Compact Access Control Labeling for Efficient Secure XML Query Evaluation

Huaxin Zhang Ning Zhang Kenneth Salem Donghui Zhuo
University of Waterloo
{h7zhang,nzhang,kmsalem,dhzhuo}@cs.uwaterloo.ca

Abstract

Fine-grained access controls for XML define access privileges at the granularity of individual XML nodes. In this paper, we present a fine-grained access control mechanism for XML data. This mechanism exploits the structural locality of access rights as well as correlations among the access rights of different users to produce a compact physical encoding of the access control data. This encoding can be constructed using a single pass over a labeled XML database. It is block-oriented and suitable for use in secondary storage. We show how this access control mechanism can be integrated with a next-of-kin (NoK) XML query processor to provide efficient, secure query evaluation. The key idea is that the structural information of the nodes and their encoded access controls are stored together so the access privileges can be checked efficiently. Our evaluation shows that the access control mechanism introduces little overhead into the query evaluation process.

1 Introduction

Access controls in relational databases are usually defined at a coarse granularity, e.g. on entire tables. In contrast, much of the work on XML access controls assumes a fine-grained model in which access controls can be specified for individual nodes. Fine-grained access control for XML gives rise to several challenges. First, the access rights specification is potentially very large: proportional to both the size of the database itself and the number of database system users. Thus, there must be a reasonably simple mechanism for specifying these rights and there must a compact way to store them.

Second, access controls must be implemented efficiently, since processing even a single query may involve many access right checks. This is not normally an issue when access controls are coarse-grained. In relational database systems, access controls are normally checked once before a query is processed. If all of the database objects (e.g. relations) on which a query depends are accessible, then the query is processed, otherwise it is rejected. In contrast, most fine-grained access control mechanisms never reject a query. Each query is answered using that portion of the database that is accessible to the user that submits the query. As a result, access controls and query evaluation are very closely related.

There is a plethora of literature on specifying fine-grained access controls on XML data using high level language for various access control needs. Instead of manually specifying access control for each XML node,the system administrator defines a set of rules and derive access controls for each node in the XML document through rule-based propagation and inferences. However, since evaluating the rules at runtime is costly, it is desirable to the net effect of these access control rules into incrementally maintainable accessibility maps, in which each XML node is labeled as either accessible or non-accessible for each subject under each action mode[12, 5]. The efficient storage and use of such maps are the problems that we address in this paper.

Some recent work [10] involves schema-based access control specifications, which do not require instance-level accessibility maps. However, schema-based approaches are not always applicable, since the schema may be non-existent, of inappropriate for the specification of the desired access controls. Additional discussion of related work can be found in Section 6.

In this paper we present a simple scheme called Document Ordered Labeling (DOL) for the compact representation of fine-grained access control information. Like Compressed Accessibility Maps (CAMs) [17], a recently proposed scheme for representing XML access control data, DOL exploits the structural locality of the access control data to achieve compression. Unlike CAM, DOL is also able to exploit correlations among the access rights of different users to achieve a substantial amount of additional compression in multi-user environments. DOL is a disk-oriented, multi-user scheme, while a CAM is intended to store a single user's access control data in memory.

The DOL access control representation is highly compatible with next-of-kin (NoK) pattern matching, which is an efficient technique for processing XML twig queries

[19]. NoK query processing uses a compact representation of document structure to evaluate some kinds of structural query constraints (e.g. parent/child relationships) very efficiently. In this paper, we show how to implement secure twig query processing by integrating DOL-based access control with NoK query processing.

We have used both real and synthetic access control data to evaluate the DOL technique. In terms of space efficiency, our results show that a single-user DOL is somewhat less compact than a single-user CAM. However, in a multi-user environment the DOL representation is much more compact than a set of per-user CAMs. In terms of query processing time, we have found that multi-user secure twig query evaluation with DOL and NoK is approximately 20% more expensive in the worst case than unsecured evaluation with NoK alone.

2 DOL: Access Control Labeling

We model an XML document as a tree in which the nodes correspond to the document's elements and the edges represent parent/child relationships among the elements. Sibling nodes in the tree are ordered. Our fine-grained access control model consists of a set of subjects [1], denoted by $\mathcal{S}$, a set of access control modes, such as read and write, denoted by $\mathcal{M}$, and the set $\mathcal{D}$ of nodes in the XML tree. These nodes are the objects to which access is to be controlled.

We assume that the net effect of an access control policy over a database instance can be captured by an accessibility function

$$\text{accessible} : \mathcal{S} \times \mathcal{M} \times \mathcal{D} \rightarrow \{\text{true}, \text{false}\}$$

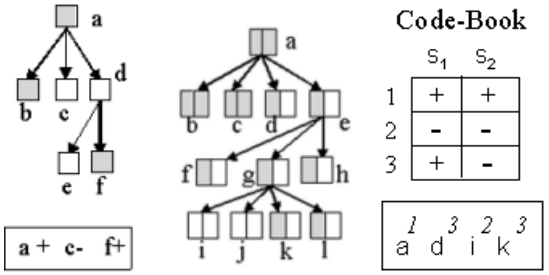
The accessibility function specifies whether a given subject can access a given data item in a given action mode. The accessibility function is often represented as an *access control matrix* [14]. For XML data, the accessibility function for a given action mode can also be represented by associating each XML node a list of subjects able to access it for that action mode. We will refer to an XML tree without access control labels as a *data tree*, and to a tree with access control labels as a *secured tree*.

