Authentic Third-party Data Publication

Prem Devanbu, Michael Gertz, Chip Martel
Department of Computer Science
University of California
Davis, California CA 95616 USA
{devanbu|gertz|martel}@cs.ucdavis.edu

Stuart G. Stubblebine*
CertCo
55 Broad Street – Suite 22
New York, NY 10004
stubblebine@cs.columbia.edu

Abstract

Integrity critical databases, such as financial information, used in high-value decisions, are frequently published over the internet. Publishers of such data must satisfy the integrity, authenticity, and non-repudiation requirements of end clients. Providing this protection over public data networks is costly. This is partly because building and running secure systems is hard. In practice, large systems can not be verified to be secure and are frequently penetrated. The negative consequences of a system intrusion at the data publisher can be severe. The problem is further complicated by data and server replication to satisfy availability and scalability requirements.

We aim to *reduce the trust required* of the publisher of large, infrequently updated databases. To do this, we separate the roles of owner and publisher. With a few trusted digital signatures on the part of the owner, an untrusted publisher can use techniques based on Merkle hash trees, to provide authenticity and non-repudiation of the answer to a database query. We *do not* require a key to be held in an on-line system, thus reducing the impact of system penetrations. By allowing untrusted publishers, our solution moves towards more scaleable publication of large databases.

1 Background

Consider an Internet financial-data warehouse, with historical data about securities such as stocks and bonds, that is used by businesses and individuals to make important investment decisions. The *owner* (or creator) of such a database might be a rating/analysis service company (such as Standard & Poors), or it might be a government agency. The owner's data might be needed at high rates, for example by user's investment tools. We focus our attention on data which changes infrequently. We assume high Query/Update ratios, with millions of queries per day. The data needs to be delivered promptly, reliably and accurately.

One approach to this problem is for the owner of the information to digitally sign the answers to users' queries, using a private signing key, sk_O. This signature is verified using the corresponding public key, pk_O. Based on the signature a user can be sure that the answer comes from the owner, and that the owner can't claim otherwise. However, there are several issues here. First the owner of the data may be unwilling or unable to provide a sufficiently reliable and efficient database service to handle the needed data rates. Second, even if the owner is willing and able to provide this service, the owner needs to maintain a high level of physical security and system security to defend against attacks. This has be done to protect the signing key, sk_O, which must be resident at the server at all times to sign outgoing data. In practice, large software systems have vulnerabilities, and keeping secret information on a publicly-accessible system is always risky. Using special hardware devices to protect the signing key will help, as would emerging cryptographic techniques like "threshold cryptography," but these methods do not fully solve the system-vulnerability problem, and can be too expensive in our domain, both computationally and financially.

*This paper is under continuous revision; rather than forwarding a copy, please get the latest from `http://seclab.cs.ucdavis.edu/~devanbu/authdbpub.pdf`

A more scalable approach is to use trusted third-party *publishers* in conjunction with a key management mechanism which allows a certified signing key of a publisher to speak for the owner. The database (or database updates) is provided securely to the publisher, who responds to user queries by signing them with it's own (certified) signing key, sk_P. Presumably, the market for useful databases will motivate publishers to provide this service, unburdening database owners of the need to do so. The owner simply needs to sign the data after each update and distribute it to the publisher. As demand increases, more publishers will emerge, or more capable ones, making this approach inherently scalable. But the approach still suffers from the problem and expense of trying to maintain a secure system accessible from the Internet. Furthermore, the user might worry that a publisher engages in deception. The user has to find a publisher that she can trust. She would have to believe that her publisher was both competent and careful with site administration and physical access to the database. She might worry about the private signing-key of the publisher, which has to be resident at the publisher's server and is therefore vulnerable to attack. To gain trust, the publisher would have to adopt meticulous administrative practices, at far greater cost. The need for trusted publishers would increase the reliance on brand-names, which would limit market competition.

In a summary of fundamental problems for electronic commerce [21], Tygar asks "How can we protect re-sold or re-distributed information ... ?" We present an approach to this problem.

2 Our Approach

We allow an **untrusted** *publisher* to provide a **verification-object** $\mathcal{VO}$ to a *user* to verify an answer to her database query. The user can use the $\mathcal{VO}$ to gain assurance that the answer is just what the database *owner* would have provided. The verification-object is based on a **summary-signature** that the owner periodically distributes to the publisher. See Figure 1.

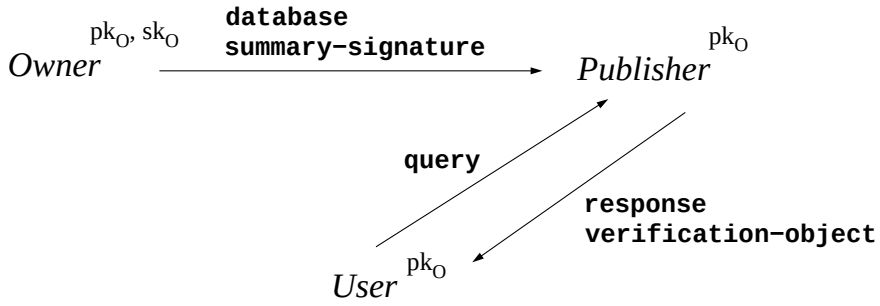

Figure 1: We partition the role of information provider into *owner* and *publisher*. The owner provides database updates and summary signatures to the publisher. The publisher is untrusted. The *user* makes inquiries with the publisher. She gets responses which can be verified using a returned *verification-object*. Superscripts denote keys known to that party. Only sk_O is secret. The client must be sure of the binding of pk_O to the owner.

