Compressed Accessibility Map: Efficient Access Control for XML

Ting Yu[*]
University of Illinois, Urbana-Champaign
tingyu@uiuc.edu

Divesh Srivastava
AT&T Labs–Research
divesh@research.att.com

Laks V.S. Lakshmanan[†]
University of British Columbia
laks@cs.ubc.ca

H. V. Jagadish[‡]
University of Michigan, Ann Arbor
jag@umich.edu

Abstract

XML is widely regarded as a promising means for data representation integration, and exchange. As companies transact business over the Internet, the sensitive nature of the information mandates that access must be provided selectively, using sophisticated access control specifications. Using the specification directly to determine if a user has access to a specific XML data item can hence be extremely inefficient. The alternative of fully materializing, for each data item, the users authorized to access it can be space-inefficient. In this paper, we propose a space- and time-efficient solution to the access control problem for XML data. Our solution is based on a novel notion of a compressed accessibility map (CAM), which compactly identifies the XML data items to which a user has access, by exploiting structural locality of accessibility in tree-structured data. We present a CAM lookup algorithm for determining if a user has access to a data item; it takes time proportional to the product of the depth of the item in the XML data and logarithm of the CAM size. We develop a linear-time algorithm for building an optimal size CAM. Finally, we experimentally demonstrate the effectiveness of the CAM for multiple users on both real and synthetic data sets.

[*]Yu's research was supported by DARPA via AFRL contract F33615-01-C-0336 and via Space and Naval Warfare Systems Center San Diego (SSCSD) grant N66001-01-18908.

[†]Lakshmanan's research was supported by grants from NSERC and NCE/IRIS.

[‡]Jagadish's research was supported in part by NSF under grants DMI-0075447 and IIS-0208852.

Proceedings of the 28th VLDB Conference,
Hong Kong, China, 2002

1 Introduction

The eXtensible Markup Language (XML) is widely regarded as a promising means for data representation, integration, and exchange, due in large part to its simplicity and capability of representing rich data structures. As companies transact business over the Internet, letting authorized customers directly access, and even modify, operational data items in XML documents over the Web offers many advantages in terms of cost, accuracy, and timeliness. However, it raises the question of security. Given the sensitive nature of business information, access must be provided selectively. Furthermore, the nature of this selective access has more and more sophisticated requirements imposed on it, as we move from B2C (business-to-consumer) e-commerce to B2B (business-to-business) e-commerce. For instance, a large corporation may be willing to let a supplier view (i) inventories for parts that it supplies (but not for other parts), and (ii) production schedules for items manufactured using this supplier's parts, but only up to the next two weeks, etc. Where suppliers participate in the design process, as many suppliers to the big automobile companies do today, a complex collection of design data must be shared with appropriate caution. The sophistication of the associated access control policy required is appreciable.

While there are certainly security issues with regard to obtaining protection against various forms of attack on the Internet, *our interest in this paper is in efficiently evaluating a stated access control policy over XML documents.*

We expect that the policy will be specified in terms of a (potentially large) set of access control rules, which specify user access to data items on the basis of user properties and categories, and on the basis of the properties of the data items in question. These rules frequently interact with one another in complex ways, with some priority mechanism or conflict resolution mechanism used for unambiguously deciding for a user, an item, and an access type whether the user has access to it or not. As a consequence, using the access control specification *directly* to determine whether or not a user has access to a particular data item can be extremely inefficient.

One possible approach for efficiently determining whether or not a user has access to a particular data item is to build a fully materialized accessibility map, which maintains, for each data item, the set of users authorized to access it; this can, however, require a large amount of space. For instance, consider a system with one million users, each of whom can access (on average) about 10% of the data. Each object, on average, would have an access list with 100,000 users, represented as 400KB per object if each user id is 4 bytes. Alternatively a single bit per user can be used to maintain an access map in one million bits, or 125 KB per object.

In this paper, we propose the *compressed accessibility map* (CAM) as a means of recording exactly this information, but in a much smaller space, for XML documents. Given a user and a data item, the CAM can be used to determine efficiently if a user has access to the data item. Our key contributions are as follows:

- We propose the novel notion of a compressed accessibility map (CAM) as a space- and time-efficient solution to the access control enforcement problem for XML data. This compression is obtained by exploiting structural locality of accessibility, i.e., data items grouped together have similar accessibility properties, on a per-user basis. The total size of the CAM is typically a small fraction of the fully materialized accessibility map.

- We develop algorithms to construct an optimal (minimum size) per-user CAM for a given accessibility map, which take time linear in the database size (number of items in the database for which accessibility is individually identified).

- Using an optimal CAM, we present algorithms for efficiently determining if a user has access to a data item. This lookup takes time proportional to the product of the depth of the item in the XML data and a logarithm of the size of the CAM.

- Finally, we experimentally demonstrate the effectiveness of the CAM for hundreds of users on both real and synthetic data sets, validating our approach to the access control enforcement problem.

The problem setting and the basic idea of the CAM are described in Section 2. Optimal CAM construction is addressed in Sections 3, 4, and 6. CAM lookup is developed in Sections 5 and 6. Experimental results are reported in Section 7. Related work is discussed in Section 8, just before summarizing our conclusions in Section 9. All proofs are omitted for reasons of space.

2 The Problem Setting

2.1 Hierarchically Structured Data and Structural Locality

A key foundation of our work is *structural locality of accessibility*, i.e., data items grouped together have similar (but not necessarily identical) accessibility properties, on a per-user basis. This is commonly observed in hierarchically-structured data used in supporting e-commerce applications, such as XML documents and file systems, where the structural organization typically results in a hierarchical grouping of closely related data items. Locality of accessibility exists not only horizontally, i.e., between entities that share the same parent, but also vertically, i.e., between parents and children. As a consequence, if an entity is accessible, then very likely (but not always) so are its ancestors.

XML has a hierarchical Document Object Model within a document. Access control is often desired at a granularity much finer than an entire XML document, perhaps even down to the level of individual elements in some cases [2, 5]. The access control policy is typically specified declaratively, using access control rules and conflict resolution mechanisms. Many proposed access control models for XML documents allow users to propagate an access control policy in a data item to its descendents unless it is overridden by more specific access control policies. Therefore, structural locality of accessibility is quite natural in XML documents.

A direct enforcement of the policy may be too inefficient, but materializing the entire policy may not be feasible, since the smaller the unit of access control, the more serious is the overhead involved in full materialization.

