Verified Query Results from Hybrid Authentication Trees

Glen Nuckolls

Department of Computer Sciences, University of Texas at Austin,
Austin, TX 78712-0233 USA
nuckolls@cs.utexas.edu
http://www.cs.utexas.edu/ nuckolls/

Abstract. We address the problem of verifying the accuracy of query results provided by an untrusted third party Publisher on behalf of a trusted data Owner. We propose a flexible database verification structure, the Hybrid Authentication Tree (HAT), based on fast cryptographic hashing and careful use of a more expensive *one-way accumulator*. This eliminates the dependence on tree height of earlier Merkle tree based proposals and improves on the VB tree, a recent proposal to reduce proof sizes, by eliminating a trust assumption and reliance on signatures. An evaluation of the Hybrid Authentication Tree against the VB tree and Authentic Publication showing that a HAT provides the smallest proofs and faster verification than the VB tree. With moderate bandwidth limitations, the HATs low proof overhead reduces transfer time to significantly outweigh the faster verification time of Authentic Publication. A HAT supports two verification modes that can vary *per query* and *per Client* to match Client resources and applications. This flexibility allows the HAT to match the best performance of both hash based and accumulator based methods.

1 Introduction

An increasing number and variety of applications and systems require access to data over a network. Third party architectures offer one way to address availability when the data source may have limited resources, by relying on a dedicated third party *Publisher* to provide responses. The related Edge Server model increases the availability of data services defined by a central server by replicating them at edge servers closer to clients. Maintaining data integrity along with high availability is a significant challenge. Providing database access introduces more complications. Typically, the number of different responses that can be generated from queries on a single data set is much larger than the data set itself. The integrity of query response involves showing that the response was correct, meaning the returned data was from the correct data set, and that the response is complete, meaning all matching data was returned.

We address the problem of verifying the accuracy of query results provided by an untrusted third party in the third party Publisher model. The *Owner* relies on the untrusted Publisher to process *Client* database queries. In the related *Authentic Publication* model [6], the Publisher is assumed to be untrusted, but the Owner is trusted, so Clients use a small digest value computed over the database by the Owner, to verify the query results. The same digest verifies many different queries on the same database. This allows the Owner to rely on, presumably cheaper, untrusted resources to handle Client requests. Authentic Publication [6] uses the organization of a tree digest, based on Merkle trees, to provide efficient verification for Clients.

Proof overhead can be significant when bandwidth is limited relative to processing power. In the original Authentic Publication method, proof size depends on the number of data points in the answer and the tree height, $O(\log N)$ additional hash values for a tree over N data points. Pang and Tan [17] proposed to eliminate this dependence on tree height with the verifiable B-tree (VB tree). This requires some degree of trust in the Publisher, relies heavily on Owner signatures and a proposed one-way digest function based on modular exponentiation. The trust assumption is problematic and the use of expensive primitives adds excessive overhead to both proof size and verification time.

In this paper we propose a flexible verification method, the Hybrid Authentication Tree (HAT), that eliminates the dependence on the tree height. Our novel method carefully incorporates fast hash functions with a more expensive cryptographic primitive, a *one-way accumulator.* The accumulator helps break the dependence on tree height without relying on Owner signatures and the lighter weight hash function speeds up the Client verification process. The design finds an efficient balance between the two primitives, using the heavier accumulators sparingly. Using the same one-time digest value, the HAT design allows the verification method to vary *per query* and *per Client* according to Client bandwidth and application requirements. One verification mode relies on both fast hash operations and the accumulator. The other mode bypasses the expensive accumulator operations, relying only on fast hash operations in a Merkle tree like method similar to Authentic Publication. In effect, we design a digest that gives us two verification schemes in one. Our analysis shows that, even for reasonable bandwidths of 1 mbps or more, the low proof overhead of the HAT improves significantly over the VB-tree and the original Authentic Publication method.

1.1 Outline of the Paper

Section 2 gives some useful background. Section 3 presents the details of the HAT construction and digest, explaining the motivation for using accumulator functions to break the dependence on tree height in Section 3.1, describing how completeness is verified in Section 3.2, and then defining the digest and verification in 3.3 and 3.4. We establish parameters for evaluation in Section 4 and give a detailed evaluation and comparison of the HAT with Authentic Publication and the VB-tree in Section 5, looking at proof overhead in 5.1, verification cost in 5.2, and bandwidth considerations in 5.3.

Section 6 describes how a HAT can bypass the use of the accumulator when advantageous. We discuss related research in 7, discuss future directions in 8 and conclude in 9.

2 Preliminary Building Blocks

We briefly describe the hash functions, one-way accumulators, Merkle trees and Authentic Publication.

2.1 Collision Intractable Hash Function

Hash functions such as MD5 or SHA-1 can be used to detect modifications in files by recomputing the MD5 or SHA-1 checksum on the file in question and comparing to the original checksum value. It is assumed to be difficult to produce another file with the same MD5 checksum or the original. This property is known as collision resistance or collision intractability. Though not formal in the cryptographic sense, we can reasonably rely on the following notion of collision resistance.

