Secure XML Publishing without Information Leakage in the Presence of Data Inference

Xiaochun Yang [*]

Department of Computer Science
Northeastern University, Liaoning 110004, China
yangxc@mail.neu.edu.cn

Chen Li [†]

School of Information and Computer Sciences
University of California, Irvine, CA 92697, USA
chenli@ics.uci.edu

Abstract

Recent applications are seeing an increasing need that publishing XML documents should meet precise security requirements. In this paper, we consider data-publishing applications where the publisher specifies what information is sensitive and should be protected. We show that if a partial document is published carelessly, users can use common knowledge (e.g., "all patients in the same ward have the same disease") to infer more data, which can cause leakage of sensitive information. The goal is to protect such information in the presence of data inference with common knowledge. We consider common knowledge represented as semantic XML constraints. We formulate the process how users can infer data using three types of common XML constraints. Interestingly, no matter what sequences users follow to infer data, there is a unique, maximal document that contains all possible inferred documents. We develop algorithms for finding a partial document of a given XML document without causing information leakage, while allowing publishing as much data as possible. Our experiments on real data sets show that effect of inference on data security, and how the proposed techniques can prevent such leakage from happening.

[*] Work partially done during a visit to UC Irvine and partially supported by NSF China No. 60173051.

[†] Supported by NSF CAREER Award No. IIS-0238586.

Proceedings of the 30th VLDB Conference, Toronto, Canada, 2004

1 Introduction

With the fast development of the Internet, there is an increasing amount of data published on the Web. Meanwhile, recent database applications see the emerging need to support data sharing and dissemination in peer-based environments [2, 12, 17, 20, 24], in which autonomous sources share data with each other. In these applications, the owner of a data source needs to publish data to others such as public users on the Web or collaborative peers. Often the data owner may have sensitive information that needs to be protected. As illustrated by the following example, if we publish data carelessly, users can use common knowledge to infer more information from the published data, causing leakage of sensitive information.

A hospital at a medical school has XML documents about its patients and physicians. Fig. 1 shows part of such an XML document represented as a tree. Each patient has a name (represented as a `pname` element), suffers from a disease (a `disease` element), and lives in a ward. Each physician has a name (`phname`), and treats patients identified by their names. For instance, physician Smith is treating patient Cathy, who has a leukemia and lives in ward W305. (We add a superscript to each node for later references.)

The hospital plans to provide the data to another department at the same school to conduct related research. Some data is sensitive and should not be released. In particular, the hospital does not want the department to know the disease of patient Alice (leukemia) for some reason. One simple way is to hide the shaded $leukemia^{(1)}$ subtree of Alice. But if it is well known that patients in the same ward have the same disease, then this common knowledge can be used by the department users to infer from the seen document that Alice has a leukemia. It is because Alice and Betty live in the same ward W305, and Betty has a leukemia. The users can do the similar inference using the information about patient Cathy, who also lives in ward W305. As a consequence, hiding the shaded $leukemia^{(1)}$ branch cannot protect the sensitive infor-

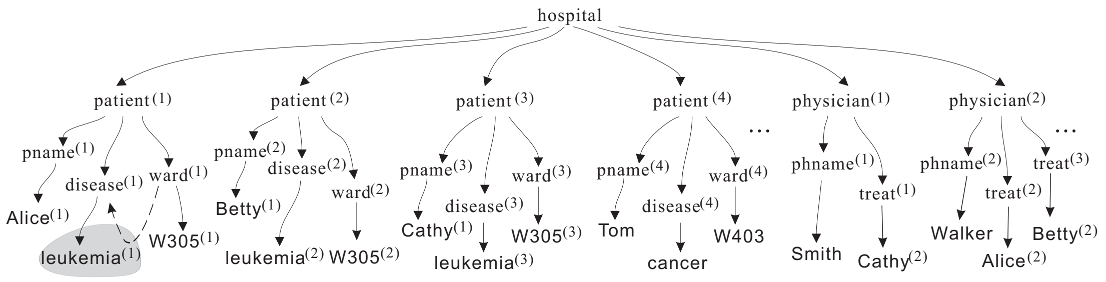


Figure 1: An XML document of hospital data. The shaded subtree is sensitive data. The dashed line shows a functional-dependency link.

mation due to this common knowledge.

One solution to this information-leakage problem is that, in addition to hiding the $leukemia^{(1)}$ branch of Alice, we also hide the $Alice^{(1)}$ branch, so that users do not know the name of this patient. Another solution is to hide the ward number $W305^{(1)}$ of patient Alice, or the ward branches of both Betty and Cathy, so that the users cannot infer the disease of Alice. A third option is to hide the disease branches of both Betty and Cathy. There are other solutions as well.

In general, publishing XML data with security requirements faces a multitude of challenges when users can infer data using common knowledge. First, how do we model data inference using common knowledge in XML documents? Such common knowledge can be represented as *semantic constraints*, which specify relationships that must be satisfied by the nodes. For instance, the common knowledge in the hospital example can be represented as a functional dependency from the `ward` elements to their corresponding `disease` elements. We thus need to understand the data-inference effects of different constraints. Some effect could be very subtle, e.g., we show in Section 3 that users could infer the existence of a new branch, even though its exact position is unknown. Such a branch could still contain sensitive information.

A second problem is: how do we compute all possible inferable data? Since users can apply constraints in arbitrary sequences to infer different documents, it is not clear what inferred documents we should consider to test if sensitive information is leaked.

A third problem is: how do we compute a partial document to be published without leaking sensitive information, even if users can do inference? As there are many possible partial documents that do not cause information leakage (a trivial one is the empty document), it is natural to publish as much data as possible without leaking sensitive information. Meanwhile, there are various kinds of constraints, and the inference result of one constraint could satisfy the conditions of another. Thus it is challenging to decide which nodes in the document should be published.

In this paper, we study these problems and make the following contributions.

- We formulate the process of data inference using common knowledge represented as semantic XML constraints. We show that there is a unique, maximal document users can infer using the constraints, which contains all possible inferred documents. We also develop a validation algorithm for testing if a partial document can leak sensitive information.
- We propose algorithms for computing a partial document to be published without leaking information, while to release as much data as possible.
- We conducted experiments using real data sets to evaluate our proposed techniques.

The rest of the paper is organized as follows. Section 2 gives the preliminaries on data security in XML publishing. In Section 3 we consider three kinds of common XML constraints, and show how each constraint can be used to infer data. Section 4 studies what data can be inferred by users using multiple constraints. We formally define information leakage, and give an algorithm for testing if a partial document is secure. In Section 5 we develop algorithms for calculating a valid partial document. Section 6 provides our experimental results on two real data sets. We conclude in Section 7.

1.1 Related Work

Recently there has been a large amount of work on data security and privacy. Miklau and Suciu [18] show a good diagram (shown in Fig. 2) to classify different settings for related studies based on trust domains. In Scenario A with a single trust domain, there are no main security issues. For client-server access control in Scenario B, the data is owned by the server. A lot of work in this setting has focused on how to respond to user queries without revealing protected data [4, 5, 15, 26]. Scenario C assumes that the client (data owner) does not trust the server, and a main problem is how to allow the server to answer clients' queries without knowing the exact data [11, 13].

