International Journal of Computational Intelligence Systems, Vol. 5, No. 3 (June, 2012), 421-433

# External and Distributed Databases: Efficient and Secure XML Query Assurance

**Andrew Clarke [1], Eric Pardede [2], Robert Steele [1]**

[1] *Discipline of Health Informatics, University of Sydney, NSW, Australia*

*E-mail: {andrew.clarke, robert.steele}@sydney.edu.au*

[2] *Department of Computer Science & Computer Engineering, La Trobe University, Bundoora, VIC 3083, Australia*

*E-mail: e.pardede@latrobe.edu.au*

Received 2 March 2011
Accepted 14 February 2012



**Abstract**

Emerging information system architectures will often be comprised of distributed systems and data repositories. As a result, providing efficient and secure query assurance over these emerging future information systems is a concern. This paper details the use of temporary time stamps and variable hash granularity to increase the efficiency of query assurance. This approach is implemented against datasets of varying type and size, including encrypted data to illustrate the potential overhead issues present in distributed systems and data repositories.

*Keywords:* Query Assurance, XML, Optimization, Distributed

## 1. Introduction

Emerging information system architectures, such as future health information systems will intrinsically include distributed systems and data repositories (Fig. 1) across multiple organizations[1]. In these information systems, XML seems particularly well suited due to its interoperability potential[2].

However, due to the nature of this approach to data organization, that is, the multi data-owner/multi server model, it is difficult to ensure the accuracy of query results. In particular, to be certain that data is not modified (correctness), the query is performed over the entire dataset (completeness) and represents the most up to date version (freshness). There has been substantial previous work in this area[3,4,5], primarily focusing on the overheads and efficiency of providing query assurance. In our previous work[6], it was shown that data overhead can be significant and efficiency can be further improved. The application of query assurance allows for detection of anomalous behavior from individual servers in the information systems. This would potentially allow for more rigorous trust decisions to be made in the distributed system, perhaps through a fuzzy approach[7].

In cases of sensitive data, there are important benefits from storing data in an encrypted format[8]. As such, it can be identified that an approach that provides query assurance and data confidentiality without substantially diminishing the usability of the information, is a necessary step. In this paper we propose an authenticated efficient query assurance

approach combined with searchable encryption.

Various proposed models for query assurance have been put forth. However, most authenticated approaches have a data-centric focus[4,5] which more closely resembles traditional database applications where data is strongly structured and search by value is of primary importance. Less attention has been given to document-centric XML, where the structure and order of the data may be meaningful as in some XML documents. Additionally, while previous approaches have in some cases combined query assurance and data confidentially, it has not extended to a searchable encryption approach. Further, though previous works have briefly suggested the use of either distributed or expiring time stamps - the method of distribution or the mechanism for setting expiry rates has not been fully explored. Most approaches suggest distribution of timestamps to users/client, but for a highly dynamic database, the associated overheads could be quite onerous if new time stamps need to be propagated.

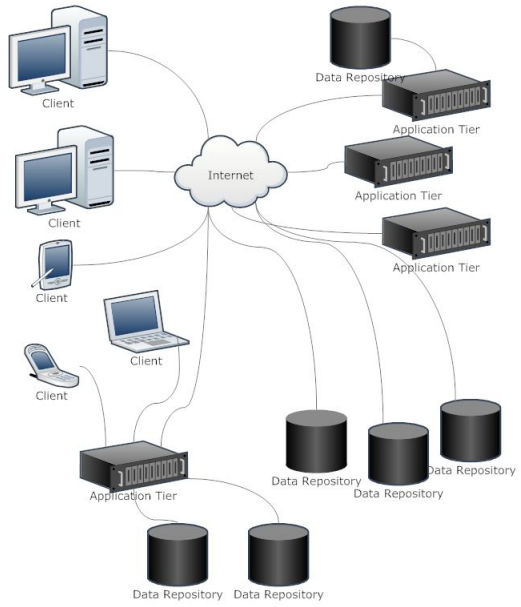


Fig. 1. External and Distributed Database Model

Of concern to all approaches is efficiency. Our approach is targeted to work efficiently with both document-centric and data-centric XML database/files, and extends previous work on freshness to identify methodology of adjustment of time stamp expiry rates in a XML database solution.

## 2. Related Work

Most approaches to query assurance can be categorized into two groups:

(i) Probabilistic - An approach whereby additional fake or duplicate data is stored in the database that is known to the data owners and the clients, but indistinguishable to the database system. The clients then make queries that extend over both the unknown and known data and if the known data is returned correctly it is assumed the unknown data is also correctly returned. This approach commonly makes use of encryption to mask the fake data from the server.

(ii) Authenticated - A more traditional approach that uses some combination of encryption, digital signing, secure hashing and time stamps. Most recent work has used the concepts of merkle hash trees[9,10,11] to make the approach more scalable and efficient.

### *2.1. Authenticated Verification*

The majority of the previous work in authenticated verification has focused on relational databases. However, most recent work has either directly targeted XML or has been compatible and will be detailed in this section.

Partially Materialized Digest Scheme (PMDS)[5] is a variation of a merkle hash tree that works on the premise that hashing operations take less time than file reads. Based on that core concept, a hash tree is created where only the higher branches of the tree are stored. When a query is made, the hash tree values are recreated to verify the result. The root of the tree is digitally signed to ensure the correctness of the model even though the majority of the tree is not stored.

