Scalable Verification for Outsourced Dynamic Databases

HweeHwa Pang Jilian Zhang Kyriakos Mouratidis
School of Information Systems
Singapore Management University
{hhpang, jilian.z.2007, kyriakos}@smu.edu.sg

ABSTRACT

Query answers from servers operated by third parties need to be verified, as the third parties may not be trusted or their servers may be compromised. Most of the existing authentication methods construct validity proofs based on the Merkle hash tree (MHT). The MHT, however, imposes severe concurrency constraints that slow down data updates. We introduce a protocol, built upon signature aggregation, for checking the authenticity, completeness and freshness of query answers. The protocol offers the important property of allowing new data to be disseminated *immediately*, while ensuring that outdated values beyond a pre-set age can be detected. We also propose an efficient verification technique for ad-hoc equijoins, for which no practical solution existed. In addition, for servers that need to process heavy query workloads, we introduce a mechanism that significantly reduces the proof construction time by caching just a small number of strategically chosen aggregate signatures. The efficiency and efficacy of our proposed mechanisms are confirmed through extensive experiments.

1. INTRODUCTION

Consider an online trading platform, fashioned after the outsourced database model, that involves three types of entities. The data aggregator (DA) disseminates live feeds from various stock, forex, commodity and mercantile exchanges through query servers (QS), possibly dispersed geographically, to a large population of users. The users trust the DA, who is responsible for the accurate and timely dissemination of information. However, the QSs that process user queries may be operated by an untrusted party, or could be infiltrated over time.

In the above scenario, there are three aspects to the *correctness* of a query answer: (a) *Authenticity*: Every value in the answer must originate from the DA. (b) *Completeness*: Every record that satisfies the query condition must be in the answer. (c) *Freshness*: The record values in the answer must be up-to-date. There is often a trade-off between correctness versus resource consumption. In some applications, it may be acceptable to compromise on the freshness requirement for the sake of computational resource preservation. Our focus is on applications that need strict correctness guarantees – in our online trading example, users who can receive updated price quotes early could act ahead of their competitors, so the advantage of data freshness would justify investing in the computing infrastructure.

To enable the user to verify his query answer, the DA constructs an authentication structure, and uploads it along with the data to the QS. Given a query, the QS returns to the user (in addition to the answer) a proof constructed from the authentication structure. Assuming that the cryptographic techniques employed in the authentication structure are secure, the degree to which the correctness requirements are satisfied hinges on the time needed for the DA to update the data and the authentication structure, for the QS to construct the proof, as well as for the user to verify the answer against the proof.

The prevalent method for authenticating query answers is to embed a Merkle hash tree (MHT) [21] into a data index. This data index is typically a B$^+$-tree for one-dimensional data (e.g. [18], [28]); for multi-dimensional data, the R-tree and the KD-tree have both been considered (e.g. in [9]). In this paper, we show that MHT has limitations in coping with updates. First, the digest of a parent node is a function of its child digests, so every data modification must propagate from the leaf to the root of the index; this generates multiple I/Os if the index resides on disk. Second, to ensure consistency, each update transaction must lock the root node in exclusive mode, thus allowing no concurrent transactions to execute. These two limitations reduce the freshness of query answers.

Besides MHT, signature aggregation is another cryptographic technique for query answer authentication. Here, the DA produces one signature for every tuple in the database. Posed a query, the proof comprises the signatures of all the tuples in the answer, condensed into a single aggregate signature. The problem with this technique is that signature generation, proof construction and user verification are all much more computationally expensive than with MHT [18]. Nevertheless, signature aggregation offers an important advantage; since a record update affects only its own signature (and that of its immediate left/right neighbors in some schemes), multiple updates can be executed simultaneously as long as they do not affect the same signatures. This potentially permits the DA to push out fresh data and authentication structures more quickly.

The objective of our work is to devise a scalable query answer authentication mechanism for dynamic databases. To avoid the locking bottleneck of MHT, we decide to build on signature aggregation. This requires us to overcome two primary challenges. First, we need to ensure the freshness of the query answers. MHT schemes involve a single signature (for the MHT root) which can be easily revoked upon updates or periodically refreshed. Both options are not viable with signature aggregation where there are as many signatures as records. The second challenge is to mitigate the higher computation costs involved in signature aggregation, relative to MHT. Our contributions include (among others) solutions to these challenges:

- We propose a correctness verification protocol that allows new records to be disseminated *immediately*, while ensuring that outdated values beyond a pre-set age can be detected. This feature relies on periodic update summaries (published by the DA asynchronously to queries and updates) that allow users to verify the freshness of records that bear old signatures. Our scheme is scalable to large databases, as the size of the summaries is proportional to the number of updates in each period (and insensitive to the number of records in the database). This is the first signature aggregation protocol that provides freshness guarantees.
- We construct authentication mechanisms for the basic relational operators of selection, projection and equi-join. Our equi-join verification is the first practical scheme. Existing methods either require a pair of boundary values to prove every unmatched record (e.g. [24]), which leads to huge correctness proofs, or they materialize the join result (e.g. [12]) and are unsuitable for dynamic databases. Our scheme provides compact correctness proofs using certified Bloom filters [5] to verify records that have no matching counterparts in the other operand relation.
- In order to reduce the proof construction cost at the query server, we introduce a signature caching scheme, called $SigCache$, that selectively retains some aggregate signatures in memory. Experiments with various query distributions reveal that $SigCache$ is able to reduce the proof computation time significantly by caching only a small number of aggregate signatures.

