# Integrity Verification of Outsourced XML Databases

Ruilin Liu, Hui (Wendy) Wang
*Department of Computer Science, Stevens Institute of Technology*
*Hoboken, NJ, 07030*
{rliu3, hwang}@cs.stevens.edu

***Abstract*—Recent years have witnessed an increasing trend of enterprises outsource their IT services to third parties. A major concern in database outsourcing paradigm is integrity verification. Two important issues of integrity verification are *correctness* and *completeness*. In this paper, we consider these two issues for outsourced XML databases. We propose a novel auditing mechanism that provides the guarantees for both of them. Our experimental results demonstrate the effectiveness and efficiency of our approaches.**

## I. Introduction

Advances in networking technologies and the continued growth of the Internet have triggered a new trend towards outsourcing data management and information technology to external service providers. However, the third-party service provider that in the *database-as-service (DAS)* model may not be trusted[10]. One of important security concerns in the database outsourcing paradigm is *integrity*; when a client receives a query result from the third-party service provider, he/she wants to be assured that the result is both correct and complete. Here *correctness* means that the answers are from the original database without being tampered with, while *completeness* means that the result includes all records in the original database that satisfy the query.

The problem of providing security guarantee to outsourced databases has received considerable interests in recent years [11], [16], [18], [20], [23]. However, most of them only consider relational databases. As large amounts of data are stored in native XML database repositories now, there rises the need to consider integrity auditing of outsourced XML databases. In this paper, we focus on the mechanisms that provide both correctness and completeness auditing for XML databases that are hosted by untrusted third-party servers.

We consider the system model that consists of three components, the *data owner* who possesses the original database, the third-party *service provider*, or the server, who hosts and manages the database from the data owner, and the *verifier* who validates the integrity of the server. Figure 1 shows the architecture of the system. We assume that the verifier can be the data owner him/herself. Thus the verifier may have limited resources such as disk space and computational power. One of our goals is to reduce the space overhead and computational complexity that is incurred by integrity auditing.

We adapt the signature-based approach [11], [13], [4] to XML databasess for correctness verification. Now the main challenge of integrity auditing is the *completeness* verification. A possible approach is to use aggregate signatures [11], [5], [18]. By this approach, to prove the completeness of the result for a range query $[a, b]$ over an ordered list $[r_1, \ldots, r_n]$, the publisher releases the two entries that are immediately below and above the query range. Then the data owner hashes and signs all consecutive pairs of tuples in the $(r_{i-1}, r_i, \ldots, r_j, r_{j+1})$, where $r_{i-1} < a \leq r_i, r_j \leq b < r_{j+1}, 1 < i \leq j < n$. By chaining tuples in this way, if the server drops any in-between tuple in the answer, the client cannot re-construct the signatures. The signature-based approach can provide robust completeness assurance, however, it only considers the completeness verification of value-based queries, thus it cannot be directly applied to the structure-based queries in XML databases.

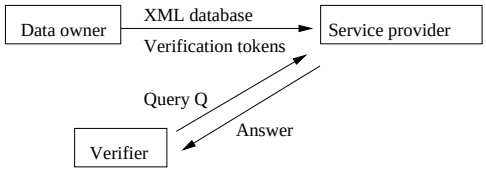



Fig. 1. System architecture

Another possible approach of completeness auditing is to store a *secure structure* of the original XML database [4]. The completeness verification is accomplished by comparing the answer from the secure structure and that from the server. Although this approach can provide auditing guarantees, the secure structure can grow to a fairly large size. Furthermore, it requires that the verifier has sufficient computation capacity of maintaining the secure structure and evaluating queries against the structure, which may not hold in our outsourcing model.

Xie et al. [23] proposed to add fake tuples into relational databases for verification purpose. We adapt their idea to XML databases. As the test queries on XML databases may cover multiple fake elements according to their structure, the challenge is to design appropriate fake elements so that the number of fake elements calculated from the test queries indeed equals to the total number of actual matching ones.

### A. Our contributions

First, we adapt the Merkle signature [14] to XML databases and propose a correctness-auditing signature scheme that prevents the attacker from modifying the databases (Section II).

