12-2006

Query-Based Watermarking for XML Data

Xuan ZHOU
L3S Research Center

Hwee Hwa PANG
Singapore Management University, hhpang@smu.edu.sg

Kian-Lee TAN
National University of Singapore

DOI: https://doi.org/10.1145/1229285.1266991

Query-based Watermarking for XML Data

Xuan Zhou
L3S Research Center
Deutscher Pavillon, Expo Plaza 1
30539 Hanover, Germany
zhou@l3s.de

HweeHwa Pang
School of Information Systems
Singapore Management University
Singapore 178902
hhpang@smu.edu.sg

Kian-Lee Tan
Department of Computer Science
National University of Singapore
Singapore 117543
tankl@comp.nus.edu.sg

ABSTRACT

As increasing amount of XML data is exchanged over the internet, copyright protection of this type of data is becoming an important requirement for many applications. In this paper, we introduce a rights protection scheme for XML data based on digital watermarking. One of the main challenges for watermarking XML data is that the data could be easily reorganized by an adversary in an attempt to destroy any embedded watermark. To overcome it, we propose a query-based watermarking scheme, which creates queries to identify available watermarking capacity, such that watermarks could be recovered from reorganized data through query rewriting. The identifier queries are tied closely to the usability of the data, and are highly resilient to reorganization and alteration attacks. In addition, our scheme considers data semantics to avoid vulnerabilities caused by redundancy, while taking advantage of the available watermark capacity. Based on this scheme, we have designed and implemented a watermarking solution for XML, and experimentally verified the effectiveness of the solution and its potential for real world applications.

1. INTRODUCTION

XML is emerging as a new standard for information representation and exchange over the internet. As increasing amounts of commercial data are exchanged or published in this data format, unauthorized duplication and distribution of the data become a mandatory concern for many applications. Examples include a job agent who would like to prevent job advertisements on his server from being stolen and posted on other web sites, and a commercial digital library that needs to safeguard its copyright over its collection of documents.

Digital watermarking is one of the most widely used measures to protect digital information from copyright infringements. By introducing indiscernible perturbations into the data, it marks the data with copyright information, through which the publisher can prove his ownership of the data. In this work, we investigate how to perform digital watermarking on XML data. As a simple example, figure 1(a) shows an XML document which contain a set of publication records. In order to embed watermarks into this document, we introduce some perturbations into the <rating> elements, whose values range from 0 to 100. In particular, we use a secure pseudo-random number generator (PRNG) to select a small number of <rating> elements and round each of their values by 2. Such small modification is usually imperceptible to users, and does not significantly affect the usage of the document. Later on, we can claim our ownership of this XML document by showing that all the <rating> elements selected by our PRNG contain only even values, which is unlikely to happen by accident.

An adversary could also modify the XML document to erase the embedded watermark. His attack will succeed if he happens to change one of the <rating> elements selected by our PRNG. However, he must achieve that in a limited number of modifications, as over-modification would make the whole document unusable. Therefore, a successful watermarking should observe two basic principles: (i) the watermark insertion does not affect the usability of the data; (ii) it is difficult to remove the embedded information without destroying the usability of the data. (In this paper, we focus on robust watermarking rather than fragile watermarking, which is used for tamper detection.)

Earlier research on robust watermarking has mostly focused on multimedia data (e.g. image [22], text [3], audio [4] and video [10]), and a number of commercial applications have been developed. However, those methodologies do not work well with XML data, as its format is so flexible that adversaries can readily reorganize the data to destroy watermark bits. In some recent studies on watermarking relational and XML data [1, 21, 8], the proposed techniques mostly focus on how to embed watermark into isolated data elements and do not adequately address the problems incurred by data reorganization. Therefore, they are either unable to fully utilize available watermark capacity or vulnerable to attacks like data reorganization and redundancy removal.

There are a number of unsolved challenges faced by watermarking XML data. They include:

(1) Identifying data elements and structures for watermarking: An XML document consists of a number of data elements and structures that link the data elements together based on their relationships, both of which could offer capacity for watermarking. Identifying these units is crucial for watermarking tasks. However, to identify each data element

```
<db>
  <book publisher="mkp">
    <title>Readings in Database Systems</title>
    <author>Stonebraker</author>
    <author>Hellerstein</author>
    <editor>Harrypotter</editor>
    <rating>75</rating>
  </book>
  <book publisher="acm">
    <title>Database Design</title>
    <author>Bernstein</author>
    <author>Newcomer</author>
    <editor>Gamer</editor>
    <rating>75</rating>
  </book>
  ...
</db>
```

(a) db1.xml

```
<db>
  <publisher name="mkp">
    <author name="Stonebraker">
      <book rating="75">Readings in Database Systems</book>
      <book rating="61">XML Query Processing</book>
    </author>
    <author name="Hellerstein">
      <book rating="75">Readings in Database Systems</book>
      <book rating="57">Relational Data Integration</book>
    </author>
    ...
  </publisher>
  <publisher name="acm">
    ...
  </publisher>
  ...
</db>
```

(b) db2.xml

Figure 1: Structure Reorganization

or structure unit, it is inadequate to treat it in isolation; rather, we should consider its relationships with other data elements and structures. As illustrated by the db1.xml in figure 1(a), if we identify each <rating> element by its value (i.e., 75), we lose the distinction between the two <rating> elements under the two different books. This would significantly reduce the effective watermark capacity. Instead, a better identifier of the <rating> element is the value of its sibling element <title>.

(2) Resilience to data reorganization and alteration: When data elements are identified through relationships and structures, an adversary could reorganize the data to prevent the elements from being correctly identified. The flexible format of XML data in particular enables it to be reorganized easily. As shown in figure 1, an adversary could redesign the schema of db1.xml into db2.xml, without losing any information. He could also alter some parts of the structure (delete or add some edges or data elements) to hinder the detection of embedded watermarks. He could even transform the XML data into relational data. Thus, the identifiers of data elements must be persistent enough to survive any form of reorganization and alteration.

(3) Identifying data redundancy: Innate redundancies within XML data could severely degrade the robustness of watermarking. For example, db1.xml contains the semantic that an editor only works for one publisher, i.e., the functional dependency "*editor* → *publisher*" holds. The FD produces many duplicated *publisher* entries that correspond to the same editor. If these duplicates are selected to embed different bits of a watermark, the watermark could be erased by making all the duplicates identical. In contrast to challenge (1) which requires data elements to be differentiated, this problem requires duplicates of the same data element to be identified and treated accordingly during watermarking.

In this paper, we present a query-based watermarking scheme for XML data that overcomes the above challenges. Our contributions include:
(i) We propose to use a set of query templates to express the usability of an XML dataset, and give the methods to quantify such usability and the corresponding distortion function.
(ii) We create a scheme for robustly identifying data elements and structure units in XML data. Our scheme creates queries as the identifiers of data elements, so that the identifiers can be adapted to reorganized data through query rewriting. The created identifiers are tied closely to the usability of the data, and highly resilient to reorganization and alteration attacks. In addition, we consider the internal semantics to avoid vulnerabilities caused by redundancy, while taking advantage of the available watermark capacity.
(iii) On top of the identification scheme, we devise a watermarking scheme for embedding copyright information in XML data, and prove its effectiveness both theoretically and experimentally.

The remainder of this paper is organized as follows. Section 2 reviews the previous works on watermarking relational and XML data. Section 3 gives the basic concepts of digital watermarking and presents a threat model for watermarking XML data. Section 4 introduces the basic construction of our watermarking system. Following that, sections 5 and 6 elaborate on the two major components of the scheme – the data element identification scheme and the watermarking algorithm. Section 7 presents an implementation of our proposed scheme and experiment results. Finally, section 8 summarizes the paper and gives directions for future research.

2. RELATED WORK

There have been a number of works [1, 21, 19, 13, 14, 15, 8] on watermarking relational database. Several watermarking schemes have been proposed to address different attacks, or to serve different purposes (e.g. traitor tracing [14, 15] and tamper detection [13]). To summarize, the attacks that have been handled by the previous works include the followings – *alteration, selection, addition, resorting* and *collusion* (collusion is not encountered in copyright protection but in traitor tracing). These attacks are also present to watermarking XML data, and we are going to address most of them (except collusion) in this paper. However, the previous works have not adequately studied the challenges posed by data reorganization and redundancy. While they proposed to identify available watermark capacity by primary keys, it is questionable whether such identifiers can survive through the reorganization and redundancy removal attacks described in section 3. (For instance, using those schemes to watermark a relation that is not normalized would result in vulnerability to redundancy removal attacks.) Moreover, without a measure of the usability of relational data and the corresponding distortion function, those schemes are unable to tie their identifiers to data usability so as to ensure their robustness. We are going to work out these problems in this paper.