In the data-publishing case (Scenario D), we mainly focus on how to publish data without leaking sensitive information. Several data-dependent approaches have been proposed [6, 7, 16]. Some existing approaches specify sensitive data by marking positive and/or negative authorizations on single data items.

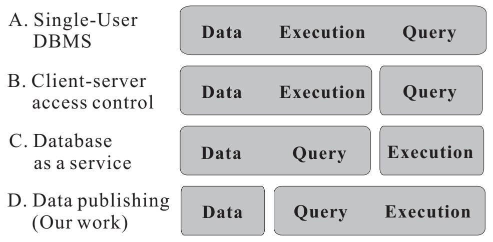


Figure 2: Different scenarios of database security based on trust domains [18].

These approaches consider each sensitive node individually. One limitation of this way of defining sensitive data is that it is not expressive to define some sensitive data such as "Alice's disease" that is independent from the specific position of the data. Another related work is [28], which shows how to compress documents to support efficient access control of sensitive XML data. Recently, Miklau and Suciu [18] study XML publishing, and develop an encryption-based approach to access control on publishing XML documents. They use an extension of XQuery to define sensitive data in a published document.

There have been works on inference control in relational databases [5, 8, 14, 23]. There are also studies on XML security control (e.g., [3, 10, 15, 28]), and they do not formally consider the effect of XML constraints. Our work formulates the process that users can infer data in XML publishing, and formally studies what additional data should be hidden in order to prevent information leakage. Notice that when we say "*hide* a branch in a document," we could either remove the branch from the document, or encrypt the branch and provide a key to the users who are allowed to access the branch. Therefore, our approach is orthogonal to whether encryption techniques are used to hide sensitive information [18].

2 Data Security in XML Publishing

We review basic concepts of data security in XML publishing. We view an XML document as a tree. An interior node represents either an ELEMENT or an ATTRIBUTE. A leaf node is of the string type, represented as a system-reserved symbol `S`, which corresponds to `PCDATA` for an ELEMENT parent node or `CDATA` for an ATTRIBUTE parent node. We do not consider the sibling order in the tree.

When publishing an XML document tree D, some nodes are *sensitive* and should be hidden from users. In our running example, the disease name of patient Alice is sensitive and should be protected. We assume such a sensitive node is specified by an XQuery, called a *regulating query*. For simplicity, we represent an XQuery as a tree pattern [1], with two types of edges: (1) a single edge represents an immediate-subelement relationship between a parent and a child (called a "c-child"); (2) a double edge represents a relationship between a node and a descendant (called a "d-child"). The tree pattern specifies the *conditions* to be satisfied when matching with the document. It has one special *sensitive node*, whose entire subtree should not be published. The sensitive node is marked with a symbol "*" in the regulating query. We make this single-sensitive-node assumption for the sake of simplicity. In general, we could allow multiple sensitive nodes in a regulating query. Such a query could be translated to multiple queries, each of which defines a single sensitive node.

A *mapping* μ from the nodes in a regulating-query tree A to the nodes in the document tree D is defined as follows: (1) if a leaf node n in A has a type or a value, then so does the corresponding node $\mu(n)$ in D; (2) if a node n is a c-child (resp., d-child) of another node v in A, then $\mu(n)$ is a child (resp., descendant) of $\mu(v)$ in D. Under mapping μ, the target subtree of the sensitive node in A is called the *excluded subtree* of A under μ. All the excluded subtrees of A under different mappings in the document are denoted as $A(D)$. According to regulating query A, all these excluded subtrees $A(D)$ should not be published.

A set of regulating queries $\mathcal{A} = \{A_1, \dots, A_m\}$ define a partial document that can be published. Such a document is calculated by removing every subtree in each $A_i(D)$. Formally, we define the set of excluded subtrees $\mathcal{A}(D) = \cup_{i=1}^{m} A_i(D)$, which represent the sensitive items in the XML tree that should not be published. The *remaining document* of document D under these regulating queries $\mathcal{A}$, denoted by $D_r(D, \mathcal{A})$, is the remaining subtree after those subtrees in $\mathcal{A}(D)$ are removed, i.e., $D_r(D, \mathcal{A}) = D - \mathcal{A}(D)$.

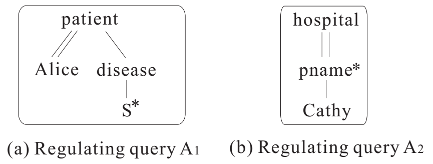


Figure 3: Example regulating queries.

For instance, the sensitive information "Alice's disease" in the hospital example can be specified by the regulating query A_1 in Fig. 3(a). The shaded subtree $leukemia^{(1)}$ in Fig. 1 is the corresponding excluded subtree. Suppose we have another regulating query A_2 as shown in Fig. 3(b), which specifies that the subtree of $pname^{(3)}$ of Cathy is sensitive. Given the set $\mathcal{A} = \{\mathsf{A}_1, A_2\}$ of the two regulating queries, the corresponding remaining subtree $D - \mathcal{A}(D)$ should exclude both subtree $leukemia^{(1)}$ and subtree $pname^{(3)}$.

3 Data Inference Using Single XML Constraints

We consider the case where users can do data inference using common knowledge represented as XML constraints. Such constraints for an XML document specify relationships that must be satisfied by the nodes in the document. XML constraints can be defined using XML schema languages, such as XML DTD, XML Schema [25], and UCM [9]. In this section, we formulate three common constraints, and show how they can be used individually to infer data.

3.1 XML Constraints

An XML constraint can be represented in the form "*conditions* $\rightarrow$ *facts*." It means that if the *conditions* on an XML document are satisfied, then those *facts* must also be true for the document. We focus on the following three types of common constraints.

- A *child constraint*, represented as $\tau \rightarrow \tau/\tau'$, means every node of type τ must have a child of type τ'.
- A *descendant constraint*, represented as $\tau \rightarrow \tau//\tau'$, means that every node of type τ must have a descendant of type τ'.
- A *functional dependency*, represented as $p/p_1 \rightarrow p/p_2$, where p, p_1, and p_2 are finite non-empty subsets of paths conforming to the document. It means that for any two subtrees t_1 and t_2 matching path p/p_1, if they have equal values in their p_1 paths, then (1) both of them have non-null, (value) equal subtrees that match p/p_2; or (2) neither of them has a subtree that matches p/p_2.[1]

Fig. 4 shows a few constraints for the hospital document. The child constraint C_1 says that each `patient` element must have a child of type `pname`. The descendant constraint C_2 says that each `patient` element must have a descendant of type `disease`. The functional dependency C_3 says that if two subtrees have the same //patient/ward value, then they must have the same //patient/disease value, i.e., patients in the same ward must have the same disease.