The approach of this paper offers an efficient and elegant solution to the problem of access control enforcement in the case of XML databases. We assume that we are given an XML database tree H, each of whose nodes is uniquely identified by a hierarchically structured identifier that allows direct determination of parent-child and ancestor-descendant relationships between tree nodes. In this paper, we use the identifier that is the concatenation, in root to leaf order, of node positions (in the prefix traversal of the tree) in the path to it from the root of the tree.[1] The nodes of

[1] If the database is a forest, our techniques can easily be applied by introducing a dummy root node and considering the forest to be a single tree.

this tree reflect the finest granularity of accessibility.

2.2 Accessibility Map

Conceptually, an access control policy induces a projection of the XML database tree for each user, consisting of the nodes the user is allowed to access according to the policy. The policy itself may be specified in terms of access control rules with conflict resolution mechanisms, explicit access control lists, or any other means appropriate. The specification mechanism is orthogonal to the efficient enforcement techniques of this paper.

Moreover, there could be many different types of access allowed, e.g., read, modify, append, etc, and policies may also specify some relationships between these access types. For instance, a user given modify access to a node may automatically also be granted read access. As far as we are concerned, all such policies are also folded in to create a final decision (of "accessible" or "inaccessible") for each node, for each user, and for each access type. We refer to this as the *accessibility map* of the database tree, formalized below.

Definition 2.1 [Accessibility Map] Let H be a database tree, U be a set of users, and A be a set of access types. The *accessibility map* is a function $M : H \times U \times A \rightarrow \{\text{accessible}, \text{inaccessible}\}$. ▮

While the fully materialized accessibility map (suitably indexed) supports rapid determination of accessibility, it can be very space-inefficient. Our goal is to identify a compact means for recording this marking, which yet affords quick lookup.

2.3 Compressed Accessibility Map

The key to obtaining a space-efficient representation for the accessibility map is to exploit the structural locality of accessibility in the XML database tree, and maintain a separate *compressed accessibility map* (CAM), for each user and access type.

Restricted to a single user and access type, the accessibility map is simply a marking that maps the nodes of the XML database tree to the set {accessible, inaccessible}. We call a database tree H together with a marking M as a *marked tree.* The basic idea of the CAM is that, instead of explicitly keeping a list of all the accessible (or inaccessible) nodes, we keep only some "crucial" nodes and place some additional information on them so that we can efficiently check whether an arbitrary node can be accessed or not by simply looking at relevant "crucial" nodes. The compression achieved by CAM relies on two observations.

First, in a "region" of the XML database tree, uniform accessibility of descendants of a node can be represented at the node itself. For instance, if each node in a subtree of the marked tree is accessible, substantial compression can be achieved if the CAM maintains the root node of the subtree with a $(d+, s+)$ label, indicating that the node is accessible ($s+$), and so are its descendants ($d+$).[2] This observation can be generalized to employ a simple node labeling mechanism, involving $d+, d-, s+$ and $s-$. The semantics of the labels is as follows. If node x carries an $s+$ (resp., $s-$), then x is accessible (resp., inaccessible). If node x carries a $d+$ (resp., $d-$) and y is a descendant of x, then y is accessible (resp., inaccessible), unless this is overridden by the label of a closer ancestor of y.

Second, it is frequently the case that, at least within a "local region" of the XML database tree, there is a hierarchical aspect to accessibility: if a node is accessible, then so are its ancestors. We call this the *ancestor accessibility* property. This observation can be utilized in the CAM as follows: if a node x carries an $s+$, then so must all its ancestors, within the local "region".

In many access control models for XML documents, a user is required to have rules that, explicitly or implicitly, propagate an element's accessibility to its descendents. Therefore, the ancestor accessibility property holds across the entire document.

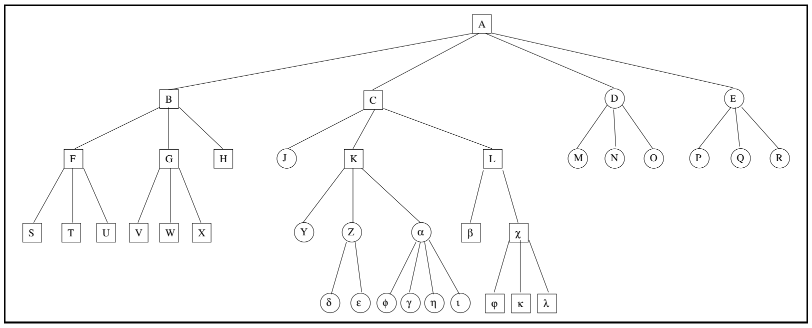
When the ancestor accessibility property does not hold over the entire marked database tree, it is easy to partition the tree into regions that satisfy the ancestor accessibility property. We call each resulting database subtree a *unit region.* The algorithm to partition a given marked database tree into unit regions is very simple: Perform a traversal of the database tree, and mark each node u that is accessible but the parent node of u is inaccessible. Each such node is called a *marker* node, since it demarcates the boundary of a unit region. The subtree rooted at a marker node (or at the root node of the entire database tree) is a unit region, excluding any descendant marker nodes and their associated subtrees. In general, we expect the number of unit regions in a database tree not to be too large.

For expositional reasons, we first develop the compressed accessibility map for a single unit region, in the next several sections, and then, in Section 6, deal with database trees comprised of multiple unit regions.

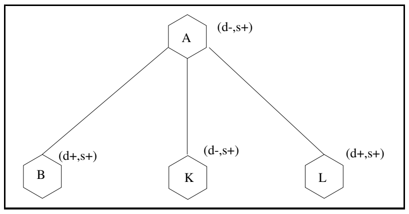
3 Unit Region CAM

Not all nodes need be labeled explicitly, since accessibility of many nodes can be determined by the d label of its closest labeled ancestor. In addition, some other nodes do not require explicit labels on account of the ancestor accessibility property, as we shall see below. Moreover, in view of the semantics of these labels and the ancestor accessibility property, it will follow that the label $(d+, s-)$ can always be replaced by $(d-, s-)$.

[2] Note that the CAM only serves to indicate whether or not a named node is accessible, not whether or not a named node exists in the database tree.



(a) A marked database tree; square nodes are accessible, round nodes are not



(b) An optimal CAM

Figure 1: Compressing a Marked Database Tree

Example 3.1 An example marked database tree, which is an abstract representation of an XML document, is shown in Figure 1(a), where square nodes are accessible and round nodes are not. The CAM of the tree is shown in Figure 1(b).