Definition 1. *A function f mapping the set of all binary strings to the set of strings of some fixed length is collision resistant if, given a "random" input value x, and the image $f(x)$, it is computationally infeasible for an adversary to compute $x' \neq x$ such that $f(x') = f(x)$.*

One informality evident is the reference to a random input. Since the size of the input is not specified, the domain cannot be uniformly sampled. The formal cryptographic definition of collision resistance handles this, and the function we rely on, SHA-1, is widely relied on for collision resistance.

When f is a hash function, we write $f(x_1, x_2)$ to denote the application of f to a single string constructed by concatenating x_1 and x_2 and some unique delimiter between them. The hash can apply to any number of strings with no ambiguity about the value and number of inputs.

2.2 Merkle Trees and Authentic Publication

We review the basic Merkle tree construction [13] since it is a common thread to many recent efforts in efficient query answer verification. The structure is a binary search tree, over a data set D of size N, with the key $\mathsf{key}(d)$ and data for each item $d \in D$ stored at the leaves. The key is simply a unique identifier for d e.g. the primary key for a relational tuple. Our example, shown on the left in Figure 2.2, uses a set of integer keys and ignores any associated data attributes. The tree is digested using a collision-resistant hash function h to produce a value $f(v)$ at each node v as follows: Starting at the leaves, the value of a leaf is its key value and the value of an internal node is the hash of the child values. Alternately we can hash the associated attributes with the key to produce the leaf value. The overall digest value of the tree, denoted Σ, is just the digest value at the root. With this digest value, an efficient proof, of size $O(\log N)$ can be given that a data item is or is not in the set. The proof consists of the intermediate hash value for each sibling of a node along the search path.

We follow the general Authentic Publication model [6]: a trusted data Owner relies on an untrusted third party Publisher to respond to Client queries on

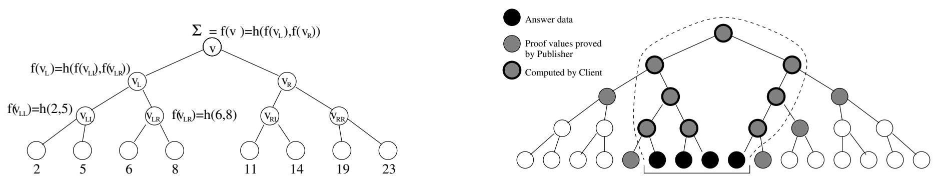

Fig. 1. Merkle hashing, left, and range query verification, right

a data set. First, a trusted Owner computes a digest of the data set (e.g. as in the binary search tree example). Next, the data is given to one or more untrusted Publishers and the digest is distributed to Clients. Publishers send additional proof values with each Client query response. Clients verify the answer by partially recomputing the root digest. Clients can send any number of queries on the data set to an untrusted Publisher and verify that the answers are the same as the Owner used to compute Σ. This approach scales well, with no security assumptions on Publishers.

2.3 An Efficient RSA Based One-Way Accumulator

Benaloh and de Mare [2] define an accumulator as a family of one-way, quasi-commutative hash functions. A hash function f is *one-way* if, given $x \in X$ and $y, y' \in Y$ it is hard to find $x' \in X$ such that $f(x,y) = f(x',y')$. The function f is *quasi-commutative* if

$$f(f(x,y_1),y_2) = f(f(x,y_2),y_1)\ \forall x \in X_k, y_i \in Y$$

We define $f(x,y_1,y_2,\ldots,y_N)$ to be $f(\ldots f(f(x,y_1),y_2)\ldots,y_N)$ for convenience. Given an initial value x, if $z = f(x,y_1,y_2,\ldots,y_N)$, the y_i values can be input into the accumulator in any order and still produce z. Now, given a value $y_i \in Y$, let z_i be the result of applying f to all the values in $Y - \{y_i\}$ with initial value x. Then $f(z_i,y_i) = z$ and, z_i serves as a proof that y_i was used to compute z.

The one-way property is weaker than strict collision resistance since the adversary can not choose y'. However, quasi-commutativity directly provides the means to break collision-resistance, and one-way is often sufficient since the values $y_i \in Y_k$ used as input to the accumulator are themselves often the result of a cryptographic collision resistant hash function. In fact, the domain set Y_k can be restricted to the result of a hash on some input. In order to forge a proof for a value that hashes to $\tilde{y}$ with respect to a collection $\{y_i\}_1^N$, an adversary would need to find an alternate proof $\tilde{z}_j$ for a value $\tilde{y}$ that can be changed, by choosing a different value to hash, but not chosen, since the hash output is unpredictable. We restrict input to accumulators to be the output of a collision resistant hash function.