The summary-signature is a bottom-up hash computed recursively over B-tree type indexes for the entire set of tuples in each relation of the owner's database, signed with sk_O. Answers to queries are various combinations of subsets of these relations. Given a query, the publisher computes the answer. To show that an answer is correct the publisher constructs a verification-object using the same B-tree that the owner had used to compute the summary-signature. This verification-object validates an answer by providing an unforgeable "proof" which links the answer to the summary-signature. Our approach has several features:

1. Besides his own security, a user need only trust the key of the owner. The owner only needs to distribute the summary-signature during database updates. So, the owner's private key can be maintained in an "offline" machine, isolated from network-based attacks. The key itself can be ensconced in a hardware token, which is used only to sign a single hash during updates.

2. Users need not trust the publishers, nor their keys. In particular, if a particular publisher were compromised, the result would *only* be a loss of service at that publisher.
3. In all our techniques, the verification-object is of size linear in the size of the answer to a query, and logarithmic in the size of the database.
4. The verification-object guarantees that the answer is correct, without any extra or missing tuples.
5. In all of our techniques, the overheads for computing the summary-signature, the $\mathcal{VO}$, and for checking the $\mathcal{VO}$ are reasonable.
6. The approach offers far greater survivability. Publishers can be replicated without co-ordination, and the loss of one publisher does not degrade security and need not degrade availability.

A correct answer and verification-object will always be accepted by the user. An incorrect answer and verification-object will almost always be rejected, since our techniques make it computationally infeasible to forge a correct verification-object for an incorrect answer. Overall, the approach nicely simplifies the operational security requirements for both owners and publishers.

3 Preliminaries

In this section we will discuss the basic notions, definitions and concepts necessary for the approach presented in this paper. In Section 3.1 we will present the basic notions underlying relational databases and queries formulated in relational algebra. In Section 3.2 we will discuss the computation and usage hash-trees.

3.1 Relational Databases

The data model underlying our approach is the relational data model (see, e.g., [6, 19]), where the owner and the publisher manage the data using a relational database nagement system (RDBMS). A *relation schema* $R\langle A_1, A_2, \ldots, A_n\rangle$ consists of a relation name R and an ordered set of attribute names $\langle A_1, A_2, \ldots, A_n\rangle$, also denoted by $schema(R)$. Each attribute A_i is defined on a domain D_i. An *extension* of a relation schema with arity n (also called *relation*, for short) is a finite subset of the Cartesian product $D_1 \times \ldots \times D_n$. The extension of a relation schema R is denoted by r. The value of a tuple $t \in r$ for an attribute A_i is denoted by $t.A_i$. We assume that with each relation schema R a set $pk(R) \subseteq \{A_1, \ldots, A_n\}$ is associated which designates the primary key. The number of tuples in a relation r is called the *cardinality* of the relation, denoted by $\mid r \mid$. A *database schema* $\mathcal{S}$ is a collection of relation schemas $\mathcal{S} = \{R_1, \ldots, R_m\}$. For a database schema $\mathcal{S}$, the extension of the relation schemas at a particular point in time is called a *database instance* (or *database*).
The basic five operators are selections, projections, unions, cartesian products and set-differences. Other popular derived operations, used in complex queries, are *natural join or equi-join* $\bowtie$, *condition join or theta-join* $\bowtie_C$ (with C being a condition on join attributes), and *set-intersection* $\cap$.

3.2 Merkle Hash Trees

We describe the computation of a Merkle Hash Tree [13] for over given relation r with relation schema $R = \langle A_1, \ldots, A_n\rangle$. For this, assume that $\mathcal{A} = \langle A_i, \ldots, A_k\rangle$ is a list of attributes from $schema(R)$. A Merkle Hash Tree, denoted by $MHT(r, \mathcal{A})$, is computed using a collision-resistant hash function h:

1. First, compute the *tuple hash* h_t for each tuple $t \in r$ thus:

$$h_t(t) \;=\; h(h(t.A_1) \;||\; \ldots \;||\; h(t.A_n))$$

The tuple hash (by the collision resistance of the hash function) functions as a "nearly unique" tuple identifier (for a hash-length of 128 bits, probability of collisions approaches 2^{-128}). We also assume a distinct "boundary tuple" t_b with artificial attribute values chosen to be the extremal tuple in the relation, with corresponding tuple hash h_b .

2. Next, compute the Merkle hash tree for relation r. For this, assume that r is sorted by the values of $\mathcal{A}$ so that for two distinct tuples $t_{i-1}, t_i \in r$, $t_{i-1}.\mathcal{A} \leq t_i.\mathcal{A}$.

$$\text{Leaf-nodes}: \qquad h^0(i) \;=\; h_t(t_i)\;,\; i = 1 \ldots \mid r \mid,$$

$$h^j(i) \;=\; h(h^{j-1}(2i-1) \;\mid\mid h^{j-1}(2i))\; for\; i \;= 1 \ldots \left\lceil (\frac{\mid r \mid}{2^j}) \right\rceil, j \;=\; 1 \ldots \lceil \log_2(\mid r \mid) \rceil$$

If a $h^{j-1}(\ldots)$ needed to compute a h^j is not defined by the lower levels, it is taken to be h_b. Thus h_b is used to indicate the boundary of the MHT.