Further, PMDS also attempts to address the issue of hash tree sprawl by implementing nesting to modify the breadth and depth of the hash tree to increase efficiency.

Nested B+ Merkle Tree[4] is a form of embedded merkle hash tree that is specifically tailored to XML data. The root tree is a path tree which preserves the

path information, though path order is not preserved and equivalent paths are collapsed into an element in the tree. Leaf nodes in the path tree contain a value tree and a parent tree. A value tree contains the index to search element by value. The parent tree stores data relating to parent elements.

Embedded Merkle Hash Trees[12] were a proposed extension to improve efficiency of authenticated query assurance through control of the tree fan-out and depth. In this approach, individual B+ trees are nodes of a larger verification tree. The intended result of this type of tree creation is that the server will be able to return smaller verification objects to authenticate the queries.

### 2.2. Probabilistic Verification

There have been a number of previous works into probabilistic verification, however the most recent have typically used encryption and fake or duplicate data[3,13] to provide query assurance.

This approach requires that the data owner and the clients are able to share knowledge of a subset of the database[3]. This known data is typically some sort of fake or duplicate data and encryption is used so that the server cannot differentiate between fake and real data. Further, to provide freshness assurance the known data needs to also be modified over time and those modifications shared between the users of the system. Previous works[3] have implemented this quite well - however the limitations of the system remain that if the server can at any point have the same knowledge as the client, the query assurance would be vulnerable. It is also not possible to disable query assurance at the client level if efficiency overrules verification in some cases. The strength of this approach is that variable levels of data can be added to provide higher or lower levels of assurance, and as a result vary the level of overhead. Further, there are no necessary modifications to make to the server.

Dual Encryption[13] is another approach to probabilistic query assurance whereby the data is encrypted and stored on the server. However, a subset of the total data is stored in duplicate and encrypted with a second key. The use of two different encryption keys is required so that though the data is identical, it is not detectable by the server. Queries can then be authenticated by querying as usual and when a duplicate block is retrieved checking the content is identical. The weakness of Dual Encryption is similar to that of any approach that utilizes encryption: most query types can not be performed as the server can not query directly over encrypted data without invalidating the query assurance scheme.

### 2.3. Summary

In summary, the related work has produced a number of possible models that have associated strengths. Probabilistic approaches can provide a reasonable and variable level of assurance. However, they can be vulnerable to being defeated if the server has the same information as any user that accesses the database. This limits the type of database model it can be applied to, as it requires trusted users. Authenticated approaches on the other hand, have not fully addressed freshness. Further, the efficiency of authenticated models on document-centric XML, or situations where value based search is not used can be problematic.

## 3. Problem Definition

In complex external and distributed systems, problems arise as it is difficult to know whether the service provider is behaving correctly. This section will briefly introduce the requirements for query assurance and the mechanisms to provide that assurance within this database model.

There are three requirements to ensure that query results are fully valid. Correctness and completeness for data centric databases have been quite well covered by previous works[4,9] with current work addressing efficiency improvements. However, freshness approaches are less well covered – especially in relation to dynamic multi-user, multi-owner databases.

- Correctness is the requirement that query results match what is stored in the database. Its goal is to detect corruption caused by server errors or modification caused by malicious operators/service provider/users.

- Completeness is the requirement that query results accurately represent the full range of matching results rather than a subset of the total result. Its goal is to ensure that the query is accurately performed on the entire database and results returned.
- Freshness is the requirement that query results represent the most up to date version of the database. As the database is likely to change overtime, the issue arises that previously valid data might be sent out even after a change has been affected. As the older data was valid at some time, it is necessary to be able to detect whether returned data is fresh.

Ideally overheads from performing query assurance would be minimal enough that the process appears transparent to the end users. However, there are many contributors to overheads, and frequently it is a balance. For example, by making overheads lower for clients, data owner/server overheads may increase.

- Client computation: Any extra user query processing overheads due to query assurance implementation. Depending on the scheme this could include: encryption/decryption, digital signature verification, hash checking, known data checking, time stamp checking and range checking. Measurement used could be processing time or CPU usage.
- Client bandwidth: Additional data overheads created by sending verification data. Measurement used would be bytes. However, overheads created compared to size of data requested is a relevant measure as increases may not be linear.
- Server computation: Any extra server processing created by query assurance. This will vary quite significantly based on the scheme as for example, schemes that require fake/false data create significantly more server overheads while typically hashing/digital signature schemes put most of the computational work onto the client/owner. The system of measurement would be server CPU load.
- Data owner computation: Any extra computation overheads for the data owner. This includes hashing, digital signatures, time stamping, fake data creation and deletion, sorting and encryption/decryption - depending on the scheme in use. The best system of measurement is likely to be the overall time spent on database maintenance.
- Server storage: Additional space required to store any verification information and functions.

The importance of these factors can vary widely based on the use of the database implementation.

## 4. Efficient Query Assurance

As covered in Section 2, methods for query assurance are broadly separated into two streams:

(i) Probabilistic - Fake or duplicate data is added to the database that is identifiable/known to the users/data owner but not to the server.

(ii) Authenticated - Hash/digital signature information is attached to databases to prove query reply authenticity.

This work proposes an approach to authenticated query assurance that differs from the previous solutions in the following ways:

- Previous solutions have focused on sort by value approach to create verification objects capable of providing completeness assurance. This is logical for data-centric XML, but has some shortcomings when applied to document-centric XML. This approach, in contrast, primarily sorts by exact path in the XML tree.
- Earlier works have rarely or briefly addressed freshness. Though expiring time stamps and distributed time stamps are suggested they have not fully been explored.
- This approach takes data overhead to be a more serious concern due to its relevance in mobile computing.
- It is compatible with searchable encryption approach, without creating detrimental overhead.

