Privacy-Aware Access Control in XML Databases

Anders H. Landberg **J. Wenny Rahayu** **Eric Pardede**

Department of Computer Science & Computer Engineering
La Trobe University, Melbourne, Australia
Email: a.landberg@latrobe.edu.au, w.rahayu@latrobe.edu.au, e.pardede@latrobe.edu.au

Abstract

With the growing use of XML for data transfer and data storage across the web, securing XML documents has become an important issue. The XML privacy and data access control issues are especially significant in XML data repositories because they typically store large collections of highly sensitive business data, health information, etc. Protecting privacy by only restricting access to individual nodes in the XML document tree is not sufficient, as combinations of nodes and aggregations thereof can lead to disclosure of sensitive information. Moreover, a mechanism is required to cope with such combined data privacy levels, as they must be validated on query-time. Extending from XML access control models, this paper proposes a privacy-aware access control model for XML with composite security levels, which adds a further level of fine-granularity to existing approaches. In order to enforce these composite security levels, we then introduce a methodology based on path-triggers. Finally, we evaluate the performance of our new approach using three different implementation techniques.

1 Introduction

Privacy-aware access control is an important issue when storing and publishing sensitive information, such as financial and health data. We define privacy-aware access control as an access control model, which aims at securing data from access that may lead to privacy violation. So far, most existing work in the area have been focusing on security access level only. The most influential works have been proposed by Damiani [5][6][7] and Bertino [2], and many other approaches have been inspired by their work [4][12][9][13][8]. These works are based on static access rule definitions, and are not targeting privacy issues in connection with access control. The fundamental concept of these approaches is to define constraints on XML elements, attributes, and links, and thus enabling only authorised users or user groups with access to those data. These constraints are specified as XPath expressions, and are associated with levels of access. However, while static access control can secure a database from unauthorised access by specifying access rules, it does not guarantee it from privacy violation, which is caused by accessing combinations of nodes, and occurs at query time.

Example: Given an XML document instance (see Figure 1) and a schema definition that describes the document, we can apply access constraints to each and every node. Treatments and personal information should have a higher level of access than the ward. Personal information should have a higher access level than treatments, because a composition of its child nodes 'ZIP', 'Age', and 'Gender', reveal more detailed information about a patient and can easier lead to disclosure of the data.

When restricting access to data, it is also important to consider how it can be queried and how the access control mechanism functions. Further, it must be ensured that after the application of a privacy-aware access control model, users such as analysts or researchers are still able to retrieve useful information from the data. This is an issue because by restricting access to particular nodes, the query result may be useless.

While the existing solution is acceptable for single node access, it fails when restricting access to composition nodes. The reason for this is because these access control policies are defined on design-time. However, combined node access restrictions cannot be predetermined, because they depend on the query. Thus, these composition node privacy constraints must be validated on query-time.

1.1 Defects of static(single) privacy constraints

In a publicly accessible hospital XML data repository, where the specific identifiers (such as name) of patients have been removed, a combination of two or more identifier attributes may lead to the disclosure of a particular patient. For example (see Figure 1), an adversary knows that Mr. President had treatments on the 1st of June 2007, but he does not know which treatments were conducted. The adversary has access to those parts of the document with a privacy level of 3 or less (see Table 1). After querying the hospital's public domain statistics on 'Treatments' and 'Date', he finds that only one male patient received treatment on this date. The treatment being dialysis, the adversary now identifies Mr. President as the patient that had dialysis on the 1st of June 2007.

One way to overcome this problem using existing access control models, is to increase the level of access for nodes that will lead to disclosure when combined, or to completely restrict access to one of the nodes. In our example, these nodes are 'Treatments' and 'Date'. In this way it can be ensured that an adversary will fail to re-identify Mr. President, because one of the two nodes is not available for querying. However, this has created a new problem. Let us assume that we previously increased the access level of the 'Date' node, so that it was no longer accessible to the adversary. As a result of this, any other data anal-

Table 1: Access authorization definitions

Node	Level of access
/PatientRecords/PatientRecord/Date/	1
/PatientRecords/PatientRecord/Treatments/	2
/PatientRecords/PatientRecord/Ward/	2
/PatientRecords/PatientRecord/Personal/	3
/PatientRecords/PatientRecord/PreviousRecords/	1