Second, we design a probabilistic *completeness* auditing mechanism. The basic idea is that the data owner stores some fake elements, called *verification tokens*, on the server together with the hosted database. To verify the integrity, the verifier sends a set of test queries $Q$ to the server. By checking whether the server returns all the verification tokens that satisfy the queries, the verifier can conclude with a probability whether the server obeys the integrity requirement. We quantify the probability that the attacker can successfully escape from being caught if he/she modifies the data. This approach avoids the expensive computation of aggregate signatures as in [11],

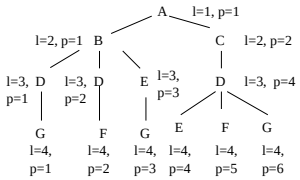



Fig. 2. The $l$ and $p$ values on XML database

[5], [18]. It is also more efficient than the secure structure technique [4], since the size of verification tokens is relatively much smaller than the size of the database (Section III).

Third, we design the *randomized* approach by which the data owner randomly generates the verification tokens. We analyze both the time and space cost at the data owner, verifier, and the server side (Section IV).

Since the randomized approach may introduce significant storage overhead at the verifier side, fourth, we propose the *deterministic* approach, which only stores a deterministic function at the verifier side instead of real verification tokens. This approach can achieve the same degree of integrity guarantee as the randomized approach, but much better time and space overhead at the data owner, verifier, and the server sides (Section V).

Last but not least, we complete our analytical results with a comprehensive set of experiments. Our experiments demonstrate both the effectiveness and efficiency of our approach (Section VI).

We assume the readers are familiar with XML and its query evaluation. We only consider the XPath queries that contain $/, //$ and []. We assume there does not exist any XML schema. We do not consider updates at present.

The rest of this paper is organized as follows. Section VII overviews the previous work related to our problem, and Section VIII concludes the paper.

## II. Correctness Auditing

For the purpose of correctness verification, we construct the *checksum* of elements. Simply constructing the checksum from the element content (tags, attributes, data values, and sub-elements) may result in that the elements on different paths are of the same checksum values, which may enable the server to change the orders of the elements. Thus we incorporate the structural information of elements into checksum. In particular, each element $e$ is labeled with a *level* value $l$ that specifies the level of $e$ in the database tree, and a *position* value $p$ that specifies its topological order within the elements at the same level. Figure 2 shows an example of labeled $l$ and $p$ values. We construct the *structural label* $(l, p)$. The checksum of the element $e$ is constructed from the structural labels:

$$c = \begin{cases} H(l \oplus H(p))||H(e.tag) & \text{if } e \text{ is a leaf} \\ ||H(e.val)||H(e.att), & \\ H(l \oplus H(p))||c(e.child_1) & \text{Otherwise;} \\ ||\ldots||c(e.child_n), & n\text{:\# of children of } e \end{cases} \quad (1)$$

Here "||" is a concatenation function, $H$ is a one-way hash function, and $\oplus$ adds the value $l$ to $H(p)$. The checksum is constructed in a bottom-up fashion; the checksum of the leaf nodes are constructed by concatenating the hash values of the structural labels and those of the content, including the tag, the data value, and the attributes. The checksum of the non-leaf nodes are constructed by concatenating the hash value of the structural labels and the checksum values of the children. We compute $l \oplus H(p)$ instead of $l \oplus p$ to guarantee the uniqueness of the computed result. We require that $H$ is an efficiently computable collision-resistance hash function [2]. It takes a variable-length input and returns a fixed length binary sequence. Furthermore, it should be difficult for the attacker to reverse the hash value, i.e., given $H(x)$, it is computational infeasible to compute $x$. Therefore, the attacker cannot easily modify the checksum.

To distinguish the true elements from the verification token elements, we construct the *signature* of the true and fake elements by applying different checksum construction mechanisms on true elements and verification tokens. In particular, let $c$ be the checksum constructed by Equation 1. We construct the signature $sig$ as following:

$$sig = \begin{cases} H(c) & t \text{ is a true element} \\ H(c)+1 & t \text{ is a verification token} \end{cases} \quad (2)$$

The structural labels and the signatures are attached to the elements and are sent to the server. On receiving the structural labels and signatures, without the possession of the one-way hash function $H$, the server cannot distinguish the true elements from the verification tokens.