In the following sections we will discuss the individual techniques that are used to provide query assurance, then their combined implementation to create our query assurance optimization approach.

### 4.1. Hash Granularity

Granularity has long been a concern for query assurance[10]. Ideally, data queried from a database would map exactly to the coinciding verification data. In practice, this can be difficult to achieve without creating large amounts of overhead. For example, digitally signing every element of an XML database separately would be expensive in CPU time for the data owner - but also for the user[10]. Even when merkle hash trees are used[5] to reduce the number of digitally signed elements (by hashing the individual elements then just signing the root node of the tree), there is still the additional concern of the large amount of verification data required to be retrieved. On the other side of this balance is the situation where coarse verification granularity is used so that the user has to retrieve much more XML data to then perform digital signature/hashing checks.

The challenge in query assurance is to match the hash granularity as closely as possible to the queries requested. If the use of the database is well known in advance this can be easily achieved, and a certain level of hash granularity can be imposed. We'll refer to this type of hash granularity as uniform database hashing. In our approach, we use variable hash granularity where there is allowance for the hashing granularity to change overtime based on usage. This occurs at the root level affecting the entire tree as well as at a branch level affecting the granularity in a particular branch. We propose that by recording for each query request, whether it had to request extra XML data (over read) or whether the query matched exactly (exact read), we can make fairly informed decisions on whether to increase or decrease hash granularity at a particular element over time.

### 4.2. Time stamps and digital signatures

Time stamps and digital signatures play a significant role in authenticated query assurance. All verification objects need to be covered by a time stamp that is digitally signed to ensure freshness, correctness and completeness of the result. This approach is quite well researched and understood. However, previous approaches only considered the use of a single time stamp/signature, with no consideration given to the following:

(i) Propagation of time stamp to users in set time stamp systems.

(ii) Calculation of appropriate expiry rate in expiring time stamp systems.

In our approach, we use multiple digitally signed expiring time stamps. Further, we consider that it is beneficial that elements that are modified frequently should be covered by a time stamp with a shorter duration than an element which is rarely modified. The extension of this is that more volatile areas of the database could have a higher assurance of freshness, both through tracking of modification and read frequencies - it also allows for specific areas to be flagged as particularly time sensitive and the time stamps to be adjusted as such.

As a further consideration, as we are now tracking the areas of high access and can place digitally signed time stamps in close proximity, the efficiency of verification can be improved as less data needs to be retrieved and less hash operations conducted. If a less selective approach was taken, the overhead for the data owner to refresh the time stamps at short expiry rates would be significant compared to the efficiency increase for clients.

The relevant variables to time stamp creation on a granule are:

(i) Modification frequency - Granules that are often updated, should have a time stamp that ensures that the freshest update is returned.

(ii) Access frequency - If a granule is accessed frequently, it is beneficial to apply a time stamp as close as possible to high access content as it reduces the bandwidth and computation required for each query.

Lastly, it is an important consideration that schemes of this type also reduce the additional time stamps as a section of the database traffic reduces. Otherwise there would be a gradual data owner overhead increase as large portions could become digitally signed. To prevent nodes from often switching states, the trigger to remove a time stamped node should be marginally lower than that to create it.

### *4.3. Encryption*

For a query assurance methodology to be fully relevant to all types of applications that are likely to be needed within an external and distributed model, compatibility with encrypted records is required[1]. In our approach we allow encryption of the records. The downside is the limitation of the types of queries that can be processed.

For the data stored within these repositories to be usable, a form of searchable encryption needs to be provided. Previous work has suggested a number of different improvements[14,15,16], but the core of most approaches is trapdoor searchable symmetrical encryption.

In our approach we implement a simple form of searchable symmetrical encryption using a trapdoor for the search keyword $T(w,s_w)$, given a keyword $w$ and the secret key $s_w$.

As records are added to the repository the $T(w,s_w)$ is generated for that record and stored in the Index tree. When a query is processed, the client encrypts the keyword, sends it to the server and the server matches it against the previously stored trapdoor value. This allows the server to accurately answer exact match queries without having knowledge of the search keyword or the contents of the record.

### *4.4. Implementation Techniques*

Query assurance implementation can either be client or server driven, or a combination. In previous work, typically probabilistic approaches have been client driven and authenticated approaches have been server driven. There has been limited interest in client driven authenticated query assurance, possibly due to the additional communication overheads. However, in our approach we consider that a client driven authenticated approach is preferable for the following reasons:

- Distribution - Verification data need not be stored on the same server (or with the same service provider) as the data repository. By separating the two services, the likelihood of receiving matching invalid data and verification objects is reduced.
- Optional - Clients could choose not to retrieve the verification data, in situations where processing/ battery/bandwidth are limited or efficiency paramount. It would be difficult for the data repository to discover this without collaborating with the verification service provider.
- Provider Trust Levels - As the verification tree is comparably small to that of the XML data, it may be possible to store the verification data with a higher QoS server.
- Scalability - As the XML and verification components are separate servers, it would be possible to have one verification server for a number of mirror XML servers.

However, the associated downsides are that the client will have to make two server connections and queries rather than one. This has associated overheads. Further, it is necessary that the client do some basic query translation to request the correct verification data from the verification server.