The sole hash value at the level $\lceil \log_2(\mid r \mid) \rceil$, is the "root hash" of the Merkle tree. In the sequel, we denote the two values $h^{j-1}(2i-1)$ and $h^{j-1}(2i))$ used to compute $h^j(i)$ as *hash siblings*; $h^j(i)$ is their *parent*. In figure 2: $h34$ is the parent of $h3$ and $h4$; $h3$ and and $h4$ are hash siblings. This construction easily generalizes to a higher branching factor $K > 2$, such as in a B^+-tree; however, for our presentation here, we primarily use binary trees. Indeed, our approach works best if the *owner* and the *publisher* build an MHT around index structures that are used in query evaluation. In this case, constructing a $\mathcal{VO}$ is a very minor overhead over the query evaluation process itself.

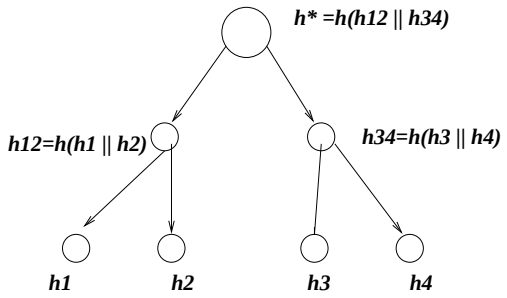

Figure 2: Computation of a Merkle hash tree

Note that (by the cryptographic assumption of a collision-resistant hash function) if the correct value of the *parent* is known to the client, the publisher cannot forge the value of its children. The *owner's* signature on the root hash value of the Merkle tree underlies the security of our approach, in the same spirit as [15].

Definition 1 (Hash Path) *Let $h^0(i)$ be a leaf node in $MHT(r, \mathcal{A})$ corresponding to a tuple $t_i \in r$. The nodes necessary to compute the hash path up to the root hash is denoted as $path(t_i)$. Such a hash path always has the length $\lceil \log_2(\mid r \mid) \rceil$ and comprises $2 * \lceil \log_2(\mid r \mid) \rceil - 1$ nodes where exactly two nodes are leaf nodes. Of these, only $\lceil \log_2(\mid r \mid) \rceil + 1$ need be provided to recompute the value at the root. Hash paths can also be provided for non-leaf nodes.*

The $\lceil \log_2(\mid r \mid) \rceil + 1$ nodes in $path(t_i)$ constitute the $\mathcal{VO}$ showing that t_i is actually in the relation. The *owner*'s signature on the root node certifies its authenticity. Indeed any interior node within the hash tree can be authenticated by giving a path to the root. Hash paths show that tuples actually exist in a relation; to show that set of tuples is complete, we need to show boundaries.

Definition 2 (Boundaries) *Any non-empty contiguous sequence $q = \langle t_i, \ldots, t_j \rangle$ of leaf nodes in a Merkle Hash Tree $MHT(r, \mathcal{A})$, has two special leaf nodes $LUB(q)$ and $GLB(q)$ that describe the lowest upper and greatest lower bound values, respectively, of q and are defined as follows:*

(1) $GLB(q) := \{t \mid t \in r \wedge t.\mathcal{A} < t_i.\mathcal{A} \wedge (\neg\exists s \in r : s.\mathcal{A} > t.\mathcal{A} \wedge s.\mathcal{A} < t_i.\mathcal{A})\}$

(2) $LUB(q) := \{t \mid t \in r \wedge t.\mathcal{A} > t_j.\mathcal{A} \wedge (\neg\exists s \in r : s.\mathcal{A} < t.\mathcal{A} \wedge s.\mathcal{A} > t_j.\mathcal{A})\}$

The above definition is somewhat simplified, and ignores tuples occurring at the edges of the MHT; that extension is straightforward. We also assume that both $GLB(q)$ and $LUB(q)$ are singletons. This can easily be accomplished by adding $pk(R)$ to the list $\mathcal{A}$ of attributes by which the leaves in $MHT(r, \mathcal{A})$ are ordered.

Any non-empty contiguous sequence $q = \langle t_i, \ldots, t_j \rangle$ leaf nodes in a Merkle Hash Tree $MHT(r, \mathcal{A})$, has a lowest common ancestor $LCA(q)$. This situation is illustrated in Figure 2. Given $LCA(q)$, one can show a hash path $path(LCA(q))$ to the authenticated root hash value. After this is done, (shorter) hash paths from each tuple to $LCA(q)$ can provide evidence of membership of q in the entire tree. This approach can also be used to provide evidence that two nodes are proximate in the tree.