Throughout most of the rest of the paper, we will assume that there is only a single access mode (i.e., $|\mathcal{M}| = 1$). Please be aware that we impose this restriction for the purposes of presentation only. The approach in this paper can be easily applied for multiple action modes in a similar way for multiple users (see more in [**?**]).

We will first present the DOL scheme for the case of a single access control subject, and then show how to generalize it to multiple subjects. Figure 1(a) shows a secured tree for a single subject, and the corresponding DOL representation of the subject's access rights. Shaded nodes are accessible to the subject, unshaded nodes are non-accessible. We define a *transition node* to be a secured tree node whose accessibility is different from its document-order predecessor (i.e., its direct previous node in document-order). As a special case, the root node of a secured tree is always a transition node. The DOL corresponding to a given secured tree is simply a list, in document order, of the tree's transition nodes, together with their accessibilities. In the DOL shown in Figure 1(a), accessible and non-accessible transition nodes are labeled with "$+$" and "$-$", respectively.

Document order is, of course, one of many possible node orders on which one could base an access control encoding like the DOL. We have chosen document order for several reasons. First, NoK query processing uses a document order encoding of document structure, and we want DOL to be compatible with NoK. Second, since XML parsers and other tools process XML data in document order, a document order encoding of access rights can be constructed on-the-fly using a single pass through a labeled XML document.

Finally, and most importantly, it allows structural locality of access controls to be exploited to reduce their size. The terms "vertical" and "horizontal" locality have been used to describe locality among parent/child nodes and among sibling nodes, respectively [17]. Structural locality is encouraged by access control specifications that propagate access rights along the hierarchical structure of the XML data, and it has been observed in real access control data [17]. Nodes that are adjacent in document order often have parent/child or sibling relationships in the document. Although this is not always the case, we expect that much of the structural locality that exists in a document's access controls will translate to locality in document order. Such locality will reduce the number of transition nodes, and hence the size of the DOL. In Section 5 we measure the impact



(a) one user (b) two users (c) DOL of two users

Figure 1. XML data with fine-grained access control and its DOL

[1] In this paper we use *subjects* to denote both *users* and *user groups*, and we use *users* to denote individuals trying to access data. The subject hierarchy, which describes group membership, is assumed to be maintained separately.

of this locality on the size of the DOL using several access control datasets.

2.1 DOL for Multiple Subjects

Aside from the structural locality of access controls for a single subject, we conjectured that different subjects in an access control system may exhibit correlated access constraints [2]. For example, subjects assigned within the same department may have similar access controls. We wish to further compress the access control labeling by taking advantage of such correlations.

Figure 1(b) shows a tree labeled with access rights for two subjects. In the figure, each node is divided into two parts, with the left part representing the access rights of one subject and the right part representing the access rights of the other. As was the case in Figure 1(a), shading represents accessibility. For example, node e is accessible to the first subject but not to the second.

We can encode these access rights in much the same way as we did for a single user, by recording a list of transition nodes. With each transition node we record its access control list. Thus, when several consecutive nodes have the same access control list, we only record it once. Furthermore, we expect the access control lists for the transition nodes will reoccur frequently throughout the secured tree. We can exploit this using dictionary compression: each distinct access control list that appears in the secured tree is recorded once in a codebook (dictionary). With each transition node in the DOL we record a reference to the appropriate access control list in the code book, rather than the access control list itself.

Figure 1(c) shows the multi-user DOL that corresponds to the secured tree in Figure 1(b). Each transition node in the list has a numeric superscript. This is the *access control code* (index into the codebook) for that transition node. The codebook itself contains three entries, because only three of the four possible distinct access control lists actually appear in the secured tree. Each codebook entry is an access control list, which we present as a bit vector with one bit for each access control subject.

The overall storage cost of DOL includes the distinct access control lists (the codebook entries) as well as the transition nodes. The number of distinct access control lists and transition nodes depends on the correlations among subjects' access controls. Generally speaking, if the access controls are not closely correlated, the transition nodes will be dense and the number of codebook entries will be large (codebook width grows linearly in the number of subjects). Suppose we have $|S|$ single subject DOLs, each having T transition nodes (in reality, each DOL would have a different number of transition nodes, but we simplify this here). In the worst case, when subjects access controls are independent, the number of distinct access control codes in the combined multi-subject DOL would grow exponentially with the number of subjects until it reaches the maximum $min(|\mathcal{D}|, 2^{|\mathcal{S}|})$. Meanwhile, the number of non-transition nodes would be:

$$|D| \times (1 - \frac{T}{|D|})^{|S|}$$

Apparently, as S goes up, the number of non-transitional nodes shrinks exponentially until each XML node becomes a transition node.

However, the real access control systems that we have studied do not exhibit this worst case behavior. We will see in Section 5 that there *do* exist strong correlations of access controls among subjects in these systems that make the overall size of DOL grow sublinearly with the number of subjects in the system.

3 Physical Representation of the DOL

In this section we describe our physical representation of the DOL, which is intended to be incorporated into an existing query processor framework called NoK [19] to optimize secure query evaluation. For this reason, we begin with a brief overview of NoK query processing.

3.1 NoK Query Modeling and Physical Storage

A NoK query processor accepts twig queries described by pattern trees and evaluates them against an XML document by pattern matching. Each successful pattern match generates a set of bindings between pattern tree nodes and data tree nodes. The query result consists of all possible bindings. For example, the pattern tree in Figure 2 will generate one match from the data tree.

