Query Racing: Fast Completeness Certification of Query Results*

Bernardo Palazzi[1,2], Maurizio Pizzonia[1], and Stefano Pucacco[1]

[1] Roma TRE University, Rome Italy
{palazzi,pizzonia,pucacco}@dia.uniroma3.it
[2] Brown University, Department of Computer Science, Providence, RI USA

Abstract. We present a general and effective method to certify completeness of query results on relational tables stored in an untrusted DBMS. Our main contribution is the concept of "Query Race": we split up a general query into several single attribute queries, and exploit concurrency and speed to bind the complexity to the fastest of them. Our method supports selection queries with general composition of conjunctive and disjunctive order-based conditions on different attributes at the same time. To achieve our results, we require neither previous knowledge of queries nor specific support by the DBMS.

We validate our approach with experimental results performed on a prototypical implementation.

1 Introduction

Advances in networking technologies and continued spread of the Internet, jointly with cost-effective offers, have triggered a trend towards outsourcing data management to external service providers, often on the Cloud. Database outsourcing is a known evidence of this trend. The outsourced database users rely on provider infrastructure, which include hardware, software and manpower, for the storage, maintenance, and retrieval of their data. That is, a company stores all its data, and possibly business-critical information, at an external service provider, that is generally not fully trusted. Actually, this approach involves several security issues that range from confidentiality preservation to integrity verification. Special attention has been posed to the problem of checking completeness of results. In fact, tuple-level integrity is easy to ensure by adopting some sort of tuple-level signature, but assessing that no malicious tuples deletion or insertion has been performed is a much harder task, if we intend to maintain DBMS-level efficiency.

Many proposal can be found in literature, some are based on *authenticated data structures* [1,2,3,4,5], some on the insertion of spurious data [6,7], some on signatures aggregation [8]. Each of them has strengths and weaknesses with respect to efficiency, privacy, kind of queries supported, etc.

* This work is partially supported by the Italian Ministry of Research, Grant number RBIP06BZW8, FIRB project "Advanced tracking system in intermodal freight transportation" and under Project "ALGODEEP: Sfide algoritmiche per elaborazioni data-intensive su piattaforme di calcolo emergenti", MIUR PRIN.

S. Foresti and S. Jajodia (Eds.): Data and Applications Security XXIV, LNCS 6166, pp. 177–192, 2010.

This paper proposes a novel technique that achieves high efficiency level in practice. We decompose the query in many simpler queries, to be concurrently run, and bind the overall efficiency to the fastest of them. We experimentally verified on a common DBMS implementation, that, in the vast majority of cases, the fastest query is also the most selective.

Many of the techniques known in literature restrict the kind of conditions supported or need to know the queries in advance to optimize data structures. Our proposal supports all conjunctive and disjunctive combination of order-based conditions on any subset of attributes.

By using *authenticated skip lists* represented into regular tables [2], our technique is easy to implement on any DBMS without need for specific support on the server. Also, if our technique should be applied to a pre-existing database, no change to its schema is needed and tables storing authenticated skip lists can also be stored on a different and independent server.

The paper is organized as follows. Section 2 briefly review the state of the art. Section 3 introduces basic background about authenticated skip lists. In Section 4 we describe query racing technique and possible optimizations. In Section 5 we comment about efficiency of our method. Section 7 presents experiments that show feasibility and scalability of our approach. In Section 6 we discuss strengths and weaknesses of our approach with respect to the state of the art.

2 State of the Art

The problem of providing provably authentic results using untrusted DBMS has been largely studied.

Some techniques known in literature provide solutions that rely on hashes and signatures or on inserting spurious data into the database, however, most of the works, rely on *authenticated data structures* [9] (*ADS*).

An ADS represents a collection of elements from an ordered domain. Supported operations are insertion, deletion, and query (equality and range). Usually, all operations require $O(\log n)$ time where n is the number of elements in the ADS. A cumulative hash (*root* or *basis*) of the whole data structure is known to the user. Query operations return a proof of correctness, of size $O(\log n)$, that basically allows the verifier to construct a hash chain from the result to the basis. Insertion and deletion update the basis which is a fingerprint of the collection stored into the ADS. The most common ADSes are Merkle hash trees [10] and authenticated skip lists [11]. Improvements to basic techniques are in [12,13].

The first use of an ADS to authenticate relational database operations is presented in [1] and later improved in [14]. The latter introduces the use of authenticated multidimensional *range trees* in order to support conjunctive queries involving multiple attributes. Usually, adoption of ADS introduces privacy problems because, to check correctness, all attributes involved in the selection condition have to be unveiled, even if some of them are supposed to be filtered out through projection. An interesting approach to solve this problem is presented by Pang et al. [5]. They propose a method to authenticate projection queries

using ADSes. An improvement of this work is presented in [4] where they also exploit condensed RSA signatures aggregation scheme introduced in [8]. A recent work that allows to preserve privacy is presented in [15].

Yang et al. [16] provide techniques for authenticated join operation. In [17] a scalable technique for verification of queries, in particular for the equi-join operation, is presented. That approach is based on Bloom filters [18].

Xie et al. [6] show a method for integrity auditing of outsourced data that uses a probabilistic approach. The proposed technique scatters some control values that are indistinguishable from real values inside the relational table.

In [19] a method to authenticate k nearest neighbors queries is introduced.