Benaloh and de Mare propose a one-way quasi-commutative accumulator based on an RSA modulus and prove it's security in [2]. Given n we define H_n by $H_n(x,y) = x^y \mod n$. H_n is quasi-commutative by the laws of exponents: $(x^{y_1})^{y_2} = (x^{y_2})^{y_1}$. To ensure H_n is one-way, the modulus n is chosen to be a *rigid* integer, meaning $n = p \cdot q$ where p and q are safe primes of the same bit size. A prime p is *safe* if $p = 2p' + 1$ and p' is an odd prime. The factorization of n is considered trapdoor knowledge and in our model is known only by the Owner. Efficient methods for choosing rigid moduli are discussed in [2] and [12].

For a set $Y = \{y_1, \dots y_N\}$ and $z = H_n(x, y_1, \dots, y_N)$, $z_i = x^{y_1 y_2 \cdots y_{i-1} y_{i+1} \cdots y_N}$ serves as a proof for $y_i \in Y$. The value of x is also chosen by the Owner, but is not trapdoor knowledge. The Owner, knowing p, q, and thus $\phi(n) = (p-1)(q-1)$, can exponentiate by first reducing the exponent mod $\phi(n)$. This is an advantage we cannot give the Publisher and Client since $\phi(n)$ easily reveals p and q. However, the Publisher can still compute a proof by exponentiating.

3 The Hybrid Authentication Tree

We combine an accumulator function with Merkle hashing to providing proofs for answer correctness and completeness that are independent of tree height is fairly straightforward. However, since accumulator operations are so much more expensive than hashing, minimizing their use requires special consideration. We then describe how completeness is verified and then present the complete digest and verification processes.

3.1 Breaking the Dependence on Tree Height

A HAT is just a binary search tree with data stored at the leaves, along with a digest procedure that incorporates a fast collision intractable hash function h and an accumulator H. We want to take advantage of the input reordering allowed by an accumulator to avoid checking the entire hash path to the root as done when using Merkle trees as in Authentic Publication [6] and related schemes. However, accumulators have larger output, typically near the 1024 bits of an RSA modulus compared to the 160 bit output of SHA-1, and take longer to verify a value y_i against a proof z_i for a set value z. For example, suppose our range returned exactly one leaf w. Instead of the client verifying that w is the correct answer by hashing from w up to the root of the tree, the client could simply verify that w was included in a final accumulation value, requiring only a constant size proof. Answer completeness would need to be addressed, but could still be done with a constant size proof. This method breaks the dependence on the height of a Merkle tree but does not scale well to larger answer sizes.

Our approach uses Merkle hashing to certain nodes of the tree and then use the accumulator to verify the values of those nodes, eliminating the hashing along remaining path to the root. One natural set of nodes to consider for a range query is the set of *canonical covering roots* (CCRs) for the range in the tree as shown in Figure 2. The set of CCRs in a search tree for a range query is the set of nodes with disjoint subtrees whose leaves are the exact answer to the range query. For a range returning T leaves, is not hard to show that there are $O(\log T)$ CCRs and they have height $O(\log T)$.

The CCRs seem like good nodes to switch from Merkle hashing to accumulation. Their size and number depend only on the answer size so they break the

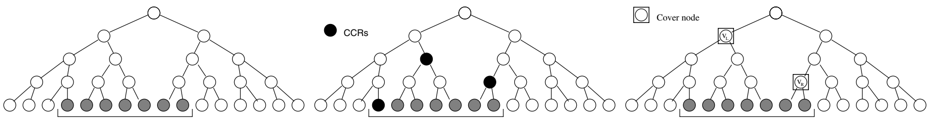

Fig. 2. The range, left, associated CCRs, center, and Covering Nodes v_L, v_R

dependence on the tree height. In fact, using CCRs provides smaller proofs than the scheme we present, but the number of accumulator computations required is still high. In order to further reduce the number of accumulator computations, we rely on the following.

Given a full binary search tree and range with T satisfying leaves, there are at most two nodes of height $O(\log T)$ whose leaves contain all the leaves in the range.

The pair with the smallest subtree is the *covering pair* for the range, or *covering node* in case there is only one. The verification switches from Merkle hashing to accumulator computations at the covering nodes (see Figure 2). This reduces the accumulator verification to one or two values.

3.2 Completeness and Covering Node Adjacency

The previous section only addressed verification that data is from the correct data set. We have not addressed how we can provide a proof that the answer is complete, containing all values within the range. The client will be able to verify that data matches the range, but not that all data matching the range has been returned.

First, note that Merkle hashing does much of this implicitly. Nodes in the middle of the range can't be left out without changing the root hash value without breaking the hash function. Verifying that no leaves were left off either end is the only requirement. Authentic publication includes the next highest and next lowest leaves in order to prove range completeness. It is easy to avoid revealing the data not in the range by providing only the data key and a hash of the data itself. However, they key, or at least some part of it, must be revealed. One alternate approach includes the split values in the hash at each node. These are checked in the verification and ensure range completeness. This approach is described in [11]. The additional split values would introduce a significant amount of additional proof overhead and it is not clear that much privacy can be gained over the inclusion of boundary key values. Privacy with completeness is an important concern, but is left to future research.