To verify the correctness, for each element in the returned answer, we require that the server must return its attached structural label and signature. Given a returned element $e$, let $(l, p)$ and $sig$ be its structural label and signature. Note that by our assumption that the query answer is element-based, i.e., $e$ must contain all of its sub-elements. Then first, the verifier reconstructs the checksum $c'$ by applying Equation 1 on $l$, $p$, and $e$. Second, the verifier computes the signature $sig'$ by applying Equation 2 on $c'$. Third, the verifier compares $sig'$ with the returned $sig$ by the server. If $sig'$ equals either $sig$ (i.e., $e$ is a true element) or $sig - 1$ (i.e., $e$ is a verification token), the verifier can conclude that the returned element originates from the owner's database. Otherwise, the verifier will inform that the database has been tampered with. If the attacker modifies either the structure or the content of the elements, it can be caught by our signature-based techniques. The complexity of correctness verification is $O(\alpha * |S_A|)$, where $\alpha$ is the time complexity of the hash function $H$, and $S_A$ is the set of elements returned by the server.

## III. Completeness Auditing

To verify whether the server returns all necessary answers, we propose a probabilistic auditing approach. Intuitively, the data owner inserts fake elements $\Delta$ as the *verification tokens* into the original XML database $D$, and sends the updated database $D+\Delta$ to the server. Meanwhile, the data owner sends the information of $\Delta$ to the verifier. How to produce $\Delta$ will be further discussed in Section IV and V. When there comes the need to verify the integrity of the server, the verifier sends a set of *test queries* $Q$ to the server. By checking against the stored verification tokens, the verifier knows in advance the tokens $Q(\Delta)$ that should be returned as part of the answer of $Q$. If the server is honest, it should return $Q(D)+Q(\Delta)$. Thus if the answer $A_S$ from the server does not satisfy that $Q(\Delta) \subseteq A_S$,

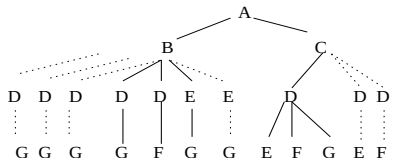



Fig. 3. An example of XML database that contains the fake elements. True elements are connected with solid lines, while fake elements are connected with dotted lines.

| ID | Verification Token | Frequency |
|---|---|---|
| $p_1$ | A/B/D/G | 3 |
| $p_2$ | A/B/E/G | 1 |
| $p_3$ | A/C/D/E | 1 |
| $p_4$ | A/C/D/F | 1 |

Fig. 4. The verification token table on the verifier side

the verifier can conclude with 100% belief that the server does not return the complete set of answers. Otherwise, the verifier concludes that the server is honest with some probability. Next, we explain how to analyze the probability.

Given a query $Q$, assume its answer contains $n$ true elements and $k$ fake elements. If the attacker removes an element $e$, the probability that this element is a true one is $n/(n+k)$. However, if the attacker returns all $k$ fake elements, the verifier cannot explicitly infer that the attacker has removed a true element. Thus the probability that the attacker escapes from detection of removing a true element is $n/(n+k)$. Furthermore, if the attacker removes $m \leq n$ elements, the probability that he/she escapes from being detected equals

$$\prod_{i=0}^{m-1}(n-i)/(n+k-i)[23] \tag{3}$$

It is straightforward that with fixed value $n$ (the number of true elements), larger $k$ results in smaller probability that the attacker can escape. Therefore, adding more fake elements can strengthen the integrity auditing guarantee. However, it will also bring additional overhead to query evaluation. There exists a trade-off between security and efficiency of query evaluation. More discussion of the trade-off can be found in the next two sections.

## IV. Randomized Approach

For the purpose of completeness auditing, the data owner generates a few fake elements at random and use these fake elements as verification tokens. In particular, the data owner randomly picks a subset of leave elements from the database. For each picked leave element, it is replicated for $k$ times, where $k$ is an integer number randomly picked from the range $[1, n]$. The value of $n$ controls the size of the verification tokens. We call these replicas of true elements as the *verification tokens*. These verification tokens are stored on both the server side and the verifier side. Figure 3 shows an example of the database that consists of both verification tokens (connected by dotted lines) and true elements (connected by solid lines).

The verification tokens are stored in a table called the *verification token table*. Figure 4 shows an example of the table. Each entry in the table consists of the verification tokens represented by their root-to-leaf paths, as well as the number that they are replicated. The table takes $O(|F|)$ space, where $F$ is the set of verification tokens of unique paths.