Li et al. [20] propose to transform B-trees, used by DBMS for indexes, into an ADS, hence requiring support from the DBMS itself. The problem of efficiently storing ADSes into a regular DBMS has been studied by Miklau and Suciu [3] for Merkle trees and by Di Battista and Palazzi [2] for authenticated skip lists.

Users may need to be sure that queries are performed on the latest version of the database. Some results on this topic are provided in [21,7].

Another work [22] focuses on authenticity and completeness guarantees of query replies, analyzes an approach for various query types, and compares it with straightforward solutions that adopt ADSes.

A technique that reduces the size of the additional information sent to users or to the client for verification purpose is presented in [23].

The authentication of outsourced data through web services and XML signatures is investigated in [24].

3 Background

In this section we provide some details on authenticated skip lists that will be used in the rest of the paper.

A *skip list* [25] is a probabilistic data structure that maintains a subset of elements of an ordered set, allowing searches and updates in $O(\log n)$ time with high probability (w.h.p.), where n is the current number of elements. A skip list for n elements has $O(\log n)$ levels w.h.p., the base level is a sorted list of all elements; a subset of these elements also appears on the second level; for each node in a given level of the skip list, a coin flip determines whether or not it will exist in the next higher level.

We call the set of nodes associated with an element a *tower*. The *height* of the tower is the level of the highest node in that tower. Each node in the structure contains pointers to the node to its right (R) and to the node below it (B). In the following, the notation $V.element$ denotes the element of node V. A search in the structure for a target element e is performed in the following way. We begin at the top left node. If $R.element > e$, then we move to B. Otherwise, we move to R. We continue this process until we are pointing to a node whose element is e (we have found the target), or we are pointing to a node on the base level whose element is greater than e (e is not contained). The nodes involved in the search identify a *search path*.

An *authenticated skip list* [11] supports authenticated versions of the skip list operations. Namely, the nodes on the base level correspond to data elements whose integrity and completeness we would like to certify in query results. Each node in the structure contains a hash value which is the *commutative cryptographic hash* (a cryptographic hash of a pair of data, whose value is independent of the ordering of the pair) of the hash values of a pair of adjacent nodes. In this way the authenticated skip list is similar to the Merkle hash tree structure. For a node V, we denote $V.hash$ the hash value stored in V, $V.level$ the level of V, and $V.height$ the height of the tower of V. The notation $h(A, B)$ indicates a commutative cryptographic hash of the values A and B.

The hash value of a particular node V in the structure is given as follows. We have two rules

Rule 1 ($V.level = 0$)**:** If $R.height = 0$ (it has only a base level node) then $V.hash = h(V.element, R.hash)$, else $V.hash = h(V.element, R.element)$.

Rule 2 ($V.level > 0$)**:** If $R.height = V.level$ then $V.hash = h(B.hash, R.hash)$, else $V.hash = B.hash$.

Application of the above rules leads to a computation of hashes that flow from bottom-right to top-left, like in the example for the element 9 (see Fig. 1). Top left node of the skip list is particularly important since its hash value, called *basis*, is an accumulation of all hashes in the whole structure. If any element in authenticated skip list changes, basis changes. Each authenticated skip list includes additional elements, minimum and maximum, and their corresponding towers ensuring that a basis exists even if our data set is empty.

Queries, for an authenticated skip list, return zero or more elements and a *proof of integrity* (also called *verification object*). Consider an element e stored in its level zero node V and its search path p. The proof of integrity of e is constituted by the hashes stored in the nodes that are on the right and below of each node in p (see Fig. 1).

If the result is empty, queried element b is not in the collection, the proof is composed by proof of two elements nearest to b in the collection according to the

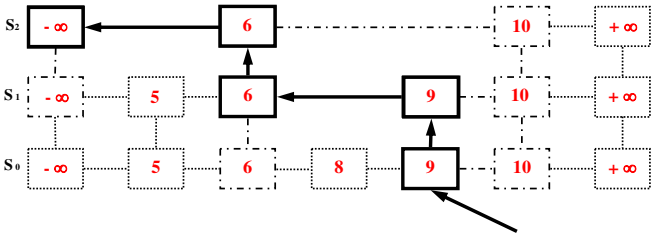


Fig. 1. In a skip list, for each element, there is a search path. For element 9 the search path is made of nodes with thick border. In an authenticated skip list, given the search path p, the proof of integrity is made of the hashes of the nodes that are on the right of p and below p. For element 9 the proof is made of hashes stored in nodes with dash-dotted border.

order. For range queries, the simplest method to provide the proof of integrity of all elements is inefficient. We provide an efficient solution in Section 4.3.

4 Completeness Certification by Query Racing

In this section we describe how to perform certified selection queries by using the query racing technique. We first describe the technique for a restricted class of selections, then we extend our techniques to general selections.

4.1 Basic Queries

Suppose to have a relational table T with attributes $a_1, \ldots, a_m$. We always assume that attribute types have orders and comparison operators which is the case in the vast majority of practical situations.

Basic Query. We call *basic query* a selection query in the following form $\sigma_{f(a_1,\ldots,a_m)}(T)$ where $f(a_1, \ldots, a_m) = \bigwedge_{i=1}^{n}(\alpha_i \star c_i)$ and $\alpha_i \in \{a_1, \ldots, a_m\}$ is an attribute of T, c_i is a constant of the the same type of α_i, and $\star$ is one of the following operators: $=, >, \geq, <, \leq$. Expressions $\alpha_i \star c_i$ are called *atoms*.