Nodes Y, Z, α and all their descendents can be inferred to be inaccessible on account of the $d-$ label at their nearest labeled ancestor, K. Node K itself can be inferred to be accessible because of its own $s+$ label. Node C would have been considered inaccessible on account of the $d-$ label at its nearest labeled ancestor, A. Nonetheless, the $s+$ label at C's child, K, causes C to be inferred to be accessible as well. ▮

A *labeling* K of a database tree H is a partial function that assigns labels of the form $(d+, s+)$, $(d-, s+)$, or $(d-, s-)$ to (some) nodes in H. We call the database tree along with its labeling, a *labeled database tree*, and denote it $\mathcal{T}$. Labeling K is complete whenever it is total, that is, each node is labeled. Given such a labeling, we can determine which nodes are accessible and which are not, based on the $s+$ or $s-$ label at the node, respectively. A (complete) labeling K is said to *respect* a marking M of a database tree H, if for each node in H, it is determined to be accessible by K exactly when it is accessible according to its marking in M. In the sequel, by a (complete) labeling for a marked tree, we mean one that respects the marking.

It is worth noting that there may be multiple complete labelings that respect any given marking of a database tree. For example, a marked database tree that has all its nodes accessible is respected both by the labeling that assigns label $(d+, s+)$ to each node, as well as by the labeling that assigns label $(d-, s+)$ to each node. The notion of induced labels, defined next, plays a role in determining which node labels in a labeled database tree are redundant.

Definition 3.1 [Induced Label] The *induced label* on a node e in a labeled database tree, written $L(e)$, is its label if one exists. Else, for unlabeled node e,

- If the nearest labeled ancestor of e has a label $(d+, s+)$, $L(e) = (d+, s+)$.
- If the nearest labeled ancestor of e has a label $(d-, s-)$, $L(e) = (d-, s-)$.
- If the nearest labeled ancestor of e has a label $(d-, s+)$,
  - If e has a descendant labeled with either $(d-, s+)$ or $(d+, s+)$, $L(e) = (d-, s+)$
  - Else $L(e) = (d-, s-)$.
- If e has no labeled ancestor, $L(e)$ is undefined. ▮

Note that in examining the labels of ancestors and descendants, only original labels should be considered, not induced labels. It is easy to see that the induced label on a node is uniquely defined, and does not depend on the order in which nodes are considered.

For example, given the labeling shown in Figure 1(b), the induced labels at (some of the) nodes in Figure 1(a) are as follows: B, F and G get $(d+, s+)$, A, K and C get $(d-, s+)$, and J, D and E get $(d-, s-)$.

Definition 3.2 [Accessible Node] A node e in a labeled database tree $\mathcal{T}$ is *accessible* if (i) it has an induced label of $(d+, s+)$ or $(d-, s+)$, or (ii) its induced label is undefined, and it has an accessible child node. Node e in $\mathcal{T}$ is inaccessible otherwise. ▮

For example, it is easily verified that, given the labeling shown in Figure 1(b), the accessible nodes of Figure 1(a) are precisely the square nodes.

A *Compressed Accessibility Map* (CAM) retains selected labels from some complete labeling such that the *accessibility* is determined correctly for each node. Note that the induced labels from the CAM may not recover the original labeling exactly.

Given a CAM (i.e., labeling) K of a marked database tree H, we can represent the CAM succinctly as a *reduced graph*, defined as follows: (i) it contains only nodes from H that are labeled in K, and (ii) for two labeled nodes x, y from H, there is an edge from x to y in the reduced graph iff x is the closest labeled ancestor of y in H, i.e., x is an ancestor of y that is labeled in K, and there is no labeled node z such that

z is a (proper) ancestor of y but a (proper) descendant of x in H. In general, the reduced graph of H is a forest instead of a tree. In that case, we add a dummy root to render it a tree. In the sequel, we identify the CAM with its reduced tree representation.

For example, consider the marked database H shown in Figure 1(a). Then the reduced tree of the labeling (CAM) K shown on the nodes of Figure 1(a) is as shown in Figure 1(b).

The next question is which labels can we leave out, while still determining unambiguously the accessibility of each node. We do this characterization using the notion of redundant labels below.

Definition 3.3 [Subsumed Label] A label at node e in a CAM $\mathcal{I}$ is said to be *subsumed* if it is identical to the induced label at e in a CAM $\mathcal{I}'$ obtained from $\mathcal{I}$ by rendering e unlabeled. ▮

Definition 3.4 [Upward Redundant Label] A label at node e in a CAM $\mathcal{I}$ is said to be *upward redundant* if

- e has an accessible (proper) descendant, and
- for every child c of e, either c is labeled in $\mathcal{I}$ or c is upward redundant. ▮

The intuition here is that the accessible descendant induces an $s+$ at e, and since all children are either labeled or upward redundant, the d-label at e is immaterial. So we do not need to record a label at e. Note that the d-label that e has, and the exact labels its children have (if at all), are immaterial as far as upward redundancy is concerned.

Definition 3.5 [Redundant Label] A label at node e in a CAM $\mathcal{I}$ is said to be *redundant* if it is either subsumed or upward redundant. ▮

Obviously, in any optimal CAM, no redundant labels are retained. How to construct one efficiently is the subject of the next section.

4 Optimal Unit Region CAM

The reduced tree representation of the CAM leads to the following natural definition:

Definition 4.1 [CAM Size] The *size* of a CAM is the number of labeled nodes in the CAM. ▮

Since the number of users with differing accessibility to the XML data can be large, it is important to minimize carefully the amount of storage required for the CAM of each user. Storage is determined by the number of labeled nodes. A given accessible projection of a database H can be represented by several equivalent labelings (CAMs). Our task is to determine a CAM of the smallest size. Conceptually, we can start with any complete labeling and delete all redundant labels. However, this naive strategy is not guaranteed to produce an optimal size CAM, owing to the following two complications.

First, given any complete labeling (that respects the database marking), it is not clear what is the order in which we should delete redundant labels: deleting one could render another no longer redundant. Consider, for example, a marked database tree that is a chain of two accessible nodes, and the labeling that labels each node $(d+, s+)$. The root's label is upward redundant, while the child's label is subsumed. Deleting either of these labels renders the other non-redundant. Indeed, in general, there are exponentially many orders in which to delete redundant labels.

Second, there are equivalent complete labelings, such that the minimal CAM afforded by one need not be the same size as the minimal CAM afforded by another. Consider, for example, the marked database tree with every node accessible, and the two complete labelings: $\mathcal{I}_1$, which labels each node by $(d+, s+)$, and $\mathcal{I}_2$, which labels each node by $(d-, s+)$. It is easy to see that these labelings are equivalent. The minimal CAM for $\mathcal{I}_1$ only labels the root node of the database tree with $(d+, s+)$. The minimal CAM for $\mathcal{I}_2$ needs to label the root and each leaf node of the database tree with $(d-, s+)$. Clearly, these (minimal CAMs) are not of the same size.