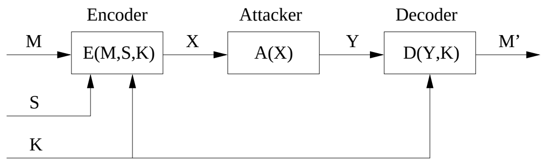


Figure 2: A Generic Watermarking Model

In fact, data reorganization is more critical with XML data, as its structure is more complex and flexible than that of relational data. Such identifiers as primary key are not always available in XML data, and it is difficult to normalize XML data to remove any innate redundancy. In [20], Sion et al. proposed an information hiding scheme for semi-structured data, in which they identify each data element by the values of its adjacent elements, and hope the identifiers could survive through reorganization. However, we do not think this intuitive method will work well for XML. As illustrated in figure 1, the neighbors of the *book* element in db1.xml actually change significantly after it is converted to db2.xml. In [17], the only work on watermarking XML data that we are aware of, the authors extended the watermarking scheme by Agrawal et. al. [1] on XML data, and deployed it on an XML compression system. However, they did not address any problems relating to data reorganization or redundancy.

David Gross-Amblard calls his work "query-preserving watermarking " [8] (this term is adapted from [11]). He considered the relational or XML data that are only partially accessible through a set of parametric queries, and studied how watermark could be inserted so that it can be detected from the results of those queries. This is different from our "query-based watermarking", in which we create queries to resiliently identify the data elements used for embedding watermarks, so that watermark could be recovered from reorganized data through query rewriting.

3. PROBLEM DEFINITION

This section enumerates the challenges for watermarking XML data, following a review of a generic watermarking model.

3.1 Watermarking

In general, a watermarking scheme embeds data with copyright information, which could later be extracted to prove its ownership. A simple model of robust watermarking process, adapted from [16], is shown in figure 2. S is the original data before being watermarked; K is the secret key used in both watermark embedding and watermark detection; and M is the watermark to be embedded in S. The watermark encoding function E embeds M into S through K, and produces watermarked data X. Attackers may try to remove M from X by modifying X into Y. However, with a resilient watermarking scheme, the owner can still extract $M' = M$ from Y through D (watermark detection function) and K.

A good watermarking scheme should satisfy the following three basic requirements:

1. Imperceptibility: The modification of the data is imperceptible to end users. In other words, the usability of the data should not be degraded too much by the watermarking procedure. Suppose $U(x, y)$ is the distortion function that measures the usability degradation caused by modifying x into y. The imperceptibility requirement could be represented by $U(S, X) < T$, where T denotes the maximum tolerable distortion.

2. Resilience: It is difficult for adversaries to remove the watermark without destroying the usability of the data. It could be represented as $P(M' \neq M|U(X, Y) < T) < \varepsilon$, which means that the probability of an unsuccessful detection is less than some tolerable error rate ε, after one or more attacks that preserve data usability.

3. Credibility: Detection of the watermark is enough to convince others that the data has been watermarked by the owner. This means that it is difficult for an attacker to falsely claim ownership by coming up with a key that can extract his intended watermark from the data. (Credibility measures the resilience of watermarking against *invertibility attacks* [5]).

There is a trade-off between resilience and credibility, in the sense that increasing credibility reduces resilience against attacks, as both are limited by the watermark capacity of the data. The means to improve both resilience and credibility is to fully utilize available watermark capacity in the data.

3.2 Threat Model

To the best of our knowledge, the followings are the various possible attacks to the watermarking of XML data:

Attack1: Data Reorganization. The attack redesigns the schema of an XML data and reorganizes the data according to the new schema, in an attempt to deter the owner from correctly identifying the data elements or structure units that contain the watermark. An example has been given in figure 1 in the Introduction. The attack also includes re-ordering the data elements and altering their relationships. Such an attack has not been adequately addressed in watermarking other types of data, e.g. image and audio, as their formats are much less variable.

Attack2: Redundancy Removal. Attackers exploit redundancies within the data to destroy the embedded watermark. This is similar to the data compression attack [7] to multimedia watermarking schemes. Redundancies within XML data are usually produced by semantics that are not considered by the schema design. An example is the functional dependency "*editor*→ *publisher*" in db1.xml of figure 1, which produces many duplicates of the *publisher* attributes. Any watermark embedded in the duplicates can probably be erased by removing the duplicates or making them identical. A possible solution is to normalize the data before it is watermarked. However, normalization of XML data is still an open research problem [2]. Furthermore, many XML data will be intentionally designed to contain redundancies, to achieve simplicity and performance.

Attack3: Addition Attack. Attackers mix the watermarked data with other related data. As with Attack1, this attack also aims to prevent the identification of watermarked data elements.

Attack4: Selection Attack. Attackers select from the data a subset that satisfies their intended usage, and discard the rest, in a hope that the watermark is lost in the process.

Attack5: Content Alteration. Attackers randomly or selectively modify some elements or structures of the data to erase the embedded watermark bits. This is the most common attack to watermarking systems.

All these attacks should be conducted under the restriction that they do not destroy the usability of data.

4. THE BIG PICTURE

Section 4.1 introduces the concept of using query templates to measure the usability of XML data. Section 4.2 gives a framework for watermarking XML data.

4.1 Measures of Data Usability

To design a watermarking scheme that satisfies the three basic requirements mentioned in section 3.1, we need to first know how to properly measure the usability of an XML dataset and find out the corresponding distortion function $U(x, y)$. In the previous works on watermarking relational and XML data, the measures of usability only consider the contents of individual data elements and ignore the structures that link them together. Those measures fail to capture the distortion caused by data reorganization.

Instead, we propose measuring usability by query templates. As the usability of XML data is a measure of whether the data can provide useful and correct information to users, we equate usability with the requirement that the results of some basic queries on the data should remain correct after watermarking or attacks. For example, users usually issue queries in the templates "*db/book[title=?]/author*" and "*db/book[author=?]/title*" to db1.xml in figure 1. If the document can no longer return correct results to these queries after watermarking or attacks, the data would be regarded as useless. Therefore, we can measure the usability of db1.xml by the consistency of the results to the two query templates before and after watermarking and attacks. This measure directly considers the requirements of end users, and takes both data contents and structures into account. With this measure, we can see that data reorganization does not necessarily impair data usability, as queries could be rewritten to retrieve the same results from the reorganized data.

In section 5, we address how to quantify the usability of XML data through a set of query templates, and define the distortion function $U(x, y)$.

4.2 The Framework

Our system for watermarking XML data comprises two schemes – an identification scheme that creates persistent identifiers for the data elements used to embed watermarks, and a watermarking scheme that inserts/detects watermarks into/from the data elements. The identification scheme is crucial for handling the challenges posed by data reorganization, and it constitutes the main contribution of this work.

Based on the requirements and the challenges for watermarking XML introduced in section 3, to ensure the quality of watermarking, the identifiers created by the identification scheme should satisfy the following criteria: (i) *Resilience*: they should be able to survive through data reorganization or alteration. As our scheme represents data usability by a number of query templates, resilience means that as long as the results for the query templates are preserved, the identifiers can correctly identify data elements. (ii) *Redundancy independence*: they should be independent of data redundancies, i.e., duplicated data elements should have the same identifier; otherwise, watermarks could be removed by redundancy removal attacks. (iii) *Differentiability*: they should be able to distinguish between data elements to achieve large watermark capacity. A larger watermark capacity helps to achieve better resilience and credibility of watermarking.