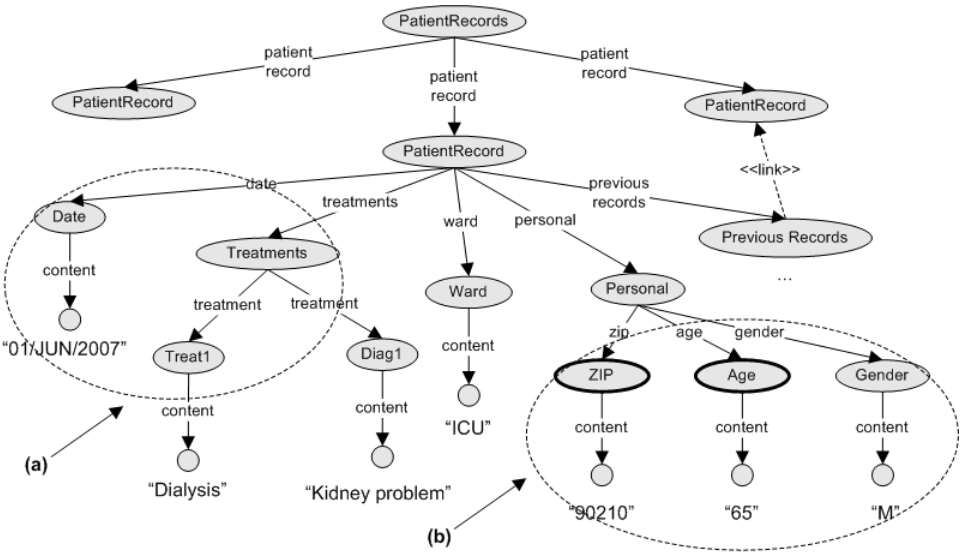

Figure 1: XML Document Instance: Patient Record

ysis queries issued by any user will not have access to the date node either, hence making the queries nearly useless (see Table 2). The same dilemma arises if we choose to restrict access to the 'Treatments' node, or some of its child-nodes.

Table 2: Inaccessible data. Access levels too high.

Node	Access Level
/../PatientRecord/Date/	**5**
/../PatientRecord/Treatments/	2
/../PatientRecord/Ward/	2
/../PatientRecord/Personal/	3
/../PatientRecord/PreviousRecords/	1

Table 3: Unnecessarily restricted data.

Node	Access Level
/../PatientRecord/	**7**
/../PatientRecord/Date/	(7)
/../PatientRecord/Treatments/	(7)
/../PatientRecord/Ward/	(7)
/../PatientRecord/Personal/	(7)

Another work-around solution to overcome this problem using existing approaches, is to identify the parent node of the sensitive nodes 'Date' and 'Treatments', and increase its level of access. An example of this scenario is given in Table 3, where the 'PatientRecord' node is given a very large access level, hence restricting access to any of its child nodes that lead to disclosure.

While this solution achieves sufficient privacy protection by restricting access to 'Date' and 'Treatments' nodes at the same time, it brings the side effect of also restricting access to these nodes individually and to all other nodes below the 'PatientRecord' node. In other words, a simple query on the 'Ward' node is not possible any longer, although this query cannot lead to privacy disclosure. Hence, this solution suffers unnecessarily restricted data and limited use.

This issue can not be avoided using existing access control models, as it is subject to the combination of nodes that is queried on run-time. Such constraints can not be pre-defined by approaches found in existing literature because they must be checked and evaluated when the query is executed. The reason behind this is that we cannot foresee which queries the user will issue, nor do we know the users' intentions.

Referring back to the access levels that we defined in Table 1, we can also see that some users (with access level less than 3) can not access the personal data such as 'Age' and 'Gender', and thus, making data analysis almost useless (see (b) in Figure 1). As far as existing works are concerned, we can only attempt to remedy the problem by specifying individual access levels to each child node of the personal node. However, this does not solve the problem of restricting composition node access.

1.2 Contributions

To overcome the defects of previous approaches, we propose a novel privacy-aware access control model for XML that captures query-time access control for combined access to nodes. The goal of security level composition is to group nodes in the XML data (see Table 4), which when accessed in combination, may lead to disclosure of individual patient records, and further to the re-identification of individual patients. While the individual access levels for the nodes within a combined security level can remain unchanged, a higher level of access will be required when attempt-

ing to access the nodes together. This means that the information in the data can be protected much stronger, yet granting less restrictive access to individual nodes, which is crucial when conducting data analysis by querying.

It is a well-known problem in the area of data privacy that a combination of attributes can lead to disclosure of sensitive information [15]. Determining the combinations of data that lead to privacy disclosure is a research field in itself and goes beyond the scope of this paper. We do not offer a new solution to arrange the data in a privacy preserving way, but propose concepts and methods to enforce these principles.

By introducing the notion of *composite security levels CSL*, our model provides a tool to define privacy-aware access constraints over sets of sensitive elements on a schema definition, or directly within the XML data itself, where a security level can be attached to a set of aggregated values or an instance of XML data. Secondly, we propose a new mechanism to implement the combined privacy-aware access control model with the use of XML triggers. We show that combined privacy-aware access constraints can be maintained in a cost-efficient manner by using this new privacy preserving access control mechanism.