Thus, finding an optimal CAM for a given database tree is a non-trivial problem. We solve it by establishing some key properties of label redundancy, and use them to devise an efficient algorithm for constructing an optimal CAM.

4.1 Order of Redundancy Removal

Lemma 4.1 (Subsumption Order Invariance): Given a CAM $\mathcal{I}$ in which the labels at nodes e and f are subsumed, it must be the case that the label at f is subsumed in the CAM $\mathcal{I}'$ obtained from $\mathcal{I}$ by making e unlabeled. ▮

It is also easy to see that the order in which different upward redundant labels are removed is immaterial, since the definition of upward redundancy simply speaks of nodes with upward redundant labels, without consideration to whether these labels are present or have been removed. With a few additional observations, see [13], we can establish:

Theorem 4.1 (Subsumption First): Given a CAM $\mathcal{I}$, let $\mathcal{I}'$ be a restriction to $\mathcal{I}$, obtained by deleting subsumed labels and upward redundant labels in a specified order. The smallest size $\mathcal{I}'$ is obtained by deleting all subsumed labels first, and then deleting ancestor-free upward redundant labels depth-first from the root in a pre-order tree traversal. ▮

4.2 Optimal CAM Construction

Definition 4.2 [Terminal Node] A node e in a CAM is called a *terminal* node if e is accessible but none of its descendants is. When a terminal node is not a leaf, we call it an *internal terminal* node. ▮

The following lemma shows that we can safely label all internal terminal nodes $(d-, s+)$, without compromising optimality.

Lemma 4.2 (Terminal Label) : There exists an optimal labeling with every internal terminal node labeled $(d-, s+)$. ▮

Definition 4.3 [Positive and Negative Nodes] A node e is *positive* (resp., *negative*) provided

- e is an accessible (resp., inaccessible) leaf, *or*
- e is an internal non-terminal node with more positive (resp., negative) than negative (resp., positive) children. ▮

A declarative description of the optimal CAM construction algorithm is in Figure 2. It follows that a node's labeling will be deferred iff it is an internal non-terminal node which is neither positive nor negative.

Since counts of positive and negative children are required to label a node in Stage 1, a bottom-up (or a post-order tree traversal) procedure is suggested. A single traversal suffices for Stage 1, except possibly for the deferred labels. But each deferred label node is visited exactly one additional time. The rules in Stage 2 are procedurally best applied top-down. Once again, saturation with each rule requires consideration of each node in the database tree at most once. Adding this up, the total time required by the above optimal CAM construction algorithm is proportional to the number of nodes in the database tree. We illustrate the algorithm in the next section.

Example 4.1 Consider the database tree in Figure 1(a), which shows a marking in the form of square (for accessible) nodes and circle (for inaccessible) nodes, for a given user and access type.

Stage 1 of the algorithm produces the following labeling: (i) all the nodes in the subtree rooted at B are with label $(d+, s+)$; (ii) J$(d-, s-)$; (iii) all the descendents of K are with label $(d-, s-)$; (iv) K$(d-, s+)$ (terminal internal); (v) all the nodes in the subtree rooted at L are with label $(d+, s+)$; (vi) C(deferred); (vii) all the nodes in the subtree rooted at D are with label $(d-, s-)$; (viii) all the nodes in the subtree rooted at E are with label $(d-, s-)$; (ix) A$(d-, s+)$. This leads to the deferred label at C being eventually set to $(d-, s+)$, once the label of the root A is decided. In Stage 2, all labels are found redundant, and removed, except those at A,B,K,L. ▮

4.3 Optimality

We have the following observations.

1. For each internal node e, the label assigned by Stage 1 of our algorithm maximizes the number of subsumed children. More precisely, the number of subsumed children is no fewer than in any equivalent labeling.
2. When a node has an equal number of positive and negative children, Stage 1 gives it a label which makes it subsumed, unless the node is the root.

The couple of observations above are easily proved, and can be used to establish the following optimality results.

Lemma 4.3 (Good Labelings) : Let K be any complete labeling of a unit region database tree T. Let $\mathcal{I}$ be any CAM associated with K. Then there is an equivalent labeling K' produced by Stage 1 of our algorithm such that there is a CAM $\mathcal{I}'$ associated with K' which is no larger. ▮

Theorem 4.2 (Optimality) : Let H be any marked unit region database tree. Then the CAM obtained using our algorithm is of the smallest size among all CAMs associated with any complete labeling of T that respects its marking. ▮

5 Unit Region CAM Lookup

Recall that the identifier for any node in the CAM, just as for the database tree, is a prefix of the identifiers for each of its descendants. As such, we can store the CAM as a trie, keeping for each node its string extension relative to its parent along with its label, and introducing additional nodes to "factor out" any parent-relative string extensions common between siblings. Where the relative string extensions are long, string B-tree techniques [9] can be used to condense these. Using these well-known data representations, it is possible to store a CAM in space that is proportional to the number of labeled nodes, independent of the database size.

Having the (trie representation of the) optimal CAM for a complete labeling K and given a node e in the database tree H, checking whether e is accessible can be done quite efficiently, using Algorithm LookUp-UnitRegion-CAM in Figure 3. Intuitively, if node e is labeled in the CAM, we look up its label, and we are done. If node e is not labeled in the CAM, we locate the nearest labeled ancestor of e in the CAM, as well as a nearest labeled descendant (if any). Finding such "relatives" is particularly easy using the trie representation of the CAM. Given the (query) identifier string for the data item in question, traverse the trie beginning from the root. Once we get to a labeled node that has no labeled child that is a prefix of the query string,

```
Algorithm Opt-UnitRegion-CAM //given a marked database tree
    // Stage 1:
    label each accessible leaf $(d+, s+)$;
    label each inaccessible node $(d-, s-)$;
    label each terminal internal node $(d-, s+)$;
    for each non-terminal internal node, label it $(d+, s+)$ if it has more positive than negative children,
        or $(d-, s+)$ if it has more negative than positive children, and defer labeling otherwise;
    whenever a node gets a definite label, propagate it to any deferred descendants;
    when the root has an equal number of positive and negative children,
        give it an arbitrary label: $(d+, s+)$ or $(d-, s+)$;

    // Stage 2:
    apply rule 2 (delete subsumed labels) to saturation;
    apply rule $1'$ (delete ancestor-free upward redundant labels) to saturation;
```