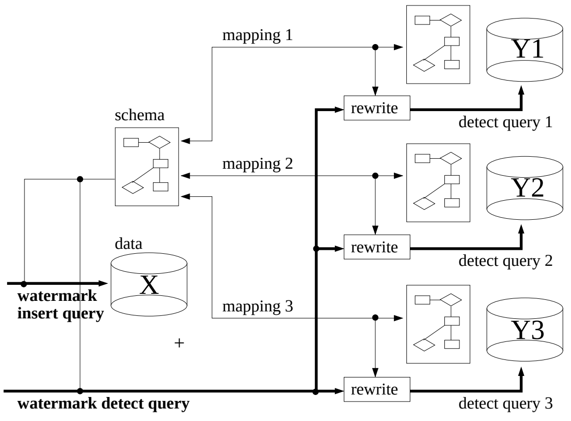


Figure 3: Watermark Insertion and Location

In consideration of these criteria, we choose to use queries as the identifiers of data elements, because a query could be easily adapted to different organizations of the same data through query rewriting. For example, an XPath query "*db/book[title='DB Design']/author*" to be conducted on db1 .xml could be rewritten into another XPath query "*db/publisher/ author[book='DB Design']@name*" to be conducted on db2.xml. Thus, we get a query-based watermarking system which works as follows:

1. Data Element Identification: Use a secret key to select a number of data elements to embed watermark bits. (These data elements should be important to the usability of the XML dataset.) Create queries as identifiers of these data elements through the identification scheme, and safeguard the set of identifiers (denoted by Q) along with the secret key.

2. Watermark Insertion: Execute the queries in Q on the original data to retrieve the data elements (see figure 3). The watermark bits are then embedded into these elements through selected watermark embedding algorithms.

3. Watermark Detection: Execute the same set of queries to retrieve the data elements embedded with watermark bits, and reconstruct the watermark from them. As the schema and the data could be reorganized by attackers, the queries may have to be rewritten for the reorganized data (figure 3). The query rewriting is conducted according to the mapping between the original schema and the new schema, which has been studied extensively in the context of data integration [18, 9, 23]. As the detection process requires query set Q, which contain a part of information of the original data, we regard our approach as semi-blind watermarking [6].

We present the identification scheme in section 5 and the watermarking scheme in section 6.

5. THE IDENTIFICATION SCHEME

This section presents our identification scheme. Section 5.1 gives the preliminary concepts of XML data, and shows that an XML dataset can be represented by a flat table. Section 5.2 shows how to quantify data usability that are expressed by query templates. Section 5.3 presents a basic algorithm that creates identifiers for data elements, and section 5.4 proves the robustness of the identifiers. Section 5.5 extends the algorithm, so it can achieve better watermark capacity.

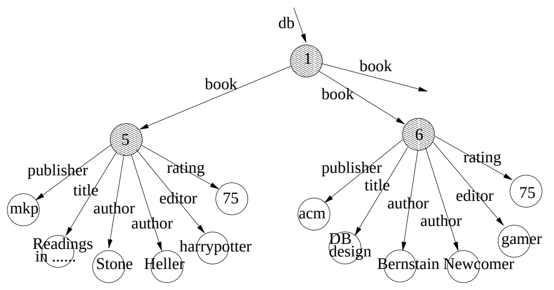


Figure 4: db1.xml as A Tree

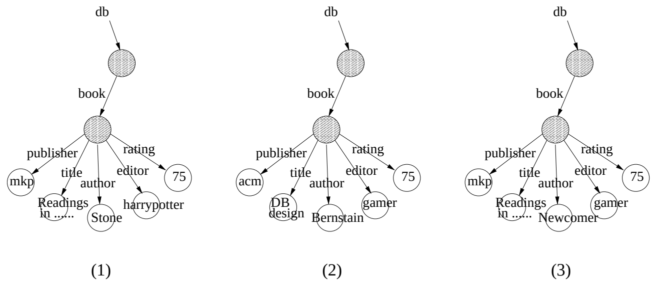


Figure 5: (1) and (2) are tree tuples, but (3) is not

5.1 Preliminaries

An XML dataset is organized in a tree structure that comprises only two types of nodes: *Leaf Node* that contains a value but does not refer to other nodes; *Non-leaf Node* that contains no value but a set of edges that refer to other leaf or non-leaf nodes. The edge pointing to a node is known as the *name* of the node. For instance, the db1.xml in figure 1 can be represented as the tree in figure 4. We distinguish between XML schema and XML tree, where the former is the definition of node name and their relationships (normally father-child relationship) and the latter is the instances of XML data that conform to a certain schema.

Path: a path is a string "$a_1/a_2/.../a_k$", where a_i is the parent of a_{i+1} in an XML schema. We denote the complete set of paths that are rooted at the root of an XML schema S by $paths(S)$. □

For example, "*db/book/publisher*" is a path in db1.xml (figure 4), and the $paths(S)$ of db1.xml include {*db, db/book, db/book/publisher, db/book/title, db/book/author, db/book/editor, db/book/rating*}. In a XML tree T, the complete set of nodes that are reachable by the path *path1* is denoted by {*path1*}.

Query: A query is a string "$a_1([sat_1])/a_2([sat_2])/.../a_k([sat_k])$", where "$a_1/a_2/.../a_k$" is a path. $[sat_i]$ is a selection criterion in the form $[p_{i1} = value_{i1},\ ...\ , p_{ik} = value_{i1}]$, where each "$p_{ij}$" is a sub-path and each $value_{i1}$ is a character string. The results to the query are the nodes in $\{a_1/.../a_k\}$ that satisfy all the sat_i criteria. □

For example, "*db/book[title = 'DB design']/author*" is a query to retrieve the authors of the book titled "DB design".

Query Template: A query template is in the form "$a_1([sat'_1])/a_2([sat'_2])/.../a_k([sat'_k])$", where $[sat'_i] = [p_{i1}, ..., p_{ik}]$ (without comparison operation). We say a query fits into a query template if they are equal after we remove all the comparison operations of the query. □

For example, "*db/book [title = 'DB design']/publisher*" fits in query template "*db/book [title]/publisher*".

Tree Tuple: Suppose T is an XML tree that conforms to a non-recursive schema S, and $paths(S) = \{p_1, ..., p_k\}$. A tree tuple is a set of nodes $\{n_{p_1}, ..., n_{p_k}\}$ in T that satisfy:

- $n_{p_i} \in \{p_i\}$, i.e. node n_{p_i} is a node reachable by path p_i.
- For each pair of paths, say p_i and p_j, if "p_j" = "p_i/a" where a is an edge, then n_{p_j} is the parent of n_{p_i} in T. □

db	**db/book**	**db/book/title**	**db/book/publisher**	**db/book/author**	**db/book/editor**	**db/book/rating**
1	5	Readings in ..	mkp	Stone	harrypotter	75
1	5	Readings in ..	mkp	Heller	harrypotter	75
1	6	DB design	acm	Bernstain	gamer	75
1	6	DB design	acm	Newcomer	game	75
......			......		......	

Figure 6: Flat Table of db1.xml

This definition of tree tuple is adapted from [2]. As illustrated in figure 5, the nodes in (1) compose a tree tuple of db1.xml. So do the nodes in (2). However, the set of nodes in (3) is not a tree tuple – the *title* node and the *author* node in (3) belong to different *book* nodes in the original db1.xml; according to the definition of tree tuple, they cannot become the children of the same *book* node in a tree tuple.

We assign each non-leaf node in an XML tree a *label* (as shown in figure 4), such that the label of two non-leaf nodes are identical if and only if the subtrees rooted at the two non-leaf nodes are identical. Then, an XML tree could be transformed into a *Flat table* (as shown in figure 6), in which each column corresponds to a path rooted at the root, and each tuple corresponds to a tree tuple.

As an example, the flat table of the XML tree in figure 4 is shown in figure 6. Because an XML tree can always be reconstructed from its flat table, the flat table does not lose any information contained in the XML tree. Representing an XML dataset as a flat table will simplify the task of quantifying the usability of the data.

Based on the concept of flat table, we define functional dependency within XML data (a similar definition could be found in [2]).

Functional dependency: Given a XML tree T that conforms to a schema S, a functional dependency (FD) is an expression of the form $P_1 \rightarrow P_2$, where $P_1, P_2 \subset paths(S)$. T satisfies $P_1 \rightarrow P_2$, if for any pair of tuples d_1, d_2 in the flat table $FT(T)$, $d_1.P_1 = d_2.P_1$ implies $d_1.P_2 = d_2.P_2$. □

For example, the FDs in db1.xml include:
db/book/title $\rightarrow$ *db/book;*
db/book $\rightarrow$ *db/book/title;*
db/book $\rightarrow$ *db/book/rating;*
db/book/title $\rightarrow$ *db/book/publisher, db/book/editor;*
db/book/editor $\rightarrow$ *db/book/publisher.*

The FD *'db/book/editor→db/book/publisher'* holds, because we assume that each editor only works for one publisher. From functional dependencies, we can infer that some nodes are duplicates of each other.