In other words, a basic query is a selection on a single table whose condition is a conjunction of equality based and order based atoms on single attributes.

In a basic query Q, if cardinality of referred attributes set $\mathrm{attr}(Q) = \bigcup_{i=0}^{n}\{\alpha_i\}$ is equal to one, we say that Q is *monodimensional* and if $\mathrm{attr}(Q) = \{a\}$ we say that Q is on a. Otherwise we say that Q is *multidimensional*.

Consider a monodimensional basic query Q on attribute a, formally $\sigma_f(T)$ where $f = \bigwedge_{i=0}^{n} a \star c_i$. There exist an expression f', logically equivalent to f, in one of the following *canonical* forms where operators $\lesssim$ ($\gtrsim$) must be intended to be either $<$ or $\leq$ ($>$ or $\geq$).

$$\texttt{true},\ \texttt{false},\ a = c,\ a \lesssim c,\ a \gtrsim c,\ (a \gtrsim c_1) \wedge (a \lesssim c_2) \tag{1}$$

where c, c_1, and c_2 are constants of the same type of a. By extension, basic query Q' which is equivalent to Q, where f is replaced by f', is said to be *canonical*.

A multidimensional basic query $\sigma_{f(a_1,\ldots,a_m)}(T)$ can be decomposed in several canonical monodimensional basic queries, by a simple syntactic processing on f, such that $\sigma_{f(a_1,\ldots,a_m)}(T) = \sigma_{f(a_1)}(T) \bigcap \cdots \bigcap \sigma_{f(a_m)}(T)$.

4.2 Completeness Certification for Multidimensional Basic Queries

Consider a table T on which basic queries should be performed. To support completeness certification of basic query results we introduce two new concepts.

Per Column Security Table. We introduce, for each attribute a of T, a *security table* S_a. Each security table represents an authenticated skip list (see Section 3) that allows the client to easily certify completeness and integrity of the result of monodimensional basic queries in canonical form on a using the techniques described in [2]. Given a monodimensional canonical basic

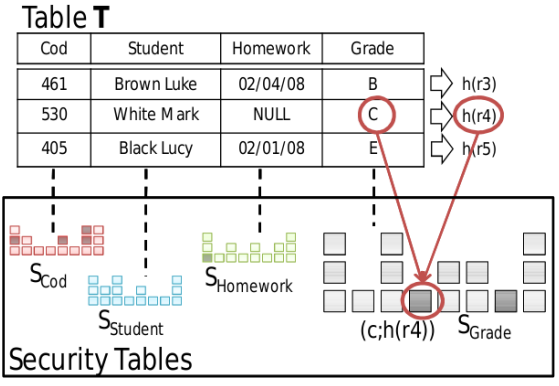


Fig. 2. Each attribute of a table is associated with a security table. Each security table represents an authenticated skip list containing all the values of that attribute with the hash of the corresponding rows.

query Q on a, we call *security query* the corresponding query $\overline{Q}$ on security table S_a whose result provides a *proof of completeness*, or *verification object*, for the result of Q. To exploit such verification object the user also needs to know a basis for table S_a. Section 4.3 shows, for any basic query, how it is possible to obtain the verification object with a single security query. For every monodimensional basic query on a, the corresponding security query on S_a can be easily built [2].

Row-Hashing. For each tuple r of T, a hash value $h(r)$ of r is computed, by hashing the concatenation of all the values of the attributes of the row. Suppose that a tuple r has value v for attribute a. The value stored into security table S_a is the pair $\langle v, h(r)\rangle$. Concerning the order of the value within the skip list represented by table S_a, pairs $\langle v, h(r)\rangle$ are ordered according to the order defined for the types of a and $h(r)$ considering v as the most significant part. This also avoids any problem about duplicate value on that attribute provided that there are no duplicated tuples in T.

We are now ready to introduce the query race technique.

Query Race. A multidimensional basic query Q is decomposed in several canonical monodimensional queries (see Section 4.1) that are concurrently executed along with their corresponding security query. The result of the fastest query is taken as a reference for computing the result of Q, other queries are aborted. The result of the fastest query is certified and processed on the client, in trusted environment, to obtain the final result.

Consider a relational table T with attributes $a_1, \ldots, a_m$ and the basic query Q on T. Without loss of generality, we suppose that all attributes $a_1, \ldots, a_m$ are involved in the condition of Q.

For each attribute a_i, consider the atoms $g_1, \ldots, g_{k_i}$ of the condition of Q that refer to attribute a_i, and let Q_i be a canonical basic query on T whose condition is $g_1 \wedge \cdots \wedge g_{k_i}$.

We now provide algorithmic details to certify completeness of result R of multidimensional basic query Q on relational table T. The following algorithm is intended to be run on the (trusted) client, for a two-party model, or on the users for a three-party model. We assume the client knows, for each security table the corresponding basis.