C_1:	//patient $\rightarrow$ //patient/pname
C_2:	//patient $\rightarrow$ //patient//disease
C_3:	//patient/ward $\rightarrow$ //patient/disease

Figure 4: Example constraints.

3.2 Data Inference Using a Single Constraint

Given a partial document P of the original document D and a constraint C, if P does not satisfy C, then users can use the condition in C to match the partial document. Whenever there is a match, users may infer more data in the document that is supposed to exist. Let $C(P)$ denote the inferred document after applying constraint C on the partial document P. We study the inference effect of different constraints.

A **child constraint** can be used to expand a partial document by adding one more branch. In this case, users know the exact location of the new branch. For instance, suppose the partial document in Fig. 5(a) is published to users. From the child constraint C_1, users know that there must be a `pname` branch, and they know the exact location of this branch, which is under the $patient^{(2)}$ element. Fig. 5(b) shows the new document $\mathsf{C}_1(P)$.

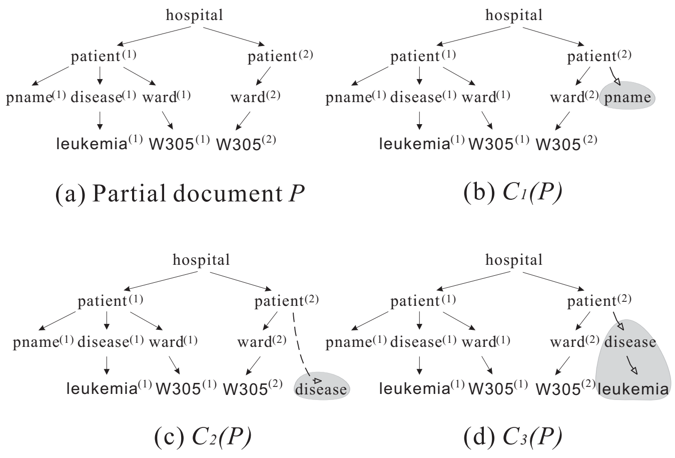


Figure 5: Inferred documents using constraints. The shaded areas represent inferred branches. The dotted edge in $C_2(P)$ represents a floating branch.

A **descendant constraint** can also be used to expand a partial document by adding a branch. In this case, however, users may not know the exact location of the new branch. Consider again the partial document in Fig. 5(a) and the descendant constraint C_2. With this constraint, users know there exists a `disease` branch, but they do not know its exact location under node $patient^{(2)}$. Users can thus add the branch to node $patient^{(2)}$, and let it "float" in the tree, as shown in Fig. 5(c). Such an inferred branch is called a *floating branch*, represented using a dotted edge.[2]

To allow floating branches in XML documents, we need to relax the definition of XML well-formedness by allowing dotted edges between elements to represent floating branches. Correspondingly, we define a mapping from a tree-pattern regulating query to an XML document (possibly with floating branches) in a straightforward way, similarly to the definition of "mapping" from an XQuery tree to a standard XML document. The only subtlety here is that the XML document could have a floating (descendant) edge, which can be the mapping image of another descendant edge in the query.

A **functional dependency** can also help users expand a tree. For a functional dependency $p/p_1 \rightarrow p/p_2$, if there are two branches t_1 and t_2 that match p/p_1, then users can use the subtree p_2 of t_1 to expand

[1]Personal communication with Marcelo Arenas and Leonid Libkin.

[2]A floating edge is similar to a descendant edge in XQuery. Here we use the word "floating" to emphasize the fact that the location of the subtree in the XML document is unknown.

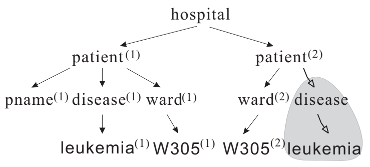
t_2, i.e., users can copy the subtree p_2 of t_1 as a subtree of t_2. (Some branches of the subtree may have already existed.) Fig. 5(d) shows the resulting document after applying the constraint C_3 to infer data.

4 Data Inference Using Multiple XML Constraints

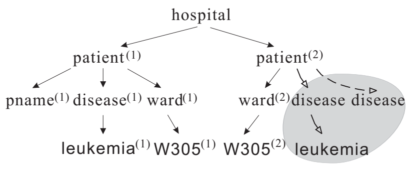
Now we study how users can use multiple constraints to infer data, which could potentially leak sensitive information. Formally, we consider a partial document P of the original document D, a set of regulating queries $\mathcal{A}$, and a set of constraints $\mathcal{C}$ known by users.

4.1 Equivalent Documents of Different Inference Sequences

Since users can do data inference with arbitrary sequences of constraints, different users could infer different results. For instance, consider the partial document in Fig. 5(a) and the two constraints C_2 and C_3 above. Fig. 6(a) shows the document after applying the sequence $\langle \mathsf{C}_3, \mathsf{C}_2 \rangle$ on the document. In particular, after applying C_3 first, we cannot use C_2 to infer any new branch, since the current document already satisfies C_2. The sequence $\langle \mathsf{C}_2, \mathsf{C}_3 \rangle$ expands the document P to the one shown in Fig. 6(b). Specifically, after applying C_2 to infer the floating branch, constraint C_3 can still be used to infer the `disease` subtree of node $patient^{(2)}$. Even though these two resulting documents look different, essentially they have the same amount of information. Intuitively, the floating branch in Fig. 6(b) says that there exists a `disease` *somewhere* under node $patient^{(2)}$. Since the document already has a `disease` element under the $patient^{(2)}$ node, this floating branch does not carry any additional information.



(a) Result of sequence $<C_3, C_2>$



(b) Result of sequence $<C_2, C_3>$

Figure 6: Inferred documents using different constraint sequences.

To formulate this notion of equivalence (or more generally, containment) between XML documents with floating (descendant) branches, we define the following concepts. (Notice this containment relationship is different from the subtree relationship.)

Definition 1 *Let D_1 and D_2 be two XML documents with possibly floating edges. We say D_1 is m-contained in D_2 if the following is true. If D_1 is treated as an XQuery, there is a mapping from query D_1 to document D_2. Such a mapping is called a* document mapping. *The two documents are called "m-equivalent" if they are m-contained in each other.*

The "m" in "m-containment" represents "mapping." Clearly any subtree of a document is m-contained in the document. The document in Fig. 5(c) is m-contained in that of Fig. 5(d). The two documents in Fig. 6 are m-equivalent. In particular, there is a document mapping from Fig. 5(b) to Fig. 5(a), which maps the floating edge to the edge from the $patient^{(2)}$ node to its $disease$ child. Intuitively, a containing document D_2 has at least the same amount of information as the contained document D_1. Since floating branches in D_1 can be mapped to the branches in D_2, they do not really represent additional information, and could be eliminated to make the document more concise.

Even though different sequences of applying the constraints to do data inference can result in different documents, there is a unique, maximal document that m-contains all these documents.