Duplicate: Given a FD $P \to p$, where $P \subset paths(S)$ and $p \in paths(S)$, for any pair of tree tuples $\{n^1_{p_1}, ..., n^1_{p_k}\}$ and $\{n^2_{p_1}, ..., n^2_{p_k}\}$, if $n^1_{p_i} = n^2_{p_i}$ for every $p_i \in P$, then n^1_p and n^2_p are duplicates of each other. (because we can deduce from $P \to p$ that $n^1_p = n^2_p$.) □

For example, owing to *db/book/editor→db/book/publisher*, if two *books* have the same *editor*, their *publishers* are duplicates of each other. Duplicates are the main cause of redundancies in XML data.

5.2 Quantifying Data Usability

As we have stated earlier, we measure the usability of XML data by a set of query templates. Through the flat table, we can quantify the usability represented by query templates, and define the distortion function used to measure the usability degradation made by watermarking or attacks.

First, a query on an XML tree can always be written into a selection query on its flat table. For example, the query "*db/book [author = 'Hellerstein', editor = 'Harrypotter']/title*" on db1.xml (figure 4) could be written into query "*SELECT db/book/title FROM FT WHERE db/book/author = 'Hellerstein' AND db/book/editor='Harrypotter'*" to be executed on the flat table in figure 6. This can be formally presented as the following proposition:

Proposition 1 A query "$a_1\big([sat_1]\big)/.../a_k\big([sat_k]\big)$" on an XML tree T is equivalent to the query "*SELECT* $a_1/.../a_k$ *FROM* $FT(T)$ *WHERE* sat_1 *AND ... AND* sat_k" on its flat table. ■

Cover Range: For a query template q = "$a_1\big([sat'_1]\big)/.../a_k \big([sat'_k]\big)$", where $[sat'_i] = [p_{i1}, ..., p_{ik}]$, its cover range $C(q)$ is the set of paths that comprise "$a_1/.../a_k$" and all "$a_1/.../a_i/p_{ij}$". □

For example, in db1.xml, the cover range of "*db/book[author, editor]/ title*" is {"*db/book/title*", "*db/book/ author*", "*db/ book/editor*"}.

Then, we quantify data usability as follows:

Usability: If a query template q is used to express the usability of an XML tree T, the quantity of the usability w.r.t. q can be measured by the integrity of $FT(T)$'s projection on q's cover range $C(q)$. (The integrity can be measured by the ratio of inconsistent tuples in this projection before and after watermarking/attacks) □

Consider a query template "*db/book [author, editor]/title*", which is used to describe the usability of db1.xml (figure 4). According to Proposition 1, for any x, y, query "*db/book [author = x, editor = y]/title*" is equivalent to the query "*SELECT db/book/title FROM FT WHERE db/book/author = x AND db/book/editor = y*" on the flat table. Therefore, the projection of the flat table on the cover range {*db/book/title, db/book/author, db/ book/editor*} is sufficient and necessary for answering all the queries that fit in "*db/book[author, editor]/title*". If "*db/book[author, editor]/title*" represents the usability of db1.xml, then the integrity of this projection quantifies the usability.

With the quantification of the usability of XML data, we have the following definition of distortion function.

Distortion: Suppose Q is a set of query templates used to express the usability of an XML tree T, and T is transformed into T' by watermarking or attacks. Then, the distortion on $q \in Q$ made by transforming T to T' is denoted by $U^q(T, T')$, which is the ratio of inconsistent tuples between the projection of $FT(T)$ over $C(q)$ and the projection of $FT(T')$ over $C(q)$, namely

$$U^q(T, T') = \frac{\pi_{C(q)}\big(FT(T)\big) - \pi_{C(q)}\big(FT(T')\big)}{\pi_{C(q)}\big(FT(T)\big)}$$

The overall distortion is $U(T, T') = \max_{q \in Q} U^q(T, T')$. □

5.3 Identifier Creation – Basic Algorithm

In this section, we introduce an algorithm to create an identifier for each data element in an XML tree. As watermarks could be embedded in an XML tree by modifying the value of its leaf nodes or changing the attributes of its non-leaf nodes, both types of nodes are to be identified. The created identifiers should satisfy the following three criteria (as mentioned in section 4.2).

Resilience: To survive reorganization attacks, the node identifiers should be closely related to data usability, such that as long as data usability is preserved, the identifiers are preserved too. In other words, the node identifiers should be able to identify most nodes correctly as long as the distortion on the data is within a tolerable threshold.

Redundancy Independence: To counter redundancy removal, duplicated (redundant) nodes should be assigned identical identifiers. We assume that redundancies in XML data are produced by functional dependencies. Although there may be redundancies that are not caused by FDs, they are beyond the scope of this paper.

Differentiability: To economize available watermark capacity, the identifiers should differentiate between nodes that are not duplicates of one another.

To create identifiers for the nodes in an XML tree (whose schema is S), the crucial step is to find a set of paths $P \subset paths(S)$ to identify each path $p \in paths(S)$. From the perspective of flat table, it is to find a set of attributes to identify each attribute of the flat table. For example, in db1.xml (figure 4), as each book node can be uniquely identified by its title, we can use "*db/book/title*" to identify "*db/book*". With this path identifier, we can readily create a query in the form "*db/book[title = ?]*" to identify each node in {*db/book*}. For instance, the identifier of the first book node in db1.xml will be "*db/book[title = 'Reading in Database Systems']*". In our algorithm, we only present how to create path identifiers and ignore the trivial step of creating identifiers for individual nodes.

Before presenting our algorithm, we first give a definition of minimum determinant.

Minimum Determinant: For a path $p \in paths(S)$, *if* there exists a set of paths $P_1 \subset paths(S)$ that satisfy: (a) each path in P_1 ends with a leaf node, (b) $P_1 \to p$, (c) there is no $P_2 \subset paths(S)$ such that $P_2 \to p$ and $P_2 \not\to P_1$, *then* P_1 is called a minimum determinant of p. □

As in db1.xml (figure 4), the minimum determinant of "*db/book*" is "*db/book/title*". The minimum determinant of "*db/book/publisher*" is "*db/book/editor*".

Figure 7 shows the algorithm to create identifier for each

```
/* we denote the cover range of $q$ by $C(q)$; */
1)  func CreateID ($S$, $FD$, $U$)
2)    foreach $p \in paths(S)$, do
3)      if there is no $q \in U$ such that $p \in C(q)$, then
4)        $ID(p) = NULL$;
5)      elseif $p$ has no minimum determinant, then
6)        $ID(p) = p$;
7)      else /* $p$'s minimum determinant is $P$ */
8)        if there is a $q \in U$ that $p \in C(q)$ and $P \subset C(q)$, then
9)          $ID(p) = P$;
10)       else
11)         $ID(p) = p$;
12) end func
```

Figure 7: Algorithm 1: Basic Identifier Creation

path in an XML tree. The input to this algorithm is the XML tree's schema S, the set of existing functional dependencies FD and a set of usability templates U that are selected to describe the usability of the XML tree. For each path $p \in paths(S)$, the algorithm outputs a set of paths to be its identifier, denoted by $ID(p)$. As presented in the algorithm, if p is not contained in the cover range of any query template, it is not contributing to the usability of the data. So we do not use the nodes in $\{p\}$ to embed watermarks, and put p's identifier as *null*. If p has a minimum determinant P, and both p and P are contained in the cover range of the same query template, we use P as p's identifier; otherwise, we use p as its own identifier. The complexity of the algorithm is $O\big(|paths(S)| \times max_{q \in U}|C(q)|\big)$.

As an example, to create identifiers for the nodes in db1.xml, we input usability templates "*db/book[title] /author*", "*db/book [title]/publisher*" and "*db/book[title]/editor*", and the schema and FDs described in section 5.1. Algorithm 1 creates an identifier for each path of db1.xml. The identifier of "*db/book*" is its minimum determinant "*db/book/title*". The identifier of "*db/book/publisher*" is itself. While it has a minimum determinant "*db/book/editor*", they are not contained in the cover range of the same query template.

5.4 Proof of Resilience

In this section, we prove that the identifiers created by above algorithm are resilient to reorganization or alteration attacks and are independent of redundancies. To prove resilience, we show that destroying identifiers will destroy data usability to the same degree.