Algorithm 1

1. Decompose basic query Q into several canonical monodimensional basic queries $Q_1, \ldots, Q_m$ on attributes $a_1, \ldots, a_m$ respectively such that for their results $R_1, \ldots, R_m$ it holds $R = R_1 \cap \cdots \cap R_m$ (see Section 4.1).
2. If among $Q_1, \ldots, Q_m$ there is one query whose condition is `false`, then result of Q is empty and complete.
3. Queries $Q_1, \ldots, Q_m$ and their corresponding security queries $\overline{Q}_1, \ldots, \overline{Q}_m$ are all concurrently executed.
4. Consider the *query pairs* $(Q_i, \overline{Q}_i)$. A query pair is considered finished when both queries are finished. Let a_j be the attribute whose query pair finishes first. As soon as query pair for a_j finishes, all other running queries are aborted. Let R_j be the result of Q_j and $\overline{R}_j$ be the result of $\overline{Q}_j$. $\overline{R}_j$ provides the verification object of R_j.
5. Certify correctness and completeness of $\overline{R}_j$ using the basis for S_{a_j} as described in Section 3 or with the optimized procedure described in Section 4.3. If check fails $\overline{R}_j$ is not genuine and it is impossible to certify R_j.
6. We consider R_j ordered according to a_j and $\overline{R}_j$ as defined before for the authenticated skip list represented in S_{a_j}. For each element $\langle v, h \rangle$ of $\overline{R}_j$ perform the following three steps.
   (a) Consider the tuple r of R_j corresponding to element $\langle v, h \rangle$ of $\overline{R}_j$ in the given orders.
   (b) Certify integrity of r by checking if $h(r) = h$. If check fails R_j is not genuine.
   (c) Evaluate condition of Q on r, if condition is true then r is in the result R of Q.
7. If all previous checks are successful R is the certified result of Q.

Theorem 1. *Algorithm 1 correctly certifies completeness and integrity of a basic query in one query round.*

Proof. (sketch) Consider query Q with result R and queries $Q_1, \ldots, Q_m$, as computed in Step 1, and their results $R_1, \ldots, R_m$. For all $i = 1 \ldots m$, $R \subseteq R_i$ holds. This implies that R can be computed starting from an arbitrary R_j, also the one that is the result of the fastest query as selected in Step 4.

From results summarized in Section 3, we assume results $\overline{R}_1, \ldots, \overline{R}_m$ of the corresponding security queries to be correctly certified as complete by Step 5.

By matching each tuple of R_j with the corresponding tuple of $\overline{R}_j$ (Step 6.a) and exploiting the row-hashing technique (Step 6.b), we certify that R_j is correct and complete. Step 6.c selects from R_j only the rows that belongs to R. Completeness and correctness of R derives from completeness and correctness of R_j and from the fact that final selection is performing in a trusted environment.

The presented algorithm provides certification of completeness of result of Q with only one query round. In this query round all queries $Q_1, \ldots, Q_m$, $\overline{Q}_1, \ldots, \overline{Q}_m$ are concurrently performed.

Alternatively, if we admit two query rounds it is possible to obtain the same result performing much less queries and transferring much less data. Step 3 (first query round) only perform security queries $\overline{Q}_1, \ldots, \overline{Q}_m$ thus avoiding to transfer potentially big tuples in results for $Q_1, \ldots, Q_m$. Step 4a select the fastest, say $\overline{Q}_j$ and abort the others. Step 4b (second query round) performs query Q_j. The rest of the algorithm is unchanged. Also, in certain circumstances, might be convenient to choose one of the security queries $\overline{Q}_1, \ldots, \overline{Q}_m$ that is expected to be the fastest.

4.3 Optimized Security Queries

Suppose to have a table T and a security table S_a on attribute a of T. Suppose to have a monodimensional canonical basic query, asking for all elements in the range $[x, y]$ (extremes may or may not be included), and its result R. We need verification object $\overline{R}$ for R. The simplest way to proceed leads to obtain a verification object for each value in the interval requiring $O(|R|)$ queries, and $O(|R| \log |T|)$ size overall for the verification object of R. However, when considering elements of a range, verification objects of consecutive elements largely overlap, as shown in Fig. 3.

In this section, we describe a procedure for getting a verification object of size $O(|R| + \log |T|)$ with $O(1)$ queries and a procedure to certify the completeness and integrity of the result $\overline{R}$ of the security query $\overline{Q}$ for range $[x, y]$.

Di Battista and Palazzi [2] show two ways to represent an authenticated skip list in a relational table: (i) a coarse-grained representation, in which each tuple stores an entire tower, and (ii) a fine-grained representation, in which each tuple stores only one level of a tower. For canonical monodimensional basic queries whose conditions contain only equalities, they show how to retrieve the verification object using $O(\log |T|)$ queries, in the coarse-grained approach, and using only one query, in the fine-grained approach.

We adopt the fine-grained representation. Let x' (y') be the first element less (greater) than or equal to x (y). The verification object $\overline{R}$ contains verification objects $V_{x'}$ and $V_{y'}$ for elements x' and y', the elements within that range (x' and y' included), and corresponding height of the tower for each of that elements. Adopting a fine-grained representation this can be done using only three queries that can be concurrently executed, namely, one query for the elements and the corresponding heights of the towers, and two queries for the verification objects of interval bounds x' and y'. Note that, in the above queries, x' and y' are not

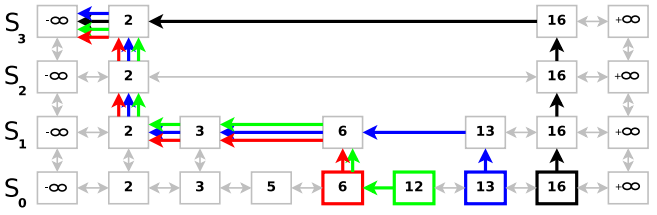


Fig. 3. Overlapping of verification objects for elements in a range

mentioned, x and y are specified instead, so that no further queries to obtain x' and y' are needed.