Theorem 1 *Given a partial document P of an XML document D and a set of constraints $\mathcal{C} = \{C_1, \ldots, C_k\}$, there is a document $\mathcal{M}$ that can be inferred from P using a sequence of constraints, such that for any sequence of the constraints, its resulting document is m-contained in $\mathcal{M}$. Such a document $\mathcal{M}$ is unique under m-containment.*

The document $\mathcal{M}$ is called the *maximal inferred document* of P using $\mathcal{C}$. The proof is in the full version of this paper [27]. Its main idea is to show the following algorithm, called CHASE, can compute a sequence of constraints that produces a document $\mathcal{M}$. This document m-contains any possible inferred document. The CHASE algorithm iteratively picks a constraint to apply on the current document. If the resulting document is not m-equivalent to the old one (e.g., new information has been inferred), the algorithm continues to apply the constraints. The algorithm terminates when no constraint can be applied to get a document that is not m-equivalent to the old document.

4.2 Information Leakage

Since users could use arbitrary sequences of constraints to do data inference, we need to prepare for the worst scenario, where a user could get the maximal inferred document. Now we formally define information leakage. Given a set of regulating queries $\mathcal{A} =$

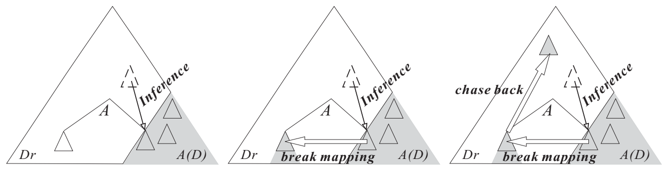
$\{A_1, \ldots, A_n\}$, a set of constraints $\mathcal{C} = \{C_1, \ldots, C_k\}$, and a partial document P of the original document D, consider the maximal inferred document $\mathcal{M}$. If there is a regulating query A_i, such that $\mathcal{M}$ can produce a nonempty answer to the query, then we say that this partial document causes *information leakage.* A partial document is called *valid* if it does not cause information leakage. To be consistent with the definition of "remaining document" $D_r(D, \mathcal{A}) = D - \mathcal{A}(D)$ in Section 2 in the case without constraints, we require that a valid document must be a subtree of $D_r(D, \mathcal{A})$.

For instance, consider the XML document in Fig. 1 and the regulating query A_1 in Fig. 3. The remaining document excludes subtree $leukemia^{(1)}$. Suppose users know all the constraints in Fig. 4. Then removing the $Alice^{(1)}$ node from the remaining document can yield a valid partial document. Alternatively, we can remove node $W305^{(1)}$ node of patient Alice, or remove the W305 nodes of both Betty and Cathy. Then users cannot use the functional dependency C_3 to infer the leukemia value of Alice.

5 Computing a Valid Partial Document

Given a document with sensitive data specified by regulating queries, in the presence of constraints, we need to decide a valid partial document to be published. There are many such valid partial documents, e.g., the empty document is a trivial one. Often we want to publish as much data as possible without leaking information. In this section, we study how to compute such a valid partial document. Since there are multiple partial valid documents, when in deciding what partial document should be published, we also need to consider application-specific requirements. For instance, if the application requires certain information (e.g., "Betty's ward number") to be published, then among the valid documents, we should publish one that did not remove this node. Here we mainly focus on finding one valid document assuming no such restriction. In case the application does require certain information be published, our solutions can be modified to take this requirement into consideration.

For simplicity, we mainly focus on the case of a single regulating query A, and the results can be extended to the case of multiple regulating queries. Fig. 7 shows the main idea of our approach. We first get the remaining document $D_r = D - A(D)$ by removing the sensitive elements $A(D)$ that match the sensitive node specified in A, possibly under different mappings. These sensitive elements are shown as shaded triangles in $A(D)$. Then we compute the maximal inferred document $\mathcal{M}$ using the CHASE algorithm in Section 4.1, and check if this document leaks any sensitive data (Fig. 7(a)). If not, we do not need to do anything, since D_r is already a valid document with maximum amount of information. Otherwise, we need to remove more nodes. As shown in Fig. 7(b), we consider each leaf node in the tree pattern A, and try to remove its target in $\mathcal{M}$ under a mapping, so that the mapping becomes "broken" after this removal. If some of the image nodes are inferred using the constraints, when removing such a node, we also need to "chase" back the data-inference process, and remove other branches to prevent such an inference (Fig. 7(c)). During the chase-back process, we find the subtree that matches the condition of the utilized constraint, and decide the branches that need to be removed.



(a) Users infer sensitive data using constraints.

(b) Remove nodes to break all mappings from A to the inferred document.

(c) If a leaf node to be removed is inferred, chase back the inference process to remove other nodes.

Figure 7: Computing a valid partial document.

One key challenge is deciding how to break mappings from the regulating query to the inferred document, and how to chase back the inference steps. In this section we present an algorithm for finding a valid partial document by constructing an *AND/OR graph* [21]. We first use an example to explain AND/OR graphs. We then discuss how the algorithm constructs such a graph, and uses the graph to find a valid partial document.

5.1 AND/OR Graphs

An AND/OR graph is a structure representing how a goal can be reached by solving subproblems. In our case, such a graph shows how to compute a valid partial document to satisfy the regulating query A. We use the following example to explain such a graph. Consider the hospital document in Fig. 1, the regulating query A_1 in Fig. 3(a), and the constraint C_3 : $//patient/ward \to //patient/disease$. The shaded part in Fig. 1 should be hidden from users. Fig. 8 shows part of the corresponding AND/OR graph for the problem of finding a valid partial document.

The graph has a special node called START, which represents the goal of computing a valid partial document. The graph has nodes corresponding to nodes in the maximal inferred document $\mathcal{M}$. Such a node in the graph represents the subproblem of *hiding* its corresponding node n in $\mathcal{M}$; that is, this node n should be removed from $\mathcal{M}$, and it cannot be inferred using the constraints and other nodes in $\mathcal{M}$. For example,

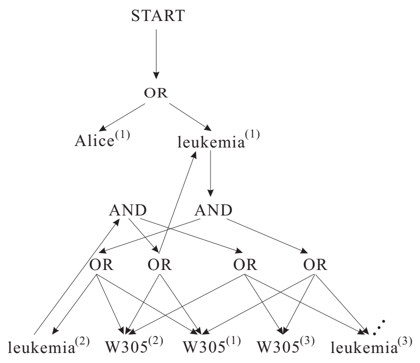


Figure 8: AND/OR graph.

the node *leukemia*$^{(1)}$ in Fig. 8 represents the subproblem of completely hiding the node *leukemia*$^{(1)}$ in $\mathcal{M}$; that is, we need to not only remove this node from the XML tree, but also make sure this node cannot be inferred using the constraint.

