INTER-NODE RELATIONSHIP LABELING: A FINE-GRAINED XML ACCESS CONTROL IMPLEMENTATION USING GENERIC SECURITY LABELS

Zheng Zhang
University of Toronto
Toronto, Ontario, Canada

Walid Rjaibi
IBM Toronto Software Laboratory
Markham, Ontario, Canada

Keywords: Authorization-transparent, fine-grained access control, label-based access control, XML relationship labeling.

Abstract: Most work on XML access control considers XML nodes as the smallest protection unit. This paper shows the limitation of this approach and introduces an XML access control mechanism that protects inter-node relationships. Our approach provides a finer granularity of access control than the node-based approaches(*i.e.*, more expressive). Moreover, our approach helps achieve the "need-to-know" security principle and the "choice" privacy principle. This paper also shows how our approach can be implemented using a generic label infrastructure and suggests algorithms to create/check a secure set of labeled relationships in an XML document.

1 INTRODUCTION

XML has rapidly emerged as the standard for the representation and interchange of business and other sensitive data on the Web. The current trend of adding XML support to database systems poses new security challenges for an environment in which both relational and XML data coexist. In particular, fine-grained access control is even more necessary for XML than for relational data, given the more flexible and less homogeneous structure of XML data compared to relational tables and rows. The additional difficulty of controlling access over XML data compared to relational data can be summarized as follows.

- The semi-structured nature of XML data, where a schema may be absent, or, even if it is present, may allow much more flexibility and variability in the structure of the document than what is allowed by a relational schema.
- The hierarchical structure of XML, which requires specifying, for example, how access privileges to a certain node propagate from/to the node's ancestors and descendants.

In almost all of the work on XML access control (Bertino and Ferrari, 2002; Damiani et al., 2002; Fan et al., 2004), the smallest unit of protection is the XML node of an XML document, which are specified by XPath fragments. Access to ancestor-descendant and sibling relationships among nodes has not been considered. An access control policy consists of positive (resp. negative) authorization rules that grant (resp. deny) access to some nodes of an XML document. The main difference between XML access control models lies in privilege propagation. Some (Bertino and Ferrari, 2002; Gabillon and Bruno, 2001) forbid access to the complete subtree rooted at an inaccessible node. Alternatively, if a node is granted access while one of its ancestor nodes is inaccessible, the ancestor node would be masked as an empty node in the XML document (Damiani et al., 2002). However, this makes visible the literal of the forbidden ancestor in the path from the root to that authorized node. This can be improved by replacing the ancestor node literal by a dummy value (Fan et al., 2004). However, this still does not solve the problem that different descendant nodes may require their ancestor's literal to be visible or invisible differently. From the differences among the above models, it is clear that defining a view that precisely describes the path leading to an authorized node is difficult. The question that begs to be asked is therefore the following: Is a node the most fine-grained entity within an XML document upon which a fine-grained access control model for XML is to be built?

We believe that the answer to this question is an unequivocal NO. We contend that the path between nodes is a better alternative upon which a fine-grained access control model for XML is to be built (Kanza et al., 2006). In other words, we con-

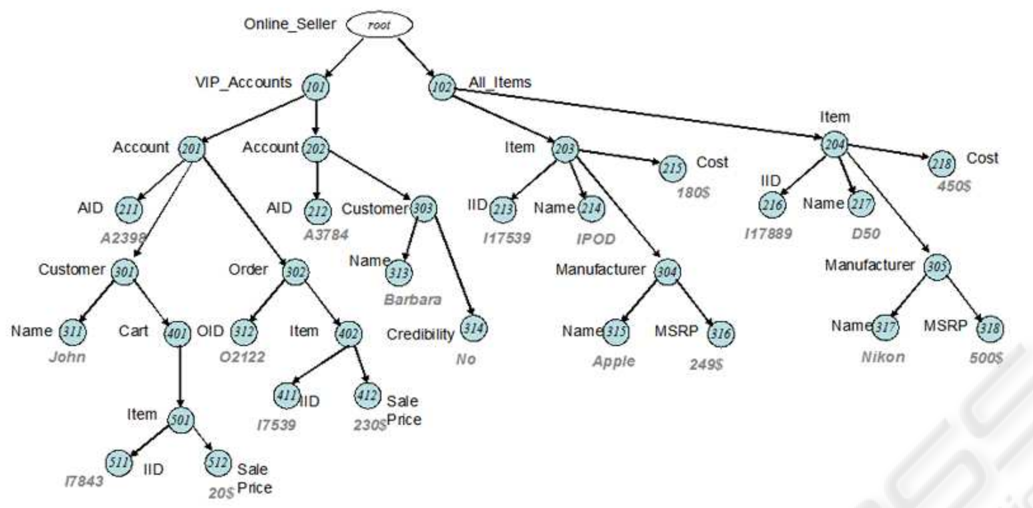

Figure 1: A document that contains information on accounts, orders and items for an online seller..

tend that ancestor-descendent relationships and sibling relationships should be considered as legitimate elements to be protected. The main advantages of our approach are as follows.