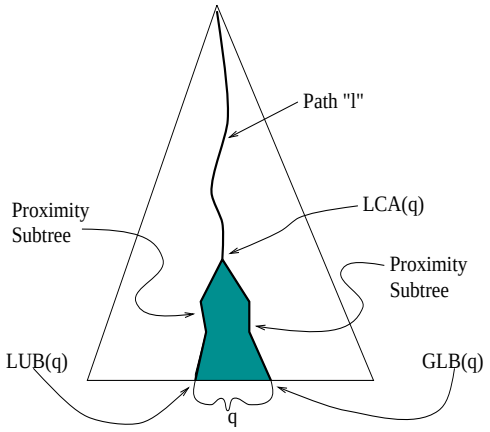

Figure 3: A Merkle tree, with a contiguous subrange q, with a least common ancestor LCA(q), and upper and lower bounds. Note verifiable hash path "l" from LCA(q) to the root, and the proximity subtrees (thick lines) for the "near miss" tuples for LUB(q) and GLB(q) which show that q is complete.

Definition 3 ***(Proximity Subtree)*** *Consider a consecutive pair of tuples (leaf nodes) s, t in $MHT(r, \mathcal{A})$, and their lowest common ancestor, $LCA(\langle s, t \rangle)$. This node, along with the two paths showing that s (respectively, t) is the rightmost (leftmost) element in the left (right) subtree of $LCA(\langle s, t \rangle)$ constitute the "proximity subtree" of s and t, denoted by $ptree(s, t)$.*

Proximity subtrees are used in boundary cases, with $GLBs$ and $LUBs$ *i.e.,* to show a "near-miss" tuple that occurs just outside the answer set lies next to the extremal tuples in the answer set. In this case, it is important to note that by construction, we just need to reveal the relevant attribute value in the "near-miss" to show that it is indeed a near miss; with just the hash of the other attributes, the tuple hash, and the rest of the proximity tree can be exhibited. This allows us to provide evidence of completeness without sending along the entire database.

We finally define desirable properties of the answer set q returned by *publisher*, in terms of the correct answer that would have been returned by *owner*.

Definition 4 *The answer given by* publisher *to a query q is* inclusive *if it contains only the tuples that would have been returned by* owner*, and is* complete *if contains all the tuples that would have been returned by* owner*.*

4 Base level Relational Queries

In this section we outline the computation of $\mathcal{VO}$ for answers to queries. We illustrate the basic idea behind our approach for selection and projection queries in Section 4.1 and 4.2, respectively. Slightly more complex types of queries (join queries) and set operators are discussed in Sections 4.3 and 4.4.

4.1 Selections

Assume a selection query of the form $\sigma_{A_i \Theta c}(r), c \in D_i$ which determines a result set $q \subseteq r$. Furthermore, assume that the Merkle Hash Tree $MHT(r, A_i)$ has been constructed. For each possible comparison predicate $\Theta \in \{=, \neq, <, >\}$, we show how the *publisher* can construct the $\mathcal{VO}$ that *client* can use to verify that the answer q is inclusive and comlete. In all the following cases, *if owner* and *publisher* construct Merkle hash trees over the same index structures used for querying, the overhead for constructing the $\mathcal{VO}$ is minor. We first consider the cases for the comparison predicate $\Theta \equiv =$ over some attribute A_i. We consider two cases: when $q = \{\}$, and otherwise. If $q = \{\}$, then the $\mathcal{VO}$ must show that tuple exists that satisfies the selection. For this, we provide paths of the two tuples that "surround" the non-existing tuple. The two tuples are determined by $GLB(q')$ and $LUB(q')$ with $q' = c$. Determining $path(GLB(q'))$ and $path(LUB(q'))$ requires searching the two associated tuples in the leaf nodes of $MHT(r, A_i)$. The proximity subtree $ptree(GLB(q'), LUB(q'))$ provides required evidence that the answer set is empty. The size of the $\mathcal{VO}$ is $O(\log_2 \mid r \mid)$.

If $q \neq \{\}$, it is a set of tuples which build a contiguous sequence of leaf nodes in $MHT(r, A_i)$. We provide a $\mathcal{VO}$ for q thus: first, identify $l := LCA(q \cup GLB(q) \cup LUB(q))$ in $MHT(r, A_i)$, and show a verifiable path from l to the root. Next, identify proximity subtrees showing that $GLB(q)$ ($LUB(q)$) occur consecutively to the smallest (largest) element of q. Now, the entire sub-tree from the elements of the set q to l can be constructed, using the hash values of the tuples in q. This verifies that the entire set occurs in the leaf nodes of the tree. To construct this subtree and to verify the root hash on the $LCA(q)$ of this subtree, the length of the $\mathcal{VO}$ is $O(\mid q \mid + \log_2(\mid r \mid))$. The proximity subtrees establish that no tuples are left out. We produce $\mathcal{VO}$s for $\Theta \equiv \neq$ (negated) and $\Theta \in \{<, >\}$ (range) queries in analogous fashion, bracketing the included tuples with boundary tuples to show inclusivity and completeness. Details are omitted here for brevity, and can be found in [8].

In [8] we present a formal proof that our construction $\mathcal{VO}$s for selections is secure:

Lemma 5 *If* publisher *cannot engineer collisions on the hash function or forge signatures on the root hash value, then if* client *computes the right authenticated root hash value using the $\mathcal{VO}$ and the answer provided for* selection *queries, then the answer is indeed complete and inclusive.*

The statement and proofs regarding the security of $\mathcal{VO}$s for operators besides selections are similar to the one presented for selections in [8]. This, however, is not an exact-security proof [2], which is left for future work.