Figure 2: Constructing an Optimal CAM

```
Algorithm LookUp-UnitRegion-CAM
    // given a CAM trie $C$, and a query node $e$ in database $H$
    let $f_1$ in $C$ denote the nearest labeled ancestor-or-self of $e$ (in $H$);
    let $f_2$ in $C$ denote the nearest labeled descendant (if any) of $e$ (in $H$);
    if ($f_1 = e$) return ($e$'s accessibility as determined by the $s*$ label of $f_1$);
    else if (label of $f_1$ is $(d-, s-)$) return ($e$ is inaccessible);
    else if (label of $f_1$ is $(d+, s+)$) return ($e$ is accessible);
    else // either $f_1$ does not exist or label of $f_1$ is $(d-, s+)$
        if (there is no $f_2$) return ($e$ is inaccessible);
        else return ($e$ is accessible); // by ancestor accessibility
```

Figure 3: Looking Up Accessibility of Node e Using a Unit Region CAM Trie

we have found the desired nearest labeled ancestor of the query node. (When we say "labeled child" of node u in the trie, we mean the nearest labeled descendant of u in the trie, ignoring any unlabeled nodes introduced on account of shared prefixes.) We simply read off the label. This node has a child corresponding to a suffix extension of the query string iff the query node has a labeled descendant. Thus, a simple trie lookup of the query node identifier is all it takes. This strategy takes time proportional to the product of the length of the query string and the logarithm of the size of the CAM. The correctness and efficiency of this algorithm are established by the following theorem.

Theorem 5.1 (CAM Lookup) : Given a CAM corresponding to a unit region database tree, Algorithm LookUp-UnitRegion-CAM correctly determines whether a specified node is accessible. Further, it does so in time proportional to the product of the depth of the node in the XML tree and log of the CAM size. ▮

By linking the nodes in the trie representation of the CAM back to the corresponding nodes in the database tree, the CAM can also be used to enumerate, for a given user and access type, the list of all data nodes accessible by the user. This can be achieved in time proportional to the size of the output plus the size of the CAM, but independent of the size of the original database tree.

6 Multiple Unit Regions

Recall, from Section 2.3, the notion of *marker nodes*, introduced to mark off regions of a database tree where the ancestor accessibility property holds locally: a marker node is an accessible child of an inaccessible node. Also recall that the unit region is a maximal subtree of the database tree that is rooted at either the tree root or at any marker node and excludes any marker descendants of the root of the unit region, as well as their descendants. We are interested in labelings that respect certain constraints.

1. $(d+, s+)$, $(d-, s+)$, $(d-, s-)$ are the only labels allowed, and
2. Every marker node has to be identified.

We call any labeling that respects these constraints a *constrained labeling*. The size of a constrained labeling is the number of nodes that are either labeled or marked by it. We seek optimal constrained labelings for database trees with multiple unit regions. We have the following proposition.

Proposition 6.1 (Marker Parent Label) : The parent of a marker node is never labeled in an optimal constrained labeling. ▮

We have the following straightforward algorithm for constructing constrained labelings for database trees

```
Algorithm LookUp-MultiRegion-CAM
    // given a CAM trie $C$, and a query node $e$ in database $H$
    let $f_1$ in $C$ denote the nearest labeled ancestor-or-self of $e$ (in $H$);
    let $f_2$ in $C$ denote the nearest labeled descendant (if any) of $e$ (in $H$);
    let $f_3$ in $C$ denote the labeled node that shares the longest common prefix with $e$;
    if ($f_1 = e$) return ($e$'s accessibility as determined by the $s*$ label of $f_1$);
    else if (label of $f_1$ is $(d-, s-)$) return ($e$ is inaccessible);
    else if (label of $f_1$ is $(d+, s+)$)
        if (($f_3$ is a marker node) and ($e$ is a descendant-or-self of the parent of $f_3$))
            // $e$ is a descendant of an IRT node
            return ($e$ is inaccessible)
        else return ($e$ is accessible)
    else // either $f_1$ does not exist or label of $f_1$ is $(d-, s+)$
        if ((there is no $f_2$) or ($f_2$ is a marker node))
            return ($e$ is inaccessible);
        else return ($e$ is accessible); // by ancestor accessibility
```

Figure 4: Looking Up Accessibility of Node e Using a Multi-Region CAM Trie

with multiple unit regions. First, perform a decomposition of the database tree H into unit region components. Reduce each unit region by deleting the subtree rooted at *the parent of* each marker node. Second, using Algorithm Opt-UnitRegion-CAM label each reduced unit region individually. Finally, take the union of these reduced unit region labelings, and mark each marker node. Call this modified algorithm Algorithm Opt-MultiRegion-CAM.

We can show that this algorithm always yields an optimal constrained labeling. A key lemma that needs to be established first is that if there is an optimal labeling of a unit region that assigns a label to its root, then Algorithm Opt-UnitRegion-CAM will produce such a labeling. The motivation is the following. We require marker nodes to be marked by all constrained labelings of the database tree. Suppose there is an optimal labeling that labels some (reduced) unit region root but the one produced by Algorithm Opt-UnitRegion-CAM does not label that root. Then the overall labeling obtained from Algorithm Opt-MultiRegion-CAM would mark an additional unlabeled node whereas in the supposed optimal labeling, we mark a labeled node. Marking a labeled node can be accomplished with one bit whereas marking a new (unlabeled) node costs additional space needed to identify the node. Thus, the overall labeling obtained from Algorithm Opt-MultiRegion-CAM would be suboptimal.

In comparing the sizes of constrained labelings, we charge for the number of nodes that are either labeled or marked. In particular, if a node is both marked and labeled, it is counted only once.[3] We now have the following theorem.