Given a test query $Q$, let $S_A$ be the answer returned by the service provider. Let $\Delta$ be the verification tokens that are stored at the verifier side. The completeness auditing at the verifier side takes four steps. First, evaluate the test query $Q$ against the verification tokens $\Delta$, and get the answer $Q(\Delta)$. Second, for each element $e$ in $S_A$, compute its checksum $c'$ from its returned structural label $sl$. Third, for each element $e$, compute its signature $sig' = H(c')$ by using Equation 2. Let $sig$ be its signature returned by the server. If $sig' = sig - 1$, $e$ is a verification token; otherwise it is a true element. Following this procedure, all verification tokens $S_F$ in $S_A$ are identified. Fourth, check whether $S_F = Q(\Delta)$. If $S_F \neq Q(\Delta)$, the verifier concludes that the service provider does not return the complete set of answers. Otherwise, the verifier concludes that the service provider may remove a tuple with the probability $p$, where $p$ is computed by using Equation 3.

### A. Cost Analysis

Let $D$ be the hosted XML database and $\Delta$ be the verification tokens. We use $\mathcal{Q}$ and $S_A$ to denote the set of test queries and their answer that is returned by the server.

**Data owner**: After the data owner outsources the database, there is no space overhead at the data owner side. For the worst case, the time complexity of generating the verification tokens is $O(|D|)$.

**Verifier**: The space overhead for storing the verification tokens is $O(|\Delta|)$. The time complexity of evaluating a test query $Q$ against a root-to-leaf verification token $t$ is $O(|Q|*|t|)$[9]. Thus the total time complexity of evaluating test queries against the verification token table, i.e., the complexity of Step 1 in the verification procedure, is $O(|Q|*|\Delta|)$. The complexity of Step 2&3 is $O(\alpha * |S_A|)$, where $\alpha$ is the complexity of the hash function $H$. The complexity of Step 4 is $O(|S_A| + |Q(\Delta)|)$. Since both $|S_A|$ and $|Q(\Delta)|$ are dominated by $O(|Q| * |\Delta|)$, and $\alpha$ is negligible, the time complexity of verifying query $Q$ is $O(|Q| * |\Delta|)$. The total time complexity of verifying a set of test queries $\mathcal{Q}$ at the verifier side is $O(|\mathcal{Q}| * |\Delta|)$.

**Server**: The space overhead for storing the hosted database is $O(|D| + |\Delta|)$. The time complexity of evaluating a test query $Q$ on the hosted database is $O(|Q| * (|D| + |\Delta|))$ [9]. Thus the total time complexity of evaluating a set of test queries $\mathcal{Q}$ is $O(|\mathcal{Q}| * (|D| + |\Delta|))$.

## V. Deterministic Approach

The randomized approach provides robust guarantees for completeness auditing. However, it is possible that a large number of verification tokens are needed for stronger security guarantees, which may bring expensive overhead in terms of both time and space at the verifier side. Thus in this section, we propose a deterministic approach that avoids storing verification tokens and evaluating XML queries at the verifier side. Before we explain the details, we first have the following theorem.

*Theorem 5.1:* Given a test query $Q$, let $S_A$ be its answer that is returned by the server and $S_F$ be the verification tokens

in $S_A$. Let $Q(\Delta)$ be the verification tokens at the verifier side that satisfies $Q$. Then if $|S_F| = |Q(\Delta)|$, then $S_F = Q(\Delta)$.

The proof of the theorem is similar as in [23]. By Theorem 5.1, instead of checking whether $S_F$ equals $Q(\Delta)$, we check whether their sizes are the same. Based on this, there is no need to store the verification tokens if $|Q(\Delta)|$ can be efficiently computed. Our goal is to design such mechanism that supports size-based completeness auditing. To achieve this goal, we propose the deterministic approach by which the number of verification tokens are decided by a deterministic function. In particular, each unique tag in the original XML database is assigned a weight $w \in [0, 1]$. We define the function $F$: $\mathcal{E} \rightarrow \mathcal{I}$, where $\mathcal{E}$ is the element set, and $\mathcal{I}$ is the set of integers. For the XML element $e$ of the root-to-leaf path $t_1/t_2/\ldots/t_n$, $F(e) = [N * \prod_{i=1}^{n} w_i]$, where $N$ is a pre-defined integer, and $w_i$ is the weight value of the tag $t_i$ in $e$.