4.2 Projections

For queries of the pattern $\pi_{\mathcal{A}}(R), \mathcal{A} \subset schema(R)$, the projection operator eliminates some attributes of the tuples in the relation r, and then eliminates duplicates from the set of shortened tuples, yielding the final answer q. There may be many different possible projections on a relation R. If *client* wishes to choose among these dynamically, it may be best to let the *client* perform the projection. The *client* will then also have to eliminate duplicates because these are not automatically eliminated in SQL, unlike the relational algebra. So in this case, the *client* is provided with the whole intermediate result $\mathcal{I}$ (or some subset thereof after intermediate selections etc) and the $\mathcal{VO}$ for $\mathcal{I}$ *before* the projection; so the $\mathcal{VO}$ will be linear in size $\mid \mathcal{I} \mid$, rather than the smaller size $\mid q \mid$ of the final result. Note also that

the projection may actually mask some attributes that the *client* is not allowed to see; if so, with just the hash of those attributes in each tuple, the *client* can compute the tuple hash, and the $\mathcal{VO}$ for $\mathcal{I}$ will still work.

Consider the case where a particular projection $\pi_{\mathcal{A}}(r)$ (which is used often) projects onto attributes where duplicate elimination will remove numerous tuples, leaving behind a small final answer q. Given the pre-projection tuple set, the *client* would have to do all this work. Now, suppose we have a Merkle tree $MHT(r, \mathcal{A})$, *i.e.,* we assume that the sets of retained attribute values can be mapped to single values (which corresponds to building equivalence classes) with an applicable total order. In this case, we can provide a $\mathcal{VO}$ for the projection step that is linear in the size of the projected result q.

Each tuple t in the result set q may arise from a set $S(t) \subseteq r$ with tuples having identical values for the projected attribute(s) $\mathcal{A}$. We must show that the set q is inclusive and complete:

1. To prove $t \in q$, we show the hash path from *any* witness tuple $y \in S(t) \subseteq r$ to the Merkle Root. However, "boundary" tuples make better witnesses, as we describe next.

2. To show that there are no tuples missing, say between t and $t', (t, t' \in q)$, we just show that $S(t), S(t'), \subseteq r$ are contiguous in the sorted order. Hash paths from two "boundary" tuples $y \in S(t)$ and $x \in S(t')$ that occur next to each other in the Merkle tree can do this.

We observe that both the above bits of evidence are provided by displaying at most $2 \mid q \mid$ hash paths, each of length $\lceil \log_2 r \rceil$. This meets our constraint that the size of the authentication evidence be bounded by $O(\mid q \mid \log_2 \mid r \mid)$.

Constructing merkle trees to provide compact $\mathcal{VO}$s for duplicate elimination with every possible projection might be undesirable. In this case, we might construct trees for only highly selective projection attributes, and leave the other duplicate eliminations to be done by the client, as with SQL.

4.3 Joins

Joins between two or more relations, specially equi-joins where relations are combined based on primary key – foreign key dependencies, are very common. We focus on pairwise joins of the pattern $R \bowtie_C S$ where C is a condition on join attributes of the pattern $A_R \Theta A_S$, $A_R \in schema(R), A_S \in schema(S), \Theta \in \{=, <, >\}$. For Θ being the equality predicate, we obtain the so-called equi-join. We show 3 different approaches, for different situations.

Given a query of the pattern $R \bowtie_C S$, one structure that supports computation of very compact $\mathcal{VO}$s for the query result is based on the materialization (i.e., the physical storage) of the Cartesian Product $R \times S$. This structure supports the three types of joins, which can all be formulated in terms of basic relational algebra operators, i.e., $R \bowtie_{A_R \Theta A_S} S := \sigma_{A_R \Theta A_S}(R \times S)$. Assume $m =\mid R \mid, n =\mid S \mid$. The verification structure for $R \bowtie_C S$ queries is constructed by sorting the Cartesian Product according to the *difference* between the values for A_R and A_S, assuming such an operation can be defined, at least in terms of "positive", "none" or "negative". This yields three "groups" of leaf nodes in the Merkle Tree: (1) nodes for $t.A_R - s.A_S$ for two tuples $t \in R, s \in S$ is 0, thus supporting equi-joins, (2) nodes where the difference is positive, for the predicate $>$, and (3) nodes where the difference is negative, for the predicate $<$. If only simple Θ joins, with $\Theta \equiv =, > or <$ are desired, there is no need to construct binary Merkle trees over the entire cross product—we can just group the tuples in $R \times S$ into the three groups, hash each group in its entirely, and append the three hashes to get the root hash. In this case, the $\mathcal{VO}$s for the three basic Θ queries would consist only of 2 hash values! For more complex queries,

If it is known that the querys will only result in equi-joins against the database, an optimized structure can be used. We will only sketch the basic concept for using this structure here. Instead of a space-consuming materialization of the Cartesian Product $R \times S$, we materialize the *Full Outer Join* $R \sqsupset\!\bowtie\!\sqsubset S$ which pads tuples for which no matching tuples in the other relation exist with null values (see, e.g.,

[6, 19]). The result tuples obtained by the full outer-join operator again can be grouped into three classes: (1) those tuples $ts, t \in R, s \in S$, for which the join condition holds, (2) tuples from r for which no matching tuples in s exist, and (3) tuples from s for which no matching tuples in r exist. Constructing a $\mathcal{VO}$ for the result of query of the pattern $R \bowtie_{A_R \Theta A_S} S$ then can be done in the same fashion as outlined above.