First of all, blocking access to a node can be addressed by blocking access to all the relationships relating to the node. For example, in Figure 1, if we want to block all access to the Account Node “202”, we could simply block access to all the paths from that node’s ancestors to the node and all the paths from the node to its sibling and descendants.

Second, blocking access to relationships helps achieve the “need-to-know” principle, one of the most fundamental security principles. This principle requires information to be accessed by only those who strictly need the information to carry out their assignments. The practice of “need-to-know” limits the damage that can be done by a trusted insider who betrays our trust. The hierarchical structure of an XML document often reveals classification information. For example, in Figure 1, the root of the left subtree of the document represents a special account type “VIP_Accounts”. Knowing an account node, say Node “201”, belongs to that subtree reveals the account type. If the smallest protection unit is a node, once we let the root of the subtree accessible, we may leak unnecessary information. For example, suppose that the relationship between the Account Node “202” and the account type “VIP_Accounts” at the root of the subtree should be protected, knowing the account type of Node “201” in the subtree reveals the account type of Node “202”. With relationship protection, we identify that the ancestor-descendant relationship between Node “101” and Node “202”, and the sibling relationship between Node “201” and Node “202” should be protected while we let the ancestor-descendant relationship between Node “101” and Node “201” be accessible.

Third, blocking access to relationships helps achieve the “choice” principle, one of the most fundamental privacy principles. At its simplest, the principle means giving clients options as to how any personal information collected from them may be used. If the smallest protection element is a node, access control over one node is propagated to its ancestor/descendant nodes (Murata et al., 2003), *i.e.*, whenever access is denied to a node, access is denied to its descendants; whenever access is granted to a node, access is granted to all its ancestors. Hence, negative access control policies over ancestor nodes give a common authorized view of the paths leading to their descendants. This violates the “choice” principle: in Figure 1, a client may want to hide the account type but not the other account information for the account with AID “A2398”. If the smallest protection element is a relationship between nodes in an XML document, we could protect the relationships between Node “101” and the nodes in the subtree rooted at Node “201”, and the sibling relationship between Node “201” and Node “202”. Then all the account information except the account type is still accessible from the root of the document tree. Moreover, there is no way to know that the subtree rooted at Node “201” is a subtree of Node “101”.

Last but not least, protecting relationships between nodes in an XML document is more expressible in terms of access control policy translation.

Contributions: The contributions made in this paper can be summarized as follows:

1. We propose an authorization-transparent fine-grained access control model that protects the ancestor-descendant and sibling relationships in an XML document. Our model distinguishes two levels of access to relationships, namely the *existence access* and the *value access*.

2. We propose a new semantics for concealing relationships in an XML document where a relationship is defined by a path in the document.
3. We propose a generic and flexible label-based access control mechanism to protect relationships. Our mechanism allows DBAs to define label-based access control policies.
4. We propose a new query evaluation mechanism to enforce our access control model.
5. We develop algorithms to check/create a secure set of labeled relationships of an XML document.