The rest of the paper is organised as follows. Section 2 summarises related work in the area of access control for XML, section 3 formalises our methodology, section 4 describes how our model is enforced with the use of XML triggers, and section 5 presents the analysis of our approach. Finally, section 6 concludes the paper.

2 Related Work

In the area of access control for XML, an early work is represented by Samarati et al. [14] that proposes an access control model for HTML documents. This approach is clearly limited by the semantic deficits of HTML as opposed to XML and as such, the model by Samarati et al. is limited in many ways when defining access policies as well as partitioning a document into meaningful segments for an access control approach.

Damiani et al. [6] propose a method to define user groups and access authorisations in XML documents. User groups can be granted different types of access (such as read, write) to certain parts of the document by defining access authorisations. These access authorisations are described based on subjects and objects. Subjects are accessing entities such as users, and objects are path expressions that identify parts in an XML document.

Damiani's paper offers a detailed approach to define data access control in XML documents. However, it is based on the existence of a schema (DTD), and cannot be applied to (schema-less) XML document instances themselves. The access policies are restrictive (either "+" or "-" for access or no access), and there is no differentiation between different levels of confidentiality of the data. By contrast, our model allows for both data-level and schema-level definition of security levels. Also, our concept of combined security levels is a novel approach in this area.

Bertino et al. [2][3] propose an XML data access model by introducing various levels of protection granularity, such as document/DTD, set of documents, (sub)elements, attributes, and links. By applying these protection levels to policies, a fine-grained and detailed access control is now possible. The authors also focus on different levels of well-formedness for XML documents, and make suggestions on which policies to apply in which case.

Our paper can be differentiated from [2] and [3] in the following areas. First, their access control policies are defined 'outside' the document instances. Although the authors claim to define access policies for document instances, this approach is ultimately still schema-based. This means that if these definitions are lost or corrupted, then the data will be unsecured and vulnerable to misuse. Our model in contrast can be applied to a schema, as well as defined in the data itself, yet maintains *propagation* (inheritance) of security levels throughout the document structure. Second, there is no mentioning about the existence of combined security levels or similar phenomena.

Kuper et al. [10] use views over XML data to restrict access. It is important to point out the significant difference between this approach and ours. By creating (static) views of the XML data, Kuper et. al. hard-code the security model, instead of (dynamically) validating it on query-time. This means that after changes in the data, the views possibly need to be updated for the security model to remain valid. On the other side, our model is immune to such changes, and any security levels and composite security levels still apply even after data modification.

A large number of works in the area of XML access control and privacy have been proposed that are inspired by the works of Damiani and Bertino [4][12][9][13][8]. The major rationale behind all approaches, however, is to bind access or privacy policies with particular nodes and/or attributes, which we will refer to as single, or static privacy levels. The reason for this is because the definition of access control and privacy protection policies and levels takes place prior to the querying of the data, hence static, and cannot be dynamically modified on query-time.

The major difference between previous works and ours is that we focus our privacy model on privacy control validation of composition nodes that is performed at run-time, rather than a static privacy definition scheme. Our model can be implemented in the XML data, and also for schema.

3 Proposed Method

This section formalises the proposed security model by focusing on the process of expanding and labelling nodes in an XML document tree. It explains how the hierarchical structure influences the security constraints throughout the document tree, and provides a methodology on how the security constraints can be implemented. We use Figure 2 to illustrate the new concepts and methodology.

3.1 Definitions and Rules

Definition 1. A *security level SL* is defined as *SL(entity) = value*, where *entity* $\in$ {*user, system, node*}, and *value* specifies the security level where $1 \leq value \leq max$.

Property 1. Unless otherwise defined, the security level *SL* along a *path* $P = N_1/N_2/.../N_m$ must be incremental or equal, such that $SL(N_i) \leq SL(N_{i+1})$ where $1 \leq i \leq m\text{-}1$.

For the following definitions and examples, we will use entity instances *E1*, *E2*, and *User1* with the properties E1 $\in$ {user,system}, E2 $\in$ {node}, User1 $\in$ {user,system}, SL(E1) = 6, SL(E2)=4, and SL(User1)=3.

Example 1. Given an XML schema as shown in Figure 2, *SL(User1) = 3* defines a security level of *value = 3* for entity *User1*. *SL(PatientRecords)=1* defines security level of *value = 1* for entity *PatientRecords*, which in this case is the *PatientRecords*

Table 4: Privacy protected. Data accessible.