Given that we stop hashing at two covering nodes, and that the quasi-commutative property of accumulators makes completeness harder, we have two tasks. Prove that the ends are complete, and prove that the leaves of the two covering nodes form a continuous range in the tree. Two nodes are adjacent if their subtrees are disjoint and have adjacent leaves (see Figure 3). If we can show

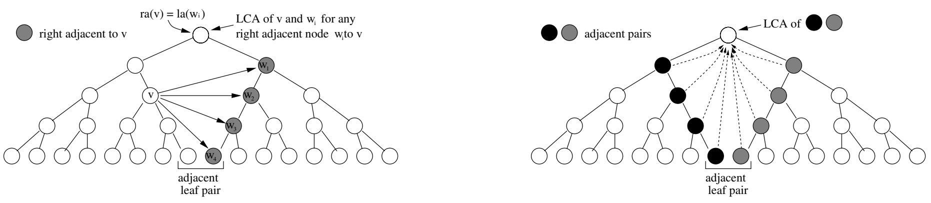

Fig. 3. Node v's right adjacent nodes and all adjacent pairs with same LCA

that our two covering nodes are adjacent, the Merkle hashing scheme used in each subtree will ensure that the range is complete. A fact about trees provides a simple and efficient way to show that any two adjacent nodes are, in fact, adjacent.

Looking at any tree, it is clear that all adjacent node pairs defined by the same adjacent leaf pair have the same least common ancestor, and that for any single node v all of the left adjacent nodes to v produce the same lca. The same holds for the right. We use $\mathsf{la}(v)$ and $\mathsf{ra}(v)$ to denote this single lca of v with all left and right adjacent nodes respectively (see Figure 3). The fact that $\mathsf{ra}(v) = \mathsf{la}(w_i)$ serves as a compact proof that v and w_i are adjacent in the tree. These left and right adjacency values are hashed in with the f_1 value at each node after the standard Merkle hash and are given to Clients to use in verification. After this, the resulting f_2 values at each node (see Section 3.3) will then be used to compute a single value using the accumulator. The final digest value is the hash of the root value of the Merkle hash and the accumulated value.

The Owner could achieve more or less the same effect as using the lca adjacency hash value by assigning some random value instead to serve as this proof, but this has a number of potential disadvantages. In particular, it prevents the Publishers from computing the digest on their own from the data and knowledge of the general digesting scheme. The hash of this lca node is computed directly from the structure and data set just as the root digest. Requiring that these random number be sent for each adjacent leaf pair adds unnecessary complication.

3.3 Digest

For each data item $d \in D$, $\mathsf{key}(d)$ and the hash output $h(d)$ are both computed in a way known to all parties from the attributes of d. Each leaf node of the binary search tree corresponds to some $d \in D$ and they appear in the tree sorted by $\mathsf{key}()$ value. Internal nodes v have left $\mathsf{lc}(v)$ and right $\mathsf{lc}(v)$ child fields, and every node has a left $\mathsf{la}(v)$ and right $\mathsf{ra}(v)$ adjacent lca node field as defined in section 3.2. If no such node exists, the field is assigned some fixed value indicating the left or right end. The digest uses a publicly known collision resistant hash function h and accumulator H to compute the final digest value Σ that the Owner provides to Clients.

1. f_1 is a standard Merkle hash using h.

$$f_1(v) = \begin{cases} h(f_1(\mathsf{lc}(v))), f_1(\mathsf{rc}(v))) & v \textit{ internal} \\ h(\mathsf{key}(d), h(d)) & v \text{ is a } \textit{leaf} \text{ with associated data } d \end{cases}$$

2. f_2 incorporates the hash values from adjacency lca nodes.
$$f_2(v) = h(f_1(v), f_1(\mathsf{la}(v)), f_1(\mathsf{ra}(v)))$$
3. A value A is computed using the accumulator H with some initial value x. If $v_1, v_2, \dots v_m$ are thee nodes of the tree,
$$A = H(x, f_2(v_1), f_2(v_2), \dots, f_2(y_m))$$
H is quasi-commutative so the node order will not affect the result.
4. Let *root* denote the root of the tree. The final digest value Σ is computed as
$$\Sigma = h(A, f_2(root))$$

3.4 Proof and Client Verification

The Client submits a range query $[a, b]$ that returns T satisfying leaf nodes. We assume that the Client has $\mathsf{key}(d)$ and $h(d)$ for data item d associated with each returned leaf v and the two boundary leaves. The Client can compute $\mathsf{key}(d)$ and $h(d)$ from the answer data except for the boundary leaves. We assume there are exactly two covering nodes denoted v_L and v_R. For a node v, z_v denotes the proof that $f_2(v)$ was used to compute the value A using accumulator H as described in Section 2.3. Verification proceeds as follows:

1. Compute $f_1(v_L)$ and $f_1(v_R)$
In addition to the $\mathsf{key}(d)$ and $h(d)$ values the Client already has, the boundary values are required and the $2 \log T$ supporting hash values for each of the covering nodes as in the Authentic Publication method.
2. Compute $f_2(v_L)$ and $f_2(v_R)$ values and check that v_L and v_R are adjacent.
Check that $f_1(\mathsf{ra}(v_L)) = f_1(\mathsf{la}(v_R))$, and thus v_L and v_R are adjacent.
Requires $f_1(\mathsf{ra}(v_L)) = f_1(\mathsf{la}(v_R))$, $f_1(\mathsf{la}(v_L))$ and $f_1(\mathsf{ra}(v_R))$
3. Check that $H(z_{v_L}, f_2(v_L))$, $H(z_{v_R}, f_2(v_R))$ both equal A.
Requires accumulator proofs z_{v_L} and z_{v_R} and value A.
4. Check that $h(f_2(root), A) = \Sigma$. Requires $f_2(root)$.

4 Proof Size and Verification Cost

We derive expressions for the proof overhead and verification in this section. In Section 5 we instantiate the hash and accumulator functions and give a detailed analysis of realistic overhead and cost values in order to compare our protocol with related proposals.

We assume that any node of the tree with M leaves has height at most $2 \log M$. We also assume that the key size is negligible. The expression $\mathsf{proof}_{\mathsf{HAT}}$ is in bits and and $\mathsf{verify}_{\mathsf{HAT}}$ is in bit operation per second. They are derived from Section 3 and the balance assumption.

$$\begin{aligned} \mathsf{proof}_{\mathsf{HAT}} &= (4 \log T) S_{hash} + 2 S_{acc} \\ \mathsf{verify}_{\mathsf{HAT}} &= (T + 2 \log T) T_{hash} + 2 T_{acc}(160) \end{aligned}$$

Hashing to the cover nodes takes at most $2T$ hashes. We assume here that the few value comparisons involved are not significant.

Table 1. Parameters used in performance analysis

Proof Overhead Parameters

N	Number of elements in data set.
T	Number of data elements in query range.
h	Height of the tree.
S_{hash}	Bit size of the hash output, SHA-1 is 160.
S_{acc}	Bit size of accumulator output. We use the recommended 1024 bit modulus for RSA here and for VB tree signatures.

Verification Cost Parameters in bit operations

T_{hash}	Compute hash output from one input block.
$T_{acc}(b)$	Verifying accumulator proof for b bit value.

Bandwidth and Processing Parameters

t	Bandwidth in bits per second.
p	Processing in bit ops per second.
T_{total}	Total time: transfer + verify.
T_{proof}	Time: transfer proof + verify.

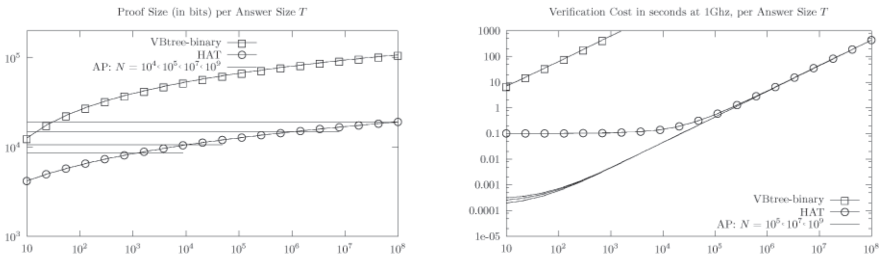

Fig. 4. Proof sizes (bits) and verification cost (seconds) calculated at 64×10^9 bit ops per second, for varying answer sizes T, both on log-log scale. Authentic Publication depends on N for proofs, N and T for verification, so we plot for several values of N

5 Performance Comparison

We compare three different range query verification methods, Authentic Publication proposed by Devanbu, et al. in [6], Pang and Tan's verifiable B-tree (VB tree) in [17] and the proposals in this paper. We focus on Client proof overhead and verification cost and then consider the effect of bandwidth.

5.1 Proof Overhead

We give expressions for the proof size in terms of bits for each scheme. A detailed derivation of the expressions is omitted due to space considerations.

$$\begin{aligned} \mathsf{proof}_{\mathsf{HAT}} &= 640 \log T + 2048 \\ \mathsf{proof}_{\mathsf{AP}} &= 640 \cdot \log N \\ \mathsf{proof}_{\mathsf{VBtree}} &= 4096 \cdot \log T - 1024 \end{aligned}$$

The proof size for the HAT was given in equation 4. We apply the parameters chosen in the previous section. The proof overhead in the Authentic Publication method consists only of supporting hash values. It is possible that split values are also used at each node, but this adds to the proof size and key values at the leaves are sufficient to give a proof that no values were omitted.

We take our VB tree analysis from [17] and use a branching factor of 2 since larger factors add significantly to proof size. We conservatively ignore overhead from projections. Projections with VB trees do not handle duplicate elimination and the other schemes produce smaller proofs in this case.