2 RELATED WORK

XML access control has been studied on issues such as granularity of access, access-control inheritance, default semantics, overriding, and conflict resolutions (Bertino and Ferrari, 2002; Damiani et al., 2002; Gabillon and Bruno, 2001; Murata et al., 2003). In particular, a useful survey of these proposals is given in (Fundulaki and Marx, 2004), which uses XPath to give formal semantics to a number of different models in a uniform way, making it possible to compare and contrast them. Almost all the recent models (Bertino and Ferrari, 2002; Damiani et al., 2002; Gabillon and Bruno, 2001) propose to restrict the user's view of a document by access control policies. In particular, authors in (Damiani et al., 2002; Gabillon and Bruno, 2001) mark each node as "accessible" or "inaccessible" in an XML document and apply conflict resolution policies to compute an authorized pruned view of the document. An alternative approach (Miklau and Suciu, 2003) defines access control policies as XQuery expressions. A user is given a modified document with encrypted data and queries are posed on this modified document. They present a new query semantics that permits a user to see only authorized data. In (Fan et al., 2004), security is specified by extending the document DTD with annotations and publishing a modified DTD. Similarly, work by Bertino *et al.* (Bertino et al., 2001) and Finance *et al.* (Finance et al., 2005) provides XML-based specification languages for publishing secure XML document content, and for specifying role-based access control on XML data (Bhatti et al., 2004; Wang and Osborn, 2004). Restricting access to nodes has also been used in XACL (IBM, 2001) and XACML (Oasis., 2005), two proposed industrial standards. Kanza *et al.* propose to restrict access to ancestor-descendant relationships (Kanza et al., 2006) and introduce authorization-transparent access control for XML data under the Non-Truman model (Rizvi et al., 2004).

3 DATA MODEL AND QUERIES

We consider an XML document as a rooted directed tree over a finite set of node literals L with a finite set of values A attached to atomic nodes (*i.e.*, nodes with no outgoing edges). Formally, a document D is a 5-tuple $(N_D, E_D, root_D, literal\_of_D, value\_of_D)$, where N_D is a set of nodes, E_D is a set of directed edges, $root_D$ is the root of a directed tree, $literal\_of_D$ is a function that maps each node of N_D to a literal of L, and $value\_of_D$ is a function that maps each atomic node to a value of A. In order to simplify the data model, we do not distinguish between elements and attributes of an XML document. We also assume that all the values on atomic nodes are of type *PCDATA* (*i.e.*, *String*).

Example 3.1 Figure 1 shows a document that contains information on accounts, orders, and items for an online seller. Nodes are represented by circles with ID's for easy reference. Values in A appear below the atomic nodes and are written in bold font.

In this paper, we use XPath (Clark and DeRose, 1999) for formulating queries and specifying relationships. XPath is a simple language for navigation in an XML document. In XPath, there are thirteen types of axes that are used for navigation. Our focus is on the *child* axis (`/`), the *descendant-or-self* axis (`//`), the *preceding-sibling* axis and the *following-sibling* 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 RELATIONSHIP ACCESS

First, we consider what it means to conceal a relationship. In general, a *relationship* is an undirected path between two nodes in an XML document. A set of *relationships* is represented by two sets of nodes. For example, the pair (C, N), where C is the set of all Customer Nodes and N is the set of all Name Nodes in Figure 1, represents the set of relationships between customers and their names. Concealing the relationships (C, N) means that for every customer c and name n in the document, a user will not be able to infer (with certainty), from any query answers, whether n is the name for c. We want this to be true for all authorized query results. Note that we are concealing the presence or absence of relationships, so we are concealing whether any of the set of pairs in (C, N) exists in the document.

Definition 4.1 (Concealing a Relationship) *Given an XML document D and a pair of nodes n_1 and n_2 in D, the relationship (n_1, n_2) is concealed if there* *exists a document D' over the node set of D, such that the following is true.*

1. *Exactly one of D and D' has a path from n_1 to n_2.*
2. *For any XPath query Q, the authorized answer set of Q over D is equal to that of Q over D'.*

We consider two kinds of relationships in an XML document, namely the ancestor-descendant relationships and the sibling relationships. Kanza *et al.* consider ancestor-descendant relationships only (Kanza et al., 2006). Sibling relationships are inferred by the ancestor-descendant relationships. Hence, when access to an ancestor-descendant relationship is blocked in their model, access to the related sibling relationships is automatically blocked.

Example 4.2 In Figure 1, suppose the relationship between VIP_Accounts Node “101” and Account Node “201” is inaccessible, then the sibling relationship between Node “201” and Node “202” is lost.

It could be necessary to preserve such sibling relationship information. For example, one policy may want to block access to the ancestor-descendant relationships between VIP_Accounts Node and Account Nodes while maintain access to the sibling relationships between the Account Nodes.

On the other hand, it might be desirable to block access to sibling relationships only. For example, one policy may want to block access to the sibling relationship between Customer and his Order.

In order to express such access control policies, we consider sibling relationships as well as ancestor-descendant relationships.