Node	Single SL	Composite SL
/PatientRecords/PatientRecord/Date/	1	5
/PatientRecords/PatientRecord/Treatments/	2	
/PatientRecords/PatientRecord/Ward/	2	
/PatientRecords/PatientRecord/Personal/	3	
/PatientRecords/PatientRecord/Personal/ZIP/	(3)	7
/PatientRecords/PatientRecord/Personal/Age/	(3)	
/PatientRecords/PatientRecord/Personal/Gender/	(3)	
/PatientRecords/PatientRecord/PreviousRecords/	1	

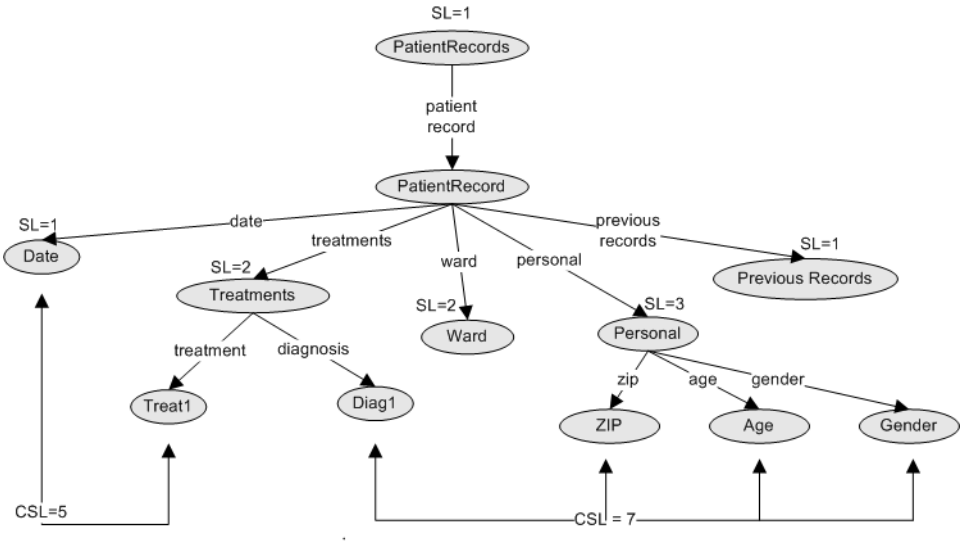

Figure 2: Privacy model applied on XML schema

node.

Rule 1.1 If a node does not specify *SL*, then its *SL* is inherited from its nearest ancestor that specifies *SL*.

Rule 1.2 Security level grouping: if *SL* is equal for all child nodes of a parent node, then this *SL* applies to the parent node.

Definition 2. Two entities *E1* and *E2* are defined as matching, when $SL(E1) \geq SL(E2)$, where $E1 \in \{user, system\}$, and $E2 \in \{node\}$.

Example 2. Given an XML schema as shown in Figure 2, and given $SL(User1) = 3$, and given $SL(Personal)=3$, $SL(User1)$ and $SL(Personal)$ are said to be matching because $SL(User1) \geq SL(Personal)$, and $User1 \in \{user, system\}$, and $Personal \in \{node\}$.

Definition 3. A *composite security level CSL* is defined as a tuple *(CN,SL)*, where *CN* is a set of *composite nodes* and a *SL* is a *security level*.
$CSL = \{CN, SL(entity) | \forall N \in CN : SL(N) \leq SL(entity)\}$.

Example 3. The *SL* of a *CSL* applies to all nodes in *CN* if all nodes in *CN* are accessed together.

Definition 4. Given a *CSL(CN, SL(entity))*, and given an entity *E1*, *E1* and *CSL (CN, SL)* are defined as *matching*, when $\forall\ N \in CN : SL(E1) \geq SL(N)$, and $SL(E1) \geq SL(entity)$, where $E1 \in \{user,system\}$.

Example 4. Given an XML schema as shown in Figure 2, and given a *CSL (CN, SL(entity))* where *CN*={*'ZIP', 'Age', 'Gender', 'Diag1'*}, and *SL(entity)=7*, and given *E1* where *SL(E1)=7*, *E1* is said to be matching with *CSL(CN,SL)* because $SL(E1) \geq SL(entity)$ and $\forall\ N \in CN : SL(E1) \geq SL(N)$.

The ancestor nodes of nodes that participate in a CSL are not affected by an increased level of privacy. This means that the SL of any ancestor nodes relative to the CSL participants remains unchanged if a CSL applies during validation. As for the descendant nodes of CSL participants, the following rule applies.

Rule 2. If a node N participates in a CSL, and if this CSL applies during a data access operation, then all descendants of this node N inherit the security level of the CSL. Exception: If a descendant node D of node N has a security level that is greater or equal to the security level of the CSL, then the security level of node D remains unchanged, and is treated according to Definition 2.