On the client, that is, in a trusted environment, the portion of the authenticated skip list for elements of $\overline{R}$, and hence their verification objects, can be computed starting from $\overline{R}$, $V_{x'}$, $V_{y'}$ and the height of the towers. The complete certification algorithm follows.

Algorithm 2

1. Certify verification objects $V_{x'}$, $V_{y'}$ as explained in Section 3.
2. Consider the values of elements e in $\overline{R}$, from the greatest value to the lowest, and for each of them perform the following steps.
   (a) Re-construct tower for element e computing hashes of each level according to the rules of the skip list data structure (see Section 3).
   (b) When a level of a tower is also present in either $V_{x'}$ or $V_{y'}$ compare the value of the hash previously known with that computed. If this check fails $\overline{R}$ is not genuine.
3. If all previous checks are successful $\overline{R}$ is correct and complete.

Theorem 2. *Algorithm 2 correctly certifies completeness of a security query asking for a range of values.*

Proof. (sketch) If Step 1 fails, at least one of the extremes of the result is not correct, as recalled in Section 3. So, suppose $V_{x'}$, $V_{y'}$ have been correctly certified and the result returned by the query within the range $[x', y']$ is not correct or is not complete, we prove that Step 2.b must fail. Let $w \in [x, y]$ be an element contained in the original skip list that is not present in the result. Let w' the greatest element belonging to the result that is lesser than w, possibly coinciding with x'. Consider the reconstruction of the original skip list as performed in Step 2.a and consider the search path for w' connecting the bottom nodes of the tower of w' and the top node of the tower for $-\infty$. According to the skip list rules (see Section 3), in the original skip list, the hash values in this path accumulates also hash values coming from w. This path must overlap the search paths for x' and y' (they overlap at least at the top node of the tower for $-\infty$). If Step 2.b does not fail on the overlapping, either the result is correct and complete or a collision for the hash function can be found comparing the reconstruction of the partial skip list and the original one.

When this algorithm is used within Algorithm 1, a further optimization can be performed. It is not needed, for the verification object $\overline{R}$, to report values contained in the authenticated skip list. Such values, as well as, row-hash values can be computed from the result of the query on the regular table (that is, from R). Note that, this can be relevant in practice, even if it does not bring any asymptotic improvement for the size of the verification object since height of towers are still needed and contribute $O(|R|)$ to it.

4.4 General Selection Queries

In this section we define a larger class of queries that we handle in Section 4.5.

Let T be a relational table with attributes $a_1, \ldots, a_m$.

General Selection Query. We call *general selection query* a query in the following form $\sigma_{f(a_1,\ldots,a_m)}(T)$ where $f(a_1, \ldots, a_m)$ is a generic boolean expression arbitrarily composed using operators $\wedge$, $\vee$, and $\neg$. Sub-expressions that do not contain those operators are called *atoms* and are in the form $\alpha \star c$ where $\alpha \in \{a_1, \ldots, a_m\}$ is an attribute of T, c is a constant of the the same type of α, and $\star$ is one of the following operators: $=, >, \geq, <, \leq$.

For any boolean formula, an equivalent boolean formula in disjunctive normal form can be obtained using elementary boolean algebra. Also, using the following elementary equivalence rules, it is possible to obtain equivalent expressions in disjunctive normal form that does not contain negation.

$$\neg(a = c) \equiv a > c \vee a < c,\ \neg(a > c) \equiv a \leq c,\ \neg(a \geq c) \equiv a < c,$$
$$\neg(a < c) \equiv a \geq c,\ \neg(a \leq c) \equiv a > c$$

A general selection query is said *canonical* if its condition is in disjunctive normal form and does not contain negation.

4.5 Completeness Certification for General Selection Queries

Consider a relational table T with attributes $a_1, \ldots, a_m$ and a general selection query Q on T. Without loss of generality, we assume Q to be canonical and having condition $\bigvee_{j=1}^{q} g_i$, where g_i is in the form $g_i = \bigwedge_{j=1}^{n_q} f_{ij}$ and each f_{ij} is an atom. To execute Q on a table T client performs the following algorithm.

Algorithm 3

1. Construct q basic queries Q_i $(i = 1, \ldots, q)$, in the form $\sigma_{g_i}(T)$
2. Basic queries $Q_1, \ldots, Q_q$ are all concurrently executed using Algorithm 1 obtaining certified results $R_1, \ldots, R_q$.
3. The result of Q is $R_1 \cup \cdots \cup R_q$.

5 Remarks on the Execution of Query Racing

Algorithms described in Section 4 assume that smaller the result of a query, shorter the time a user/client has to wait for it. Even if this assumption sounds

reasonable, several aspects affect the waiting time of a user. In Section 4, we implicitly assumed that concurrent queries either are run on distinct processor or are fairly scheduled such that at each query is given roughly the same amount of CPU/Disk time. Also, current systems are rather complex. The DBMS server interacts with network, and operating systems in ways that are hard to predict, also, indexes can greatly speed up some queries with respect to others.