An AND/OR graph contains hyperlinks, called *connectors*, which connect a parent with a set of successors (nodes or connectors). (To avoid confusions, we use "nodes" in the AND/OR graph to refer to the START state and other vertices corresponding to some elements in the document $\mathcal{M}$, while we use "connectors" to refer to those vertices representing AND/OR relationships between a parent and its successors.) There are two types of connectors. An *OR connector* from a parent p to successors $s_1, \ldots, s_k$ represents the fact that, in order to solve problem p, we need to solve one of the subproblems $s_1, \ldots, s_k$. For instance, the OR connector below the START node in Fig. 8 shows that, in order to achieve the goal, we can either hide node *Alice*$^{(1)}$ or hide node *leukemia*$^{(1)}$. An *AND connector* from a node p to successors $s_1, \ldots, s_k$ represents the fact that solving problem p requires solving *all* the subproblems $s_1, \ldots, s_k$. For instance, the AND connector below the *leukemia*$^{(1)}$ represents the fact that hiding node *leukemia*$^{(1)}$ requires solving two subproblems. The first one, represented as an OR connector, is to hide one of the nodes $W305^{(1)}$, $W305^{(2)}$, and *leukemia*$^{(2)}$. This subproblem is due to the fact that, with the constraint C_3, users can use these three nodes to infer *leukemia*$^{(1)}$ (from Betty's information). Similarly, the second subproblem, also represented as an OR connector, is to hide one of the nodes $W305^{(1)}$. $W305^{(3)}$, and *leukemia*$^{(3)}$. Both nodes *leukemia*$^{(2)}$ and *leukemia*$^{(3)}$ have an AND connector similar to that of *leukemia*$^{(1)}$. For simplicity we do not draw the AND-connector structure of node *leukemia*$^{(3)}$.

To compute a valid partial document, we first search in the graph for a *solution graph*, which has the following properties: (1) It is a connected subgraph including the START node. (2) For each node in the subgraph, its successor connectors are also in the subgraph. (3) If it contains an OR connector, it must also contain one of the connector's successors. (4) If it contains an AND connector, it must also contain all the successors of the connector. After computing a solution graph G, for each non-connector node in G, we remove the corresponding node in the XML tree $\mathcal{M}$. In addition, we remove the nodes in $A(D)$, due to the assumption that these nodes must be removed (see Section 4.2). The final tree is a valid partial document.

Fig. 9 shows two solution graphs for the hospital AND/OR graph. The first one corresponds to a valid partial document that excludes the *Alice*$^{(1)}$ node. The second one corresponds to a valid partial document that excludes nodes *leukemia*$^{(1)}$, $W305^{(2)}$, and $W305^{(3)}$. By default, the corresponding partial documents do not include the node *leukemia*$^{(1)}$ due to assumption that this sensitive node must be removed.

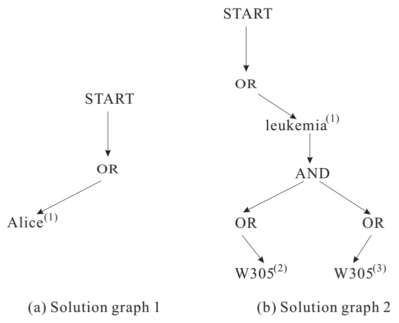


Figure 9: Solutions graphs for the hospital example.

In the rest of the section we give details of how to construct such an AND/OR graph, and how to find a valid partial document by using a solution graph.

5.2 Constructing an AND/OR Graph

Consider a document D, a set of constraints $\mathcal{C}$, and a regulating query A. Let $\mathcal{M}$ be the maximal inferred document that causes information leakage, i.e., there is at least one mapping from A to $\mathcal{M}$. We construct an AND/OR graph in three steps.

Step 1: To avoid information leakage, we need to break all the mappings from A to $\mathcal{M}$. If there is only one such mapping μ, as illustrated by the example in Fig. 8, we introduce an OR connector from the START node to a set of nodes, each of which corresponds to the image node (under μ) of a leaf node in A. (We choose leaf nodes of A in order to remove as few nodes as possible.) If there are more than one such mapping, for each of them we introduce an OR connector to its successor nodes in a similar manner. We then add an AND connector from the START node to these OR connectors, representing the fact that we must break all these mappings.

Step 2: For a node n in the AND/OR graph, its corresponding node in $\mathcal{M}$ could be inferred due to the constraints. In this step, we chase back these data inferences, and add nodes to the AND/OR graph to show how to break such an inference. Now we give the detail of how to break the inference of each type of constraint.

Child constraint: If node n can be inferred using a child constraint $\tau \rightarrow \tau/\tau'$, then n must be of type τ', and its parent p node in $\mathcal{M}$ must be of type τ. In order to break this inference, we need to remove the parent node p. Therefore, in the AND/OR graph we add an AND connector from node n to a new node p. (If an AND connector connects a node a to only one node b, for simplicity we can replace this connector with a single edge from a to b.)

Descendant constraint: If node n can be inferred using a descendant constraint $\tau \rightarrow \tau//\tau'$, then n must be of type τ', and it must be a floating branch in the inferred document $\mathcal{M}$ from a node a of type τ. In order to break this inference, we need to remove the node a. Therefore, in the AND/OR graph we add an AND connector from node n to a new node a.

Functional dependency: Consider the case where node n can be inferred by a functional dependency $p/t_1 \rightarrow p/t_2$. That is, node n is of type p/t_1 in the inferred document $\mathcal{M}$, and there exist nodes n_2, n', and n'_2 of types p/t_2, p/t_1, and p/t_2, respectively, such that n_2 is equal to n'_2 (as values) and n is equal to n' (as values). In this case, in order to break this inference, we need to remove one of n_2, n', and n'_2. Thus we add an OR connector from node n in the AND/OR graph to new nodes of n_2, n', and n'_2. Notice that the functional dependency can be used to infer node n with different sets of n_2, n', and n'_2. In this case, as illustrated by the example in Fig. 8, we introduce an AND connector from the node n to the corresponding OR connectors, each of which corresponds to such a set.

In the process of breaking the inferences, we may need to add new nodes (and connectors) to the AND/OR graph. If the nodes are already in the AND/OR graph, we can just add the necessary links to these existing nodes. In addition, for each newly added node, we still need to check if it can be inferred using constraints. If so, we need to repeat this process by adding necessary nodes and connectors. This step will repeat until each node in the graph either cannot be inferred, or have the necessary successors to break possible inferences for this node.

Step 3: In this step, we consider the fact that removing a node from the inferred document $\mathcal{M}$ also requires removing all its descendants. Thus we identify the ancestor-descendant relationships among all the nodes in the AND/OR graph, add an AND connector from each node to its descendant nodes (if any).

We use another example to show how to construct an AND/OR graph. Consider the document shown in Fig. 10(a), the regulating query in Fig. 10(b), and the following constraints.