The above definitions and rules have described our privacy model. In the following section, Algorithm 1 represents the access control mechanism. It is used to detect security levels in the XML data, and to match these against the accessing entity's security level.

Algorithm. The algorithm first validates basic SL's (security levels), and in the same process, identifies and validates composite security levels. The CSL validation is divided into three parts: (i) identification of CSLid's (CSL tags), (ii) identification of participating nodes for these CSL id's, and (iii) the actual matching of the CSL's security level (if the CSL applies) against the accessing entity's security level.

Function query_node_instances(D, path_expr) re-

Algorithm 1: Matching Algorithm

```
Data:
D : XML document
selected_nodes: Set of selected node types as
path expressions
SL(accessing_entity): accessing entity's security
level
Result: b: returns true if access is granted,
        false otherwise
begin
    foreach path_expr in SN do
        node_list ⟵ query_node_instances(D,
        path_expr)
        foreach node in node_list do
            if node.SL > SL(accessing_entity)
            then
                return false
            else
                continue
            if has_attribute(node, "CSLid") and
            not node.CSLid ∈ csl_list then
                part_nodes ⟵
                query_dct_node_typ(D, all_paths,
                "CSLid")
                if selected_nodes ⊆ part_nodes
                and node.CSL >
                SL(accessing_entity) then
                    return false
                else
                    continue
            else
                add(csl_list, node.CSLid)
    return true
end
```

turns a set of node instances from document D as specified by path expression *path_expr*. Function has_attribute (node, "CSLid") returns true if *node* has attribute *"CSLid"*, false otherwise. Function query_dct_node_typ(D, all_paths, "CSLid") returns a set of all distinct node types from document D that have attribute *"CSLid"*. Function add(csl_list, node.CSLid) adds attribute value *node.CSLid* to list *csl_list*.

In regards to Rule 2, this algorithm is applied recursively to all descendant nodes of a given context node, which is being validated during data access.

4 XML Triggers for Privacy-Aware Access Control

Single and composite security level validation as proposed in the matching algorithm, can efficiently and effectively be realised using XML triggers. We have introduced the notion of XML triggers as a mechanism to maintain XML data consistency after a sequence of updates [11]. In this section, we adopt a triggering mechanism to activate a certain access policy each time a query that may violate privacy rules is executed. We believe that this is an effective and unique way of dynamically firing an access control in the database layer.

4.1 Single security level validation trigger (SSLVT)

A single security level is either hard coded into the document as an attribute 'SL' (document scope), or the security level is defined outside the document with an XPath expression. In the latter case, the security level applies to the particular element in all documents that are associated with that schema (schema scope).

To create a mechanism for validation of this single security level, we first create a new path-trigger, and apply it to the node (expressed by *tp*) that is to be validated upon access. It requires the trigger's event to be 'access', so whenever there is an access to this node, then the trigger is fired. The event 'access' is not available in [11], therefore we must modify the trigger's event options accordingly.

The target path *tp* of a SSLVT is the XPath expression that specifies the node to be protected by the single security level, and has the form {<XPath expr>}.

The next step is to adjust the trigger's action-body appropriately. The attribute that stores the security level must now be compared to the accessing entity's security level. The latter must be either hard-coded into the trigger, or retrieved/passed dynamically to the trigger on run-time. Function

```
Trigger definition: SSLVT := {e, c, a}
e := {access tp}
c := {SL(entity) < $con/@SL} ; for document scope
c := {SL(entity) < node_SL($con)} ; for schema scope
a := {RETURN $con/*[@SL <= SL(entity)]}
```

node_SL(path_expr) takes as input an XPath expression and returns the security level for it. SL(entity) specifies the security level of the accessing entity.

Although we only consider read-operations (read-only queries) for our privacy model, it can be clearly seen that the path-trigger mechanism is capable of supporting other operations such as 'insert', 'delete', and 'update' as well. Compared with other approaches [6][2], these operations would be similar to 'write', 'remove', and 'replace' respectively.

Example. As an example trigger implementation, we first declare a security level for target node $tp = /../PatientRecord/Ward$, and set its SL = 1. This implies that when we declare the trigger, the event will be $e = accesstp$. We must explictly specify which implementation type we use, so that the correct condition is chosen. In this case we chose schema type. Thus, the condition of the trigger will be $c := \{SL(entity) < node\_SL(\$con)\}$, where $con is the XPath expression specifying the context node that is being evaluated by the trigger.

4.2 Composite security levels validation trigger (CSLVT)

If two or more elements participate in a composite security level, then these elements specify two additional attributes, namely the id of the CSL, and the security level of the CSL. These values can either be directly hard-coded into XML data (document scope), or specified in an XML schema definition (schema scope). Alternatively, a CSL can be defined as a set of XPath expressions that specify the participating nodes in the CSL, and a security level (a number) that applies for the CSL.