Assumption 1: given a flat table T, and two cover ranges C_1 and C_2 such that $C_1 \subset C_2$, the tuples in T's projection over C_1 have the same expected number of replicas in T's projection over C_1. (This assumption holds in many real world data.) ■

Proposition 2: Suppose $p \in paths(S)$, and the identifier created by Algorithm 1 for p is P. If $X\%$ of the nodes in $\{p\}$ are not identifiable after an attack, then the distortion made by this attack to the XML data is at least $X\%$.
Proof. Based on Algorithm 1, p and P must be covered by the cover range of the same query template, denoted by $C(q)$. If $X\%$ of the nodes in $\{p\}$ are not identifiable after the attack, then there are $X\%$ of the tuples destroyed in the projection of the flat table on $P \cup p$. As $P \cup p \subset C(q)$, there are also about $X\%$ of tuples destroyed in the projection of the flat table on C (owing to the Assumption 1). Therefore, the distortion made by the attack on q is $X\%$, i.e.,$U^q(T, T') = X\%$. According to the definition of distortion, the distortion made by the attack on the whole XML data is at least $X\%$. ■

To prove redundancy independence, we show that duplicated nodes are always identified by the same identifier.

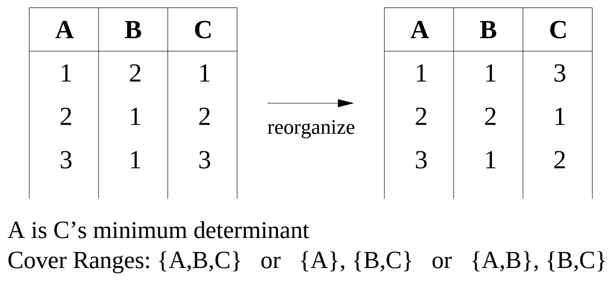
Proposition 3: Suppose $p \in paths(S)$. If two nodes $n_1, n_2 \in \{p\}$ are duplicates of each other, then n_1 and n_2 will be assigned the same identifier by Algorithm 1.
Proof. Based on the definition of duplicate, there must be a FD $P1 \rightarrow p$, and n_1, n_2 must belong to two tree tuples $\{n^1_{p_1}, ..., n^1_{p_k}\}$ and $\{n^2_{p_1}, ..., n^2_{p_k}\}$, such that $n_1 = n^1_p, n_2 = n^2_p$ and $value(n^1_{p_i}) = value(n^2_{p_i})$ for all $p_i \in P1$. If the identifier of p is itself, since $value(n^1_p) = value(n^2_p)$, n_1 and n_2 will be assigned the same identifier. If the identifier of p is its minimum determinant $P2$, according to the definition of minimum determinant, $P1 \rightarrow P2$. Therefore $value(n^1_{p_j}) = value(n^2_{p_j})$ for all $p_j \in P2$, and again n_1 and n_2 will be assigned the same identifier. ■

5.5 Identifier Creation – Extended Algorithm

According to Algorithm 1, the identifier of a path p is either p itself or its minimum determinant. Obviously, using the minimum determinant as identifier could further differentiate the nodes in $\{p\}$ than using p itself. However, to counter reorganization attacks, Algorithm 1 allows minimum determinant to be used only if p and its minimum determinant are contained in the cover range of the same query template. This condition is overly strict and limits the differentiability of created identifiers. Our first extension to Algorithm 1 is to relax the condition of using minimum determinants, so as to improve the differentiability of the created identifiers.

Suppose an XML tree T that conforms to schema S and $paths(S) = \{A,\ B,\ C\}$. Figure 8 shows the flat table of T, with A, B and C as its attributes. Suppose the minimum determinant of C is A, and the query templates to express the usability of T is U. We consider whether to use A as C's identifier in different scenarios.
Scenario 1: U contains only one query template $U = \{q\}$, and its cover range $C(q)$ is $\{A, B, C\}$. In this scenario, it is safe to use A as C's identifier. According to Proposition 2, to destroy the identifiers, an attacker would have to modify the flat table's projection on $\{A, C\}$, which is not possible without changing the projection on $\{A, B, C\}$.
Scenario 2: U contains two query templates $U = \{q_1, q_2\}$, and $C(q_1) = \{A\}$ and $C(q_2) = \{B, C\}$. In this scenario, it is unsafe to use A as C's identifier. As shown in figure 8, a simple attack is to reorder the records in columns B and C, which could change the projection on $\{A, C\}$ entirely while preserving the projections on $\{A\}$ and $\{B, C\}$. This scenario has been considered by Algorithm 1.
Scenario 3: U contains two query templates $U = \{q_1, q_2\}$, and $C(q_1) = \{A, B\}$ and $C(q_2) = \{B, C\}$. In this scenario, it is possible to use A as C's identifier. An attacker cannot arbitrarily change the projection on $\{A, C\}$, as it would either destroy the projection on $\{A, B\}$ or destroy the projection on $\{B, C\}$. In fact, to preserve data usability, the attacker can only change some tuples that have identical B fields. If the replication factor of the records in column B is very low, only a small number of tuples can be changed, so it is still practical to use A as C's identifier. This scenario is ignored by Algorithm 1.

Based on the above observation, we introduce the con-



A	B	C
1	2	1
2	1	2
3	1	3

$\xrightarrow{\text{reorganize}}$

A	B	C
1	1	3
2	2	1
3	1	2

A is C's minimum determinant
Cover Ranges: {A,B,C} or {A}, {B,C} or {A,B}, {B,C}

Figure 8: Example of Reorganization (after the reorganization, the projections on {A} and {B,C} are preserved, but not for {A,B,C} or {A,C})

cepts of γ-related and γ-closure, which will be used in identifier creation to achieve better watermark capacity.

γ-related: Given two query templates q_1 and q_2, whose cover ranges $C(q_1)$ and $C(q_2)$ overlap. In the projection of the flat table on $C(q_1) \cup C(q_2)$, if the replication factor of the records in the columns covered by $C(q_1) \cap C(q_2)$ is less than γ, then we say that q_1 and q_2 (or $C(q_1)$ and $C(q_2)$) are γ-related to each other. $\square$

γ-closure: Given a set of query templates Q and their cover ranges $C(Q)$, a γ-closure of cover ranges, denoted by clo, is a subset of $C(Q)$, such that each pair of $C(q_1), C(q_2) \in clo$ is γ-related, and there is no $C(q_3) \notin clo$ that is γ-related to any cover range in clo. $\square$

When two cover ranges $C(q_1)$ and $C(q_2)$ are γ-related and γ is very small, it is difficult to modify the flat table's projection on $C(q_1) \cup C(q_2)$ without modifying the usability projection on $C(q_1)$ or $C(q_2)$. Thus, if a path and its minimum determinant are covered by a γ-closure of cover ranges, it is still safe to use the minimum determinant as the identifier of the path. This enables us to use more minimum determinants to achieve a larger watermark capacity.

Our second extension concerns some paths that do not appear on the right side of any functional dependency, e.g. "db/book/author" in db1.xml. According to the definition of duplicate, if a path p does not appear on the right side of any FD, the nodes in $\{p\}$ will not contain duplicates. It seems that we can use anything as p's identifier, without worrying about redundancy problem. However, if the parents of some nodes in $\{p\}$ are duplicates of each other, we can still deduce that these nodes are identical. Therefore, in our extended algorithm, we use the path itself and the identifier of its longest prefix path as its identifier, e.g. ID("db/book/author") = {ID("db/book"), "db/book/author"} = {"db/book/title", "db/book/ author"}.

Finally, figure 9 shows the extended algorithm for identifier creation, which offers higher watermark capacity than Algorithm 1. In the algorithm, a user only needs to set γ to a sufficiently small value so that it is difficult for an attacker to disable the identifiers through reorganization. The complexity of the algorithm is also $O\big(|paths(S)| \times max_{q \in U}|C(q)|\big)$.

To summarize, section 5 presents a scheme that can create resilient identifiers for the data elements in an XML dataset. The identifiers are tied closely to the query templates used to represent the data usability. After reorganization attack, as long as the query templates (after being rewritten) can return correct results, the identifiers (through query rewriting) can correctly identify the data elements from the reor-

```
func CreateID ($S$, $FD$, $U$, $\gamma$)
  generate all cover ranges of U and their $\gamma$-closures;
  foreach $p \in paths(S)$, do
    if there is no $q \in U$ such that $p \in C(q)$, then
      $ID(p) = NULL$;
    if $p$ has no minimum determinant, then
      if there is no FD $P_1 \rightarrow p$, then
        $ID(p)$ = $ID(p$'s longest prefix$) + p$;
      else
        $ID(p) = p$;
    else /* $p$'s minimum determinant is $P$ */
      if $p$ and $P$ are contained in a $\gamma$-closure, then
        $ID(p) = P$;
      else
        $ID(p) = p$;
end func
```

Figure 9: Algorithm 2: Extended Identifier Creation

ganized data. In addition, the identifiers are independent of redundancies caused by functional dependencies, while being able to differentiate data elements to the full extent.