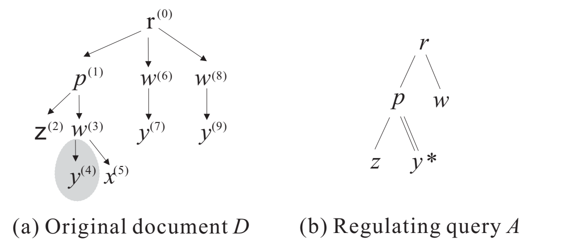


Figure 10: A document and a regulating query.

$$\begin{aligned} C_1&: \quad //w \rightarrow //w/y; \\ C_2&: \quad //p \rightarrow //p//y; \\ C_3&: \quad //p \rightarrow //p/z. \end{aligned}$$

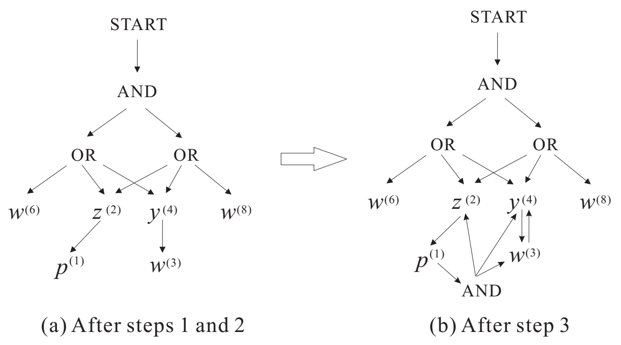


Figure 11: AND/OR graph for Fig. 10.

The corresponding AND/OR graph after steps 1 and 2 is shown Fig. 11(a). In particular, in step 1, we consider the two mappings from A to the maximal inferred document (i.e., the whole document in this example). The images of leaf nodes in A under the two mappings are $\{z^{(2)}, y^{(4)}, w^{(6)}\}$ and $\{z^{(2)}, y^{(4)}, w^{(8)}\}$, respectively. Thus we add one AND connector and two OR connectors to show the need to break both mappings. In step 2, we consider all constraints that can infer these nodes. Constraint C_3 can be used to infer node $z^{(2)}$ from $p^{(1)}$. Thus we add an AND connector (simplified as a single edge) from node $z^{(2)}$ to $p^{(1)}$ in the graph. Similarly, we add an AND connector from node $y^{(4)}$ to $w^{(3)}$ due to constraint C_1. Notice that node $y^{(4)}$ cannot be inferred from $p^{(1)}$ using constraint C_2, since this constraint can only infer a floating branch of $y^{(4)}$ from $p^{(1)}$, and this branch has already been merged with the path from $p^{(1)}$ to $y^{(4)}$.

In step 3, we add those ancestor-descendant links, and the final graph is shown in Fig. 11(b). In particular, node $p^{(1)}$ is an ancestor of nodes $z^{(2)}$, $w^{(3)}$, and $y^{(4)}$, and we add an AND connector from this ancestor to these descendants. Node $w^{(3)}$ is a parent node of $y^{(4)}$, so we add an edge from $w^{(3)}$ to $y^{(4)}$.

5.3 Computing a Valid Partial Document Using the AND/OR Graph

We search within the constructed AND/OR graph for a solution graph, which can be used to produce a valid partial document. We want to remove as few nodes as possible. If it is computationally expensive to find a solution graph with the minimum number of nodes to remove, we can use heuristics to search for a solution graph. For instance, we can adopt the depth-first search strategy as follows. Initially we add the `START` node to the solution graph G. We mark it SOLVED and add it to a stack. In each iteration, we remove the top element e from the stack. There are four cases: (1) e is a node without any successors. Then we do nothing. (2) e is a node with successors. We add these successors and the corresponding edges to the solution graph G. For each of the successors that is not marked SOLVED, we mark it SOLVED and add it to the stack. (3) e is an AND connector. We add all its successors and the corresponding edges to G. For each of its successors that is not marked SOLVED, we mark it SOLVED and add it to the stack. (4) e is an OR connector. Then we choose one of its successors, add this successor and the corresponding edge to G. If the successor is not marked SOLVED, we then mark it SOLVED and add it to the stack. We repeat the process until the stack becomes empty.

After finding a solution graph, for each of its nodes, we remove the corresponding node in the maximal inferred document $\mathcal{M}$. We also remove the nodes in $A(D)$, and the final document is a valid partial document. For example, Fig. 12 shows a solution graph for the AND/OR graph in Fig. 11(b).

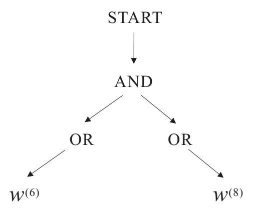


Figure 12: A solution graph.

Remarks: So far we have described how to construct a *complete* AND/OR graph and then search for a solution graph. Often we only need to find just one solution graph. Thus we can find a such solution graph without constructing the complete AND/OR graph. That is, we search for a solution graph as we construct the AND/OR graph "on the fly." There has been a lot of work on heuristic searches in AND/OR graphs [21, 22], such as GBF, GBF*, AO, AO*, and etc. These heuristics can be adopted for efficiently finding a solution graph.

6 Experiments

We conducted experiments to evaluate the effect of data inference on security and the effectiveness of our proposed techniques. In this section we report the experimental results.

6.1 Setting

Data sets: We used two real XML data sets. The first one was from the University of Illinois.[3] It had a file "course_washington.xml" with course information. It contained 3,904 course elements with 162,102 nodes including elements and text nodes. The second data set is from DBLP.[4] It contained more than 427,000 publications (at the time of the experiments), represented as an XML file with about 8,728,000 nodes, including elements, attributes, and their text nodes.

XML Constraints: In general, constraints known to users are defined according to common knowledge [19] or the possible schema of the application domain. We analyzed the two data sets, and found that many constraints were valid. In the experiments, we used the constraints shown in Fig. 13 as examples. They were assumed to be common knowledge known to users. For instance, in data set 1, each course must have a title, represented as the constraint "//course → //course/title." In data set 2, two papers with the same title should have the same authors. (Some conference papers were also published as journal articles with the same title and authors.) Such a fact could be represented as a functional dependency "//dblp/pub/title → //dblp/pub/author," when we treated different types of publications as a single type called "pub."

Data set 1: **course_washiongton.xml**
Child constraints
//course → //course/title
//course → //course/section
//session → //session/day
//session → //session/time
//session → //session/place
Descendant constraints
//course → //course//credits
//course → //course//instructor

Data set 2: **dblp.xml**
Functional dependency
//dblp/pub/title → //dblp/pub/author

Figure 13: Sample constraints.

Which part of an XML document is sensitive depends on the application. In our experiments, we considered sensitive information defined by two types of regulating queries. The first type of regulating queries ("Type 1") were manually defined as XQuery expressions after we analyzed the semantics of different nodes. The following are two examples.

- A_1: In course_washington.xml, hide codes of all courses.
- A_2: In dblp.xml, hide authors who published papers in 2001.

[3] anhai.cs.uiuc.edu/archive/data/courses/washington/
[4] www.informatik.uni-trier.de/~ley/db/

The second type of regulating queries ("Type 2") were generated by randomly marking a set of nodes in the document as sensitive data. The goal is to see the relationship between the percentage of sensitive nodes and the amount of leaked information.