The NoK query processor first partitions the pattern tree into *NoK* subtrees, each containing only parent-child or following-sibling relationships (the so-called "next-of-kin" relationships) among its nodes. Then the processor finds matches for these NoK subtrees from the data tree. Finally it combines the matched results using structural joins on the ancestor-descendant relationship. For example, the pattern tree in Figure 2 would be split into two NoK subtrees, each matches to a fragment in the data tree. The two fragments found are thus connected by the ancestor-descendant relationship between nodes a and h.

The NoK query processor uses a physical representation of the data tree that allows it to match NoK subqueries very efficiently. The structure of the data tree is stored separately from the node values in a compact representation. It is encoded by listing the nodes in docu-

[2] There may also exist correlations among action modes, but in this paper we restrict our attention to subject correlations only. We believe our approach can also exploit correlations among action modes

ment order, with embedded markup to indicate where subtrees begin and end. For example, the structure of the data tree of Figure 2 would be encoded using the following string[3]: $(a(b)(c)(d)(e(f)(g)(h(i)(j)(k)(l))))$, where the nesting of parentheses captures the nesting of subtrees. This document-order string is decomposed into blocks for storage on disk. Each block has a header with meta data for that page (e.g. the number of nesting parentheses for the first node in the page).

It can be seen that nodes connected by "next-of-kin" relationships are clustered in this physical representation and thus such nodes are more likely to be located in the same physical block. The net effect is that a NoK query processor can match a NoK pattern using just a few I/O operations[19].

3.2 Integrating Access Control Data

Our approach is to physically cluster the access control data with the NoK structural data. Specifically, our scheme for physical representation of the access control data consists of the following three components:

- The DOL codebook is maintained in memory for fast accessibility lookup. If the codebook grows beyond the capacity of memory, each accessibility lookup may result in an extra physical page read for loading the codebook entry. However, our results in Section 5 shows in practice the codebook will be quite small.
- The DOL transition nodes are embedded into the NoK structural data. Figure 3 shows the embedding for the secured tree of Figures 1(b) (the page header does not show NoK meta-data for simplicity). For the purposes of the illustration, we have assumed that these data are spread across three disk blocks. In the physical encoding, we treat the first node in each block as if it were a transition node, regardless of whether it is actually a transition node. The access control code for this initial transition node is stored in the block header, which is described next. These initial transition nodes ensure that we can determine the access rights of any node using only the codes in that node's block.
- For each disk block, there is a small access control header which contains two items. The first is the access control code for the first data node in the block. The second is a "change" bit which is set if there is at least one transition node (other than the initial node) in the block, and cleared otherwise. By keeping all the page headers in memory (our statistics show we only need 30Mb to 100Mb as page header for processing 1Tb XML data consisting of 10 billion nodes), the NoK query processor can implement I/O optimizations which we shall describe shortly.

[3] Actually, the string is further compacted by eliminating all of the open parentheses, which are redundant.

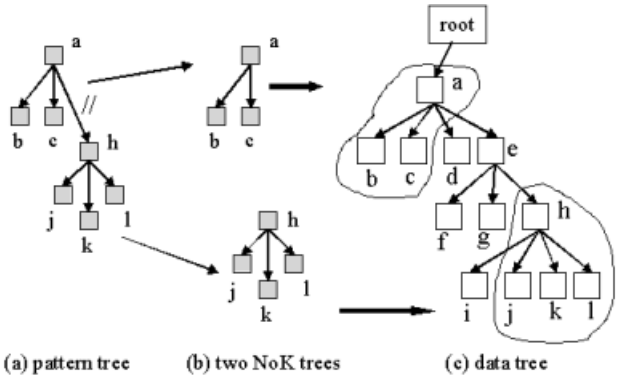


Figure 2. A Pattern Tree with two NoK trees matched to data tree

3.3 Access Lookup

To check the accessibility of a node d for subject s, the query processor locates the transition node that precedes node d (if d is not itself a transition node). Since the first node in every block is a transition node, the transition node will be found in d's block. That transition node's access control code is then used to identify an entry in the in-memory access control codebook. The s-th bit in that code book entry indicates the accessibility of the node for subject s. As we will see in Section 4, the NoK query processor checks nodes' accessibility while it matches NoK query patterns. Provided that d's disk block has been loaded (piggy-backed) for query evaluation by the NoK evaluator, the access control check for d requires no additional I/O.

In some cases, the query processor could make use of the in-memory DOL page header to avoid unnecessary page reading: if the starting transition node in the header indicates non-accessible to the user, and the "change" bit in the header is not set (meaning there is no other transition nodes in this page), all nodes in that page are non-accessible to the user. Thus the query processor could avoid load that page.

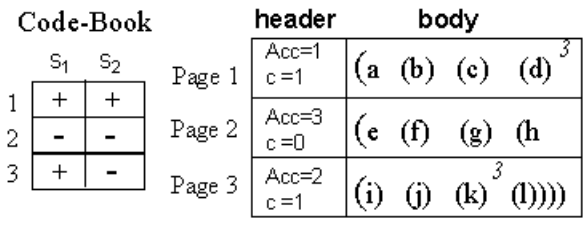


Figure 3. DOL at physical level

3.4 DOL Updates

We consider two types of update operations to the access control representation: **accessibility update** and **structural update**. The first type of update refers to changes in the accessibility function itself, e.g.,, a change to the accessibility of a single node (e.g.,, adding read permission for a given subject), or a change to the accessibility of all of the nodes in a document subtree. The second type of update occurs when we modify the structure of the data tree, e.g.,, insertion or deletion of a node or a subtree (we assume the nodes inserted have access controls already), or moving a node or a subtree.