#### *4.4.1. Initialization*

On adding an XML document to the database server, Path and Index trees are created as in Fig. 2. The Path tree contains signed time stamp at the root node and read/modification counts. The leaf nodes of the Path tree contain a hash of the XML data to which it refers. Branches of the Path tree contain hashes of their child elements.

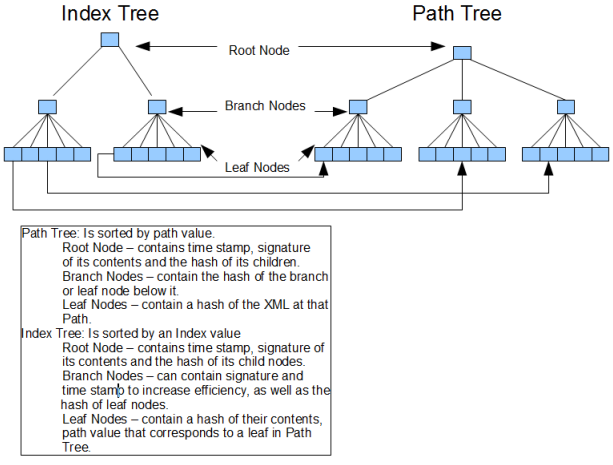


Fig. 2. Verification Tree Structure

It is possible to change the depth and breadth of this verification tree by adjusting the maximum child nodes per branch. The leaves are sorted based on absolute path, so elements are retrieved by path based query.

In our approach, all XML files need a Path tree for query assurance. If completeness proofs are required for value based searches, an Index tree would be added. The Index tree contains a nested tree for each search variable. Leaf nodes of the Index contain the absolute path to the XML element. The branches of the tree act in the same manner as the Path tree, whereby the branches contain the hash value of their children. As the Index tree contains only the path of the XML element, it essentially links to the Path tree. As the hash of the contents is not contained in the Index tree, both index and path verification data is required to validate a query.

#### 4.4.2. *Accessing*

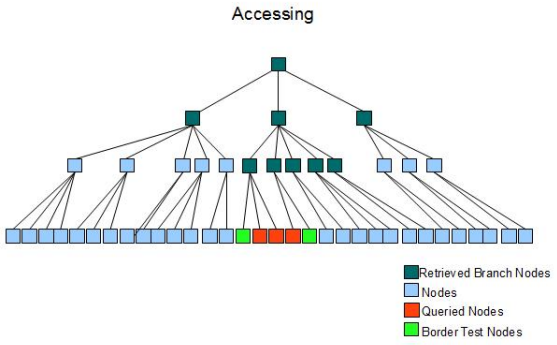


Fig. 3. Verification Tree Accessing

In our approach accessing is client driven. That is, the client makes the XML query, and also an associated verification query. In the case of the verification query, the server returns the verification object for the requested item and boundary cases on either side to provide completeness assurance as shown in Fig. 3. This verification data is then compared against the XML query result. If the time stamp is valid, the hash and digital signature can be verified. As the verification object includes boundary values at either end (border test nodes), the query result is verified as accurate.

#### 4.4.3. *Updates*

On updating of the XML data, two updates need to be created - one for each server. The XML update occurs as usual. In the verification update, the element is changed in the Path tree and the parent recalculated until a signed node is reached. However, the Index tree does not need to be recalculated on each update, only in cases where:

(i) The exact path of an element changes.

(ii) The search value on which an element is indexed changes.

This allows for cheaper updates and appends where the value is not indexed. However, insertion and deletion of elements would be more expensive.

#### 4.4.4. *Refreshing*

In our approach, we consider that for keeping time stamps fresh a master data owner model is relevant. So, in systems where there are multiple data owners, there is a single data owner responsible for refreshing the time stamps. However, any data owner can update data and refresh time stamps when they make a change. The process taken is that a list of time stamped nodes are retrieved, the data owner resigns these nodes with a fresh time stamp, and then the time stamp nodes are updated back into the tree.

#### 4.4.5. *Maintenance*

Maintenance is an additional step that is not required for query assurance. However, it allows for optimization techniques to be adjusted on the verification tree. It could be run as part of a scheduled backup service or more often to optimize the verification trees to changing query habits. An example of a Tree prior and post maintenance is shown in Fig. 4.

In our approach, during maintenance the hash granularity can be increased or decreased based on past usage. The expiry rate of time stamps is adjusted based on modification rates. Further, additional time stamped and digitally signed nodes are created to optimize the verification object size - and

the children per branch can be adjusted to modify the depth and breadth of the tree.

The maintenance process begins with the XML documents being retrieved in their entirety along with their associated Index/Path trees. The first adjustment is that of hash granularity. Based on over reads and exact reads on the verification tree versus the XML document that was recorded in the verification data, the hash granularity is adjusted or kept the same.