6.2 Amount of Leaked Information Defined by Regulating Queries of Type 1

We first evaluated how much sensitive information was leaked due to constraints. The information was defined using regulating queries of type 1. We measured the number of sensitive nodes defined by each regulating query of type 1, and the number of leaked nodes after the user does data inference. For example, in the course_washington.xml document, 3,904 `code` elements were defined as sensitive. However, using the given constraints, no information can be inferred because no constraint can be used to infer any `code` element. In the dblp.xml document, there were 63,653 authors who published papers in 2001, and their names were defined as sensitive by regulating query A_2. If we just published the document by removing these sensitive names, users could use the functional dependency "//dblp/pub/title $\rightarrow$ //dblp/pub/author" to infer 977 such author names. In particular, the corresponding publications of these inferred authors have been published in year 2001, whereas there exist some other (journal) publications that have the same title of these publications, but they were published in other years. Using the functional dependency, users can infer some of these hidden author names.

6.3 Amount of Leaked Information Defined by Regulating Queries of Type 2

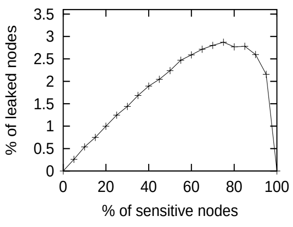
We let the percentage of sensitive nodes specified by regulating queries $\mathcal{A}$ of type 2 vary from 0 to 100%. We assumed that users know the constraints in Fig. 13. We measured how much information can be leaked if the system just published the document $D - \mathcal{A}(D)$ by removing the sensitive nodes $\mathcal{A}(D)$ (their subtrees). We used the validation approach discussed in Section 4 to compute the number of leaked sensitive nodes. We considered different factors that can affect the amount, such as the types of constraints, the number of constraints, and the number of nodes that satisfy the conditions in the constraints. The percentage of leaked nodes is calculated as

$$\frac{\text{number of leaked sensitive nodes}}{\text{number of nodes in the whole document}}.$$

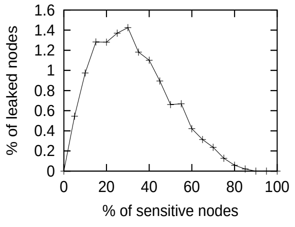
Effect of Different Constraints

Fig. 14 shows the relationship between the number of leaked nodes and the number of sensitive nodes for different types of constraints.

Child and descendant constraints: Fig. 14(a) shows the results for the child and descendants in Fig. 13 for document course_washington.xml. As the percentage of sensitive nodes increased, the percentage of leaked nodes increased to a peak (about 2.8%), when the percentage of sensitive nodes was around 82%. The reason for this increase is that more nodes inferred by users could be sensitive. If the number of sensitive nodes further increased, then the number of leaked nodes started to drop. The result is not surprising, since there were fewer nodes in the remaining document, and it became less likely for users to do data inference, causing less information to be leaked.



(a) Child and descendant constraints in course_washington.xml



(b) Functional dependencies in dblp.xml

Figure 14: Effect of different kinds of constraints.

Functional dependencies: We used dblp.xml to test the effect of the functional dependency shown in Fig. 13. Since the number of nodes satisfying the functional dependency was relatively small compared to the document size, we chose a subdocument by using a subset of the represented publications, such that the subdocument had 8% of its publications satisfying the functional dependency. We then varied the number of sensitive nodes in the subdocument. The result is shown in Fig. 14(b). Similarly to the previous case, as the percentage of sensitive nodes increased, the percentage of leaked nodes increased quickly to a peak value of 1.4%, when the corresponding percentage of sensitive nodes was around 26%. The number of leaked nodes also started to drop as we increased the percentage of sensitive nodes, for the similar reason as in the case of child/descendant constraints. Notice that in this case, the percentage of sensitive nodes that yielded the peak value was smaller than that of the previous case. The main reason is that functional dependencies often need more nodes to be satisfied than child/descendant constraints.

Effect of Number of Constraints

We then evaluated the effect of the number of constraints (corresponding to the amount of common knowledge) on data inference. We chose different numbers of constraints and tested how they affected the number of leaked elements. We considered two cases for the document course_washington.xml.

- Using 4 constraints: We assumed that users know the first three child constraints and the first one descendant constraint in Fig. 13.
- Using 7 constraints: We assumed that users know the five child constraints and the two descendant constraints in Fig. 13.

The results in Fig. 15 are consistent with our intuition. As the number of constraints increases, there is more common knowledge known by users. Then they can infer more data, causing more information to be leaked. For both cases, as the percentage of sensitive nodes increased, the percentage of leaked nodes increased to reach a peak before it started to drop.

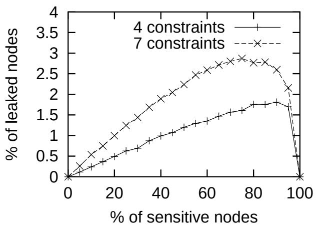


Figure 15: Different numbers (4 and 7) of constraints in course_washington.xml.

Effect of Nodes Satisfying Constraints

Different partial documents could have different numbers of nodes that satisfy the conditions in the constraints. We studied the effect of this number on data inference and information leakage. In the experiments, we chose a subdocument of the dblp.xml document with about 800, 000 nodes as a test document. We adjusted the number of nodes satisfying the condition in the functional dependency in Fig. 13. We adjusted this number to about 40, 000, 64, 000, and 80, 000, by removing some of the represented publications. Fig. 16 shows the results, where V is the number of nodes satisfying the condition in the functional dependency. Not surprisingly, as we increased V, the percentage of leaked nodes increased, since more such nodes can help users infer more sensitive nodes.

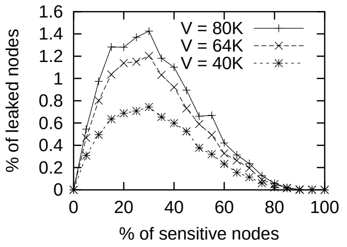
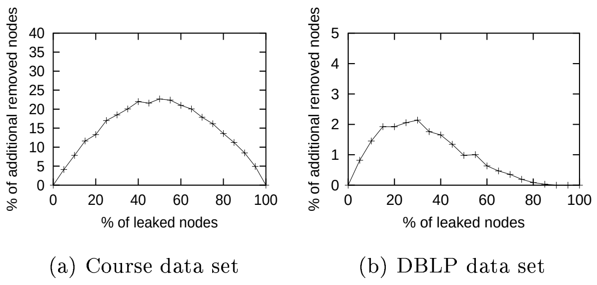


Figure 16: Effect of number of nodes satisfying the condition in the functional dependency.

6.4 Removing Nodes to Prevent Leakage

We used the method described in Section 5 to compute a valid partial document without information leakage. In this experiment we measured how many nodes the algorithm decided to remove to compute a valid partial document. Based on the analysis in Section 6.2, we know that the nodes that need to be removed highly depend on the regulating queries. If there are many ways the conditions in the regulating queries can be mapped to the document, then many nodes are sensitive and should be hidden. Whereas, if the conditions are very selective, few nodes satisfying these conditions, and few nodes need to be hidden. Furthermore, different kinds of constraints have different effects on what nodes should be removed.