We distinguish two levels of access to relationships, namely the *existence access* and the *value access*. In value access, information about a relationship indicates a node whose ID is “v_a” and whose literal is “A” is related to a node whose ID is “v_b” and whose literal is “B”. For example, the pair (C, N) is a value access to the relationships between Customer Nodes and Name Nodes. In existence access, information about a relationship is basically the same as information of value access but lacks at least one of the values “v_a” and “v_b”. In other words, existence access to a relationship returns whether a node of some literal is related to some node. For example, existence access could indicate a node whose literal is “A” is related to a node whose literal is “B”. Obviously, if a relationship is not accessible under existence access, then the relationship is not accessible under value access.

Example 4.3 Consider the relationship between the account with AID “A2398” and its customer name in Figure 1. The value access to this relationship returns that Node “201” whose literal is “Account” is related to Node “311” whose literal is “Name” and whose value is “John”. The typical queries that will return this information are:

Q_1: *//Account[AID=“A2398”],*
Q_2: *//Account[AID=“A2398”]/Customer/Name.*

Now consider an existence access to this relationship: a query Q_3 wants to return all the accounts’ AID’s that have a customer name. The fact that “A2398” is returned tells us that there exists a customer with name under the account with AID “A2398”, but it does not tell us what the customer’s name is, nor the Node ID “311”. In other words, Q_1 and Q_3 reveal that Node “201” whose literal is “Account” is related to some node n whose literal is “Name”, where n is a child of some node whose literal is “Customer” and which is a child of Node “201”.

Q_3: *//Account[Customer/Name]/AID.*

In the next section, we show how to specify ancestor-descendant and sibling relationships and attach access labels to them.

5 ACCESS CONTROL POLICY SPECIFICATION

Our access control model uses a generic, flexible label infrastructure (Rjaibi and Bird, 2004) where a label has only one component “access level”. The value of the component can be “EXISTENCE”, “VALUE”, or “NULL”. The ranks of these values are as follows: “EXISTENCE” > “VALUE” > “NULL”. We distinguish two types of labels: *Access labels* and *Path labels*. Access labels are created and assigned to database users, roles, or groups along with the type of access for which the access label is granted (*i.e.*, Read/Write). For simplicity, we consider only users in this paper. We call *read (resp. write) access label* an access label associated with the Read (resp. Write) access type. Path labels are created and attached to paths of an XML document. When a user or a path is not associated with a label, the “NULL” label is assumed for that user or path.

Example 5.1 The following statement creates and grants the “EXISTENCE” access label to a database user Mike for the Read access type.

```
GRANT ACCESS LABEL EXISTENCE
TO USER Mike FOR READ ACCESS
```

The following statement revokes the “EXISTENCE” read access label from Mike.

```
REVOKE ACCESS LABEL EXISTENCE
FROM USER Mike FOR READ ACCESS
```

Access to an XML document is based upon the labels associated with the paths of the XML document and the label associated with the user accessing the document via the paths. A label access policy consists of label access rules that the database system evaluates to determine whether a database user is allowed access to an XML document. Access rules can be categorized as Read Access rules and Write Access rules. The former is applied by the database system when a user attempts to read a path in an XML document; the latter is applied when a user attempts to insert, update or delete a path in an XML document. In both cases, a label access rule is as follows:

Access Label $\langle$operator$\rangle$ Path Label

where the operator is one of the arithmetic comparison operators $\{=, \leq, <, >, \geq, \neq\}$.

Example 5.2 The following statement creates a label access policy that *(1)* does not allow a user to read a path unless his read access label is larger than or equal to the path label, *(2)* does not allow a user to write a path unless his write access label is equal to the path label.

```
CREATE LABEL POLICY XML-FGAC
READ ACCESS RULE rule
  READ ACCESS LABEL ≥ Path LABEL
WRITE ACCESS RULE rule
  WRITE ACCESS LABEL = Path LABEL
```

Recall value access to a relationship returns more information than existence access. An "EXISTENCE" label protects existence and value access. A "VALUE" label protects value access only. Therefore, if a user with a "NULL" read access label wants to existence access a path with a "VALUE" path label, access should be allowed since this existence access does not return the complete relationship information from value access. We call this the DEFAULT policy. This policy only applies to Read Access since any Write Access involves real node ID's (*i.e.*, existence access is impossible). This policy could coexist with other policies such as XML-FGAC to give a more complete authorized answer set of a query.

Example 5.3 Assume the relationship in Example 4.3 has a "VALUE" path label. If a user with a "NULL" read access label asks query Q_3, the existence access to the relationship should be allowed.