Even if in this context it is impossible to provide any theoretical statement about the time a query takes to complete, we experimentally verified that the time increases with the size of the result in most of the common cases. In certain particular cases, the results of monodimensional queries might turn out to be much bigger of the result of the main query (for example, selecting a particular date when day, month, and year are stored in distinct attributes). In these cases, concerning security queries, the set of attributes that are responsible for the problem can be treated like a single attribute, solving the efficiency problem.

The main contributions to the waiting time are the time taken by the DBMS to compute the result and the time taken by the network to transfer it.

If network is a bottleneck, our assumption is reasonable: supposing that all queries traverse same network, transferring more data implies longer wait time.

Suppose the DBMS server is the bottleneck. If the system is highly parallel with respect to CPU and disks, which is the case for large clouds, we can assume each query does not have to compete with the others for CPU and Disk, hence concerning waiting time we can assume it runs alone. If the system is not highly parallel, we can always suppose that they get the same share of CPU and disk. In both cases, the important thing to understand is if the time taken by a DBMS to complete a query, behave monotonically with respect to query size result. This is investigated experimentally in Section 7.

6 Comparison with the State of the Art

In this section we briefly discuss our results with respect to the state of the art according to criteria expressed in Section 2.

We provide completeness certification for general multidimensional selection queries on dynamic databases, having as condition any boolean expression with any order-based operator on any set of attributes.

This result is also achieved in other papers [14,20,4,6,16,17] but with different trade offs. We now briefly compare our work with each of them.

Our approach allows us to answer any selection query with the same efficiency, while Devanbu et al. [14] adopt range trees optimized for a specific set of queries to be decided when the authenticated database is created.

Concerning efficiency, we require only one query-round, do not mandate any (symmetric or asymmetric) encryption, except for the basis, and complexity verification is bounded to the fastest query which is very often the most selective (see Section 7. Pang et al. [4,17] require to sign each tuple and/or to compute complicated hashes which may be a burden for client and users.

The technique described by Yang et al. [16] provide verification objects of unpractical size and uses many query-rounds.

Xie et al. [6] describe an efficient and flexible probabilistic method, however, they do not assure to detect all malicious changes, in particular for punctual ones.

The solution provided by Li et al. [20] requires a customized DBMS while our approach can use a plain DBMS.

Our method does not hinder the possibility to adopt other complementary techniques known in literature. In particular, concerning privacy, results described in [5] and also order preserving encryption can be adopted. Concerning freshness, results from [7] can be used.

7 Experimental Evaluation

This section shows experimental results that aim to validate techniques and discussions presented in Sections 4 and 5, and to report the performance of our prototypical implementation. We based our experiments on two data sets. The first data set, called *artificial*, is randomly built. We created tables with number of tuples ranging from $10,000$ to $1,000,000$ and with number of attributes ranging from 1 to 100. All attributes have type string (MySql type `varchar`). The second is the *Adult Data Set* publicly available from Machine Learning Repository [26]. It has 14 attributes and 32,561 tuples.

7.1 Validation of Monotonicity

Our first aim is to provide a first empirical validation to the hypothesis that the time taken by a DBMS server to execute a query monotonically increases with result set size (see Section 5).

We considered 13 different queries on Adult Data Set, the result set size is recorded for each of them. We run each query ten times and we recorded execution time for each run. We performed our experiments on a MySQL DBMS server (ver. 5.0) running on a Linux system (kernel version 2.6.24 with cfq disk scheduler), on a small laptop with 2GB of ram and no other significant load on the machine. Time measures were taken by the MySQL profiling system (using `SHOW PROFILES`).

We performed the whole experiment in four possible situations that encompass having or not having indexes on the (six) attributes involved by the queries, and exploiting or not exploiting caches in MySQL (cache disabled by issuing `SET GLOBAL query_cache_type=OFF`) and Linux (cache cleaned before each query by issuing `sync; echo 3 > /proc/sys/vm/drop_caches`).

Results are summarized in Fig. 4. It is possible to see that, in our experiments, monotonicity is roughly respected in all situations. We can see that when caches are available, performances are more predictable (smaller min-max ranges), see Figs. 4(b) and 4(d). In our experiments, indexes do not provide very much improvement while augmenting the possibility of non-monotonic behavior,