Theorem 6.1 (Optimality) : Let H be any marked database tree, possibly with multiple unit regions. Then the CAM obtained by applying Algorithm Opt-MultiRegion-CAM is of the smallest size among all constrained labelings of H. ▮

CAM lookup for a database tree with multiple unit regions is quite similar to the case for a single unit region, and is presented in Algorithm LookUp-MultiRegion-CAM, in Figure 4. The key difference with Algorithm LookUp-UnitRegion-CAM is that the lookup algorithm for multiple regions needs to account for marker nodes defining unit region boundaries. Specifically, our main lookup theorem is now restated as:

Theorem 6.2 (Multi-Region CAM Lookup) : Given a CAM corresponding to a multi-region database tree, Algorithm LookUp-MultiRegion-CAM correctly determines whether a specified node is accessible. Further, it does so in time proportional to the product of the depth of the node in the XML tree and logarithm of the CAM size. ▮

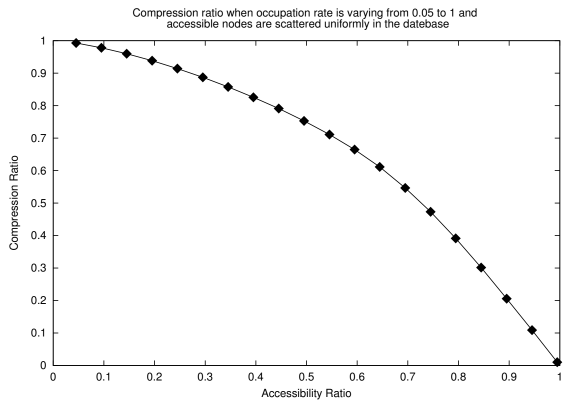
7 Experimental Verification

To validate the utility of a compressed accessibility map, we ran a variety of experiments that are intended to be indicative of how our techniques would perform for complex XML data that specifies access control information. We could not find a large production XML system with well-defined access control information; hence, we used synthetic XML data. However, hierarchical file systems (with real access control data) were readily available. Although our techniques are designed for supporting efficient access control for XML data, we made the (reasonable) assumption that the nature of access locality in XML data would be similar to that in hierarchical file systems.

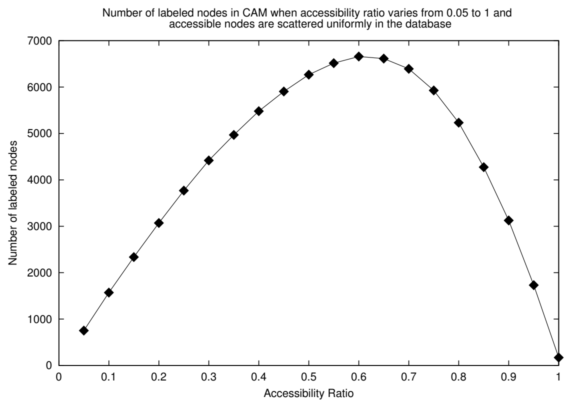
7.1 Experiments with Synthetic XML Data

To show how access locality affects the compression ratio, we experimented with synthetic XML data.

[3] It costs one extra bit to mark a labeled node, but this can safely be neglected. What is important is that it costs more to mark an unlabeled node.



(a) Compression Ratio versus Accessibility Ratio



(b) Space cost of CAM versus Accessibility Ratio

Figure 5: Accessible Nodes are Uniformly Distributed

The Data:

We used XML documents generated using IBM AlphaWorks' XML generator [7]. Several parameters are provided by the XML generator that can be used to control the structure of the generated XML documents (e.g., fanout and depth). We ran our experiments for a variety of XML documents and obtained essentially similar results. We report here results for a large representative XML tree, with 16811 nodes, with variable fanout (maximum = 60, minimum = 1, average = 2, not including the leaf nodes), and variable distances between leaves and root (average depth of 8), for different types of access locality.

No Access Locality:

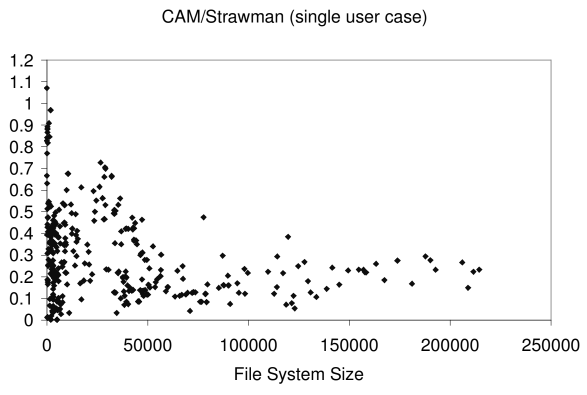
We first dealt with the case where accessible nodes are uniformly randomly distributed in the XML data. Note that access locality is the worst in this case. Figures 5(a) and 5(b) show the compression ratio and number of labeled nodes of the CAM, respectively, when the accessibility ratio is varied from 0.05 to 1. (The *accessibility ratio* is the fraction of nodes in the database that are accessible.)

From Figure 5(a), we can see that the larger is the accessibility ratio, the better is the compression ratio achieved by the CAM. The reason is that, while the space requirement of the fully materialized accessibility map grows linearly with the accessibility ratio, the space requirement of the CAM grows much more slowly. When the accessibility ratio is small, the fully materialized accessibility map requires less space. Therefore, even though the CAM does not require much space, the compression ratio is quite high. On the other hand, when the accessibility ratio is large, the fully materialized accessibility map requires far more space than the CAM, which makes the compression ratio much better. In particular, when the accessibility ratio is close to 1, accessible nodes tend to be close to each other, which results in better locality. In this situation, because of the way CAM takes advantage of access locality, it labels only a few nodes, thus takes little space cost.

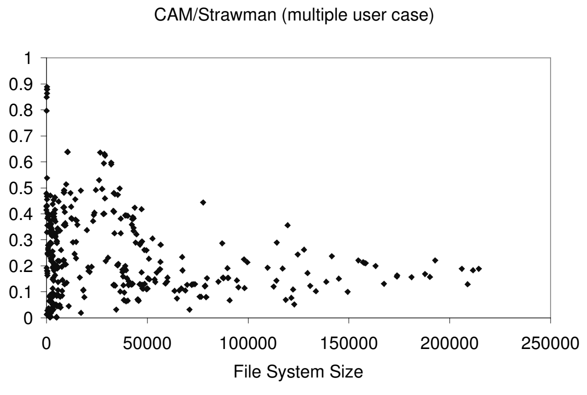
Varying Access Locality:

Here, we present experimental results for the case when the database has better access locality. In our synthetic XML data, a node is called *friendly* (resp., *non-friendly*) if the objects in the subtree rooted at that node have a high (resp., low) probability of being accessible. We call the subtree rooted at a friendly node (resp., non-friendly node) a *friendly area* (resp., *non-friendly area*). Two parameters of interest are af and anf, which define the access probability of a node in a friendly area and in a non-friendly area, respectively. Initially, we set the root to be a friendly node. To allow for multiple unit regions, we use two additional parameters. The *friend ratio*, or fr, is the probability that a node is a friendly node given that its parent is a non-friendly node. Similarly, the *reverse ratio*, or rr, is the probability that a node is a non-friendly node given that its parent is a friendly node.