The strategy for implementing a path-trigger that validates a composite security level is similar to the above described one. The differences are that to validate a CSL, we must first identify the closest common ancestor node (this will be the context node for the trigger [11]), and then implement the trigger's action-body in such a way that all participating nodes are retrieved, and their security level for the CSL is matched against the accessing entity's security level.

The target path *tp* of a CSLVT is the path that specifies the closest common ancestor node, relative

to all the participating nodes in *CN*, and has the form {<XPath expr>}.

The reason why we need to find the closest common ancestor node of all participating nodes, is because from this point in the XML document structure, the number of node traversals to each of the participating nodes, is minimised. Also, the path-trigger will automatically traverse all context paths relative to the context node, and in this way allow direct access to possible participants of CSLs.

```
Trigger definition: CSLVT := {e, c, a}
e := {access tp}
c := true
a := { IF $path IN S(CN) THEN mark($path) END IF;
IF getMarked(CSL id) EQUALS S(CN) THEN
IF SL(entity) < SL(CSL id) THEN
THROW EXCEPTION
END IF;
END IF; }
```

Function mark(path_expr) marks a path as belonging to the CSL that is to be validated. Function getMarked(CSL id) returns the set of marked nodes for the CSL with *CSL id*.

The most time consuming task in this process is to determine whether a CSL applies or not, e.g. whether all or a subset of the retrieved nodes by a query are all participating in one and the same CSL. To do this, we must first retrieve the set of nodes that participate in the CSL with the respective id that is to be validated. Then, this set of nodes must be compared against the set of participating nodes that are actually retrieved during the query. If the sets match, then the CSL applies, and the security level of CSL and accessing entity can be matched against each other.

Example. Referring to Figure 2 and Table 5, this example shall demonstrate a practical application of the CSLTV mechanism. Table 5 lists a number of sample queries that are issued by accessing entities. Each of the entities have an associated security level that is used to determine whether they have access to the XML contents specified by the query.

The 'Public' user profile can access nodes that have an SL below their own (query 1), but when attempting to access nodes on which a CSL applies (query 2), then access will be denied. In this case the selection in the query is made based upon the @SL attribute.

5 Analysis

For a more realistic and fair comparison of our method, we conducted the tests in three different modes. These modes are (i) 'Cold-run', (ii) 'View', and (iii) 'Trigger' respectively. In 'Cold-run' mode, security levels and composite security levels are validated on query-time, and are implemented as preconditions of the respective queries. In 'View' mode, only composite security levels are validated on run time, single security levels are implemented using views of the XML data. In 'Trigger' mode, XML triggers are used to implement the privacy model. It is obvious that a system without access control will perform better, as access control mechanisms add performance overhead.

5.1 Experimental Environment

The tests described in this section were conducted using the above described implementation using XML triggers on a machine equipped with a 2.0GHz Intel Pentium M 760 processor and 1GB DDR2 memory, running Windows XP professional as operating system. We used the Sigmod-Record in XML format [1] to build the XML repository, with the main document holding a total of 23047 nodes. The XPath queries were conducted in such a way that the CSLs were activated and executed.

We define a *computational cost unit* as the cost of traversing a node in the XML document tree.

5.2 Performance Results

The results of our tests in respect to composite security levels can be summarised as follows. Both the cold-run (CR) mode and the view (VW) mode showed a linear increase by 5 cost units per additional CSL that had to be validated. The trigger (TR) mode performed at a linear increase by 3 cost units, and therefore 60% better than the former traditional modes.

As we expected, the computational cost of validating an incremental number of CSLs increases linearly. Maintaining a privacy preservation model will always incur an overhead, but the results show us that the mechanism we use helps to significantly reduce the cost. We can clearly see that the trigger mode outperforms both other modes, since pre-processing of single security levels is performed when the trigger is created, and context paths can be pre-cached by the path-trigger to save computation time on query-time.

Traditional approaches rely on the 'Cold-Run' mode, where all access validation is performed at query-time. Some more recent approaches that use views [10][1] rely on what we call 'View' mode, where a set of access control and privacy constraints are 'hard-coded' into the database or repository as a view. While the latter of these two modes performs slightly better than the cold-run, it still suffers the deficit of being incapable of supporting the composite security levels to a satisfactory degree.

The tests in path-trigger mode achieve much better results, because the trigger creates a dynamic compilation of the participating node paths when it is created. This has the effect that query-time assembly of participating nodes is reduced, and hence computation time can be kept to a minimum. Further computational cost is saved when composite security levels overlap, which will be discussed in the next section. This scenario of overlapping nodes for CSLs is common, because generally a set of quasi-identifying nodes in combination are the reason for privacy disclosure of sensitive nodes. Therefore, CSLs will most likely apply to these sets of quasi-identifiers with one or more additional sensitive nodes.