We first look at changing accessibility of a single node. Suppose we are to set the accessibility of a node to "accessible" for certain subject. We need to locate the nearest preceding transition node. If that transition node's access control code indicates "accessible" for the subject, we stop. Otherwise, we mark the original node as a new transition node and update its access control code to be "accessible" for that subject. We may need to add that access control code to the codebook if it is not already there. Finally we need to mark the following node to be a new transition node with the same access control code as the preceding transition node's access control code. All these operations occur in memory after loading that page(assuming the codebook is in memory). Thus the cost for update a specific node is a page read followed by a page write flushing the updates to disk. However, if we are to set the accessibility of a whole subtree, the number of page I/O will be much smaller than updating each of the nodes in the subtree separately. This is because the physical representation clusters these nodes consecutively in the pages, and the the pages are logically consecutive (thus more likely to be physically close to each other). Suppose each page can hold B nodes, the cost for updating accessibility of a subtree with N nodes would be the N/B pages reads (and writes).

It is worth mentioning that all updates to DOL have the *update locality* property, i.e., an update to a subtree only affects the nodes within the pair of transition nodes that surround the subtree. This property guarantees that updates are confined within a contiguous region of the affected data.

Updates may also affect the amount of space used to store access control data by increasing or decreasing the number of transition nodes. However, the following proposition applies to all of the types of updates that we have presented, including the subtree updates

Proposition 1 *For each of the above operations (accessibility update or structural update), the number of transition nodes of the new DOL will be at most* 2 *more than the number of transition nodes in the original data (and the data to be inserted).* □

In addition to the updates to the XML data, we need to consider updates to $\mathcal{S}$, the set of access control subjects. With DOL, it is relatively simple to add a new subject who has no (initial) access rights, or whose access rights initially match those of some existing subject. This can be accomplished by simply adding an additional column to each entry in the in-memory codebook. No changes to the embedded transition nodes and the references are required. Deletion of a subject can also by accomplished within the codebook. This may leave unnecessary codes embedded in the structural data, since the deletion of a subject may decrease the number of transition nodes. However, any such redundancy can be corrected lazily.

4 Secure Query Evaluation

Our semantics for secure query evaluation are identical to those used by Cho et al[7]. Recall that the (unsecured) evaluation of a twig query Q returns all of the possible sets of bindings of query pattern nodes to data nodes. Secure evaluation of Q for subject s eliminates from this result any sets of bindings that include data nodes that are inaccessible to s.[4] For example, the pattern tree shown in Figure 2 will return a single set of bindings if nodes a, b, c, h, j, k and l in the data tree are all accessible to the subject s. It will return no bindings if any of those nodes are inaccessible to s. Note that the accessibility of nodes d, e, f, g and i has no impact on the secure evaluation of the particular query shown in Figure 2.

4.1 DOL for Secure NoK Pattern Matching

In Section 3.1 we described that a NoK query processor works by first decomposing a pattern tree into NoK subtrees, and then attempting to match each NoK subtree to the data (by using B+ trees on the subtree root's value or tag names to start the matching). One node in the NoK pattern tree is set as *returning node*, which means the nodes in the data tree that matches to this node should be returned as result of this pattern matching.

The secure NoK pattern matching algorithm is shown in Algorithm 1. We use unordered XML data (no ordering between siblings in pattern tree) for ease of presentation only, though we use ordered pattern tree in real experiments. The input parameter *proot* is the current node from the NoK pattern tree, and *sroot* is the current document node that is being matched to *proot*. The third parameter R is set to $\emptyset$ initially and will contain a list of data tree nodes (in document order) that match the returning node.

To match NoK subtrees, the query processor uses a recursive navigational approach, starting with an initial match

[4] In practice, the actual access rights of a user may be determined by combining the access rights of one or more access control subjects from $\mathcal{S}$. For example, a user's access rights may include her own plus those any groups of which she is a member.

from the data for the root of the NoK pattern tree. It then proceeds by recursively matching children of *proot* to children of *sroot* [5]. The subroutines FIRST-CHILD and FOLLOWING-SIBLING use the block-oriented physical encoding of the document structure to return the first child of the *sroot* in document-order, or the next sibling of the current node, respectively. The subroutine ACCESS (line 6) checks the accessibility of the child of the current document subtree root to be matched. Since a node's accessibility is checked immediately after it is loaded (by FIRST-CHILD or FOLLOWING-SIBLING), and since its access control code will be found on the same page as the node itself, no additional I/O will be required for node accessibility checks. Note that the pre-condition of Algorithm 1 is the $sroot$ nodes be accessible. This means before we use Algorithm 1 to recursively match NoK pattern trees, the root of the NoK data tree should be checked to make sure it is accessible.

According to the query evaluation semantics given early in Section 4, we can skip the recursion on the child if the child is not accessible.

After NoK subtree matches are located, they can be structurally joined based on ancestor-descendant relationships. Since the nodes in the NoK subtrees are already checked for accessibility, the structural-join algorithm does not need to check accessibility any more. We have the following theorem:

Theorem 1 *Algorithm ϵ-NoK, together with any non-secured structural join algorithm, securely evaluates XML twig query* □

[5] This top-down recursive pattern matching needs only $O(|P| \times |D|)$ time to find *all* matches, where P is the size of the pattern tree and the D is the size of the document [19].