To have better locality, fr should be small which will result in few unit regions in the whole database. In our experiment, we set fr to be 0.05. To get various accessibility ratios, we vary rr from 0 to 1. af and anf are set to be 0.98 and 0.02 respectively. Figure 7 shows the compression ratio when accessibility ratio changes from 0.07 to 0.98. It is quite clear that when accessibility ratio is very small (around 0.1), the compression ratio achieved by the CAM is not good. However, as the accessibility ratio increases, the compression ratio improves dramatically. Even for a relatively low accessibility ratio like 0.25, the compression ratio is still very impressive. Compared with Figure 5(a), we see clearly how access locality greatly affects the compression ratio that the CAM can achieve.



(a) Single user



(b) Multiple users

Figure 6: Compression Ratio versus File System Size

7.2 Experiments with Real File System Data

The Data:

We obtained access control information on 433 UNIX file systems at a large university. Our sample included a variety of file systems, ranging from file systems on "old" servers, with faculty users who had been with the University a long time, to new file systems on machines purchased recently. The file systems included the personal systems of a variety of users, students, visitors, and faculty, as well as data associated with several large data intensive projects and much system software.

The Metrics:

To demonstrate the space-efficiency of our CAM approach, for each file system, we compare the storage cost of the CAM with that of the fully materialized accessibility map, for multiple users. In the CAM approach, three CAMs (for read, write and execution permissions respectively) are maintained for each user. Thus, the total space cost of the CAM approach is the sum of the sizes of the CAMs of each user. In the fully materialized accessibility map, each object in the database maintains an access control list for each permission, which lists user IDs of the users who have read/write/execution access to that object, based on the standard UNIX semantics. Therefore, the space cost is the sum of the sizes of these lists over all the objects. Given a user set U, the *compression ratio* is calculated as the total space cost of the CAM approach for U over that of the fully materialized accessibility map.

Compression Ratio Results:

First, we examined the compression ratio for a single user. Figure 6(a) shows the compression ratio versus file system size for the worst of twenty users that we tested. We can see that when a file system is reasonably large (over 25,000 files), in most cases the compression ratio is quite good (around 0.2). Also, it is fairly clear that most cases of poor compression were for small file systems. Thus, one observes a trend towards better compression ratios as file system size increases.

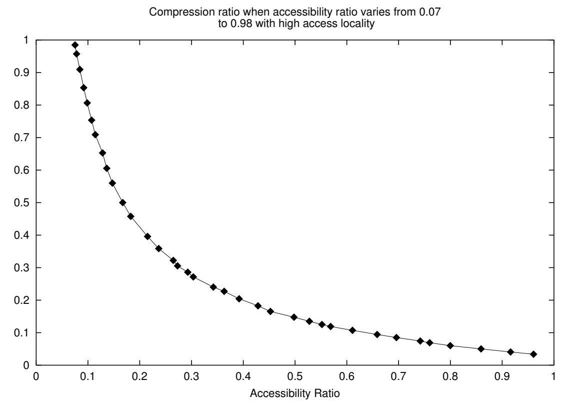


Figure 7: Compression ratio versus Accessibility Ratio when Locality is High

Figure 6(b) presents compression ratio versus file system size when U is set to be all the users of a file system. It is quite heartening that the chart of compression ratio for the multi-user case is very similar to that of the single user case in both shape and trend. The reason is that most users have access to only a small portion of the file system. Therefore they tend to have similar compression ratios which makes the overall compression ratio for multiple users similar to that of a single user.

To illustrate this point more clearly, we ran another experiment on one file system which is of size 78501 and with 294 users. Figure 8 shows how the compression ratio varies as the size of U increases. In the beginning, most users in U are superusers. Therefore,

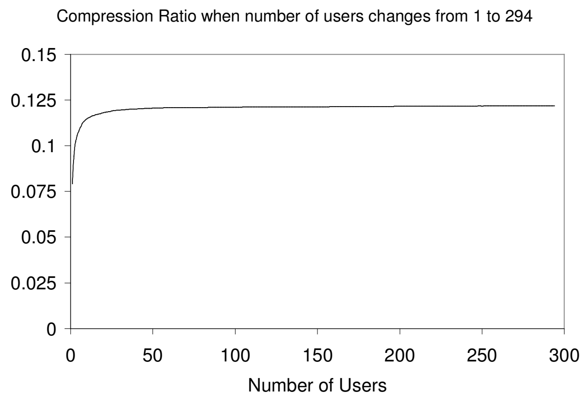


Figure 8: Compression Ratio versus # of Users

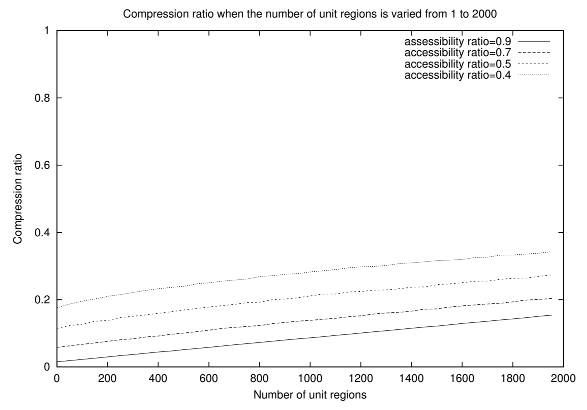


Figure 9: The effect of multiple regions

the compression ratio is very good. As more and more ordinary users are added to U, the compression ratio goes up to around 0.12 and remains constant even when U's size increases.

Another interesting observation is that the worst case behavior for a single user (shown in Figure 6) is for the user login "guest". This is because the access locality for a guest user (files with read permission for "other") is poorer than that of an ordinary user. By locality, we mean that the accessible objects tend to be located in one or more subtrees instead of being randomly scattered throughout the whole file system.

7.3 The Effect of Multiple Unit Regions

We also ran experiments over synthetic data to study how the number of unit regions in an XML database will affect the compression ratio that CAM can achieve. Our previous experiment results (figure 5(a) and 7) show that CAM's compression ratio partially depends on the accessibility ratio of the database. In order to demonstrate clearly the effect of multiple unit regions, it is desirable to only vary the number of unit regions while keeping the accessibility ratio relatively constant. To achieve this end, we introduce a parameter *propagation ratio* (pr for short) in the experiments, which controls the probability that a node is accessible when its parents is accessible. Given an XML database and the desired number of unit regions k, we first randomly select k nodes as marker nodes. Then in each unit region, we assign each node's accessibility according to pr. By this method, two databases with different number of unit regions will have relatively the same accessibility ratio as long as the same pr is used. Note that if the number of unit regions are comparable to the size of the XML database, the accessibility ratio will inevitably increase when number of unit regions increases. Therefore, we ran the experiments over a database with size 20000 and vary the number of unit regions from 1 to 2000. Figure 9 shows the compression ratio under different accessibility ratios. We can see that with the same accessibility ratio, the more unit regions a database has, the bigger the compression ratio CAM achieves. This agrees with our intuition that better locality (i.e., fewer unit regions) yields better compression ratio. Notice that because of the high locality of accessibility in the dataset, even when the accessibility ratios are pretty low, the corresponding compressing ratios are still very satisfactory. It is also interesting to observe that the compression ratio for each accessibility ratio increases linearly almost at the same rate along with the number of unit regions, which suggests that the impact of multiple unit regions is orthogonal to that of accessibility ratio.

