locally valid query Q, holds $Q(D_{sc}) = Q(D_u)$.

Intuitively, we can think of a document D_{sc} as a result of moving some subtrees of the original document, from one part of the document to another. Thus, for Node 301, we can either leave it below Node 201 or move it to be below Node 202. However, we cannot move Node 301 to be below Node 203. On the conceptual level, this is because Node 203 is a course that belongs to a different department from the department to which Node 301 is related. This is because if we move Node 301 to be below Node 203, we will have two ancestors to Node 301 (Node 101 and Node 102) that are not on the same path. In a tree this should not happen.

In the computation, we check for each student, how many courses it can be related to, as was done for Node 301. In our example, each student has two such courses. Thus, the relationships that ρ defines are 2-concealed.

We present now the algorithm—Compute-k—that for any coherent set of rules R and a document D, computes a maximal value k such that k-concealment can be guaranteed for the relationships that are specified by the rules of R.

Compute k (D, R)
Input: a document D and a coherent set of rules R;
Output: a maximal k such that there is a k-concealment for each rule in R;

Initially, we set $k = \infty$. We iterate over all the rules of R. Given a rule $\rho = (x_1, x_2)$ in R, we denote by A the set $x_1(D)$; and by B the set $x_1x_2(D)$. We iterate over all the elements of B. For each element $b \in B$ we count the number of nodes $a \in A$ such that we can move b to be below a (shortly, we will explain how). Let k_b be this count. Then, if $k_b < k$, we set k to be k_b. At the end of all the iterations, k is returned.

We now explain how we count, for a given $b \in B$, the number of nodes $a \in A$ such that we can move b to be below a. We iterate over the elements of A and try to "attach" b below each one of these elements. The test for b and a is as follows. We start by computing the pruning by R of the transitive closure of D—$prune_R(\bar{D})$. We then try to connect b to a using only edges between restricted pairs, *i.e.*, we add only edges that will be removed in the pruning by R. This produces an expansion D'' of D such that $Q(D'') = Q(D)$ for every query Q that is locally valid w.r.t. R and D.

The following observations are important. First, for every $a' \in A$, in $prune_R(\bar{D})$ there does not exist any edge from a' to b. This is because of the rule ρ. Furthermore, since R is coherent, in $prune_R(\bar{D})$, there is no path from a' to b, and no path from a' to any ancestor or descendant of b. Hence, there is a subtree T_b in $prune_R(\bar{D})$ that contains b and is disconnected (*i.e.*, not reachable by a directed path) from any $a' \in A$. What we need to test is the possibility to connect the root of T_b to a or to a descendant of a, using only edges that are removed in the pruning, such that *(1)* there is no element of A, other than a, on the path from a to the root of T_b, and *(2)* the following two tests are passed. First, a test for making sure that we will eventually produce a tree or a graph that can be extended to be a tree. Secondly, a test for checking that by adding T_b below a, we do not create a relationship (*i.e.*, an ancestor-descendant pair) between two nodes that were not related in D and do not form a restricted pair.

To ensure that we are able to extend the new graph D'' to be a tree, we need to verify that the nodes of T_b do not have two ancestors that are not on one path. To that end, we define X_b to be the nodes x such that x is not in T_b and in D'' there is a descendant edge from x to some node in T_b. For the node a to be an ancestor of b, a must be below all the nodes X_b (a cannot be above any one of the nodes X_b since there is no path from a to any node in T_b; also, a must be on one path with all the nodes X_b). The test succeeds if one of the following cases occurs. *(1)* In D'', a is below all the nodes X_b. *(2)* There is an ancestor y of a that does not have a parent in D'', such that y can be connected to a node below the nodes X_b, using an edge that is removed in the pruning by R, without creating a path between a non-restricted pair.

In the second test, we simply check that for every pair of nodes connected by a path, after moving T_b, either they are connected by an edge in $prune_R(\bar{D})$ or they are a restricted pair according to R. If this test, or the previous test, fails, we do not increase k_b. Otherwise, we increase k_b by one. □

An important advantage of the algorithm Compute-k is that it has a polynomial time complexity. The following theorem shows the correctness of the algorithm Compute-k.

Theorem 1. *Given a document D and a coherent set of rules R, Algorithm Compute-k computes a value k such that the followings hold.*

1. *All the relationships that are defined by rules of R are k-concealed.*
2. *There is a rule in R that defines a relationship which is not $k+1$-concealed.*

Theorem 1 shows that when a coherent set of rules is used, it can be tested for a given document D, whether 2-concealment, or even k-concealment for some $k > 2$, is provided. When k-concealment is provided for D and R, the following holds. Suppose that e_1 and e_2 are two elements such that the association between them should be concealed, *i.e.*, there is a rule in R that specifies the relationship (A, B), where $e_1 \in A$ and

$e_2 \in B$. Then, a user who sees the answers to locally valid queries will not be able to tell with certainty if the two elements e_1 and e_2 are connected in D. This is because 2-concealment guarantees that there are two documents D_1 and D_2 such that in D_1 the two elements e_1 and e_2 are connected, while in D_2, the two elements e_1 and e_2 are not connected. Furthermore, $Q(D_1) = Q(D_2)$, for any locally valid query.

8 Conclusion

We presented an authorization-transparent access-control mechanism for XML under the Non-Truman model. Our mechanism uses rules, which are formulated using XPath expressions, for specifying element relationships that should be concealed. We defined the semantics of rules with respect to a document and with respect to a schema. Coherency of a rule set was defined and discussed. A set of rules is coherent if concealed relationships cannot be inferred from non-concealed relationships. We showed how to construct a coherent set of rules. Finally, we presented the notion of k-concealment, which is a modification of k-anonymity to our model. We showed that when a coherent set of rules is used, k-concealment can be tested efficiently.