We considered the constraints in Fig. 13. We considered regulating queries that have a condition that is either a single element (the element is sensitive) or a path (the root of the path is sensitive). We randomly chose certain percentage of the nodes as sensitive nodes, and applied our algorithm to decide what nodes should be removed to avoid information leakage. We could choose a node in a mapping image of the regulating query to remove, or we could chase back the process from an inferred sensitive node and remove other nodes as well. In the case there were different ways to remove nodes, we randomly selected one solution. For each solution, we measured how many nodes needed to be removed. We ran 10 rounds to compute the average percentage of nodes that can be removed. When we removed an interior node, we also counted its descendants in the number of nodes.



(a) Course data set (b) DBLP data set

Figure 17: Additional removed nodes to get a valid partial document.

Fig. 17 shows the results for both data sets. The document course_washington.xml with child and descendant constraints required relatively more nodes to be removed compared to that of document dblp.xml with a functional dependency. The reason is the following. For the child/descendant constraints, when we chased back their inference process, we had to choose a parent or ancestor to remove, which can remove a lot of child/descendant nodes. On the other hand, for the functional dependency, when we chose a node to break the inference process, we could often find a leaf node (or a node close to leaf nodes) to remove, thus the total number of removed nodes was smaller.

Summary: Our experiments show that sensitive information could be leaked during XML publishing if common knowledge (constraints) is not considered carefully. The amount of leaked information depends on the number and type of regulating queries, the number and type of constraints, and the number of nodes satisfying the conditions in the constraints. Our proposed techniques can measure how much sensitive data is leaked, and can also compute a valid partial document without information leakage.

7 Conclusions

In this paper, we studied the effect of data inference using common knowledge (represented as XML constraints) on data security in XML publishing. We formulated the process how users can infer data using three types of common XML constraints. We showed that there is a unique, maximal document that contains all possible inferred documents. We developed algorithms for finding a partial document of a given XML document without causing information leakage. Our experiments on real data sets showed that effect of inference on data security, and how the proposed techniques can avoid such leakage.

Acknowledgments: We thank Marcelo Arenas and Leonid Libkin for clarifying the definition of XML functional dependencies. We thank Lei Shi and Ge Yu for their discussions and help on the implementation.

References

[1] S. Amer-Yahia, S. Cho, L. V. S. Lakshmanan, and D. Srivastava. Minimization of Tree Pattern Queries. In *SIGMOD Conference*, 2001.

[2] P. A. Bernstein et al. Data management for peer-to-peer computing: A vision. In *WebDB*, 2002.

[3] E. Bertino, B. Carminati, and E. Ferrari. A Secure Publishing Service for Digital Libraries of XML Documents. In *ISC*, pages 347–362, 2001.

[4] J. Biskup and P. Bonatti. Controlled Query Evaluation for Known Policies by Combining Lying and Refusal. In *Foundations of Information and Knowledge Systems*, pages 49–66, 2002. LNCS 2284.

[5] A. Brodskyand, C. Farkas, and S. Jajodia. Secure Databases: Constraints, Inference Channels, and Monitoring Disclosures. *TKDE*, 12(6):900–919, 2000.

[6] E. Damiani, S. D. C. D. Vimercati, S. Paraboschi, and P. Samarati. Design and Implementation of an Access Control Processor for XML Documents. *Computer Networks*, 33(1-6):59–75, June 2000.

[7] E. Damiani, S. D. C. D. Vimercati, S. Paraboschi, and P. Samarati. A Fine-Grained Access Control System for XML Documents. *ACM Transaction on Information and System Security*, 5(2):169–202, 2001.

[8] S. Dawson, S. D. C. D. Vimercati, P. Lincoln, and P. Samarati. Minimal Data Upgrading to Prevent Inference and Association Attacks. In *PODS*, 1999.

[9] W. Fan, G. M. Kuper, and J. Siméon. A Unified Constraint Model for XML. *Computer Networks*, 39(5):489–505, 2002.

[10] A. Gabillon and E. Bruno. Regulating Access to XML Documents. In *Proceedings of the 15th Annual IFIP WG 11.3 conference on Database Security*, July 2001.

[11] H. Hacigumus, B. R. Iyer, C. Li, and S. Mehrotra. Executing SQL over Encrypted Data in the Database Service Provider Model. In *SIGMOD*, 2002.

[12] A. Y. Halevy et al. Schema mediation in peer data management systems. In *ICDE*, 2003.

[13] B. Hore, S. Mehrotra, and G. Tsudik. A privacy preserving index for range queries. In *VLDB*, 2004.

[14] S. Jajodia and C. Meadows. Inference Problems in Multilevel Secure Database Management Systems. In *Inofrmation Security: An Integrated Collection of Essays*, pages 570–584, 1995.

[15] S. Jajodia, P. Samarati, M. L. Sapino, and V. S. Subrahmanian. Flexible Support for Multiple Access Control Policies. *ACM Transaction on Database Systems*, 26(2):241–286, 2001.

[16] M. Kudoh, Y. Hirayama, S. Hada, and A. Vollschwitz. Access Control Specification Based on Policy Evaluation and Enforcement Model and Specification Language. In *Symposium on Cryptograpy and Information Security (SCIS)*, 2000.

[17] C. Li, J. Li, and Q. Zhong. RACCOON: A peer-based system for data integration and sharing. In *ICDE*, 2004.

[18] G. Miklau and D. Suciu. Controlling Access to Published Data using Cryptography. In *VLDB*, 2003.

[19] G. Miklau and D. Suciu. A Formal Analysis of Information Disclosure in Data Exchange. In *SIGMOD*, 2004.

[20] W. S. Ng et al. PeerDB: A P2P-based system for distributed data sharing. In *ICDE*, 2003.

[21] N. J. Nilsson. *Principles of Artificial Intelligence*. Morgan Kaufmann, 1994.

[22] J. Pearl. *Heuristics: intelligent search strategies for computer problem solving*. Addison-Wesley Longman Publishing Co., Inc., 1984.

[23] X. Qian and T. Lunt. A Semantic Framework of the Multilevel Secure Relational Model. *IEEE Transactions on Knowledge and Data Engineering*, 9(2):292–301, 1997.

[24] The Hyperion Project. Computer Science Dept., University of Toronto.

[25] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. XML Schema Part 1: Structures. W3C. http://www.w3.org/TR/xmlschema-1, 2000.

[26] M. G. W. Fan, C.-Y. Chan. Secure XML Querying with Security Views. In *SIGMOD*, 2004.

[27] X. Yang and C. Li. Secure XML Publishing without Information Leakage in the Presence of Data Inference (Full Version). Technical report, UC Irvine, 2004.

[28] T. Yu, D. Srivastava, L. V. S. Lakshmanan, and H. V. Jagadish. Compressed Accessibility Map: Efficient Access Control for XML. In *VLDB*, pages 478–489, 2002.