8 Related Work

Bertino et al. [2] and Damiani et al. [5] deal with access control for XML documents. In particular, Damiani et al. develop an approach for expressing access control policies using XML syntax. The semantics of access control to a user is a particular view of the document(s) as determined by the relevant access control rules. They provide an elegant algorithm for computing this view using tree labeling. Our work is complementary to theirs in that we focus on efficient materialized policy evaluation rather than policy specification.

Commercial relational DBMSs support access control (e.g., see [12, 17]), and there is considerable literature on database security (e.g., see [6]). Access control techniques can broadly be divided into mandatory control techniques and discretionary control techniques. Important representative work in the former includes [16, 14, 19, 3]. Most of the work here uses some kind of belief logic-based semantics, where a secure database corresponds to a set of databases: subject s's database has precisely those tuples that s believes hold w.r.t. its clearance level. A CAM can be used to capture this view efficiently, if tuples had a suitable hierarchical organization imposed on them. However, most mandatory techniques tend to be relatively simple.

The idea behind discretionary control is that control is effected via views that are defined in the native query language of the data source. The view definitions may be used to explicitly define permissions or denials, depending on the type of "meta-policy" (such as "denials-take-precedence") one wants to adopt. Database systems have tended to lean more towards discretionary access control. The view definitions here can often be quite complex, and expensive to compute, so materialization as in a CAM can be quite valuable.

The history of work on discretionary access control begins as early as System R [11, 8]. Notable extensions and improvements include the ORION model [18], a model due to Gal-Oz et al. [10], and the model of Bertino et al. [1].

More recently, there has been interest in managing access control across multiple stores. For instance, Jajodia et al. [15] propose a datalog-like specification language, and show that many of the known meta-policies can be easily encoded in their language, with the intention that different meta-policies can be enforced for different data stores. Once more, materialization of the result of executing their specifications can be of value.

9 Conclusion

Transacting business over the Internet using XML is becoming more and more of a reality as we move towards a world of inter-networked data, with applications requiring access to data on the net. In this setting, there is a need for much more sophisticated types of access control than is permitted by simple firewalls. We envision that the policies (rules) that define access control to such networked data will interact in complex ways warranting mechanisms for efficient determination of whether or not a user has access to a given data item, given an access type. The compressed accessibility map is an important step towards realizing this vision.

The technical contributions of this paper include the design of a linear-time algorithm for finding an optimal CAM (for a given user and access type) for XML databases. With the aid of real-life and synthetic data for multiple users, we demonstrated the substantial savings a CAM can effect on the storage requirements for efficient access control policy enforcement.

In the full version of the paper [13], we also show that simple updates (additions/deletions) to a database can dramatically change the composition of an optimal CAM. However, with a small price (of a CAM 1 worse than the optimal), we can maintain near-optimality, very efficiently.

An interesting open problem is to take advantage of the commonalities between the access rights of users with respect to parts of the data to optimize the overall space consumed by the CAMs, while still guaranteeing fast lookup time.

References

[1] E. Bertino, P. Samarati, and S. Jajodia. An extended authorization model for relational databases. *IEEE TKDE*, 9(1):85–101, Jan. 1997.

[2] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Controlled access and dissemination of XML documents. In *ACM WIDM*, 1999.

[3] K. S. Candan, V. S. Subrahmanian, and S. Jajodia. Secure mediated databases. In *ICDE*, 1996.

[4] E. Damiani, S. di Vimercati, S. Paraboschi, and P. Samarati. Design and implementation of an access control processor for XML documents. In *WWW9/Computer Network 33(1-6)*, 59–75, 2000.

[5] E. Damiani, S. di Vimercati, S. Paraboschi, and P. Samarati. Securing XML documents. *EDBT*, 2000.

[6] S. Dawson, S. di Vimercati, P. Lincoln, and P. Samarati. Minimal data upgrading to prevent inference and association attacks. In *PODS*, 114–125, 1999.

[7] A.L. Diaz, D.Lovell. XML Generator. http://www.alphaworks.ibm.com/tech/xmlgenerator, September, 1999.

[8] R. Fagin. On an authorization mechanism. *ACM TODS*, 3(3):310–319, Sep. 1978.

[9] P. Ferragina and R. Grossi. The string B-tree: A new data structure for string search in external memory and its applications. *Journal of ACM*, 46(2):237–280, Mar. 1999.

[10] N. Gal-Oz, E. Gudes, and E. B. Fernandez. A model of methods access authorization in object-oriented databases. In *VLDB*, 1993.

[11] P. C. Griffiths and B. Wade. An authorization mechanism for a relational database system. *ACM TODS*, 1(3):242–255, Sep. 1976.

[12] Informix Software Inc., Menlo Park, CA. *Informix Online/Secure Security Features User's Guide*, 1993.

[13] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and T. Yu. Compressed accessibility map: Efficient access control for XML. Available from http://www.research.att.com/~divesh/papers/cam-full.ps.

[14] S. Jajodia and R. Sandhu. Toward a multilevel secure relational data model. In *PODS*, 1991.

[15] S. Jajodia, P. Samarati, V. S. Subrahmanian, and E. Bertino. A unified framework for enforcing multiple access control policies. In *SIGMOD*, 1997.

[16] T. F. Lunt. Secure distributed data views, vol. 1-4. Palo Alto, Calif, 1989. SRI International.

[17] Oracle Corp., Redwood City, CA. *Oracle7 Server Administrator's Guide*, December 1992.

[18] F. Rabitti, E. Bertino, W. Kim, and D. Woelk. A model of authorization for next-generation database systems. *ACM TODS*, 16(1):88–131, March 1991.

[19] M. Winslett, K. Smith, and X. Qian. Formal query languages for secure relational databases. *ACM TODS*, 19(4):626–662, 1994.