5.6 Some Remarks on Usability Templates

Selecting appropriate query templates to represent the usability of an XML document is crucial to the success of watermarking. If one chooses too many query templates that are not important to users, it would impair the resilience of watermarking, because attackers can reorganize the data without regard to these templates, such that some watermark identifiers created upon these templates are destroyed. On the other hand, if the important query templates are not all taken into account, we could lose some watermark capacity. In practice, the templates selection can be conducted by domain experts, following guideline – (1) all selected templates should be important; (1) all important query templates should be included.

6. THE WATERMARKING SCHEME

On top of the above identification scheme, we devise a watermarking scheme for embedding watermarks in data elements. Watermark can be embedded in (i) leaf nodes by modifying their values and (ii) non-leaf nodes by adding to or deleting their child nodes. For example, the first 'book' in db1.xml contains two 'author' attributes. We can change the attribute number to 1 by deleting an 'author', or to 3 by adding a fake 'author'. Information is embedded through such additions and deletions. Our watermarking scheme makes use of both leaf nodes and non-leaf nodes.

As an XML dataset may contain various data types with unique characteristics, our scheme employs different watermarking algorithms for them. A number of existing watermarking algorithms can be adopted directly, such as the algorithms for watermarking text [3], numeral [21], and image [22]. As it is not the objective of this paper to design watermarking algorithms for these different data types, we shall not discuss them further. Instead, we assume some common properties of the leaf nodes and non-leaf nodes, and present a "example" watermarking algorithm for XML data. We assume that each leaf node contains bits of different significance, such that modifying a more significant bit would result in a larger deviation from its original value. Similarly, the attributes of a non-leaf node also have different significance.

6.1 Preparation

```
proc embed($\phi$, $k$, $wm$)
   foreach leaf node $A$, do
      for $i$=1 to |LSB($A$)|, do
         $r$=hash($i$,$A$,$k$) mod P; /*P is a large prime*/
         if $r < \phi \times P$, then
            $j$=hash($i$,$A$,$k$) mod |$wm$|;
            $b_i(A)$=$wm_j$;
            add $ID(A)$ to $Q$;
   foreach non-leaf node $B$, do
      for $i$=1 to |LSA($B$)|, do
         $r$=hash($i$,$B$,$k$) mod P;
         if $r < \phi \times P$, then
            $j$=hash($i$,$B$,$k$) mod |$wm$|;
            if |$a_i(B)$| mod 2$\neq$ $wm_j$, then
               delete a $a_i(B)$;
            add $ID(B)$ to $Q$;
end proc
```

Figure 10: Algorithm 3: Watermark Insertion

In the preparation step, the user first discovers all the functional dependencies based on the schema of the XML data, and specifies a number of query templates that represent the usability of the data. He also determines the potential leaf nodes and non-leaf nodes to be used to embed watermarks, and the MSB/MSA (most significant bits/attributes) and the LSB/LSA (least significant bits/ attributes) for each type of nodes. Then, he inputs the information to the identification scheme, which creates an identifier for each potential leaf node or non-leaf node to be watermarked. (As shown in Algorithm 1 and Algorithm 2, if a node is not contributing to the data usability, its identifier will be empty and it will not be used to embed watermarks.)

When creating node identifiers, we only use MSB/MSA (instead of full values) in the selection criteria of the created identifying queries, so that slight modification of the values or attributes would not affect the accuracy of identification and at the same time retain the differentiability of the created identifiers. The LSB/LSA are the bits/attributes that could sustain a certain distortion, so they are used to embed watermarks. It is worth noting that "least significant bits" is different from "insignificant bits"; over-modifying the LSB/LSA will also destroy the usability of data.

6.2 Watermark Insertion

After preparation, each LSB/LSA constitutes a potential object to embed a watermark bit. However, to preserve data usability, only a small fraction of the LSB/LSA is selected for modification. Let ϕ denote the proportion of the selected LSB/LSA. A secure hash function $h()$ is used to assign to each LSB/LSA a hash value, which decides whether that LSB/LSA is used to embed a watermark bit. The hash value of the ith LSB/LSA of node A is

$$hash(i, A, k) = h\big(h(ID(A)) \otimes h(i) \otimes h(k)\big)$$

where $ID(A)$ is the query to identify A and k is a secret key. We denote the watermark by wm, and the ith bit of wm by wm_i. In addition, we denote the ith LSB of a leaf node A by $b_i(A)$, and the ith LSA of a non-leaf node B by $a_i(B)$. Q is a set of queries to identify the selected leaf/non-leaf nodes for embedding watermarks. Q and k are secrets that the user keeps, and are used for watermark detection subsequently. The watermark embedding algorithm is shown in figure 10.

The input of the algorithm is ϕ (the proportion of LSB/LSA used to embed watermark), the secret key k and the watermark wm. First, it embeds watermark bits into the leaf nodes. For each LSB in a leaf node, a hash value is calculated that decides whether that LSB is to be used for watermarking. If so, this LSB embeds the (hash(i,A,k) mod $|wm|$)th bit of the watermark. Next, the algorithm embeds watermark bits into the non-leaf nodes. The same selection criterion is used to determine the LSA of non-leaf nodes for watermarking, where watermark bits are embedded through attribute deletion. The complexity of the algorithm is $O\big(n \times max(LSB, LSA)\big)$, where n is the number nodes in the XML tree being watermarked, and $max(LSB, LSA)$ denotes the maximum number of LSB and LSA on each node.

```
proc detect($Q$, $\phi$, $k$, $wm$)
   foreach query ID($A$) in $Q$, do
      if $A$ is a leaf node, then
         for $i$=1 to |LSB($A$)|, do
            $r$=hash($i$,$A$,$k$) mod P; /*P is a large prime*/
            if $r < \phi \times P$, then
               $j$=hash($i$,$A$,$k$) mod |$wm$|;
               $q$ = rewrite(ID($A$));
               retrieve $C$ by query $q$;
               if $b_i(C)$=1, then
                  $wm_t[j]$ += 1;
               else $wm_f[j]$ += 1;
      else if $A$ is a non-leaf node, then
         for $i$=1 to |LSA($A$)|, do
            $r$=hash($i$,$A$,$k$) mod P;
            if $r < \phi \times P$, then
               $j$=hash($i$,$A$,$k$) mod |$wm$|;
               $q$ = rewrite(ID($A$));
               retrieve $C$ by query $q$;
               if |$a_i(C)$| mod 2=1, then
                  $wm_t[j]$ += 1;
               else $wm_f[j]$ += 1;
end proc

proc construct($wm$, $v$)      /*$\frac{1}{2} < v < 1$*/
   for $i$=0 to |$wm$|-1, do
      $x = \frac{wm_t[i]}{wm_t[i]+wm_f[i]}$;
      if $x > v$, then
         $wm_i$ = 1;
      else if $x < 1 - v$, then
         $wm_i$ = 0;
      else $wm_i$ = undefined;
end proc
```

Figure 11: Algorithm 4: Watermark Detection

6.3 Watermark Detection

Watermark detection is the reverse process of watermark insertion. As the nodes are identified via queries, query rewriting is the initial step of watermark detection: for selecting the watermarked nodes (see section 4), the user rewrites the queries on the original data into queries on the reorganized data according to the mapping between the original and reorganized schemas. As query rewriting is beyond the scope of our work, we omit the details. The basic techniques can be found in [12, 9, 18]. While query rewriting can only be done in a semi-automatic way, it does not affect the soundness of our watermarking schema. The watermark detection algorithm is shown in figure 11.

Each query in Q is rewritten and executed on the target data to retrieve the nodes that carry the watermarks. Each LSB/LSA of the nodes is decided by the selection criterion whether it has been watermarked. If so, its value is used to vote whether the corresponding watermark bit is 1 or 0. For each bit of the watermark, two buckets, $wm_t[i]$ and $wm_f[i]$, are used to count the votes. $wm_t[i]$ records the number of votes supporting $wm_i = 1$ and $wm_f[i]$ records the number of votes supporting $wm_i = 0$. Finally, the watermark is reconstructed based on a majority rule with υ as the threshold. The complexity of the algorithm is $O\big(|Q| \times max(LSA, LSB)\big)$.