Next, we introduce how the labels are attached to paths in an XML document. First, attaching a label to ancestor-descendant paths are specified by an SQL statement in the following form:

ATTACH *path_label* ANCS *path*$_1$ DESC *path*$_2$,

where *path*$_1$ and *path*$_2$ are two XPath expressions. Notice expression *path*$_2$ is a relative XPath expression w.r.t. *path*$_1$. The two expressions specify pairs of ancestor nodes (*i.e.*, *path*$_1$) and descendent nodes (*i.e.*, *path*$_1$/*path*$_2$). Expression *path_label* is a label.

Example 5.4 The following expression attaches "EXISTENCE" path labels to the relationships between Account Nodes and their Customers' Name Nodes in Figure 1.

ATTACH EXISTENCE ANCS *//Account*
DESC */Customer/Name*

The following expression attaches a "VALUE" path label to the relationship between the Item Node with Name "IPOD" and its Cost Node in Figure 1.

ATTACH VALUE ANCS *//Item[Name = "IPOD"]*
DESC *//Cost*

For sibling relationships, we consider the *preceding-sibling axis* and *the following-sibling axis* in XPath. Thus, attaching a label to sibling paths are specified by XPath expressions in the following form:

ATTACH *path_label*
NODE *path*$_1$ PRECEDING-SIBLING *path*$_2$
FOLLOWING-SIBLING *path*$_3$,

where *path*$_1$, *path*$_2$ and *path*$_3$ are three XPath expressions. Notice expressions *path*$_2$ and *path*$_3$ are two relative XPath expressions w.r.t. *path*$_1$. The expressions specify relationships between some nodes (*i.e.*, *path*$_1$), and their preceding siblings (*i.e.*, *path*$_1$/*preceding-sibling* :: *path*$_2$) as well as the relationships between the nodes and their following siblings (*i.e.*, *path*$_1$/*following-sibling* :: *path*$_3$). Notice the *PRECEDING-SIBLING* expression and the *FOLLOWING-SIBLING* expression do not have to appear at the same time.

Example 5.5 The following expression attaches a "VALUE" path label to the relationship between the Account whose Customer has Name "Barbara" and its preceding sibling.

ATTACH VALUE
NODE *//Account[Customer/Name = "Barbara"]*
PRECEDING-SIBLING *Account*

Note that the SQL statement to detach a label from an ancestor-descendant path or a sibling path is similar to the SQL statement to attach a label to those paths except that ATTACH is replaced by DETACH.

6 QUERY EVALUATION

In authorization-transparent access control, users formulate their queries against the original database rather than against authorization views that transform and hide data (Motro, 1989). In (Rizvi et al., 2004), authorization transparent access control is categorized into two basic classes, the *Truman model* and the *Non-Truman model*. In the Truman model, an access control language (often a view language) is used to specify what data is accessible to a user. User queries are modified by the system so that the answer includes only accessible data. Let Q be a user query, D be a database and D_u be the part of D that the user is permitted to see, then query Q is modified to a safe query Q_s such that $Q_s(D) = Q(D_u)$. We call $Q_s(D)$ the *authorized answer set* of Q over D. In contrast, in the Non-Truman model, a query that violates access control specifications is rejected, rather than modified. Only *valid* queries are answered.

Our model is an authorization-transparent Truman model. We allow users to pose XPath queries against the original labeled XML document. The evaluation of an XPath query over a labeled XML document has two parts. First, we change the usual XPath query semantics as follows. If a child axis occurs, the evaluation follows a parent-child path; if a descendant-or-self axis occurs, the evaluation follows an ancestor-descendant path; if a preceding-sibling axis occurs, the evaluation follows a preceding-sibling path; if a following-sibling axis occurs, the evaluation follows a following-sibling path.

Second, we need to make sure that for each path accessed, a user is allowed access to that path based on the path label and the user's access label. Suppose a path P has a path label L_1 and a user Mike has a read access label L_2. According to the XML-FGAC policy, *(1)* if L_2 is "EXISTENCE", Mike could read the path P regardless of the value of label L_1; *(2)* if L_2 is "VALUE", Mike could read the path P if L_1 is not "EXISTENCE"; *(3)* if L_2 is "NULL", Mike can only access paths with "NULL" labels; if the DEFAULT policy coexists, Mike could ask queries to existence access the path P if L_1 is "VALUE". The discussion for Write Access is similar. The above logic is inserted into the query access plan. When the access plan is executed, the access rules from the label access policy associated with the labeled XML document are evaluated for each path accessed in the document. This approach allows the cached access plan to be reused because the access labels of the user who issued the query are acquired during runtime.