Suppose R and S have B-tree indices over the join attributes, and these trees have been Merkle-hashed; also suppose the size of the join result is q. We can now provide larger $\mathcal{VO}$s of size $O(q \log m + q \log n)$ *without materializing the cross product* $R \times S$. This is done by doing a "verified" merge of the leaves of the two index trees. Whenever the join attributes have the right θ relation, witness hash paths in the trees for R and S are provided to show inclusiveness of the resulting answer tuples; when tuples in R or S are skipped during the merge, we provide a pair of proximity subtrees in R or S respectively to justify the length of the skip. This conventional approach to joins gives rise to larger $\mathcal{VO}$s than the approach described above, but at reduced costs for *publisher* and *owner*.

4.4 Set Operations

All set operations involve two relations u and v. We may assume that u and v are intermediate results of a query evaluation, and are subsets of some relations r and s respectively, and that r and s are each sorted (possibly on different attributes) and have its own Merkle tree $MHT(r, \mathcal{A})$ and $MHT(s, \mathcal{A}')$, the root of which is signed as usual. We consider unions and intersections.

Union. In this case, the answer set is $q = u \cup v$. The *client* is given $\mathcal{VO}$s for u and v, along with $\mathcal{VO}$s for both; *client* verifies both $\mathcal{VO}$s, and computes $u \cup v$. This can be done in $O(u + v)$ using a hash merge. Since $\mid q \mid$ is $O(\mid u \mid + \mid v \mid)$, the overall $\mathcal{VO}$, and the time required to compute and verify the answer, are still linear in the size of the answer.

Intersection. The approach for union, however, does not produce compact $\mathcal{VO}$s for set intersection. Suppose $q = u \cap v$ where u and v are as before: note that often $\mid q \mid$ could be much smaller than $\mid u \mid + \mid v \mid$. Thus, sending the $\mathcal{VO}$s for u and v and letting the *client* compute the final result could be a lot of extra work. We would like a $\mathcal{VO}$ of size $O(\mid q \mid)$. If Merkle trees exist for u and v, we can do inclusiveness in $O(\mid q \mid)$: *publisher* can build a $\mathcal{VO}$ for q with $O(\mid q \mid)$ verification paths, showing elements of q belong to both u and v. Completeness is harder. One can pick the smaller set (say u) and for each element in $u - q$, construct a $\mathcal{VO}$ show that it is $\notin \ v$. In general, if u and v are intermediate results not occurring contiguously in the same Merkle tree, such a $\mathcal{VO}$ is linear in the size of the smaller set (say u). Consider for example a set of tuples $\langle \texttt{name}, \texttt{age}, \texttt{salary} \rangle$, where one wishes to select tuples in a specific salary and age range. Assume then that u has been obtained by performing a selection based on `salary`, and v based on `age`. u and v would be verified by $\mathcal{VO}$'s resulting from different Merkle hash trees: one sorted by `salary`, and one sorted by `age`. Computing the intersection $u \cap v$ would result in a $\mathcal{VO}$ with size linear in $\mid u \mid$: this $\mathcal{VO}$ would provide inclusiveness evidence (in u and v) for each element of $u \cap v$, and would show completeness by showing that each remaining element in $(u - (u \cap v))$ is not in v. This again leaves us with the unsatisfactory situation of a $\mathcal{VO}$ being linear in the size of a potentially much larger intermediate result (if $\mid u \mid \gg \mid u \cap v \mid$). A similar problem occurs with set differences $u - v$.

We have not solved the general problem of constructing $\mathcal{VO}$'s linear in the size of the result for intersections and set differences. Indeed, the question remains as to whether (in general) linear-size $\mathcal{VO}$s *can* even be constructed for these objects. However, we have developed an approach to constructing linear-size $\mathcal{VO}$s for a common type of intersection, *range query*, such as the $\langle \texttt{name}, \texttt{age}, \texttt{salary} \rangle$ range query discussed above. This is accomplished using a data structure drawn from computational geometry called a *multi-dimensional range tree.* This also supports set differences over range querys on several attributes.

In d-dimensional computational geometry, when one is dealing with sets of points in d-space, one could ask a d-space range query. Consider a spatial interval $(< x_1^1, x_2^1 > \ldots < x_1^d, x_2^d >)$: this represents a axis-aligned rectilinear solid in d-space. A query could ask for the points that occur within this solid. Such problems are solved efficiently in computational geometry using so-called *Range Trees* (See [10], Chapter 5). We draw an analogy between this problem and a database query of the form

$\sigma_{c_1^1 < A_1 < c_1^2}(r) \bigcap \ldots \bigcap \sigma_{c_d^1 < A_d < c_d^2}(r)$

where $\{A_1, \ldots A_d\} \subseteq schema(R)$ for a relation R. We use the multi-dimensional range tree ($mdrt$) data structure to solve such queries and provide compact verification objects. For brevity, we omit the full details of our approach. However, in [8], we show how to construct $\mathcal{VO}$s for "d"-way range queries such as the ones shown above. We also argue that the size of these $\mathcal{VO}$s, for a relation with size n, is $O(\mid q \mid d \log n + \log^d n)$. The $mdrt$ itself uses $O(n \log^{d-1} n)$ space and can also be constructed in time $O(n \log^{d-1} n)$. While the data structure arises out of computational geometry, it can be used with any domain that has enough structure to admit a total order. Full details are availabe in [8]

5 Pragmatic Issues, Related Work and Future Research

We now examine some pragmatic considerations in using our approach, as well as related work and future research.