Algorithm 1 ϵ-NoK Pattern Matching

NPM$(proot, sroot, R)$

Pre-condition: sroot is accessible

```
1:  if proot is the returning node
2:     then LIST-APPEND(R, sroot);
3:  S ← all children of proot;
4:  u ← FIRST-CHILD(sroot);
5:  repeat
6:        if ACCESS(u) = TRUE
7:           then for each s ∈ S that matches u with
                     both tag name and value constraints
8:                do
9:                   b ← NPM(s, u, R);
10:                  if b = TRUE
11:                    then S ← S \ {s};
12:       u ← FOLLOWING-SIBLING(u);
13:    until u = NIL or S = ∅
14: if S ≠ ∅
15:    then R ← ∅;
16:       return FALSE;
17: return TRUE;
```

4.2 Alternative Access Control Semantics

One other secure semantics is defined in [11], which specifies that a subtree rooted at a non-accessible node can not provide answers even if it contains accessible nodes. For example, the pattern tree in Figure 2 will not find any matches from the data tree if node e is not accessible while all the remaining nodes are accessible.

Therefore, we not only need to check the ancestor-descendant (AD) relationship between nodes, but also the accessibility of all the nodes from the ancestor to the descendant. The join must be aborted if there is one non-accessible node on the path. However, the nodes between the ancestors and descendants are not necessarily clustered on the same physical pages as the NoK subtrees, so this checking may involve lots of page reads.

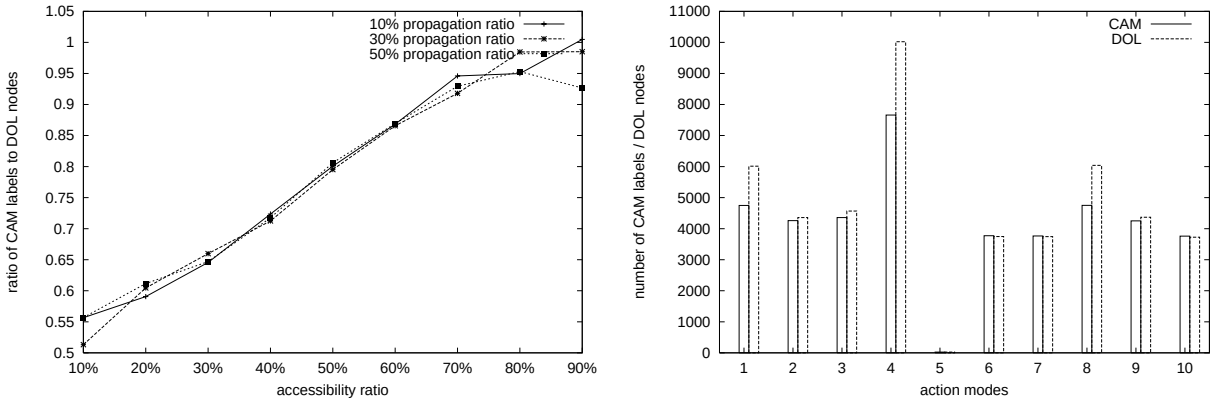
In [18] we developed a secure structural join algorithm based on the widely accepted *Stack_Tree_Desc* (STD) algorithm [2]. We demonstrate both theoretically and empirically that our secure structural join algorithm aggressively prunes un-secured matches and only load each page once if necessary, regardless of the accessibility distribution of the document.

5 Performance Evaluation

We evaluate the DOL technique using both synthetic and real access control data. We generated synthetic access controls on XMark benchmarks [1] by randomly choosing some nodes from the document as *seeds*, and then labeling these seeds as accessible or non-accessible. We simulate horizontal structural locality by randomly setting the seeds' direct siblings with the same accessibility, provided that the siblings are not themselves seeds. Then, we simulate vertical structural locality by propagating accessibilities of labeled nodes to their descendants using the Most-Specific-Override policy [12], i.e., a node inherits its accessibility from its closest labeled ancestor. We always choose the document root as seed to ensure all nodes be labeled. The access controls in the data are affected by two parameters. The *propagation ratio* determines percentage of nodes that are seeds while the *accessibility ratio* determines the percentage of seeds that are accessible.

In addition, we used two sets of real multi-user access control data. The first data set describes the access control information from a production instance of OpenText LiveLink[6], which provides web-based collaboration

[6] LiveLink is a trademark of OpenText Corporation.



(a) synthetic data

(b) LiveLink system

Figure 4. CAM labels and DOL transition nodes for single subject

and knowledge management services in a corporate intranet. The LiveLink system has 371547 data items in a tree-structure with an average depth of 7.9 and a maximum depth of 19. The system has a total of 8639 access control subjects (users and groups).

The second data set consists of the access control data from a multiuser Unix file system at the University of Waterloo. This system has 182 users and 65 user groups, and includes more than 1.3 million files/directories. Although neither of these systems stores actual XML data, both provide tree-structured data models and instance-level access controls. For the purposes of our experiments, we treat these systems as surrogates for real multi-user access controlled XML databases.

5.1 Compression Ratio

We first evaluate DOL for a single subject. Since CAM [17] is the state of the art compact labeling for single subject access controls, we compare DOL with CAM. We first use an XMark document of 17133 nodes with synthetic access controls produced by different accessibility and propagation ratios. Our metric is the ratio of the number of CAM nodes to the number of DOL transition nodes (the codebook size is trivial for one subject). Thus, values less than 1.0 favor CAM and those greater than 1.0 favor DOL.