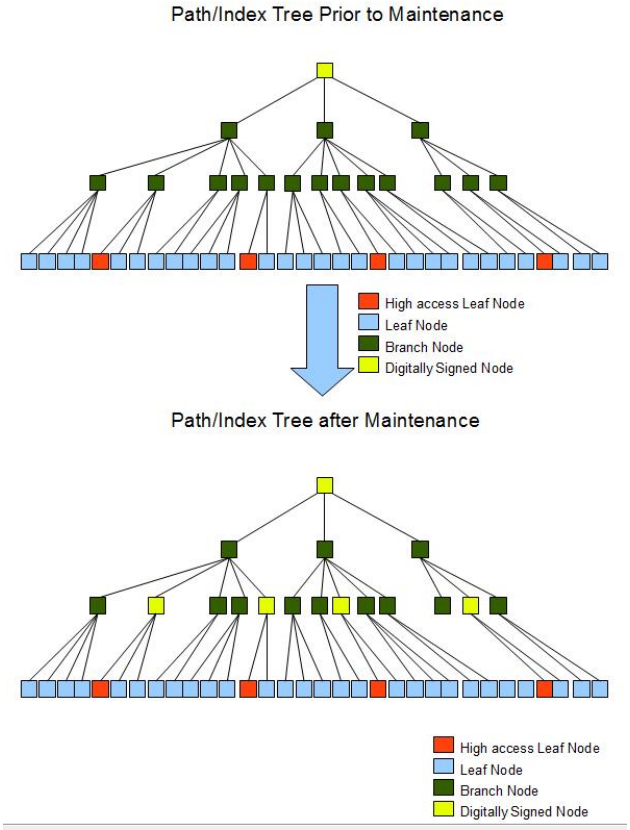


Fig. 4. Pre and Post Maintenance Tree Structure

Based on this approach, if the query habits remain the same after a few maintenance runs, the most efficient level of hash granularity can be achieved. Further, if those habits later change, the hash granularity will adjust.

The second stage of the maintenance process is time stamp management. Modification rates are compared to existing expiry rates on time stamps. The goal is to set expiry rates at 10-20% lower than that of the average time between modifications. Lower is preferred as it is expected that most databases with have peak and trough times during a cycle.

Finally, to increase efficiency, temporary time stamps are added at branches nearby highly accessed nodes. In this approach, the leaves of the verification tree are traversed from first to last and any highly accessed nodes that occur within proximity are signed at their common ancestor. Through this process, the most common queries have a significantly smaller data overhead than that of a random query due to having to retrieve far less verification data. An example of this change in structure is shown in Fig. 4

## 5. Experiment and evaluation

### *5.1. Experiment Setup*

The test setup was conducted on a x3 AMD 710 with 3GB of RAM. The client and both server implementations were run on the one machine. To more accurately simulate verification overheads in comparison to XML DB queries, a full featured native XML database was utilized. In this case a Xindice data collection[17] running on a tomcat 6 server. A client and verification server was developed in Java with 1.6.0 runtime library. The verification tree object is a custom class built on top of the DefaultMutableTreeNode class. The tree objects loosely follow a B+ tree structure - with the characteristics that the tree is self balancing and all records are stored in leaf nodes.

Additionally, to demonstrate the applicability of this approach over an encrypted data set a key-value store was used to store encrypted records - in this case an Apache Cassandra database running as a single node. This was acceptable as encrypting records limit the types of queries that can be performed, though this is minimized with the implementation of a searchable encryption[16,15] approach. As a result, there is no advantage to storing the encrypted records in a native XML database, and a more streamlined database implementation can be utilized.

To perform the encryption of the data a 128-bit

AES specification was used. To create the verification trees and objects RSA encryption with a 1024 key length and SHA1 hashing were implemented.

#### *5.1.1. Data setup*

The data used in the experiment was of three distinct types:

(i) The CIA World Fact Book[18] is used to demonstrate a mid sized data-centric XML file.

(ii) A collection of ten RSS documents to demonstrate smaller document-centric XML files.

(iii) 100000 Continuity of Care (CCR) XML documents. The approximate total size of the CCR files is 3 GB. Prior to storage these records are encrypted using a 128-bit AES, encoded into base64 then stored between CDATA tags in XML format to avoid issues with unparseable XML records.

The use of various sizes and types of XML files is to demonstrate the effectiveness of the optimization approach on common data types. Additionally, the CIA World Fact Book and RSS documents are stored in a Xindice data collection, while the encrypted CCR records are stored in an Apache Cassandra database. CCR XML files were chosen to illustrate a domain where query assurance across encrypted data might be increasingly important[1]. Additionally, XML files are often suggested as a suitable format for distributed systems, such as health information systems due to their potential for interoperability[2].

#### *5.1.2. Query Setup*

For the Xindice datasets, a range of different query types are attempted. Path, exact match and range search are conducted to give a more broad result. The sample set that is used on the XML/verification servers consists of a limited number of queries performed hundreds of times for a total of 20000 queries performed on each XML data set. The sample queries are performed both pre and post maintenance to measure the improvement gained by the query assurance optimization. As there are two databases in use, and each database is queried both pre and post maintenance, and by path and index verification trees, the total number of queries performed is 80000.

For the Cassandra dataset, since the records are stored in an encrypted format, with trapdoor searchable encryption implemented to provide basic query functionality, only exact match queries are performed. In this case, two separate query sets are conducted. Query Set 1 involved performing 500000 queries pre and post maintenance. This was considered a best case scenario where the pre and post maintenance queries are identical. In addition, Query Set 2 involved performing 500000 queries pre and post maintenance. However, the pre and post maintenance queries were randomized to demonstrate the effectiveness of the maintenance optimization techniques even where there is no pattern to client queries.