Query processing flexibility. What querys can be handled? A typical SQL "**select** ... **from** ... **where** ..." can be thought of one or more joins, followed by a (combined) selection, followed by a projection. A multi-dimensional range tree can be used for both efficient evaluation and construction of compact $\mathcal{VO}$s for such queries. Specifically, consider a query that involves the join of two relations R and S, followed by a series of selections and a final projection. Let's assume a Theta-join over attribute A_1 (A_1 occurring in both relations), followed by selections on attributes A_2 and A_3, and a final projection on several attributes, jointly represented by A_4 (as discussed in Section 4.2). Full details are deferred to [8]. However, to summarize briefly: such a query can be evaluated by constructing a multi-dimensional range tree, beginning with the join attributes, followed by trees for each selection attribute, and perhaps finishing with a tree for some selective projection attributes.

Conventions. It is important to note that all interested parties: the *owner*, the *publisher* and the *client*, share a consistent schema for the databases being published. In addition there needs to be secure binding between the schema, the queries and the query evaluation process over the constructed Merkle trees. A convention to include this information within the hash trees needs to be established. All parties also need to agree on the data structures used for the $\mathcal{VO}$. It is also important that the *publisher* and the *client* agree upon the format in which the $\mathcal{VO}$ together with the query result is encoded and transmitted. Tagged data streams such as XML provide an attractive option.

Recent Query Evaluations. Verifiers must verify that query evaluation is due to an "adequately recent" snapshot of the database and not an old version. We assume the technique of recent-secure authentication [20] for solving this problem. Risk takers (e.g. organizations relying on the accuracy of the data) specify freshness policies on how fresh the database must be. The digital signatures over the database include a timestamp of the last update as well as other versioning information. Clients interpret the timestamps and verify the database is adequately recent with respect to their freshness policies.

Query flexibility. For efficient verification of query answering, we make use of different trees over sorted tuple sets. Without such trees, our approach cannot provide small verification objects. This points to a limitation of our approach—only queries for which Merkle trees have been pre-computed can be evaluated with compact verification objects. Our approach cannot support arbitrary interactive

querying with compact verification objects. Arbitrary interactive querying, however, is *quite rare* in the presence of fixed applications at *client* sites.

In practice, however, data-intensive applications make use of a fixed set of queries. Indeed, via mechanisms such as embedded SQL (see, e.g., [19]) database queries are compiled into applications. These queries can still make use of parameters entered by a user and which are typically used in selection conditions. Our approach is compatible with such applications. Essentially, client applications commit *a priori* the queries they wish to execute; the owner and the publisher then pre-compute the required Merkle hash trees to produce short verification objects.

So while our approach cannot provide compact verification objects in the context of arbitrary interactive database querying, it is quite compatible with the widely-used practice of embedding predetermined (and parameterizable) queries within data-intensive applications. In case there is a demand for a great many different projections, then we can only use Merkle trees for the most selective attributes, and in other cases let the *client* do the projection himself.

Costs and Trade-offs. If the *owner* and *publisher* can use the same data structure for processing queries and for computing $\mathcal{VO}$s, the additional effort of providing verification is greatly reduced. The Merkle hash values can be computed while building the B-tree index data structures, and can be updated while updating the indices. These computations can be reproduced at both the *owner* and *publisher* sites. Likewise, the $\mathcal{VO}$s objects can be easily extracted at minor additional cost during normal query-processing.

Some (not all) of our approaches to Joins (materializing cross-products or full outer joins) are expensive; however this cost is amortized over more efficient query processing, and more compact $\mathcal{VO}$s. In addition, this approach is similar to quadratic physical structures (e.g., clustered indices) currently used in databases and warehousing applications, and offers similar performance benefits. In addition, although the construction of multi-dimensional range trees is expensive, it leads to more efficient computation of (and smaller $\mathcal{VO}$s for) range-query type of set-intersections.

Trade-offs between query performance and storage costs are common. We introduce a new factor, the cost of computing and verifying $\mathcal{VO}$s; this cost, fairly modest in many cases, comes with the *benefit* of additional security, *i.e.,*, the elimination of the need to trust the database publisher. Further research will provide other possible implementation and design choices.

Future Work. To our knowledge, this is a new perspective on authentic data publication; many issues await exploration. We present a short list here: a) Possible lower bounds on the size of verification objects for different operations b) more efficient and/or secure underlying physical data structures c) Exact security analyses [2] of the precise costs of attacking our approach d) generalization to non-relational data models (*e.g.,* XML, OODB, etc.) e) approaches to supporting re-engineering of existing databases for authentic re-publication f) Other protocol models, e.g., perhaps with partially trusted publishers, or quorums.