Based on the deterministic function $F$, the deterministic approach works as following:

**Setup**: The data owner decides the weight assignment of tags and the deterministic function $F$. He/she sends the weight assignment scheme and $F$ to the verifier.

**Construction of verification tokens**: The data owner randomly picks some leaf elements from the original database. For each chosen element $e$, it is replicated for $F(e)$ times. The replicated elements, i.e., the verification tokens, are sent to the server with the hosted database.

**Verification**: For a test path query $Q$ that consists of tags $t_1, \ldots, t_n$, the verifier computes $k = [N * \prod_{i=1}^{n} w_i]$, where $w_i (1 \leq i \leq n)$ is the weight of the tag $t_i$ in $Q$. The value $k$ is considered as $|Q(\Delta)|$, the total number of verification tokens that satisfy the query $Q$.

It is straightforward that if the test queries only consist of root-to-leaf paths and the "/" (parent-child) predicate, then any weight assignment scheme works. However, if the test queries either contain the paths that are not root-to-leaf or contain the "//" (ancestor-descendant) predicate, they may match multiple verification tokens of different paths. Careless assignment of weights may result in wrong conclusion of the number of verification tokens that satisfy the test queries. Example 5.1 gives more details.

*Example 5.1:* Given the XML instance in Figure 3 (only considering the data connected with solid lines), assume that the $w_D$ = 0.3, $w_E$ = 0.1, while other tags are assigned the weight 1. Given $N$ = 10, the number of verification tokens $A/B/D/G$ equals $N * w_A * w_B * w_D * w_G$ = $10 * 1 * 1 * 0.3 * 1 = 3$. Similarly, the number of verification tokens $A/B/E/G$ equals 1. Then for the test query $Q_1 : A//G$, the deterministic function $F$ will return the number of the verification tokens that satisfies $Q_1$ as $N * w_A * w_G$ = $10 * 1 * 1$ = 10, which does not match the real answer 4. Thus the weight assignment scheme is not correct. ▌

To find the appropriate weight value scheme so that the number of fake elements calculated from the test queries indeed equals to the total number of actual matching verification tokens, first, we define the *valid* weight assignment.

*Definition 5.1:* **[Correct weight assignment]** We say the weight $(w_1, \ldots, w_n)$ of the tags $(t_1, \ldots, t_n)$ is *valid* if for any path $p : t_i//t_2//\ldots//t_j$, it satisfies that

$$\prod_{i=1}^{k} w_i = \sum_{j=1}^{|P|} \prod_{s=1}^{|P_j|} w_s, \tag{4}$$

where $w_i$ is the weight value of the tag $t_i$, and $P$ is the set of paths that only contain / and $p$ has an embedding to.

The existence of the embedding from the path $p$ to the path $q$ implies that the answers of $p$ contains that of $q$. Thus the number of verification tokens that satisfies the path query $p$ that only contains / must equal to the sum of those of all queries that are contained in $p$. In general, Equation 4 should be satisfied for all possible test queries. However, the number of possible test queries are exponential to the number of unique tags in the database, which may result in the computation of weight assignment of expensive complexity. Thus we assume that the verifier and the data owner pre-define the test queries together. The data owner only has to compute the weight assignment of the tags in the test queries by using Equation 4. Example 5.2 gives more details.

*Example 5.2:* Given the original XML instance in Figure 3 (only considering the data connected with solid lines), assume the data owner picks the elements $A/B/D/G$, $A/B/E/G$, and $A/C/D/E$ to make replicas. The tag $F$ is assigned a weight 0. The test queries are defined as $\{A//E, A//D\}$. We have:

$$\begin{cases} w_A * w_E = w_A * w_B * w_E + w_A * w_C * w_D * w_E \\ \text{(A//E covers A/B/E/G and A/C/D/E)} \\ w_A * w_D = w_A * w_B * w_D + w_A * w_C * w_D \\ \text{(A//D covers A/B/D/G and A/C/D/E)} \end{cases}$$

A valid weight assignment scheme is $w_B = w_C = 0.5$, $w_D$ = $w_E$ = 1. $w_A$ and $w_G$ can be any value in $(0, 1]$. ▌

Note that there always exists at least one valid scheme, in which the test queries are root-to-leaf path queries that only contain /.

### A. Cost Analysis

Let $D$ be the original XML database, $\Delta$ be the fake elements, and $E$ be the set of unique tags in $D$. We use $S_A$ to denote the answer that is returned by the server.