Figure 4 shows results for returned answer set sizes up to 10^8 and data set sizes up to 10^9 for Authentic Publication. We assumed a tree balanced within a factor of two at each subtree. The HAT has smallest overhead of any of the schemes and is close to AP only when T is near N. HATs have roughly 3 to 4 times smaller proofs for small answer sizes.

5.2 Verification Cost

We use values derived from a detailed analysis, omitted here, and round conservatively. As before, these expressions were derived from a detailed analysis omitted due to space considerations.

$$\begin{aligned}
\mathsf{verify}_{\mathsf{HAT}} &= 28,500 \cdot T + 6.5 \times 10^8 \\
\mathsf{verify}_{\mathsf{AP}} &= 57,000 \cdot \log N + 28,500 \cdot T \\
\mathsf{verify}_{\mathsf{VBtree}} &= 4 \times 10^9 \cdot T + 10^{10} \cdot \log T - 4 \times 10^9
\end{aligned}$$

We assume a 1024 bit modulus for the one-way hash used in the VB tree and the same modulus for the Owner RSA signatures. We also choose a low 16 bit public exponent for efficiency. As before, a binary branching factor is the most efficient instantiation of a VB tree, and projection does not improve the comparison. For Authentic Publication, Clients simply hash to the root using, for our comparison, SHA-1. Verification depends on N in this case.

The Authentic Publication verification is only has faster than a HAT for answer sizes up to 10^5. The VB tree is the slowest and is nearly infeasible for answer sizes more than 100,000. For answers of size less than 10,000, a HAT remains roughly constant, dominated by the modular exponentiation on the two 160 bit hash values to check against the accumulator proof value. Only one exponentiation may be required. Clearly our method is more practical than the VB tree. For answer sizes above 10^5, a HAT is as efficient as the Authentic Publication, yet has smaller proof size.

5.3 Bandwidth Versus CPU

When bandwidth is limited in comparison to processing resources, proof overhead is a larger factor. We compare the total time to transfer and verify proofs with and without full data transfer for HATs and Authentic Publication. Both comparisons are of interest since the Client may obtain data and proof from different sources or channels. The Client may also want a proof without needing all data sent. For an expired result similar to the current version, transmitting changes may be much smaller than the data set.

However, we still compare transmission including data as well. We assume that the data size per item is small, only 1024 bits, or 128 bytes. Clearly for much larger data sizes, this transmission cost will dominate, but it is useful to consider smaller, but reasonable sizes.

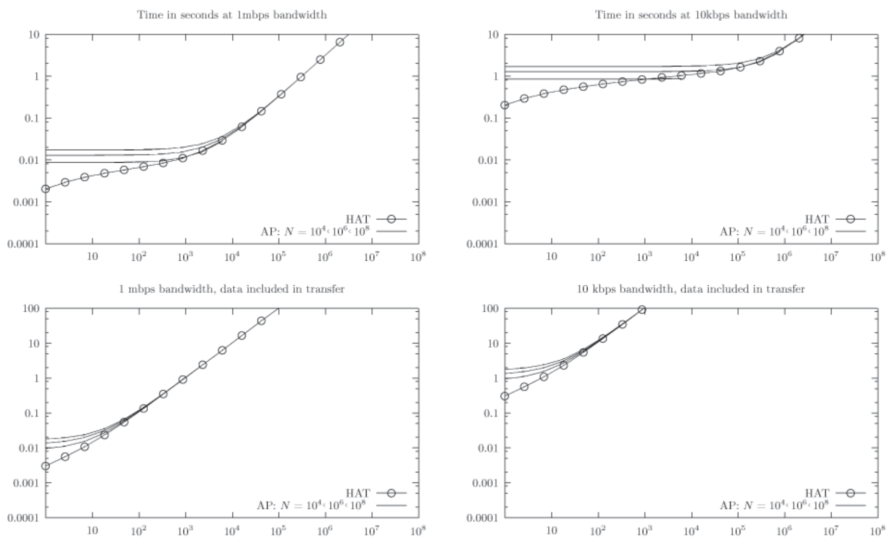

Fig. 5. Transmission time in seconds vs number T of items in answer, assuming a 1 gigahertz 64 bit processor at 64×10^9 bit ops per second

$$T_{total} = \frac{1024 \cdot T}{t} + \frac{\mathsf{proof}}{t} + \frac{\mathsf{verify}}{p}$$

T_{proof} excludes the first term of T_{total}. We keep the 1 GHz 64 bit processor assumption since varying bandwidth will have largest affect and varying both parameters adds too much complexity. Figure 5 shows total transfer and verification time for proof only and data included for average bandwidths to the Publisher of 1 mbps and 10 kbps. For small answer sets, the HAT takes between 3 and 10 times less time depending on the size N and if the data is included. The improvement is similar for both bandwidth values, but a little better as expected for the lower value. The improvement is sustained up to answer sizes of 10^4 for 1 mbps and over 10^5 for 10 kbps, for proof transmission only, and for answer sizes of less than 100 when answer data is included. Clearly, when bandwidth is very high, much higher than 1 mbps, Authentic Publication's verification time will win for smaller answers, but for reasonable bandwidth assumptions, low proof overhead shows significant improvement for smaller answer sizes.