For an XML document, there is an ordering, *document order* (Clark and DeRose, 1999), defined on all the nodes in the document corresponding to the order in which the first character of the XML representation of each node occurs in the XML representation of the document. This ordering information may leak information as shown in the following example.

Example 6.1 Let us look at Figure 1 again. Suppose one security policy wants to block public access to the sibling relationships between the Customer Nodes and their Order Nodes. Suppose the following queries are allowed to return their answers in document order: *//Customer* and *//Order*. Then the order of Customer output might match the order of Order output, hence leaks secret information. The situation becomes worse if the document has a registered schema and the schema shows publicly that each customer has a fixed number, say 2, of orders. In this case, the association between a Customer and his Orders is completely leaked.

To prevent an information leak based on document order, we shuffle the output as follows. Each node in the output will receive a random number. And the nodes will be output based on the order of their assigned random numbers.

In sum, the processing algorithm to be inserted in the access plan for a labeled XML document with XML-FGAC and DEFAULT policies is as follows.

Algorithm: Insert Read and Write Access logic into a query access plan for a labeled XML document.

1. Fetch the user's Access Labels for Read and Write actions (*e.g.*, from a system catalog table).
2. For all paths accessed, do the following.
   (a) If it is a Read Access and READ Access rules do not permit access, skip the path unless *(1)* the Read Access Label is "NULL", *(2)* the Path Label is "VALUE", and *(3)* it is an existence access.
   (b) If it is a Write Access and Write Access rules do not permit access, skip the path.
3. Shuffle output.

Example 6.2 Suppose the document in Figure 1 has two labels attached to its paths as specified in Example 5.4 and the label access policies are XML-FGAC and DEFAULT. Suppose a database user Mike with a read access label "EXISTENCE" asks the query Q_1: *//Account[Customer/Name]*. The query access plan checks the following paths:

1. the paths P_1 from the root of the document to Account Nodes, *i.e.*, *//Account*,
2. the paths P_2 from Account Nodes to their descendant Name Nodes via Customer Nodes, *i.e.*, ANCS *//Account* DESC */Customer/Name*,
3. the paths P_3 from Customer Nodes to their children Name Nodes, *i.e.*, *Customer/Name*.

Paths P_1 and P_3 have "NULL" labels, hence, access is allowed. Paths P_2 have "EXISTENCE" labels. Mike could read them since his read access label is "EXISTENCE". Read access to P_2 is denied for any other labels and the authorized answer set is empty.

Next, suppose another user John with a read access label "VALUE" asks the query Q_2: *//Item//Cost*. The query access plan checks the following paths:

1. the paths P_1 from the root of the document to the Item Nodes, *i.e.*, *//Item*,
2. the paths P_2 from the Item Nodes to their descendant Cost Nodes, *i.e.*, ANCS *//Item* DESC *//Cost*.

Paths P_1 have "NULL" labels, hence, access is allowed. For P_2, one path P_{21} has a "NULL" label; the other path P_{22} has a "VALUE" label as it is ANCS *//Item[Name="IPOD"]* DESC *//Cost*. John could read P_2 if his read access label is "VALUE". John could read P_{21} but not P_{22} if his read access label is "NULL". Hence, the authorized answer set is "450\$". However, even if John's read access label is

"NULL", the following query from John will still return the complete answer to Q_3: *//Item[Cost]*. This is because Q_3 only existence accesses the paths P_2, *i.e.*, the authorized answer set only indicates there exist Cost children Nodes for the Item Nodes "203" and "204", but no information about the values and node ID's of the Cost Nodes is leaked.

7 CREATE A SECURE SET OF LABELED RELATIONSHIPS

Our goal is to allow users to label node relationships and let them be sure that what they want to conceal is truly concealed from the users whose access labels do not satisfy the label access policy with the path labels. 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.

Let us see an example of four cases where a relationship could be inferred from a pair of non-concealed relationship.

Example 7.1 In Figure 1, suppose it is known that Account Node "201" is a descendant of VIP_Accounts Node "101" and Customer Node "301" is a descendant of Account Node "201". Then, there is no point to conceal the ancestor-descendant relationship between VIP_Accounts Node "101" and Customer Node "301".

Suppose it is known that Customer Node "301" is a descendant of VIP_Accounts Node "101" as well as Account Node "201". Since there is only one path from the root of the document to Account Node "201", there is no point to conceal the ancestor-descendant relationship between VIP_Accounts Node "101" and Account Node "201".