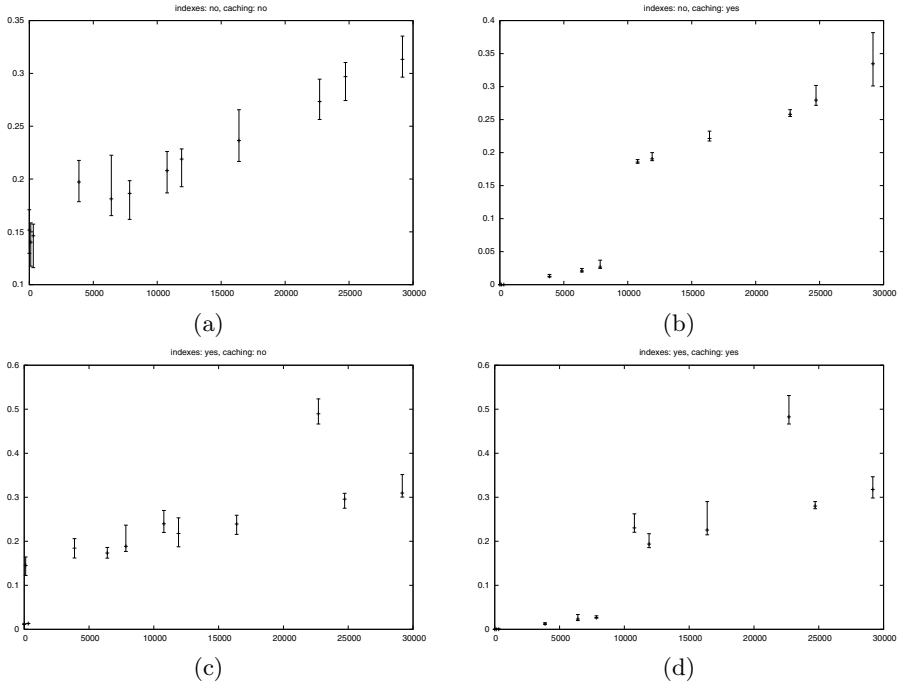


Fig. 4. Results of experiments performed on Adult Data Set to validate monotonicity of query execution time vs. size of the result. Abscissae report the size of the result set of the query, ordinates report average, minimum, and maximum time taken by the DBMS server to execute the query. The four charts show measure in the following four situations: (a) neither indexes nor caches, (b) no index and caching activated, (c) indexes available and caching not active, (d) indexes available and caching active.

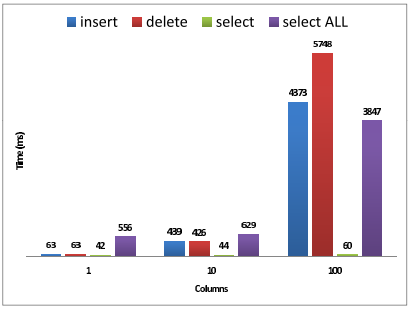
see Figs. 4(c) and 4(d). The most evident misbehavior is the query with whose result set size is 22,696 which is much slower than the others and whose behavior can be hardly explained considering the cardinality (nine) of the values of the single attribute on which the query performs selection.

7.2 Performance

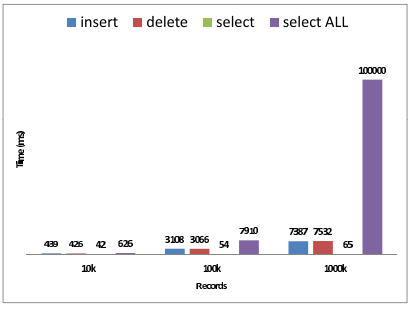
We now intend to show the performance of certified queries using our prototypical implementation.

The following experiments were performed on a dual core CPU 2.10GHz with 3 Gb of RAM and one hard disk (5,400 rpm Serial ATA) running Linux 2.6.24 (Ubuntu^{TM}8.04, 32 bit). We used the MySQL DBMS (version 5.0.45). The prototypical software is written in Java^{TM}and uses MySql JDBC Connector Java-bean 5.1.15 to connect to the DBMS.

Our prototypical implementation, adopts the algorithm described in Section 4.1 using the coarse-grained representation. This means that optimizations described



(a) Varying number of attributes (10,000 tuples).



(b) Varying number of tuples (10 attributes).

Fig. 5. Scalability analysis for operation insert, delete, select, and select-all. (a) Variation of performace while increasing the number of attributes. (b) Variation of performances while increasing the number of tuples.

in Section 4.3 are only partially implemented, hence, data presented in this section are pessimistic with respect to potential performance of the proposed technique. Namely, the number of query rounds to obtain a verification object for a (range) query is $O(\log |T|)$, where $|T|$ is the number of tuples in the table. After the verification object is received, verification is performed as described in Section 4.3. Select-all queries are more optimized in the sense that even if the skip list representation is coarse-grained, the verification object is obtained using a single query.

We first report tests about scalability performed on Artificial data set and then report performances on Adult Data Set taken from real life.

Scalability. We analyze performances of our prototype for the following four operations: insertion (*insert*), deletion (*delete*), single value selection (*select*), and full table selection (*select-all*).

For each operation and for each size of the artificial data set, we performed the operation 10 times. Time measurements have been performed within the software, and hence it accounts for transmission time to obtain data and verification object and computation time get certification of completeness and integrity.

Results in Fig. 5(a) show scalability with respect to number of attributes and in Fig. 5(b) show scalability with respect to number of tuples.

As figures show, our techniques perform very well for selections, since verification object is obtained with a few query rounds in our implementation. Also select-all queries scale well considering the amount of data involved. Theoretically, augmenting the number of attributes, the cost of computing row-hash increases linearly with the number of attributes. However, the time taken for computing row-hash is negligible with respect to query execution time.

Insertion and deletion scale poorly with respect to the number of attributes, see Fig. 5(a), since each change has to be performed on each security table. On the contrary, scalability, with respect to the number of tuples, is fairly good

Query Condition	Result Size (tuples)	Result Size (bytes)	Plain Execut. (ms)	Certified Execut. (ms)	Verification (ms)
`makemoney = '>50K' AND age BETWEEN '17' AND '25'`	114	14029	323	1363	203
`makemoney = '>50K' AND age BETWEEN '17' AND '25' AND race = 'Amer-Indian-Eskimo'`	2	276	254	392	14
`makemoney = '>50K' AND workclass = 'Private' AND nativecountry = 'United-States' AND sex = 'Male'`	3879	470016	1852	14543	1877

Fig. 6. Execution time for some example queries on the Adult Data Set

since authenticated skip lists are quite efficient. Also note that, these kind of operations might greatly benefit of a more parallel platform in which changes to the security tables can be concurrently performed. In our experiment, the time taken to perform insertion or deletion is basically the sum of the time taken to perfrom the operation on each security table.