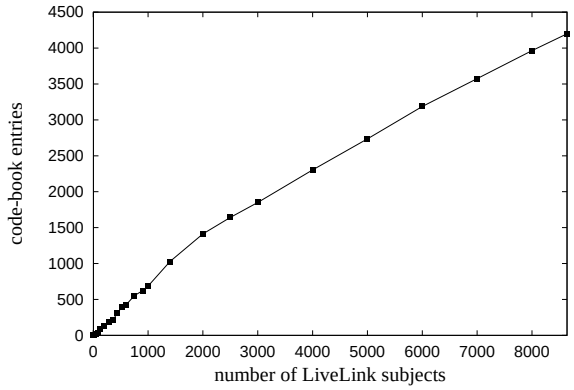
Figure 4(a) shows the comparisons as the accessibility ratio varies from 10% to 90%. We tried three propagation ratios with these different accessibility and the results are similar. When accessibility ratio is low (few nodes are accessible), the number of CAM nodes is around 53% of the number of DOL transition nodes. As accessibility goes up, this difference becomes smaller. A close look at the test results (not shown) reveals that CAM's compression ratio is asymmetric to accessibility ratio. The number of CAM nodes reaches maximum at 60% accessibility ratio, but the number of CAM nodes at 10% accessibility ratio is only 1/3 of CAM nodes at 90% accessibility ratio. On the other hand, DOL's compression ratio is symmetric around 50% accessibility ratio (with the most number of transition nodes at 50%).

We also compare single user CAM and DOL for LiveLink data. The LiveLink system supports ten different access modes. For each of the ten access modes we sample a number of users and built CAM and DOL for each single user. The number of DOL labels/CAM nodes for an average user is shown in Figure 4(b), with the ten access modes shown on the horizontal axis. In the worst cases, DOL had 20-25% more nodes than CAM. In other cases, the two schemes performed about the same.

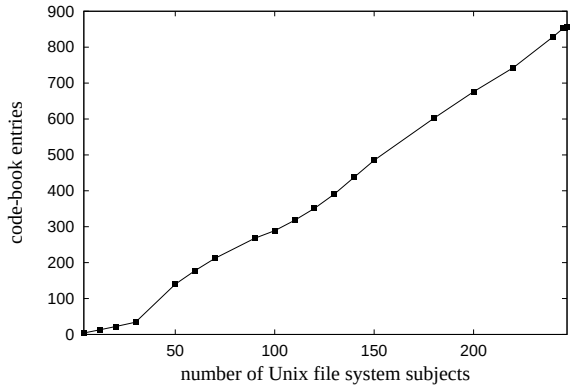
Our performance metric implicitly assumes that CAM nodes and DOL transition nodes are the same size. In practice, however, the DOL nodes are likely to be much smaller. This is because CAM stores the access rights separately from the data. As a result, each CAM node must include a reference to a document node and pointers to the node's children in the CAM, in addition to the access control information itself. In contrast, DOL, which piggybacks access control information into the document encoding, stores only an access control code per transition node. Thus, although CAM may have fewer nodes than DOL, the total space required for CAM may be greater.

5.1.1 Multiple User Environment

We used our two real data sets to evaluate the space efficiency of DOL in multi-user environments. To get a sense of how the codebook size might vary as a function of the number of subjects, we selected a number of subjects randomly and computed DOL codebooks for the selected subjects only. In Figures 5(a) and 5(b) we have plotted the number of codebook entries as a function of the cardinalities

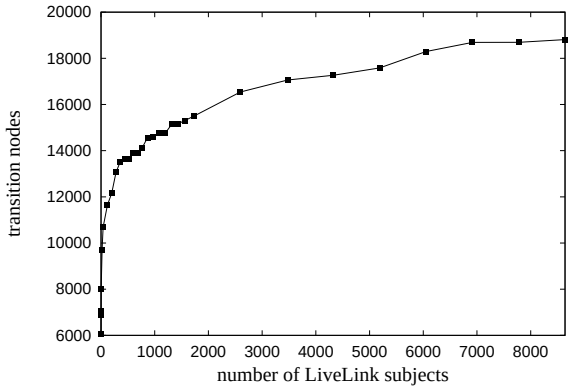


(a) LiveLink

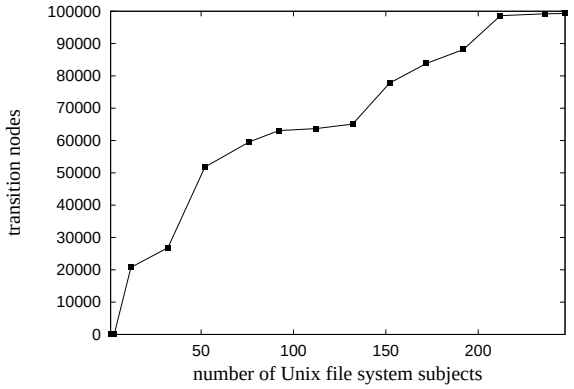


(b) Unix file system

Figure 5. Codebook entries for multi-subjects



(a) LiveLink



(b) Unix file system

Figure 6. Transition nodes for multi-subjects

of these subsets. If subjects' access controls were uncorrelated, we would expect to see exponential growth in the number of codebook entries as subjects increased. However, our results show that the growth is much slower in practice. With all $8000+$ subjects, the LiveLink system required around 4000 codebook entries. At 1000 bytes per codebook entry (one bit per subject for all 8000 subjects), the complete LiveLink codebook would occupy only about 4MB of memory. The Unix system required about 855 codebook entries for 247 subjects, with an overall size of only 25KB.