**Data owner**: There is no space overhead at the data owner side. The time complexity of computing weights is $O(|D|)$.

**Verifier**: The verifier stores the set of unique tags in the verification tokens and their weights. Thus the space overhead on the verifier side is $O(|T|)$, where $T$ is the set of unique tags in the verification tokens. For each test query $Q$, the time complexity of computing $|Q(\Delta)|$ is $O(|Q|)$. The time complexity of identifying verification tokens from $S_A$ is $O(\alpha * |S_A|)$, where $\alpha$ is the complexity of the hash function $F$. The time complexity of comparing whether $|Q(\Delta)| = |S_F|$ is constant. Thus the time complexity for a test query $Q$ at the verifier side is $O(|Q| + \alpha * |S_A|)$. The total time complexity for a set of test queries $\mathcal{Q}$ is $O(|\mathcal{Q}|$ + $\alpha * |S_A|)$.

**Server**: The space overhead for storing all fake elements is $O(|D|+|\Delta|)$. The time complexity of evaluating the test query $Q$ on the hosted database is $O(|Q| * (|D| + \Delta|))$ [9]. Thus the total time complexity of evaluating a set of test queries $\mathcal{Q}$ is $O(|\mathcal{Q}| * (|D| + |\Delta|))$.

From the comparison of the space and time overhead of both randomized and deterministic approaches, we observe that the deterministic approach reduces much overhead at the verifier side. due to the requirement of correct weight assignments, the deterministic approach might only support test queries that consists of root-to-leaf paths with / predicate solely. Thus there exists a trade-off between the types of test queries that can be verified and the performance of the auditing procedure.

## VI. Experiments

In this section, we measure both efficiency and effectiveness of our approach.

### A. Setup

We used two real datasets, *SIGMOD record* and *university courses*, downloaded from the XML repository of University of Washington [1]. To evaluate the effect of size of verification tokens to verification, we choose $p = 0\%, 1\%, 5\%, 10\%$ and $15\%$ of leaf elements that are replicated as verification tokens.

### B. Space Overhead

First, we measure the space overhead by correctness verification. The result shows that for *SIGMOD record* dataset ($467KB$), the total size of the structural labels and signatures is $216KB$, the $46\%$ of size growth; while for the *university course* dataset ($2MB$), the structural labels and signatures take the $0.8MB$ space, the $40\%$ of the original database. Although they are considerable large compared with the original database, it is acceptable as they only occur at the server side.

Second, we measure the size of verification token table for the randomized approach. Figure 5 shows the result. We observe that the increase of the database size is much faster than that of the verification token table. This is because all verification tokens in the database that are of the same path correspond to one entry in the verification token table. This is beneficial to the verifier since more verification tokens do not necessarily increase space overhead at his/her side.

### C. Time Performance

We measure the time for query evaluation alone, for correctness verification, and for completeness verification. Figure 6 reports the results on both datasets. First, it is straightforward that more verification tokens always require more time to be evaluated and verified. Second, the time of evaluating both correctness and completeness are negligible compared to the time of query evaluation, i.e., our integrity auditing approach does not incur much overhead to query evaluation. Third, the time to validate completeness by the deterministic approach is always much faster than that by the randomized approach. This proves the efficiency of our deterministic approach.

## VII. Related Work

After Hacigumus et al. [10] raised the security issues in the scenario of database outsourcing, there are several notable works in recent years. These works address different aspects of integrity in the database outsourcing model. For instance, [5], [16] focus on the *correctness* aspect of the integrity, while [11], [18], [23], [20] study the problem of *completeness* assurance. In particular, Li et al. [11] uses Merkle hash tree [14] to audit the completeness of the query answers, Pang et al. [18] proposes assigning signatures in a chain of paired tuples. Both work can provide robust completeness assurance, however, the computation of signatures can be expensive, especially for the range queries that cover large number of tuples. Furthermore, it only considers the completeness verification of value-based queries in the relational databases and cannot be directly applied to the structure-based queries in XML databases. Sion [20] proposes a mechanism called the *challenge token* and uses it as a probabilistic proof that the server has executed the query over the entire database. But their scheme does not provide authentication guarantee: the server could pass the challenges and yet still return false query results. Xie et al. proposed to add fake elements into the database for verification [23]. We follow their path. However, they only consider relational databases. We adapt their idea to XML databases. The challenge is to generate appropriate weight schemes for the deterministic approach.