Suppose it is known that Account Node "201" and Account Node "202" are the children of VIP_Accounts Node "101", then there is no point to conceal the sibling relationship between Account Node "201" and Account Node "202".

Suppose it is known that VIP_Accounts Node "101" has a descendant Customer Node "301" and the customer has a sibling Order Node "302", then there is no point to conceal the ancestor-descendant relationship between VIP_Accounts Node "101" and Order Node "302".

We say a set of labeled relationships/paths in an XML document D is *not secure* w.r.t. a path label L if one of the following four cases happens.

1. Case 1: D has three nodes, n_1, n_2 and n_3 s.t. the ancestor-descendant path from n_1 to n_2 and the ancestor-descendant path from n_2 to n_3 have labels $L_{12} < L$ and $L_{23} < L$. The ancestor-descendant path from n_1 to n_3 has a label $L_{13} \geq L$.
2. Case 2: D has three nodes, n_1, n_2 and n_3 s.t. the ancestor-descendant path from n_1 to n_3 and the ancestor-descendant path from n_2 to n_3 have labels $L_{13} < L$ and $L_{23} < L$. The ancestor-descendant path from n_1 to n_2 has a label $L_{12} \geq L$.
3. Case 3: D has three nodes, n_1, n_2 and n_3 s.t. n_1 is the parent of n_2 and n_3, the parent-child path from n_1 to n_2 and the parent-child path from n_1 to n_3 have labels $L_{12} < L$ and $L_{13} < L$. The sibling path from n_2 to n_3 has a label $L_{23} \geq L$ or the sibling path from n_3 to n_2 has a label $L_{32} \geq L$.
4. Case 4: D has three nodes, n_1, n_2 and n_3 s.t. the ancestor-descendant path from n_1 to n_2 has a label $L_{12} < L$, and either the sibling path from n_2 to n_3 has a label $L_{23} < L$ or the sibling path from n_3 to n_2 has a label $L_{32} < L$. The ancestor-descendant path from n_1 to n_3 has a label $L_{13} \geq L$.

There is a simple test to verify that a set of labeled relationships/paths in an XML document D is not secure w.r.t. a path label L. The test starts by computing three ternary relations R_1, R_2 and R_3. The first two columns store the start/end nodes of paths. The third column stores the label associated with paths (if a label is missing, then it is a NULL value). In particular, R_1 stores all ancestor-descendant paths in D, R_2 stores all parent-child paths in D, and R_3 stores all sibling paths in D.

1. Case 1 is true for a path label L iff the expression $\pi_{\$1,\$5}(R_{1,L} \bowtie_{\$2=\$1} R_{1,L}) - R_{1,L}$ is not empty where $R_{1,L}$ is $\sigma_{\$3<L}(R_1)$.
2. Case 2 is true for a path label L iff the expression $\pi_{\$1,\$4}(R_{1,L} \bowtie_{\$2=\$2} R_{1,L}) - R_{1,L}$ is not empty where $R_{1,L}$ is $\sigma_{\$3<L}(R_1)$.
3. Case 3 is true for a path label L iff the expression $\pi_{\$2,\$5}(R_{2,L} \bowtie_{\$1=\$1} R_{2,L}) - R_{3,L}$ is not empty where $R_{2,L}$ is $\sigma_{\$3<L}(R_2)$ and $R_{3,L}$ is $\sigma_{\$3<L}(R_3)$.
4. Case 4 is true for a path label L iff the expression $\pi_{\$1,\$5}(R_{1,L} \bowtie_{\$2=\$1} R_{3,L}) - R_{1,L}$ is not empty where $R_{1,L}$ is $\sigma_{\$3<L}(R_1)$ and $R_{3,L}$ is $\sigma_{\$3<L}(R_3)$.

Furthermore, we give intuitive conditions to construct a secure set of labeled relationships for an XML document. If we ignore the directions of ancestor-descendant and sibling paths, all these paths form cycles in an XML document. To assign a path label L to a relationship between two nodes n_1 and n_2 in an XML document D, we must make sure, for every cycle that includes the path from n_1 to n_2, either there is another path whose label $L' \geq L$, or n_1 and n_2 are descendants of some nodes in the cycle and n_1, n_2 are not children of the same parent. Both cases ensure there is uncertainty whether a relationship between two nodes n_1 and n_2 exists: the first case by having another path missing in the cycle, while in the second case, the fact that n_1 and n_2 are descendants of some nodes in the cycle introduces uncertainty except when they are children of the same parent, in which case the sibling relationship between n_1 and n_2 is leaked.