Traditionally, access control has been performed using views. Rules can be used either instead of views or in addition to views. There are cases where rules can be written concisely while using view is cumbersome. For example, when only a small fraction of the data should be concealed from users. Yet, when most of the data should be concealed, defining the policy using views might be easier than using rules. Rules, however, have the advantage that when the set is coherent, we can guarantee that concealed relationships cannot be inferred from the results of valid queries. Importantly, our solutions can be used to verify not only queries but also views.

Future work includes showing how to integrate our access-control rules with existing XPath query processors. Another important challenge is to adapt our mechanism to XQuery and XSLT.

Acknowledgments

We thank the Natural Sciences and Eng. Research Council of Canada for their support, and the anonymous reviewers for their careful comments.

References

1. R. J. Bayardo and R. Agrawal. Data privacy through optimal k-anonymization. In *Proc. of the 21st ICDE*, pages 217–228, 2005.
2. E. Bertino, S. Castano, and E. Ferrari. On specifying security policies for web documents with an XML-based language. In *Proc. of the 6th SACMAT*, pages 57–65, 2001.
3. E. Bertino and E. Ferrari. Secure and selective dissemination of XML documents. *ACM TISSEC*, 5(3):290–331, 2002.
4. L. Bouganim, F. Dang-Ngoc, and P. Pucheral. Client-based access control management for XML documents. In *Proc. of the 30th VLDB*, pages 84–95, 2004.

5. D. Chamberlin, J. Clark, D. Florescu, J. Robie, J. Siméon, and M. Stefanescu. XQuery 1.0, June 2001. W3C standard. Available at `http://www.w3.org/TR/xquery`.
6. SungRan Cho, S. Amer-Yahia, L. V.S. Lakshmanan, and D. Srivastava. Optimizing the secure evaluation of twig queries. In *Proc. of the 28th VLDB*, pages 490–501, 2002.
7. J. Clark. XSLT 1.0. W3C standard. Available at `http://www.w3.org/TR/xslt`, 1999.
8. J. Clark and S. DeRose. XPath 1.0. Available at `http://www.w3.org/TR/xpath`.
9. E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A fine-grained access control system for XML documents. *ACM TISSEC*, 5(3):169–202, 2002.
10. E. Damiani, S. Samarati, S. di Vimercati, and S. Paraboschi. Controlling access to XML documents. *IEEE Internet Computing*, 5(6):18–28, 2001.
11. W. Fan, C. Chan, and M. Garofalakis. Secure XML querying with security views. In *Proc. of the 23rd ACM SIGMOD*, pages 587–598, 2004.
12. B. Finance, S. Medjdoub, and P. Pucheral. The Case for access control on XML relationships. In *Proc. of the 14th CIKM*, pages 107–114, 2005.
13. I. Fundulaki and M. Marx. Specifying access control policies for XML documents with XPath. In *Proc. of the 9th ACM SACMAT*, pages 61–69, 2004.
14. A. Gabillon and E. Bruno. Regulating access to XML documents. In *Proc. of the 15th IFIP WG11.3*, pages 299–314, 2001.
15. S. Godik and T. Moses. eXtesible Access Control Markup Language (XACML) Version 1.0. Available at `http://www.oasis-open.org/committees/xacml`, 2003.
16. S. Hada and M. Kudo. XML Access Control Language: provisional authorization for XML documents. Available at `http://www.trl.ibm.com/projects/xml/xacl`.
17. A. Meyerson and R. Williams. On the complexity of optimal k-anonymity. In *Proc. of the 23rd PODS*, pages 223–228, 2004.
18. G. Miklau and D. Suciu. Controlling access to published data using cryptography. In *Proc. of the 29th VLDB*, pages 898–909, 2003.
19. G. Miklau and D. Suciu. Containment and equivalence for a fragment of XPath. *Journal of the ACM*, 51(1):2–45, 2004.
20. G. Miklau and D. Suciu. A formal analysis of information disclosure in data exchange. In *Proc. of the 23rd ACM SIGMOD*, pages 575–586, 2004.
21. A. Motro. An access authorization model for relational databases based on algebric manipulation of view definitions. In *Proc. of the 5th ICDE*, pages 339–347, 1989.
22. S. Rizvi, A. O. Mendelzon, S. Sudarshan, and P. Roy. Extending query rewriting techniques for fine-grained access control. In *Proc. of the 23rd ACM SIGMOD*, pages 551–562, 2004.
23. A. Rosenthal and E. Scoire. View security as the basis for data warehouse security. In *Proc. of the 2nd DMDW*, Stockholm (Sweden), 2000.
24. A. Rosenthal and E. Scoire. Administering permissions for distributed data:factoring and automated inference. In *Proc. of the 15th IFIP WG11.3*, pages 91–104, 2001.
25. L. Sweeney. k-anonymity: a model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 10(5):557–570, 2002.
26. XML. W3C standard. Available at `http://www.w3c.org/XML`.
27. XML Schema. W3C standard. Available at `http://www.w3c.org/XML/Schema`.
28. Wanhong Xu and Z. M. Özsoyoglu. Rewriting xpath queries using materialized views. In *Proc. of the 31st VLDB*, pages 121–132, 2005.
29. C. Yao, X. S. Wang, and S. Jajodia. Checking for k-anonymity violation by views. In *Proc. of the 31st VLDB*, pages 910–921, 2005.
30. T. Yu, D. Srivastava, L. V.S. Lakshmanan, and H. V. Jagadish. Compressed accessibility map: efficient access control for XML. In *Proc. of the 28th VLDB*, pages 363–402, 2002.