6 Flexible per Query Tradeoffs: A Hybrid Tree

The HAT digest structure supports a per query, per client choice of two verification modes, both equally compatible with a single computation of the digest value. We described the accumulator mode in Section 3.4 and evaluated it in later sections. The hash only mode bypasses the accumulator checks and simply hashes to the root as in the Authentic Publication scheme, treating the covering nodes the same as any other node. The only difference is that the Client hashes in the accumulated value A with the root f_2 value at the end. This option is available *per query* and effectively produces a hybrid scheme able to bypass the accumulator step if advantageous.

The analysis and comparisons of Section 5 demonstrate one advantage of this flexibility. It also allows Publishers responding to a heterogeneous Client set to choose and adapt a verification strategy tailored to the Client's (possibly changing) resources, bandwidth, and application requirements.

7 Related Work

As discussed in Section 2.2 the proposed techniques are based on the original work by Merkle [13]. Naor and Nissim [15] made refinements in the context of certificate revocation. Goodrich and Tamassia [7] and Anagnostopoulos et al. [1] developed authenticated dictionaries. Authentic Publication was introduced in [6] where they showed how to securely answer some types of relational database queries. Devanbu, et al. described authentication for queries on XML document collections in [5]. Bertino, et al. focus in detail on XML documents in [3] and leverage access control mechanisms as a means of providing client proofs of completeness. They also expand supported query types and provide a detailed investigation of XML authentication issues including updates, and address important implementation issues.

Martel, et al. [11] establish a general model, Search DAGs, and provide security proofs for authenticating a broad class of data structures. Goodrich et al. [8] show that a broad class of geometric data structures fit the general model in [11] and thus have efficient authenticated versions. Similar Merkle hash tree based techniques have been used by Maniatis and Baker [10] and most recently Li, et al. [9] for secure storage. Buldas, et al. [4] describe methods that support certificate attestation when there is no trusted Owner. Nuckolls, et al. show how to extend the techniques in [6] to a distributed setting, allowing a collection of Owners to rely on an untrusted Publisher or other untrusted resources to achieve the same effect as a single trusted Owner while distributing the costs among the trusted parties.

Pang and Tang [17] proposed the VB tree as a way of eliminating dependence of proof size on the tree height, but require some trust of the Publisher since they only ensure that answer data is a subset of the original set. They also incorporate more expensive signatures and a proposed one-way function based on modular exponentiation. We discuss the VB tree in our performance analysis section and compare it to our proposed method.

Proving that values are not excluded in set membership queries, Micali, et al. [14] have shown how to construct zero-knowledge databases that preserve privacy, in addition to authentication. Ostrovsky, et al. [18] tackle the same problem for multi-attribute queries using a multidimensional range tree (MDRT).

Accumulators were proposed by Benaloh and de Mare [2] more constructions followed in [16]. Many of the RSA based accumulators contain a trap door and a proposal for accumulators without trapdoors has been given by Sander in [19].

8 Future Directions

We'd like to extend the methods here to a wider range of structures. So far, no systematic approach has been suggested for extending query verification techniques to the relational model. Devanbu et al. [6] discuss projection and joins, Martel et al. [11] present a solution for multidimensional range trees that support multi-attribute queries, and [17] suggest an approach to projections, but do not address the issue of duplicate values. There are many remaining issues with integrity on all types of relational queries.

The HAT relies on an accumulator with trapdoor information that must be kept secret by the Owner. Accumulators without trapdoors exist [19] but are not efficient enough for our purposes. An efficient quasi-commutative accumulator without a trapdoor would speed up our verification time.

Efficiently eliminating the dependence on the tree size lends increased flexibility to the verification process. This improves support for different types of Client applications and is worth exploring in the short term. One example is supporting Client verification of data in an order appropriate to the end application. The proposal in this paper easily adapts to provide priority verification for subranges of a query and then subsequently, if required, full query verification, and should also support incremental verification for small changes to previous requests.

We are also hopeful about the prospect of incorporating privacy in efficient database query authentication. Another promising goal is to bridge the differences between the work started in [14,18] and the current state of the more efficient authentication only methods, including the proposals of this paper. In addition to privacy, we hope to see progress on the longer term goal of ensuring that databases operated entirely by an untrusted and possibly distributed third party, satisfy arbitrary and general security requirements.

9 Conclusion

We carefully evaluated three methods: 1) fast hashing, specifically Authentic Publication over a binary search tree, 2) our hybrid authentication tree in accumulator mode, and 3) the VB tree. We show that both Authentic Publication and our accumulator based method significantly out perform the VB tree. We also show that the HAT always produces a smaller proof size than Authentic Publication and improves by a factor of 3 or 4 for smaller answer sizes.