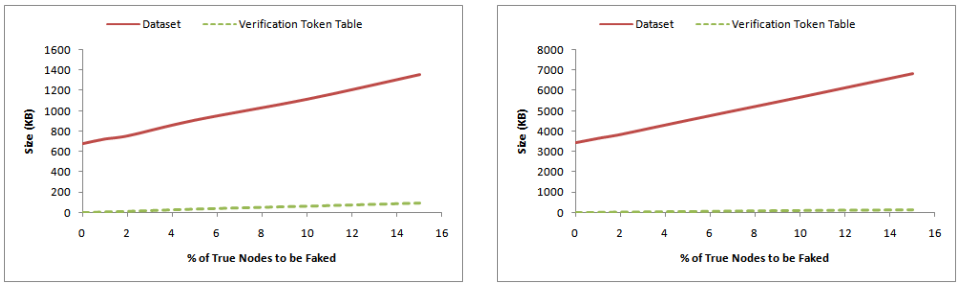
There are a few work on verification of query assurance for XML databases [13], [4], [6]. Devandu et al. [6] uses the Merkle hash technique for authenticating XML data. Bertino et al. [4] proposes a technique based on the Merkle hash technique for selective dissemination of XML data in a third party distribution framework. Kundu [13] proposes a signature scheme for tree structures to ensure efficient integrity in third-party distribution frameworks. They only address the *correctness* aspect of the integrity, but cannot be used to solve the *completeness* issue. Nguyen et al. studied the query assurance issue for outsourced XML databases [17]. They proposed to convert XML databases to relational databases so that the existing signature-based techniques on relational databases as aforementioned can be applied to verify the completeness and correctness. Thus it has the same drawback as expensive overhead of computing the aggregate signatures. Bertino et al. [4] studied the security issue in the scenario of third-party distribution of XML documents. They propose to construct the *secure structure* of the database for later verification. the secure structure can grow to a fairly large size for large XML databases. Furthermore, it requires that the verifier has sufficient computation capacity of maintaining the secure structure and evaluating queries against the structure, which may not hold in our outsourcing model.

There are other security issues in outsourcing database models that are orthogonal to our problem, for example, secure releasing of outsourced XML databases [22], enforcing access control policies on published XML documents [3], [15], and querying encrypted outsourced XML databases (we refer the readers to [21] for a survey of this topic).

## VIII. Conclusion

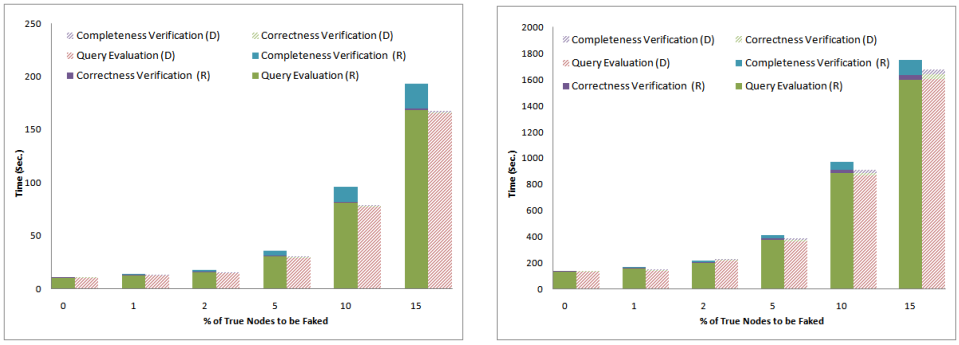
In this paper, we propose a solution for integrity auditing of the outsourced XML databases. It addresses two issues, *correctness* and *completeness* of query evaluation. By our approach, the correctness auditing is achieved by signature scheme, while the completeness auditing is realized by inserting some fake elements, i.e., verification tokens, into the outsourced database and then analyzing the fake elements that

[1] http://www.cs.washington.edu/research/xmldatasets/www/repository.html




SIGMOD Dataset University Record Dataset

Fig. 5. Space Overhead of Verification Token Tables by the Randomized Approach.