Performances on real data. We consider some queries on Adult Data Set. Table 6 reports, for each query, result set size (tuples and bytes as reported by MySQL), time taken by a plain execution, total time taken by an execution with certified completeness and integrity, and time spent by client for result certification. For this experiments, we used the same non-optimized prototype that may perform several query rounds for each security query.

8 Conclusion

We presented a method that largely improves, with respect of the state of the art, the class of queries that can be authenticated using a conventional DBMS, without modifying pre-existing schemas and data, and without knowing the structure of the queries in advance. As future work we intend to exploit the same ideas in the context of queries involving join operations.

References

1. Devanbu, P.T., Gertz, M., Martel, C.U., Stubblebine, S.G.: Authentic third-party data publication. In: DBSEC, pp. 101–112 (2001)
2. Di Battista, G., Palazzi, B.: Authenticated relational tables and authenticated skip lists. In: DBSEC, pp. 31–46 (2007)
3. Miklau, G., Suciu, D.: Implementing a tamper-evident database system. In: ASIAN: 10th Asian Computing Science Conference, pp. 28–48 (2005)
4. Pang, H., Jain, A., Ramamritham, K., Tan, K.: Verifying completeness of relational query results in data publishing. In: SIGMOD Conf., pp. 407–418 (2005)

5. Pang, H., Tan, K.L.: Authenticating query results in edge computing. In: Proc. of the 20th Int. Conference on Data Engineering, pp. 560–571 (2004)
6. Xie, M., Wang, H., Yin, J., Meng, X.: Integrity auditing of outsourced data. In: VLDB, pp. 782–793 (2007)
7. Xie, M., Wang, H., Yin, J., Meng, X.: Providing freshness guarantees for outsourced databases. In: EDBT, pp. 323–332. ACM, New York (2008)
8. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. Trans. Storage 2(2), 107–138 (2006)
9. Tamassia, R.: Authenticated data structures. In: Di Battista, G., Zwick, U. (eds.) ESA 2003. LNCS, vol. 2832, pp. 2–5. Springer, Heidelberg (2003)
10. Merkle, R.C.: A certified digital signature. In: Brassard, G. (ed.) CRYPTO 1989. LNCS, vol. 435, pp. 218–238. Springer, Heidelberg (1990)
11. Goodrich, M.T., Tamassia, R., Schwerin, A.: Implementation of an authenticated dictionary with skip lists and commutative hashing. In: Proc. DISCEX II, pp. 68–82 (2001)
12. Buldas, A., Roos, M., Willemson, J.: Undeniable replies for database queries. In: Proc. Intern. Baltic Conf. on DB and IS, vol. 2, pp. 215–226 (2002)
13. Goodrich, M.T., Tamassia, R., Triandopoulos, N.: Super-efficient verification of dynamic outsourced databases. In: Malkin, T.G. (ed.) CT-RSA 2008. LNCS, vol. 4964, pp. 407–424. Springer, Heidelberg (2008)
14. Devanbu, P., Gertz, M., Martel, C., Stubblebine, S.G.: Authentic data publication over the Internet. Journal of Computer Security 11(3), 291–314 (2003)
15. Singh, S., Prabhakar, S.: Ensuring correctness over untrusted private database. In: EDBT 2008, pp. 476–486. ACM, New York (2008)
16. Yang, Y., Papadias, D., Papadopoulos, S., Kalnis, P.: Authenticated join processing in outsourced databases. In: SIGMOD 2009, pp. 5–18. ACM, New York (2009)
17. Zhou, Y., Salehi, A., Aberer, K.: Scalable delivery of stream query results. PVLDB 2(1), 49–60 (2009)
18. Bloom, B.H.: Space/time trade-offs in hash coding with allowable errors. Communications of the ACM 13, 422–426 (1970)
19. Cheng, W., Tan, K.: Authenticating knn query results in data publishing. In: Proc. 4th Int. Workshop on Secure Data Management, pp. 47–63 (2007)
20. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: ACM SIGMOD, pp. 121–132 (2006)
21. Dang, T.K.: Ensuring correctness, completeness, and freshness for outsourced tree-indexed data. Information Resources Management Jrnl., 59–76 (2008)
22. Narasimha, M., Tsudik, G.: Authentication of outsourced databases using signature aggregation and chaining. In: Li Lee, M., Tan, K.-L., Wuwongse, V. (eds.) DASFAA 2006. LNCS, vol. 3882, pp. 420–436. Springer, Heidelberg (2006)
23. Yang, Y., Papadopoulos, S., Papadias, D., Kollios, G.: Spatial outsourcing for location-based services. In: ICDE, pp. 1082–1091 (2008)
24. Polivy, D.J., Tamassia, R.: Authenticating distributed data using Web services and XML signatures. In: Proc. ACM Workshop on XML Security (2002)
25. Pugh, W.: Skip lists: A probabilistic alternative to balanced trees. In: Workshop on Algorithms and Data Structures, pp. 437–449 (1989)
26. UCI Machine Learning Repository, University of California, Irvine, School of Information and Computer Sciences (2007), `http://www.ics.uci.edu/~mlearn/MLRepository.html`