6.4 Analysis

This section analyzes the credibility level offered by the watermarking scheme and its resilience to various attacks.

Credibility reflects how confident we are that the data has been purposely watermarked by the owner. It measures the resilience of a watermarking scheme against invertibility attacks – given a dataset and a specific watermark wm, what is the difficulty (probability) for an attacker to figure out a Q and a k that can detect wm from the dataset? If the hash function $h()$ used in our watermarking algorithm is sufficiently secure, it will be impossible for an attacker to find a k that can generate a Q to extract the correct wm. What an attacker can only do is to randomly guess the k. Based on our algorithm, the probability of a successful guess is $p = \frac{1}{2^{|wm|}}$, where $|wm|$ is the length of the watermark. If the watermark contains 32 bits, this probability will be 2^{-32}, which is small enough to defend against such random guesses. Therefore, the credibility of our watermarking scheme can be high if we can insert an watermark of an adequate size.

Attack4 (selection) can succeed only if all the LSB/LSAs that have been embedded with one of the watermark bits are removed. Given N bits of LSB/LSAs in the dataset, on the average $\frac{N\phi}{|wm|}$ bits are used to embed a single watermark bit. If an attacker selects $X\%$ of the data and discards the rest, the probability of a successful attack is approximately $1 - \big[1 - \big(1 - X\%\big)^{\frac{N\phi}{|wm|}}\big]^{|wm|}$. With $N = 5 \times 10^5$, $|wm| = 20$ and $\phi = 0.5\%$, even if the attacker discards 90% of the data, the probability of a successful attack is only about 4×10^{-5}.

In Attack5 (alteration), to preserve data usability, the attacker is only allowed to introduce a small distortion to the dataset. It means that he can only alter a small fraction of the LSB/LSAs. Then, the resilience of our algorithm against this attack is measured by the probability that modifying a certain fraction of the LSB/LSA destroys the watermark. If the attacker randomly reverses $X\%$ of the LSB/LSA, the probability of a successful attack is $1 - \big[\sum_{i=\upsilon n}^{n} \binom{n}{i} (1-X\%)^i X\%^{(n-i)}\big]^{|wm|}$, where $n = \frac{N\phi}{|wm|}$. With $N = 5\times10^5$, $\upsilon = \frac{1}{2}$, $|wm| = 20$ and $\phi = 0.5\%$, even if the attacker reverses 35% of the LSB/LSAs, the probability of a successful attack is only about 3×10^{-3}.

The same applies to Attack1 (reorganization), in which an attacker reorganizes the data to prevent watermarked nodes from being correctly identified. Proposition 2 has shown that destroying node identifiers will destroy data usability to the same degree. Therefore, if the maximum tolerable distortion is $X\%$, the attacker is only allowed to disable $X\%$ of the node identifiers. The probability of a successful attack is also $1 - \big[\sum_{i=\upsilon n}^{n} \binom{n}{i} (1-X\%)^i X\%^{(n-i)}\big]^{|wm|}$, where $n = \frac{N\phi}{|wm|}$. This indicates that reorganization attacks are actually no more powerful than alteration attacks.

Attack2 (redundancy removal) does not apply to our watermarking scheme, as the node identifiers created by our identification scheme are independent of the redundancies inside the data.

Attack3 (addition) is not able to compromise our watermarking scheme either. An attacker can merge the watermarked dataset with some relevant dataset, and conduct reorganization afterwards. However, as long as the usability of the data is preserved, through Q we can still locate the watermarked nodes from the merged dataset and retrieve the watermark from them.

An attacker may choose to perform several kinds of attacks simultaneously. However, this will not improve his chance of success, as the destruction made by the attacks to data usability will accumulate.

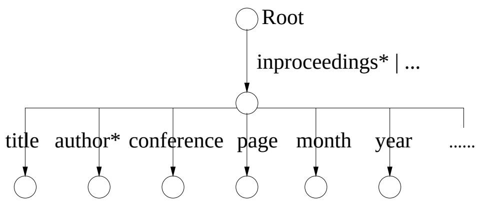


Figure 12: Schema of DBLP Dataset

7. EXPERIMENT

We have implemented a system for watermarking XML, named WmXML, based on the architecture in figure 3. A demonstration of this system has been published in [24].

Section 6.4 has theoretically analyzed the effectiveness of our watermarking system. This section presents the results of some experiments on this system, which are intended to establish how much incremental watermark capacity our watermarking scheme achieves, and how resilient the scheme is against attacks. The XML data we used in our experiments is the DBLP dataset, which encodes the information of more than 600,000 publications. Its schema is shown in figure 12.

7.1 Watermark Capacity

We conducted experiments to confirm that our system is able to further differentiate the data elements in a semistructure document than the naive method of identifying each data element directly by its value. In the experiments, we used our system to create identifiers for all the leaf nodes in the DBLP data set, and compared their quantity with those created by the naive method. Figure 13 compares the number of identifers created by the two methods for three types of leaf nodes – *author*, *pages* and *year*. Obviously, our method is able to differentiate and identify many more leaf nodes than the naive approach. The contrast is especially significant for those nodes with small values, e.g. *year*. As the naive approach does not differentiate between leaf nodes containing the same value, it loses a large amount of watermark capacity.

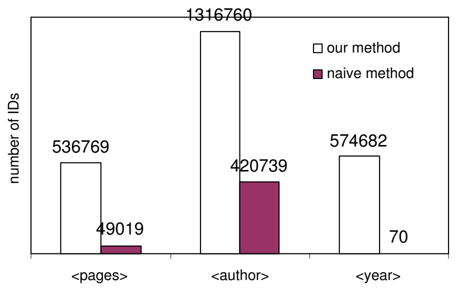
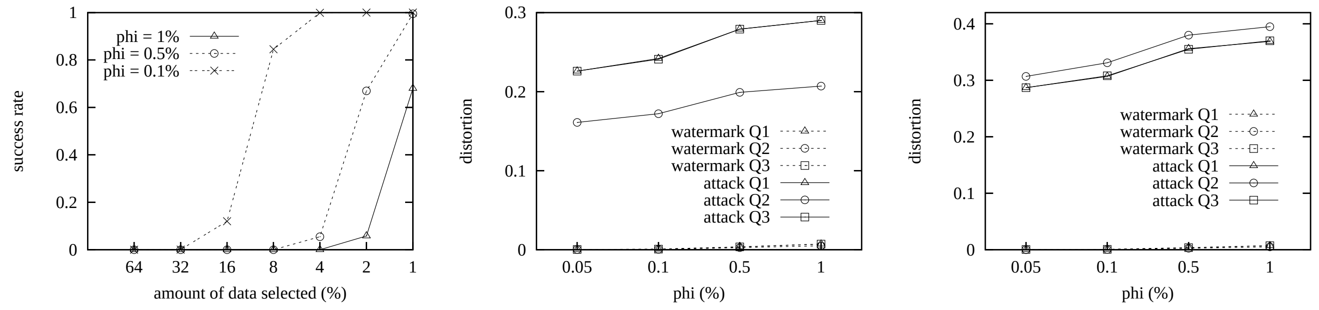


Figure 13: Watermark Capacity



(a) Attack4: Selection (b) Attack5: Alteration (c) Attack1: Reorganization

Figure 14: Resilience to Various Attacks

7.2 Resilience

We also conducted a set of experiments to study the resistance of our scheme to various kinds of attacks. To simplify our study, we only embedded watermarks by modifying the number of *author* attributes under each *inproceedings* node. (We assume that a very small proportion (say 0.1%) of incorrect *author* attributes is acceptable for most users.) The parameters of the data set and our watermarking procedure are given in table 1.

According to DBLP web-site (http:// dblp.uni-trier.de/), we summarize the usability of the bibliographic data into three applications:
1: find the authors of a given publication;
2: find the publications of a given author;
3: find the publications of a particular conference or journal.
These can be captured by the query templates in table 1. Thus, our experiments were designed to assess whether attackers could remove the watermarks without destroying the results to the three usability query templates. If that is not achievable, we consider our system to be secure. An attack's damage to the three types of usability is measured by the *distortion* defined in section 5.2, which measures the percentage of mismatch between the flat table's projections on the query templates before and after attacks.