Related Work. The use of Merkle hash trees for authentication of data is not new. This work is most closely related to the work of Naor & Nissim [15] for revocation. Haber and Stronetta [9] use similar techniques for timestamping. Similar schemes [18] have also been used for micropayments. All these schemes (including ours) share a common theme of leveraging the trust provided by a few digital signatures from a trusted party over multiple hashes, hash paths or hash trees, with the goal of protecting the integrity of the content, with efficient verification, since hashes are more efficient than digital signatures. However, the use of such trees for authentic data publication is new.

There is quite bit of related work in the general area of database security, particularly on access control, statistical querying etc [5, 12]. Anderson [3] discusses an approach to third-party publication of data in $files$, but without querying over the contents. Again, to our knowledge, this particular problem of authentic database publication has remained unexamined.

Finally, our approach can be viewed as providing "proof-carrying" [16] answers to database queries; however, unlike [16] our $\mathcal{VO}$s are based on secure hashing, rather than on formal logic.

6 Conclusion

We have explored the problem of authentic third party data publication. In particular, we have developed several techniques that allow untrusted third parties to provide evidence of *inclusive* and *complete* query evaluation to clients without using public-key signatures. In addition, the evidence provided is linear in the size of the query answers, and can be checked in linear time. Our techniques do involve the construction of complex data structures, but the cost of this construction is amortized over more efficient query evaluation, as well as the production of compact verification objects. Such pre-computation of views and indexing structures are not uncommon in data warehousing applications [17].

Our techniques suggest the use of a single hash function. In particularly high-integrity applications where tolerance of failure is very low, one can use multiple one-way hash functions to construct each Merkle tree. Clients requiring higher levels of integrity may check more than one hash computation.

However, our techniques are restricted currently to the relational model. Our techniques do not allow interactive querying, but can be used with embedded queries in applications. We cannot currently construct linear-size $\mathcal{VO}$s general SQL queries, such as ones including arbitrary intersections; we also leave open the (lower-bound) question as to whether such $\mathcal{VO}$s are possible. We believe, however, that our techniques are a start on an important problem area, and subsequent work will perhaps remove some of these limitations.

References

[1] N.M. Amato and M.C. Loui. Checking linked data structures. In *Proceedings of the 24th Annual International Symposium on Fault-Tolerant Computing (FTCS)*, 1994.

[2] Mihir Bellare. Practice-oriented provable security. In G. Davida E. Okamoto and M. Mambo, editors, *Proceedings of First International Workshop on Information Security (ISW 97)*, volume 1396 of *LNCS*. Springer Verlag, 1997.

[3] R. J. Anderson. The Eternity Service. *In Proceedings of Pragocrypt*, 1996.

[4] M. Blum, W. Evans, P. Gemmell, S. Kannan, and M. Noar. Checking the inclusiveness of memories. *Algorithmica*, 12(2/3):225–244, 1994. Originally appeared in *FOCS* 91.

[5] S. Castano, M. Fugini, G. Martella, P. Samarati Database Security Addison-Wesley, 1995

[6] C.J. Date. An Introduction to Database Systems (7th Ed), Addison-Wesley, 1999.

[7] C.J. Date and H. Darwen. A Guide to the SQL Standard (4th Ed), Addison Wesley, 1997.

[8] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic third-party data publication. (`http://seclab.cs.ucdavis.edu/~devanbu/authdbpub.pdf`), 1999.

[9] S. Haber and W. S. Stornetta. How to timestamp a digital document *J. of Cryptology*, 3(2), 1991.

[10] M. D. Berg , M. V. Kreveld, M. Overmars and O. Schwarzkopf. *Computational Geometry.* Springer-Verlag, New York.

[11] S. Jajodia, P. Samarati, V. S. Subramanian, E. Bertino, A Unified Framework for Enforcing Multiple Access Control Policies *Proceedings ACM SIGMOD*, 1997.

[12] T. Lunt, (Ed.) Research Directions in Database Security Springer-Verlag, 1992

[13] R.C. Merkle. A certified digital signature. In *Advances in Cryptology–Crypto '89*, 1989.

[14] J. Melton, A.R. Simon. Understanding the New SQL. Morgan Kaufmann, 1993.

[15] M. Naor, K. Nissim. Certificate Revocation and Certificate Update. *Proceedings, 7th USENIX Security Symposium*. 1998.

[16] G. Necula. Proof-carrying code. In *Proceedings of POPL 97*. ACM SIGPLAN, 1997.

[17] W.H. Inmon. Building the Data Warehouse John Wiley & Sons, 1996.

[18] S. Charanjit and M. Yung. Paytree: Amortized Signature for flexible micropayments *Second Usenix Workshop on Electronic Commerce Proceedings*, 1996

[19] A. Silberschatz, H. Korth, S. Sudarshan: Database System Concepts, McGraw-Hill, 1997.

[20] S. G. Stubblebine. Recent-secure authentication: Enforcing Revocations in distributed systems IEEE Computer Society Symposium on Security and Privacy, 1995.

[21] J. D. Tygar Open Problems In Electronic Commerce *Proceedings ACM SIGMOD PODS*, 1999.