HT Proof size is also smaller for answers significantly smaller than the database size. Authentic publication, as expected, has more efficient verification, but when bandwidth and transmission time is considered, the smaller HAT proofs provide up to 10 times faster overall verification time than the Authentic Publication method for smaller answer sizes. Even assuming infinite bandwidth, for answer sizes around 10^5 or higher, the HAT matches the performance of fast hashing in the Authentic Publication method. Our scheme is also competitive when considering storage overhead, digest computation, proof retrieval/generation, and updates.

For high bandwidths, Authentic Publication will provide faster verification, but our analysis shows there are significant benefits from a hybrid scheme that takes the best of the fast hashing and accumulator methods. Good tradeoff values can be analytically determined, changed on the fly *per query* with changing network conditions and client preferences.

Our analysis assumes a nearly balanced tree. Skewed trees could make our method even more attractive since hash trees performance depends on the length of paths to the root. Many applications and specialized data structures, such as XML document structures, are often not balanced. A HAT may provide flexibility in the data structure choice, reducing or eliminating the effect of unbalanced trees on Client costs.

Acknowledgments

The author thanks Mike Dahlin and Felipe Voloch for helpful discussions and the valuable comments of the anonymous reviewers.

References

1. A. Anagnostopoulos, M. T. Goodrich, and R. Tamassia. Persistent authenticated dictionaries and their applications. *Lecture Notes in Computer Science*, 2200:379, 2001.
2. J. Benaloh and M. de Mare. One-way accumulators: A decentralized alternative to digital signatures. In *Advances in Cryptology - EUROCRYPT '93*, number 765 in LNCS, pages 274–285, 1994.
3. E. Bertino, B. Carminati, E. Ferrari, B. Thuraisingham, and A. Gupta. Selective and authentic third-party distribution of xml documents. *IEEE Transactions on Knowledge and Data Engineering*, 16(10):1263–1278, 2004.
4. A. Buldas, P. Laud, and H. Lipmaa. Eliminating counterevidence with applications to accountable certificate management. *Journal of Computer Security*, 10:273–296, 2002.
5. P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. G. Stubblebine. Flexible authentication of xml documents. In *Proceedings of the 8th ACM Conference on Computer and Communications Security (CCS-8)*, pages 136–145, 2001.
6. P. T. Devanbu, M. Gertz, C. U. Martel, and S. G. Stubblebine. Authentic publication over the internet. *Journal of Computer Security*, 3(11):291–314, 2003.
7. M. Goodrich, R. Tamassia, and A. Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. *DISCEX II*, 2001.
8. M. T. Goodrich, R. Tamassia, N. Triandopoulos, and R. Cohen. Authenticated data structures for graph and geometric searching. In *Topics in Cryptology - CT-RSA 2003, Proceedings of the Cryptographers' Track at the RSA Conference*, volume 2612 of *LNCS*, pages 295–313, 2003.
9. J. Li, M. N. Krohn, D. Mazires, and D. Shasha. Secure untrusted data repository (sundr). In *Proceedings of the 6th Symposium on Operating Systems Design and Implementation*, pages 91–106, 2004.
10. P. Maniatis and M. Baker. Enabling the archival storage of signed documents. In *Proceedings of the USENIX Conference on File and Storage Technologies (FAST 2002)*, pages 31–45, Monterey, CA, USA, Jan. 2002. USENIX Association.

11. C. U. Martel, G. Nuckolls, P. T. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.
12. A. J. Menenzes, P. C. van Oorschot, and S. A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.
13. R. Merkle. Protocols for public key cryptosystems. In *Proceedings of the IEEE Symposium on Security and Privacy*, pages 122–134. IEEE Computer Society Press, 1980.
14. S. Micali, M. O. Rabin, and J. Kilian. Zero-knowledge sets. In *Proceedings of the 44th Symposium on Foundations of Computer Science (FOCS 2003)*, pages 80–91, 2003.
15. M. Naor and K. Nissim. Certificate revocation and certificate update. *IEEE Journal on Selected Areas in Communications*, 18(4):561–570, 2000.
16. K. Nyberg. Fast accumulated hashing. In *Proceedings of the Third Fast Software Encryption Workshop*, number 1039 in LNCS, pages 83–87, 1996.
17. H. Pang and K.-L. Tan. Authenticating query results in edge computing. In *Proceedings of the 20th International Conference on Data Engineering (ICDE'04)*, 2004.
18. A. S. Rafail Ostrovsky, Charles Rackoff. Efficient consistency proofs for generalized queries on a committed database. In *Proceedings of the 31st International Colloquium on Automata, Languages and Programming (ICALP 2004)*, volume 3142 of *LNCS*, pages 1041–1053, 2004.
19. T. Sander. Efficient accumulators without trapdoor. In *Proceedings of the Second International Conference on Information and Communication Security - ICICS'99*, number 1726 in LNCS, pages 252–262, 1999.