The other major storage concern is the number of DOL transition nodes. Figures 6(b), 6(a) show the numbers of transition nodes required for the LiveLink and Unix systems as the number of subjects increases (using the same methodology that was used for Figures 5(a) and 5(b)). Figure 6(a) shows a close to linear trend in the growth of transition nodes. For over 8000 subjects, the number of transition nodes is only about 4 times larger than the number for a single subject. For Unix file system, we see a similar situation in Figure 6(b), in which the number of transition nodes of 247 subjects is only twice as many for 50 subjects. Recall that the total number of nodes in the LiveLink system is 371547, and the total number in the Unix system is about 1.3 million. Thus, the density of transition nodes is less than 1 in 10 for both systems (for all the subjects). These results indicate that the access rights for different subjects are highly correlated in real world.

To compare the overall storage cost between DOL and CAM, we first look at a single subject in LiveLink (under action mode 1): DOL needs about 6000 transition nodes while CAM needs 4500 labels. However, for all 8639 subjects in the same system under the same action mode, DOL needs 18800 transition nodes while CAM needs 8639×4500 labels, a difference of three orders of magnitude. Assuming each DOL transition node requires a 2 byte access control code (for the 4000 codebook entries), and each CAM label takes 2 bits for its accessibility encoding, and *(unrealistically)* only 1 byte for node pointers, the DOL's total space requirement will be a 4MB codebook plus a trivial 40KB embedded transition nodes, while CAM's will be

Q1	/site/regions/africa/item[location][name][quantity]
Q2	/site/catetories/category[name]/description/text/bold
Q3	/site/categories/category/name[description/text/bold]
Q4	//parlist//parlist
Q5	//listitem//keyword
Q6	//item//emph

Table 1. Queries

46.6MB. For the Unix file system the situation is similar. Clearly, correlation among the subjects contributes substantially to compression effectiveness.

5.2 Query Evaluation

Since DOL is the only disk-oriented access control model for secure XML data evaluation, we compare its performance with the non-secured NoK query processor for fairness. We implement both ε-NoK, ε-STD, and the non-secure versions of the NoK and STD algorithms using Java 1.5. All of the experiments were conducted using a PC with a Pentium III 997MHz CPU, 512MB RAM, and 40GB hard disk running Windows XP.

Our test data is a 50Mb XMark instance (832911 element nodes) with synthetic access controls. The data is stored on disk with each page at 4K bytes. The benchmark queries are shown in Table 1. The top three queries represent three classes of NoK pattern trees: those with branches at the end (Q1), in the middle (Q2), or a single path (Q3). The bottom three queries are for ancestor-descendant structural joins and represents those having descendants located closely (Q4), medium distantly(Q5), distantly(Q6) from the ancestors.

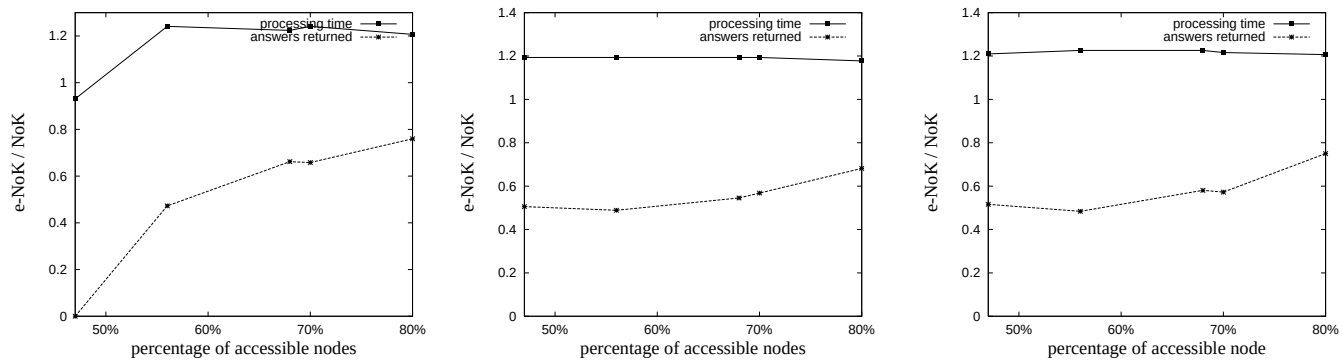
Figures 7(a),7(b) and 7(c) show the performance of ε-NoK algorithm. The two lines in each figure depict the ratio of processing time and answers returned between the ε-NoK and non-secure NoK algorithms. In most situations the the processing time of the ε-NoK algorithm is only around 20% more than the non-secure NoK algorithm, and does *not* depend on the accessibility ratio. This is still true when majority of the document is accessible (thus most answers of the original NoK algorithm are returned, and nodes in these answers are all checked). The reason is that accessibility checking does not require extra I/O for ε-NoK algorithm. Only when the accessibility ratio filters most of the answers that are originally returned by the non-secure NoK algorithm, the secured NoK algorithm could save some page I/O by checking the in-memory DOL page headers, and thus works faster than the non-secure NoK.