Table 6: CSL Overlap, n=10

k	max CSL	ovlp	ovlp/CSL (%)
2	45	9	20%
3	120	36	30%
4	210	84	40%
5	252	126	50%
6	210	126	60%
7	120	84	70%
8	45	36	80%
9	10	9	90%
10	1	1	100%

5.3 Scalability

The maximum number of composite security levels with a cardinality k that can be applied to an XML

[1] Available for download at http://www.sigmod.org/record/xml

Table 5: Sample XPath queries

Query	Accessing Entity	Access
/PatientRecords/PatientRecord/*[@SL=1]	Public (SL=4)	yes
/PatientRecords/PatientRecord/*[@SL<=3]	Public (SL=4)	no
/PatientRecords/PatientRecord/*[position()<3]	Researcher (SL=6)	yes
/PatientRecords/PatientRecord/*[position()>1]	Researcher (SL=6)	no
/PatientRecords/PatientRecord/	Admin (SL=10)	yes
/PatientRecords/PatientRecord/	Web-Service (SL=4)	no

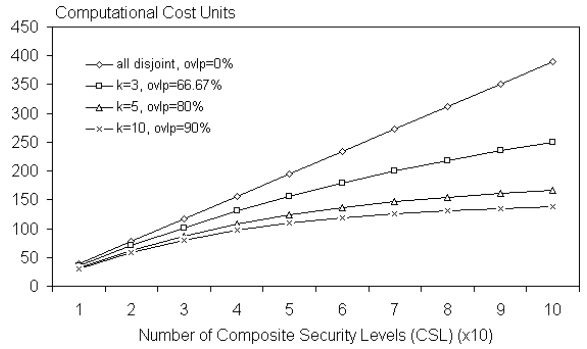

Figure 3: Validating CSLs

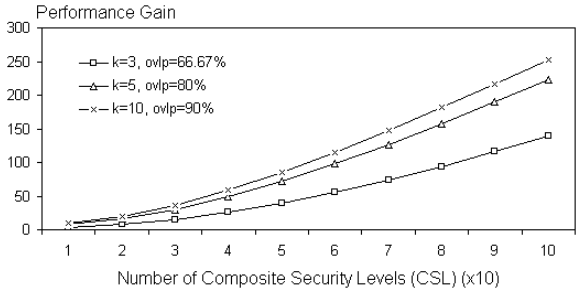

Figure 4: Performance gain

structure containing n leaf nodes is specified by the binomial coefficient $C_n^k = \binom{n}{k} = \frac{n!}{k!(n-k)!}$, which is also known as the *choose function*. We apply this function to our method to measure how many CSLs will possibly be generated based on a given XML structure.

Table 6 presents the maximum number of CSLs that can be generated with a given cardinality k, the number of nodes that overlap for the CSLs, and the percentage of overlapping nodes per CSL. From these results we derive a cost model that represents the percentage of overlap per CSL: $CSL_{ovlp} = \frac{100}{n}$ with parameter n that denotes tree cardinality. We calculate the number of overlapping nodes with the formula $n_{ovlp} = C_{n-1}^{k-1} = \binom{n-1}{k-1} = \frac{(n-1)!}{k!(n-k)!}$.

The performance gain is dependent on the cardinality of the grain of the XML document, to which the CSLs are being applied. As an example, if we have a document structure as illustrated in Figure 2, then the grain of the document is a 'PatientRecord' node, and the cardinality n thereof is 7, given that there is only one 'Treat1' and 'Diag1' node below the 'Treatments' node. The performance gain that can be achieved in this document yields $ovlp = \frac{100}{7} = 14.3\%$.

The maximum CSL overlap percentage MAX(CSL_{ovlp}) is dependent on the cardinality of the CSLs, and is calculated using the following formula. $CSL_{ovlp} = 100 * (1 - \frac{1}{k})$. An increasing possible overlap of CSLs occurs with increasing variable k and constant n. For example, given two CSLs with k=2 each, then the overlap can be at most 50%, and given three CSLs with k=3 each, then $MAX(CSL_{ovlp}) = 66.67\%$.

A low number of CSLs indicates a likelihood of the CSLs to be disjoint. With increasing number of CSLs, the likelihood of overlapping among these CSLs increases, and hence brings performance gain (see Figure 3). The first (upper, 0% overlap) graph show performance for the worst-case when there is no overlap (all CSLs disjoint), in which case the performance is a linear graph with a constant increase in computational cost for each additional CSL.