The first set of experiments is intended to study the resilience of our system against Attack4 (selection), in which an attacker selects a part of the bibliographic data that satisfies his intended purpose and discards the rest. We simulated the attack through random selection. Figure 14 (a) depicts the *success rate* of this attack (the probability that a watermark is removed by the attack) when selecting different proportions of data. If $\phi = 0.1\%$ (i.e., 0.1% of the *inproceedings* nodes are used to embed watermarks), the attack succeeds only when the selected data is less than 16%. When $\phi = 1\%$, even if an attacker selects only 2% of the data, he is not likely to remove the embedded watermark. As discarding too much data directly compromises the usability of DBLP, our watermarking scheme is highly resilient to selection attacks.

The second set of experiments aims to study the resilience of our system against Attack5 (alteration). To simulate the attack, we randomly selected some *inproceedings* nodes, and modified their *author* attributes in order to reverse the possible embedded watermark bits. We then tested whether the attack successfully removed the watermark (success rate $\geq 50\%$) without destroying the three query templates mentioned previously. We also varied parameter ϕ to observe how it increased the difficulty of the attacks. Figure 14 (b) shows the *distortion* incurred by successful attacks, in contrast to those incurred by the watermarking procedure. As expected, when ϕ is larger (more nodes are selected to embed watermark bits), both watermarking procedures and attacks incur higher distortion. However, even when $\phi = 1\%$, the distortion made by watermarking are still insignificant ($\approx 0.5\%$). In contrast, the distortion made by Attack5 are very significant ($20\% \leq$ distortion $\leq 30\%$), and can hardly be accepted by users. Thus, it is difficult for Attack5 to remove watermark without destroying the usability of the data.

The last set of experiments was designed to study the resilience of our system against Attack1, which attempts to reorganize the structure to prevent the watermarked nodes from being identified correctly (although this has been analyzed in section 5.4). As mentioned earlier, we only embedded watermark bits into the *inproceedings* nodes. The identifier created by our system for *inproceedings* is "*Root/ inproceedings[title]*". To simulate the attacks, we randomly exchanged the *title* attributes to disable the identification of the *inproceedings* nodes. We then tested whether the attack succeeded (success rate $\geq 50\%$) without destroying the results to the three query templates. The results are shown in figure 14 (c). In accordance with the analysis in section 5.4, reorganization degrades data usability in proportion with the identifiers it disables. The distortion made by the reorganization attack are very significant ($30\% \leq$ distortion $\leq 40\%$). To disable the identification of watermarked node, the reorganization attacks have to incur the same degree of destruction to data usability as alteration attacks (Attack5).

Parameter	Value
Size of DBLP data set	245 MBytes
Number of bibliographies in DBLP	575744
Number of different authors in DBLP	420739
ϕ(phi) – % of nodes to embed watermark	0.05%∼1%
v – parameter in algorithm 4	0.5

Notation	Query Templates
Q1	"*Root/inproceedings[title]/author*"
Q2	"*Root/inproceedings[author]*"
Q3	"*Root/inproceedings[conference]/title*"

Table 1: Experiment Parameters

8. CONCLUSION

In this paper, we proposed a query-based watermarking technique for XML data. We showed that data usability could be represented by query templates, and data element identifiers could be created through the query templates and functional dependencies, so that the identifiers are highly resilient against data reorganization and redundancy removal, while making the best of available watermark capacity. We devised and implemented a watermarking system for XML based on the proposed technique, and experiments results confirmed its resilience to various attacks.

There are some issues for further investigation. (1) The query templates we used to represent data usability are templates of simple selection queries. More complex queries involving joins and aggregations could also be incorporated. (2) Different user groups may have different definitions of data usability. It would be interesting to design a watermarking scheme that can counter attacks from different user groups. (3) There may be redundancies in the data that are not caused by functional dependency. Their influence on watermarking needs to be studied further.

9. REFERENCES

[1] R. Agrawal and J. Kiernan. Watermarking relational data: framework, algorithms and analysis. *The VLDB Journal*, 12(2):157–169, 2003.

[2] M. Arenas and L. Libkin. A normal form for xml documents. In *Proceedings of the twenty-first ACM symposium on Principles of database systems*, 2002.

[3] M. Atallah, V. Raskin, C. Hempelmann, M. Karahan, R. Sion, and K. Triezenberg. Natural language watermarking and tamperproofing. In *Proceedings of the 5th International Information Hiding Workshop*, 2002.

[4] L. Boney, A. H. Tewfik, and K. N. Hamdy. Digital watermarks for audio signals. In *Proceedings of International Conference on Multimedia Computing and Systems*, pages 473–480, 1996.

[5] S. Craver, N. Memon, B.-L. Yeo, and M. M. Yeung. On the invertibility of invisible watermarking techniques. In *ICIP '97: Proceedings of the 1997 International Conference on Image Processing (ICIP '97) 3-Volume Set-Volume 1*, page 540, Washington, DC, USA, 1997. IEEE Computer Society.

[6] E. Elbasi and A. M. Eskicioglu. A dwt-based robust semi-blind image watermarking algorithm using two bands. In E. J. Delp, III and P. W. Wong, editors, *Security, Steganography, and Watermarking of Multimedia Contents VIII*, pages 777–787, Feb. 2006.

[7] C. Fei, D. Kundur, and R. Kwong. The choice of watermark domain in the presence of compression. In *Proceedings of International Conference on Information Technology: Coding and Computing*, pages 79–94, 2001.

[8] D. Gross-Amblard. Query-preserving watermarking of relational databases and xml documents. In *Proceedings of the twenty-second ACM symposium on Principles of database systems*, pages 191–201. ACM Press, 2003.

[9] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[10] F. Hartung and B. Girod. Watermarking of uncompressed and compressed video. *Signal Processing*, 66(3):283–301, 1998.

[11] S. Khanna and F. Zane. Watermarking maps: hiding information in structured data. In *SODA '00: Proceedings of the eleventh annual ACM-SIAM symposium on Discrete algorithms*, pages 596–605, Philadelphia, PA, USA, 2000. Society for Industrial and Applied Mathematics.

[12] M. Lenzerini. Data integration: a theoretical perspective. In *PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 233–246, New York, NY, USA, 2002. ACM Press.

[13] Y. Li, H. Guo, and S. Jajodia. Tamper detection and localization for categorical data using fragile watermarks. In *DRM '04: Proceedings of the 4th ACM workshop on Digital rights management*, pages 73–82, New York, NY, USA, 2004. ACM Press.

[14] Y. Li, V. Swarup, and S. Jajodia. Fingerprinting relational data: schemes and specialties. *IEEE Transaction on Dependable and Secure Computing*, 2(1):34–45, 2005.

[15] S. Liu, S. Wang, R. H. Deng, and W. Shao. A block oriented fingerprinting scheme in relational database. In *ICISC'04: International Conference on Information Security and Cryptology*, 2004.

[16] P. Moulin and J. O'Sullivan. Information-theoretic analysis of information hiding. *IEEE Transactions on Information Theory*, 49, 2003.

[17] W. Ng and H.-L. Lau. Effective approaches for watermarking xml data. In *DASFAA'05: International Symposium on Database Systems for Advanced Applications*, 2005.

[18] Y. Papakonstantinou and V. Vassalos. Query rewriting for semistructured data. In *Proceedings of the 1999 ACM SIGMOD*, pages 455–466. ACM Press, 1999.

[19] R. Sion. Proving ownership over categorical data. In *20th International Conference on Data Engineering (ICDE'04)*, page 584, 2004.

[20] R. Sion, M. Atallah, and S. Prabhakar. Resilient information hiding for abstract semi-structures. In *Proceedings of Workshop on Digital Watermarking*, volume 2939/2004, pages 141–153, 2003.

[21] R. Sion, M. Atallah, and S. Prabhakar. Rights protection for relational data. In *Proceedings of the 2003 ACM SIGMOD*, pages 98–109. ACM Press, 2003.

[22] M. D. Swanson, B. Zhu, and A. H. Tewfik. Transparent robust image watermarking. In *Proceedings of the 1996 SPIE Conf. on Visual Communications and Image Proc.*, volume III, pages 211–214, 1996.

[23] C. Yu and L. Popa. Constraint-based xml query rewriting for data integration. In *Proceedings of the 2004 ACM SIGMOD*, pages 371–382. ACM Press, 2004.

[24] X. Zhou, H. Pang, and K.-L. Tan. WmXML: A system for watermarking xml data (demo). In *VLDB'05: the 31st International Conference on Very Large Databases*, pages 1318–1321, 2005.