6 Related Work

Jajodia et al [12] and Bertino et al [5] define models that are capable of describing a wide spectrum of access control policies such as positive and negative authorization, propagation policies, conflict resolving, closed versus open world assumptions. The IBM XACL project [13] proposed a XML access control language for authorizing access (read, write, create, delete) to fine-grained XML data. It is capable of defining propagations, conflict resolving and provisional authorization. Bertino et al [6] proposed a model that is capable of describing various access control policies. Their model also addresses the problem for "push" queries, which is for massive distribution of data to subscribers. Damiani et al[8] proposed access control query modeling specific for XML documents to facilitate secure information flow for the Web. A similar framework is implemented in the Author-X project [4]. Gabillon and Bruno [11] define view-based semantics for secure query evaluation. They define a secured view for each user group, and queries are applied against the secure views. However, their approach prunes all subtrees with a non-accessible root, regardless whether there are accessible nodes in that subtree or not. Cho et al [7] define a more relaxed pattern-matching based semantics. This semantics allows answers to come from a subtree whose root is non-accessible. They use schema information to rewrite queries to optimize query evaluation time. Stoica et al [16] use secured views and secure data schema (both generated from original data schema and access control policies) for answering XML queries. Similarly, Fan et al [10] uses secured views generated from DTD schemas and access control rules for secure query evaluation. The access control rules in these two approaches are based on the data schema, which must exist. However, instance-based access control is necessary when there is no schema, or when the desired access controls cannot be described by schema level specifications. Lee et al [15] propose a secure XML evaluation framework in which access controls are specified on the XML model but enforced in an underlying relational database. However, the XML data instance needs to be first sliced into the relational model. There is also work [3, 9] on efficient dissemination of sensitive XML data using pattern matching. The DOL approach can be similarly used for dissemination of XML data to multiple users. The difference is that DOL works on arbitrarily fine-grained sensitive data at instance level.

7 Conclusion

Our paper presents a compact XML access control labeling scheme called DOL that supports efficient secure query evaluation. Our labeling scheme exploits both access control structural locality within the XML data and correlations



(a) Query 1 (b) Query 2 (c) Query 3

Figure 7. Performance between ε-NoK and NoK as a function of node accessibility

between users' access rights. Physically, access controls are embedded into the representation of the document structure. The physical layout makes it easy to embed into streaming XML data as control characters and many one-pass algorithms on streaming XML data can be made secure. Our experiments demonstrate its storage compactness in multi-user environments and its ability to support secure query evaluation efficiently.

Acknowledgements Special thanks to Frank Tompa of University of Waterloo for feedback. Research supported by Communications and Information Technology Ontario (CITO), the Natural Sciences and Engineering Research Council (NSERC), and the Open Text Corp.

References

[1] The XML benchmark project. Available at http://www.xml-benchmark.org.

[2] S. Al-Khalifa, H. V. Jagadish, N. Koudas, and J. M. Patel. Structural Joins: A Primitive for Efficient XML Query Pattern Matching. In *Proc. 18th Int. Conf. on Data Engineering*, 2002.

[3] M. Altinel and M. J. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *Proc. 26th Int. Conf. on Very Large Data Bases*, pages 53–64. Morgan Kaufmann Publishers Inc., 2000.

[4] E. Bertino, S. Castano, and E. Ferrari. Securing XML Documents with Author-X. *IEEE Internet Computing*, 5(3):21–31, 2001.

[5] E. Bertino, B. Catania, E. Ferrari, and P. Perlasca. A Logical Framework for Reasoning about Access Control Models. *ACM Transactions on Information and System Security*, 6(1):71–127, 2003.

[6] E. Bertino and E. Ferrari. Secure and Selective Dissemination of XML Documents. *ACM Transactions on Information and System Security*, 5(3):290–331, August 2002.

[7] S. Cho, S. Amer-Yahia, L. V. S. Lakshmanan, and D. Srivastava. Optimizing the Secure Evaluation of Twig Queries. In *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 490–501, Aug. 2002.

[8] E. Damiani, S. De Capitani di Vimercati, S. Paraboschi, and P. Samarati. A Fine-grained Access Control System for XML Documents. *ACM Transactions on Information and System Security*, 5(2):169–202, 2002.

[9] Y. Diao and R. T. Peter M. Fischer, Michael J. Franklin. YFilter: Efficient and Scalable Filtering of XML Documents. In *Proc. 18th Int. Conf. on Data Engineering*, pages 341–353, 2002.

[10] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure XML Querying with Security Views. In *Proc. ACM SIGMOD Int. Conf. on Management of Data*, 2004.

[11] A. Gabillon and E. Bruno. Regulating Access to XML Documents. In *Proc. 15th Int. Conf. on Database and Application Security*, pages 299–314. Kluwer Academic Publishers, 2002.

[12] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible Support for Multiple Access Control Policies. *ACM Trans. Database Sys.*, 26(2):214–260, 2001.

[13] M. Kudo and S. Hada. XML Document Security based on Provisional Authorization. In *7th ACM Conference on Computer and Communication Security*, pages 87–96, 2000.

[14] B. Lampson. Protection. In *ACM Operating Systems Review*, pages 8–24, 1974.

[15] D. Lee, W.-C. Lee, and P. Liu. Supporting XML Security Models Using Relational Databases: A Vision. In *Proc. 1st Int. XML Database Symposium*, pages 267–281, 2003.

[16] A. G. Stoica and C. Farkas. Secure XML Views. In *Proc. 15th Int. Conf. on Database and Application Security*, pages 133–146, 2002.

[17] T. Yu, D. Srivastava, L. V. Lakshmanan, and H. V. Jagadish. Compressed Accessibility Map: Efficient Access Control for XML. In *Proc. 28th Int. Conf. on Very Large Data Bases*, pages 478–489, 2002.

[18] H. Zhang, N. Zhang, K. Salem, and D. Zhuo. Compact Access Control Labeling for Efficient Secure XML Query Evaluation. Technical report, Department of Computer Science, University of Waterloo, 2004.

[19] N. Zhang, V. Kacholia, and M. T. Özsu. A Succinct Physical Storage Scheme for Efficient Evaluation of Path Queries in XML. In *Proc. 20th Int. Conf. on Data Engineering*, 2004.