There is another possible information leak due to *singleton-source disclosure* (Kanza et al., 2006). In short, a user can infer that two nodes n_1 and n_2 are related in a document D when *(1)* the path from the root of document D to node n_2 must go through a node whose literal is A, *(2)* the only node with literal A in document D is node n_1. An algorithm to test singleton-source disclosure has been proposed in (Kanza et al., 2006) and we will not repeat it here.

8 CONCLUSION

This paper has introduced a fine-grained access control model for XML data using generic security labels. Our model is based on inter-node relationship labeling and provides finer-grained access control than traditional node labeling approaches, hence helps achieve the “need-to-know” security principle and the “choice” privacy principle. We propose a new semantics for concealing relationships in an XML document under the Truman model. To enforce our model, we provide a new query evaluation algorithm and suggest algorithms to check/create a set of secure labeled paths for an XML document.

Our future work includes implementing our model and validating its effectiveness and performance using real-life XML access control user cases. An important challenge is adapting our mechanism to XQuery, general XML document graphs and XML schemas.

ACKNOWLEDGEMENTS

We thank NSERC and IBM Toronto CAS for their support, and Renée J. Miller for her careful comments.
Trademark: IBM is a trademark or registered trademark of International Business Machines Corporation in the United States, other countries, or both.
Disclaimer: The views expressed in this paper are those of the authors and not necessarily of IBM Canada Ltd. or IBM Corporation.

REFERENCES

Bertino, E., Castano, S., and Ferrari, E. (2001). On specifying security policies for web documents with an xml-based language. In *SACMAT*, pages 57–65.

Bertino, E. and Ferrari, E. (2002). Secure and selective dissemination of xml documents. *ACM Trans. Inf. Syst. Secur.*, 5(3):290–331.

Bhatti, R., Bertino, E., Ghafoor, A., and Joshi, J. (2004). Xml-based specification for web services document security. In *IEEE Computer*, volume 4 of *37*, pages 41–49.

Clark, J. and DeRose, S. (1999). XML Path Language (XPath) version 1.0. Available at `http://www.w3.org/TR/xpath`.

Damiani, E., de C. di Vimercati, S., Paraboschi, S., and Samarati, P. (2002). A fine-grained access control system for xml documents. *ACM Trans. Inf. Syst. Secur.*, 5(2):169–202.

Fan, W. F., Chan, C. Y., and Garofalakis, M. N. (2004). Secure xml querying with security views. In *SIGMOD*, pages 587–598.

Finance, B., Medjdoub, S., and Pucheral, P. (2005). The case for access control on xml relationships. Technical report, INRIA. Available from http://www-smis.inria.fr/dataFiles/FMP05a.pdf.

Fundulaki, I. and Marx, M. (2004). Specifying access control policies for xml documents with xpath. In *SACMAT*, pages 61–69.

Gabillon, A. and Bruno, E. (2001). Regulating access to xml documents. In *Working Conference on Database and Application Security*, pages 311–328.

IBM (2001). Xml access control. *http://xml.coverpages.org /xacl.html*.

Kanza, Y., Mendelzon, A., Miller, R., and Zhang, Z. (2006). Authorization-transparent access control for xml under the non-truman model. In *EDBT*, pages 222–239.

Miklau, G. and Suciu, D. (2003). Controlling access to published data using cryptography. In *VLDB*, pages 898–909.

Motro, A. (1989). An access authorization model for relational databases based on algebraic manipulation of view definitions. In *ICDE*, pages 339–347.

Murata, M., Tozawa, A., Kudo, M., and Hada, S. (2003). Xml access control using static analysis. In *CCS*, pages 73–84. ACM Press.

Oasis. (2005). Oasis exensible access control markup language (xacml 2.0). *http://www.oasis-open.org/ committees/xacml*.

Rizvi, S., Mendelzon, A., Sudarshan, S., and Roy, P. (2004). Extending query rewriting techniques for fine-grained access control. In *SIGMOD*, pages 551–562.

Rjaibi, W. and Bird, P. (2004). A multi-purpose implementation of mandatory access control in relational database management systems. In *VLDB*, pages 1010–1020.

Wang, J. Z. and Osborn, S. L. (2004). A role-based approach to access control for xml databases. In *SACMAT*, pages 70–77.