The next three graphs represent performance for CSL cardinalities k=3, k=5, and k=10 respectively. It can be seen that with increasing overlap percentage, the performance stabilises and will become constant once there is a 100% overlap of CSLs.

Figure 4 shows that with moderate CSL cardinalities, a substantial performance gain can be achieved.

Summary

The performance evaluation revealed that the new proposed concept of composite security levels does not add any substantial overhead to the access control mechanism. This is particularly visible in the comparison of cold-run and view modes against trigger mode. Whereas the first modes only slightly vary in cost, the trigger mode clearly outperforms both other modes. So we can say that with increasing number of CSLs of variable depths, participating nodes, and levels of distribution, the performance remains stable. This means that an implementation of our method does not add much more cost than already existing access constraint models, but provides a range of additional new functionalities.

Moreover, the scalability analysis showed that there exists an upper limit of CSLs that can be applied to XML documents, and that there is a potential performance gain of up to 50% for the validation of overlapping CSLs. Our derived cost model can be used to forecast performance gain, and consequently the expected computation time can be calculated.

6 Conclusion and Future Work

This paper has proposed (i) instance/schema level data access control for XML documents, (ii) composite security levels, and (iii) levels of access, offering a more rigorous concept for XML access control. The definitions and methodologies that are introduced in this paper were used to incorporate privacy-aware data access control into an existing XML data repository using XML triggers. Hence, XML data access control has been extended, so that XML document nodes can be assigned individual security levels, and combinations of nodes can also be given an additional (higher) level of security. A case study and analysis were used to demonstrate and quantify performance of the proposed concepts. Hereby, the new access control method was compared for three different test modes. Our scalability analysis showed that CSLs can be implemented in a cost-effective manner.

There are several avenues for future work. First, we want to show how the CSL approach can be integrated into query processing algorithms of (native) XML databases. In this way we wish to prove the applicability of the CSL mechanism. Second, we plan to conduct extensive experiments and case studies using more complex XML data in order to further prove scalability and robustness.

The conclusion of this paper is, that the proposed concepts of data access control have proven to be a beneficial extension to existing access models and XML, and that these new concepts also can be implemented and are scalable.

References

[1] P. Ayyagari, P. Mitra, D. Lee, P. Liu, and W.-C. Lee. Incremental adaptation of xpath access control views. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 105–116, 2007.

[2] E. Bertino, S. Castano, E. Ferrari, and M. Mesiti. Specifying and enforcing access control policies for xml document sources. *World Wide Web*, 3(3):139–151, 2000.

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

[4] J. Crampton. Applying hierarchical and role-based access control to xml documents. In *SWS '04*, pages 37–46. ACM, 2004.

[5] E. Damiani, S. D. C. di Vimercati, S. Paraboschi, and P. Samarati. Securing xml documents. In C. Zaniolo, P. C. Lockemann, M. H. Scholl, and T. Grust, editors, *EDBT*, volume 1777 of *Lecture Notes in Computer Science*, pages 121–135. Springer, 2000.

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

[7] E. Damiani, M. Fansi, and A. G. an Stefania Marrara. A general approach to securely querying xml. In *Computer Standards & Interfaces*, pages 379–389, 2008.

[8] W. Fan, C.-Y. Chan, and M. Garofalakis. Secure xml querying with security views. In *SIGMOD '04: Proceedings of the 2004 ACM SIGMOD international conference on Management of data*, pages 587–598. ACM, 2004.

[9] S. K. Goel, C. Clifton, and A. Rosenthal. Derived access control specification for xml. In *XMLSEC '03: Proceedings of the 2003 ACM workshop on XML security*, pages 1–14. ACM, 2003.

[10] G. Kuper, F. Massacci, and N. Rassadko. Generalized xml security views. In *SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies*, pages 77–84, New York, NY, USA, 2005. ACM Press.

[11] A. H. Landberg, J. W. Rahayu, and E. Pardede. Extending xml triggers with path-granularity. In *WISE*, pages 410–422, 2007.

[12] B. Luo, D. Lee, W.-C. Lee, and P. Liu. Qfilter: fine-grained run-time xml access control via nfa-based query rewriting. In *CIKM '04*, pages 543–552. ACM, 2004.

[13] P. Röder, O. Tafreschi, and C. Eckert. History-based access control for xml documents. In *ASIACCS '07: Proceedings of the 2nd ACM symposium on Information, computer and communications security*, pages 386–388. ACM, 2007.

[14] P. Samarati, E. Bertino, and S. Jajodia. An authorization model for a distributed hypertext system. *IEEE Trans. Knowl. Data Eng.*, 8(4):555–562, 1996.

[15] X. Xiao and Y. Tao. Anatomy: Simple and effective privacy preservation. In *VLDB*, pages 139–150, 2006.