### *5.2. Initialization*

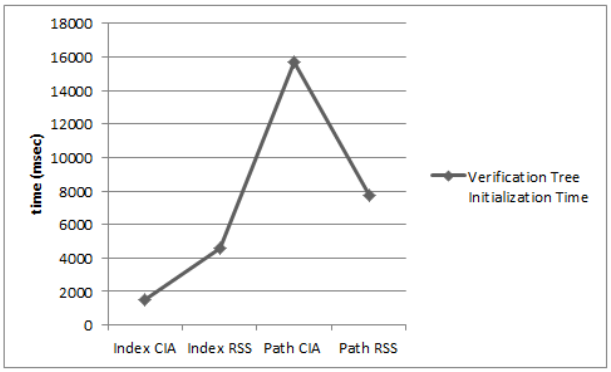


Fig. 5. Xindice Verification Tree Setup Time

During our tests of the verification tree initialization time, we found overall that the setup time was not significant as shown in Fig. 5. The verification tree creation involved populating 10 path verification trees and 10 index verification trees. The time required to setup each index tree was reasonable in comparison to that of the path verification. This is due to the index tree population using some of the resources of the path verification initialization. It follows, that increasing the number of index veri-

fication trees would not be a major overhead growth area. In our previous work[6] we found the initialization time increased depending on the variation in hash granularity and further, there can be some improvement gained by increasing children per branch as this leads to less branch splits during verification tree initialization. However, in this implementation we used fine granularity and a smaller number of children per branch.

### *5.3. Client Query*

After initialization of the Xindice verification trees, the initial benchmark client queries were conducted. Measurements were taken for data and CPU time. It is shown in Fig. 6 that CPU time overhead to verification object creation is not significant compared to that already used by the XML database. In total, initialization, verification and maintenance were less than 10% of the Xindice query runtime with initialization being the most significant of those numbers at 7%. However, as shown in Fig. 7, the data overheads were very significant compared to the XML data returned - with an overall overhead of 89.5% pre-maintenance reduced to 53.4% after maintenance.

The trends whereby overheads increase as number of elements in the database increase are quite clearly defined in Fig. 9. In a unexpected result, CIA World Fact Book path verification data returned was larger than the amount of XML data returned, which emphasizes the potential inefficiencies of authenticated query assurance.

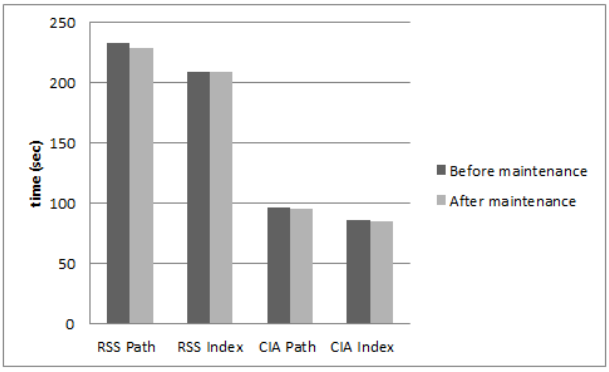


Fig. 6. Xindice Experiment Computation Breakdown

The results for the Cassandra data sets also showed a reduction in CPU time at 23.61% prior to optimization and 15.4% (Fig. 8) after optimization as a percentage of total query time. The average query time over the 500000 queries performed during the pre and post optimization phase was 2.01ms and 1.93ms respectively.

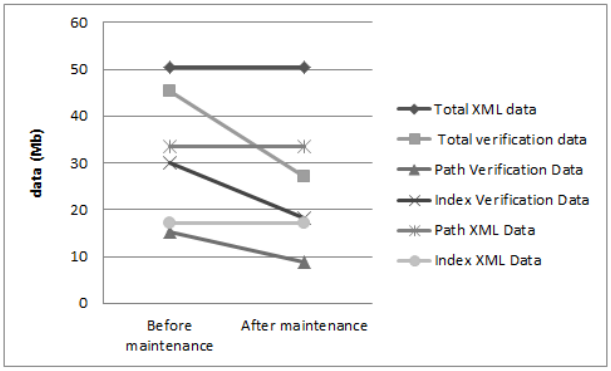


Fig. 7. Xindice Verification data overheads

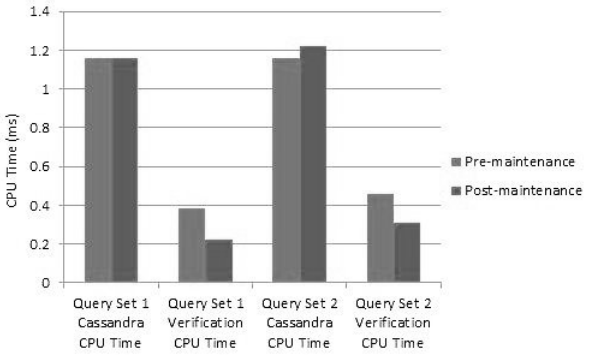


Fig. 8. Cassandra Experiment Computation Breakdown

### *5.4. Maintenance*

The maintenance stage, which adds additional digital signatures and time stamps at lower branches of the verification tree in proximity to high access leaves, garnered the following results: The actual execution time of the maintenance stage, though quite short compared to the entire test run at 0.5% of the runtime, would be too costly to perform in real time. Further, the maintenance time increases as the size of the database increases. This is expected and acceptable because as shown in Section 5.5, the larger databases have the most significant overhead

concerns and could benefit more from efficiency improvements.