SIGMOD Record Dataset University Record Dataset

Fig. 6. Time Overhead. R: randomized approach; D: deterministic approach.

show up in the query result. We formally analyze the probabilistic guarantee of the completeness auditing mechanism and propose two approaches, namely the *randomized* and the *deterministic* approaches, to realize our mechanism.

As the future work, first, we will implement other possible approaches, for example, merkle tree based techniques, and compare the performance of these approaches with ours. Second, our current results are limited to path queries. How to extend the current solution to support general XML queries is an interesting research direction. Third, we plan to extend our solution to support update. We also plan to extend our work to address the other important issues, for instance, the freshness (i.e., the returned answer results are up-to-date), in the database outsourcing model.

## REFERENCES

[1] S. Amer-yahia, S. Cho, L. V. S. Lakshmanan, D. Srivastava. Minimization of Tree Pattern Queries. In Proc. of SIGMOD, June 2001.
[2] S. Bakhtiari, R. Safavi-Naini, and J. Pieprzyk. Cryptographic hash functions: A survey. Technical Report 95-02, Department of Computer Science, University of Wollongong, 1995.
[3] E. Bertino, E. Ferrari. Secure and selective dissemination of XML documents. ACM Transactions on Information and System Security, 5(3), 290-331, 2002.
[4] E. Bertino, B. Carminati, E. Ferrati, B. Thuraisingham, A. Gupta. Selective and Authentic Third-Party Distribution Of XML Documents. TKDE, Vol 16, No. 10, 2004.
[5] P.T. Devanbu, M. Gertz, C. U. Martel, S.G. Stubblebine. Authentic third-party data publication. DBSec, 2000.
[6] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, S. G. Stubblebine. Flexible authentication of XML documents. CCS 2001.
[7] R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In Proc. of the 23rd Intl. Conf. on Very Large Data Bases, August 1997.
[8] G. Gottlob, C. Koch, R. Pichler. XPath Query Evaluation: Improving Time and Space Efficiency. In ICDE Conference, 2003.
[9] G. Gottlob, C. Koch, R. Pichler. Efficient Algorithms for Processing XPath Queries. VLDB 2002.
[10] H. Hacigumus, B. R. Iyer, C. Li, and S. Mehrotra. Executing sql over encrypted data in the database-service-provider model. In SIGMOD Conference, 2002.
[11] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In SIGMOD Conference, 2006.
[12] R. Kaushik, P. Shenoy, P. Bohannon, and E. Gudes. Exploitin Local Similarity for Efficient Indexing of Paths in Graph Structured Data. In Proc. of the 18th Intl. Conf. on Data Engineering, February 2002.
[13] A. Kundu, E. Bertino. Structural Signatures for Tree Data Structures. PVLDB 2008.
[14] R. C. Merkle. A certified digital signature. In Proc. of Advances in Cryptology (CRYPTO), pages 218-238, 1989.
[15] G. Miklau, D. Suciu. Controlling Access to Published Data Using Cryptography. VLDB 2003.
[16] E. Mykletun, M. Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. In NDSS. The Internet Society, 2004.
[17] V. H. Nguyen, T. K. Dang, N. T. Son, J. Kung, Query assurance verification for dynamic outsourced XML databases. In Proc. of the 2nd International Conference on Availability, Reliability and Security, 2007
[18] H. Pang, A. Jain, K. Ramamritham, and K.L. Tan. Verifying completeness of relational query results in data publishing. In SIGMOD Conference, 2005.
[19] T. Milo and D. Suciu. Index structures for Path Expressions. In Proc. of the 7th Intl. Conf. on Database Theory, January 1999.
[20] R. Sion. Query execution assurance for outsourced databases. In VLDB, 2005.
[21] O. Unay, T. I. Gundem, A Survey on Querying Encrypted XML Documents for Databases as a Service, SIGMOD record, Vol. 37, No. 1, 2008.
[22] H. Wang, L. V. S. Lakshmanan. Efficient Secure Query Evaluation over Encrypted XML Databases. VLDB 2006.
[23] M. Xie, H. Wang, J. Yin, X. Meng. Integrity Auditing of Outsourced Data. VLDB 2007.
[24] XSL Transformations (XSLT) 1.0, http://www.w3.org/TR/xslt.
[25] XML Path Language (XPath) 2.0, http://www.w3.org/TR/xpath20/.
[26] XQuery 1.0: An XML Query Language, http://www.w3.org/TR/xquery/.