### *5.5. Post Maintenance client Query*

In this stage, a query set similar to that processed in the initial benchmark was re-processed. We found from our results Fig. 9 that there was an overall verification data reduction of 59% for the Xindice datasets. As expected the XML data retrieved remained the same. In terms of individual breakdowns, the RSS collection had its overhead reduced by 30.5%, the CIA World Fact Book by 46.5% and the encrypted CCR files by 70.8% and 28.8% for Query Set 1 and 2 respectively (Fig. 10). The best to worst overhead goes from an initial 204% increase to 111%. Further, the computational time was also marginally improved by 1% in the Xindice datasets and 8.21% for the Cassandra datasets but as the verification data query time was significantly less than the XML data query, the overall runtime was not effectively changed.

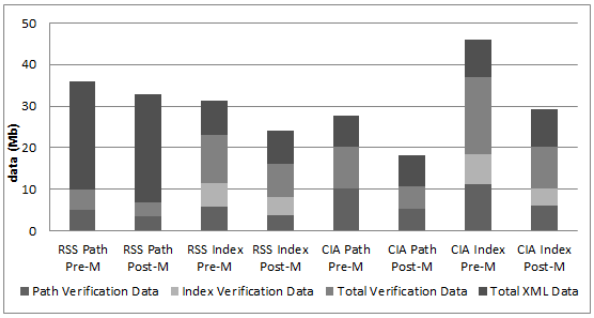


Fig. 9. Xindice Maintenance Performance Comparison

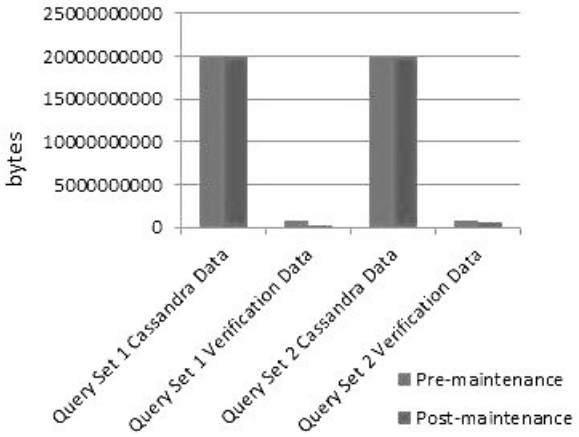


Fig. 10. Cassandra Maintenance Performance Comparison

### *5.6. Discussion*

Through our implementation we have shown that it is possible to provide significant efficiency improvements to query assurance over XML through the use of variable time stamps and hash granularity. This is a dynamic optimization approach and could be used in tandem with a more static optimization through other related work. The balance is that of long term global improvement of the entire database from static approaches, compared to short term small portion efficiency increases.

A further issue of this model suggested in this work is the use of exact paths to map from Index tree to Path trees. The use of exact paths means that the trees in some circumstances display array like qualities. That is, inserting or deleting elements anywhere other than on the end of a sequence can be quite expensive due to the need to also alter any following records in that same sequence. However, the upside of using exact path to map between the trees is that there are less required updates. As Index trees only need to be updated when the exact path or the index value changes. This allows for more relaxed time stamps on the Index trees as updates are less frequent - which further improves efficiency by allowing for more temporary time stamps for the same given amount of refreshing overhead.

The approach used in this paper assumes that to some extent client usage habits will be pattern based. This allows for optimization based on usage pattern with weightings on the most active records. However, in the case of a random usage pattern, the effectiveness of this approach would be significantly reduced as shown in the Cassandra implementation example. So while still providing query assurance it would do so in a more inefficient manner.

Also important and relating to our work, is that computational time of verification and maintenance was found to be not as crucial due to its relative smaller values compared to that of the XML query operations. This is an important distinction as it prompted us to look into data overhead which has more noticeable effect on the client and was an important and somewhat overlooked metric of query

assurance.

Further, the Cassandra results indicate that the overhead cost of applying an authenticated query assurance model in combination with searchable encryption is not detrimentally high. The use of CCR XML files is well suited to demonstration of sensitive information that may be stored. Additionally, though CCR files are relatively large for individual records, they occupy the middle ground of health record types where medical imaging and results would be significantly larger.

Finally, there is the limitation on the decision making data. In our study we used the read and modification counts on the verification server to make decisions on varying hash granularity and time stamp placement and duration. Though it is simple to ensure the modification count integrity from changes by a non privileged user through digitally signing of such data, there is no practical way to provide the same certainty for read counts as they need to be updated by clients' actions. Clients in this model do not have access to re-sign the verification tree. Our concession to this weakness was to give higher weighting to modification counts over read counts. However, there is the potential for exploitation of this to make the database behave in an inefficient manner. The rational for this is limited as it would also increase the load on the verification server. In terms of ways to reduce or detect this form of exploitation, auditing and then crosschecking data between verification and XML servers to detect inconsistencies is a possibility, but would be insufficient if the servers collaborated. A further approach could be to collect opt-in data from a percentage of users and extrapolate that as a predicted usage model and detect aberrant behavior.

## 6. Conclusions and Future Work

This work explored the area of secure and efficient query assurance in external and distributed XML databases. Procedures to provide that assurance while reducing data overheads compared to similar approaches were investigated. In particular, an exact path approach to storing verification data, and an indexing scheme for verification with dynamic optimization through temporary time stamps and variable levels of hash granularity were considered. Further, time stamp creation and the issues involved with read/modification counts were explored.

To gauge the effectiveness of this approach a test implementation was constructed. The query assurance tests were run along side a native XML database (Xindice) to more accurately assess overheads. This implementation test was composed of 80000 queries performed over a range of different sample files, ranging from a collection of small RSS feeds to large XML files. Additionally, to show the applicability of this approach to encrypted records 100000 CCR records were stored in a Cassandra data store and queried 500000 times pre and post maintenance through a keyword trapdoor encryption modification to the verification tree. This allows for the implementation to perform basic queries on the encrypted data, while preserving the confidentiality of the keyword and stored data.

It was of particular interest that our results showed that the CPU time overheads of query assurance were not significant compared to that of XML queries. The results indicated that the most significant area of verification overheads would then be data overhead. Further, our results showed that the data overhead could be significantly reduced by the use of temporary time stamps to optimize the verification tree to the queries it is performing. Reductions of as much as 70.8% occurred in large files.

In the future, an interesting study would be using the methods in this work with that of other approaches (embedding/nesting) in an adaptive and self maintaining/balancing way that applies the most efficient range of optimizations based on the databases' current type and usage, while historic usage patterns affect the modifications in an attempt to predict and optimize for upcoming usage.

An extension of this is investigating the effect that this verification approach has on more complex FLWOR (FOR LET WHILE ORDER RETURN) XQuery expressions. Previous work and this paper have focused on path and range searches, rather than on the effect of join type queries.

Finally, there remains the challenging future task of implementating query assurance as part of a com-

plete approach to database security. This involves the consideration of the other areas of external and distributed database security: data confidentiality, privacy, secure auditing and secure and efficient storage.

## References


1. A. Clarke and R. Steele, "Secure and reliable distributed health records: Achieving query assurance across repositories of encrypted health data," in *HICSS*, IEEE Computer Society, 2012.
2. R. Steele, W. Gardner, D. Chandra, and T. S. Dillon, "Framework and prototype for a secure xml-based electronic health records system," *IJEH*, vol. 3, no. 2, pp. 151–174, 2007.
3. M. Xiey, H. Wang, J. Yin, and X. Meng, "Providing freshness guarantees for outsourced databases," in *Proceedings of the 11th international conference on Extending database technology: Advances in database technology*, vol. 261, pp. 323–332, 2008.
4. V. H. Nguyen and T. K. Dang, "A novel solution to query assurance verification for dynamic outsourced xml databases," *Journal of Software*, vol. 3, no. 4, pp. 9–16, 2008.
5. K. Mouratidis, D. Sacharidis, and H. Pang, "Partially materialized digest scheme: an efficient verification method for outsourced databases," *Very Large Data Base*, vol. 18, no. 1, pp. 363–381, 2009.
6. A. Clarke and E. Pardede, "Outsourced xml database: Query assurance optimization," in *AINA*, pp. 1181–1188, IEEE Computer Society, 2010.
7. S. Schmidt, R. Steele, T. S. Dillon, and E. Chang, "Building a fuzzy trust network in unsupervised multi-agent environments," in *OTM Workshops*, vol. 3762 of *Lecture Notes in Computer Science*, pp. 816–825, Springer, 2005.
8. M. Czapski and R. Steele, "Strengthening privacy and confidentiality protection for electronic health records," in *Web Technologies, Applications, and Services* (M. H. Hamza, ed.), pp. 35–40, IASTED/ACTA Press, 2005.
9. G. T. Einar Mykletun, Maithili Narasimha, "Authentication and integrity in outsourced databases," *ACM Transactions on Storage (TOS)*, vol. 2, no. 2, pp. 107–138, 2006.
10. P. T. Devanbu, M. Gertz, C. U. Martel, and S. G. Stubblebine, "Authentic third-party data publication," in *Database Security*, pp. 101–112, 2000.
11. M. Narasimha and G. Tsudik, "Authentication of outsourced databases using signature aggregation and chaining," in *Lecture Notes in Computer Science 3882*, pp. 420–436, 2006.
12. F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin, "Dynamic authenticated index structures for outsourced databases," in *Special Interest Group on Management Of Data Conference* (S. Chaudhuri, V. Hristidis, and N. Polyzotis, eds.), pp. 121–132, ACM, 2006.
13. H. Wang, J. Yin, C.-S. Perng, and P. S. Yu, "Dual encryption for query integrity assurance," in *Conference on Information and Knowledge Management* (J. G. Shanahan, S. Amer-Yahia, I. Manolescu, Y. Zhang, D. A. Evans, A. Kolcz, K.-S. Choi, and A. Chowdhury, eds.), pp. 863–872, ACM, 2008.
14. P. van Liesdonk, S. Sedghi, J. Doumen, P. H. Hartel, and W. Jonker, "Computationally efficient searchable symmetric encryption," in *Secure Data Management* (W. Jonker and M. Petkovic, eds.), vol. 6358 of *Lecture Notes in Computer Science*, pp. 87–100, Springer, 2010.
15. I. R. Jeong, J. O. Kwon, D. Hong, and D. H. Lee, "Searchable encryption with *keyword-recoverability*," *IEICE Transactions*, vol. 92-D, no. 5, pp. 1200–1203, 2009.
16. H. S. Rhee, J. H. Park, W. Susilo, and D. H. Lee, "Trapdoor security in a searchable public-key encryption scheme with a designated tester," *Journal of Systems and Software*, vol. 83, no. 5, pp. 763–771, 2010.
17. "Apache xindice." http://xml.apache.org/xindice/, 2009.
18. "Sample dataset – cia factbook: Country data." http://www.dbis.informatik.uni-goettingen.de/lopix/lopix-mondial.html, 2009.