The rest of this paper is organized as follows. The next section covers background on cryptographic primitives and related work. Section 3 presents our authentication mechanisms for standard relational operators. Our signature caching scheme is introduced in Section 4. Experiment results are reported in Section 5. Finally, Section 6 concludes the paper.

2. BACKGROUND

2.1 Cryptographic Primitives

Our authentication schemes as well as existing methods (covered in Section 2.2) build on the following cryptographic primitives.

One-Way Hash: A one-way hash function, denoted as $h(.)$, works in one direction; it is easy to compute the value $h(m)$ for a message m, but computationally infeasible to find a message m that hashes to a given $h(.)$ value. We refer to $h(m)$ as the hash or digest of m. A common hash function is SHA [31] with 160-bit digests. Over time, longer digests are expected to be used to compensate for the increasing computational power of the adversaries.

Cryptographic Signature: A cryptographic signature protocol is a tool for verifying the origin, authenticity and integrity of signed messages. The protocol involves a pair of public and private keys. Only the holder of the private key can use it to generate cryptographic signatures on messages. The corresponding public key is distributed openly, for anyone to verify a message against its signature. RSA [29] and ECC [6] are two standard signature algorithms. We refer to a cryptographic signature simply as signature.

Merkle Hash Tree (MHT): The Merkle hash tree is a method for collectively authenticating a set of messages [21]. Consider the example in Figure 1, where the owner of messages m_1, m_2, m_3, m_4 wishes to authenticate them. The MHT is built bottom-up, by first computing the leaf nodes N_i as the digests $h(m_i)$ of the messages, where $h(.)$ is a one-way hash function. The value of each internal node is derived from its two child nodes, e.g. $N_{1,2} = h(N_1|N_2)$, where $|$ denotes concatenation. Finally, the digest $N_{1,2,3,4}$ of the

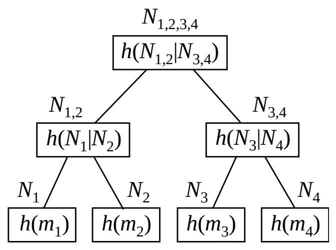


Figure 1: Example of a Merkle Hash Tree

root node is signed. The tree can be used to authenticate any subset of the data values, in conjunction with a verification object (VO). For example, to authenticate m_1, the VO contains $N_2, N_{3,4}$ and the signed root $N_{1,2,3,4}$. Upon receipt of m_1, any addressee may verify its authenticity by first computing $h(m_1)$ and then checking whether $h(h(h(m_1)|N_2)|N_{3,4})$ matches the signed root $N_{1,2,3,4}$. If so, m_1 is accepted; otherwise, m_1, N_2, $N_{3,4}$ and/or the signed root have been tampered with. The MHT is a binary tree, though it can be extended to multiway trees and directed acyclic graphs [20].

Bloom Filter: A Bloom filter [5] supports membership checks on a set of b key values $R = \{r_1, r_2, \ldots, r_b\}$. To construct a Bloom filter with m bits, we choose k independent hash functions h_1, h_2, ..., h_k, each with a range of $[1, m]$. For every $r_i \in R$, the filter bits at positions $h_1(r_i), h_2(r_i), \ldots, h_k(r_i)$ are set to 1. To check whether a given r is in R, we examine the bits at $h_1(r), h_2(r), \ldots, h_k(r)$. If any of the bits is 0, r cannot be in R; otherwise, there is a high probability that r is in R. In other words, false positives are possible, but not false negatives. The false positive rate is

$$FP = \left(1 - \left(1 - \frac{1}{m}\right)^{kb}\right)^k \approx \left(1 - e^{-kb/m}\right)^k \qquad (1)$$

Mathematically, FP is minimized at $k = (m \times ln2)/b$, so $FP = 0.6185^k$. Given the value of b and the target FP rate, we can set k and m accordingly.

Elliptic Curve Cryptography (ECC): The mathematical operations of ECC are defined over the elliptic curve $y^2 = x^3 + ax + \beta$, where $4a^3 + 2\beta^2 \neq 0$. Each combination of a and β produces a different elliptic curve. All points (x, y) that satisfy the above equation lie on the elliptic curve.

The elliptic curve over a finite operator group G_p [3] is specified by the tuple $\langle p, a, \beta, g, n\rangle$, where

- p is the order of (i.e., the number of elements in) G_p. SEC [8] specifies p to be a prime with 112 to 521 bits.
- a and β define the elliptic curve $y^2 \mod p = x^3 + ax + \beta \mod p$.
- $g = (g_x, g_y)$ is a point on the chosen elliptic curve and a generator for G_p.
- n is the order of (i.e., the number of points on) the elliptic curve.

The above specification is known to all parties who participate in the ECC protocol. [16] provides some examples of suitable elliptic curves. Multiple ECC signatures can be aggregated using the *Bilinear Aggregate Signature (BAS)* scheme [7, 6]. BAS enables any set of message-signature pairs to be combined in arbitrary order into a single signature, and then to be collectively verified.

According to [17], an ECC signature that is 160 bits long provides comparable security to a 1024-bit RSA [29] signature. Using shorter signatures enables ECC to achieve lower storage overhead. ECC has been adopted as a replacement for RSA public key cryptography by various government agencies in the US, UK, Canada and other NATO countries [32],[13]. In the industry, an ECC Interoperability Forum has been created to ensure that ECC products from participating vendors (including Certicom, IBM, Microsoft, RSA, Sun, Verisign, etc.) will integrate seamlessly. As part of the effort to promote widespread use of ECC, Sun Microsystems has donated ECC code to OpenSSL and the Network Security Services (NSS) library; this brings ECC to the Apache web server and Mozilla browsers, and potentially many other products.

2.2 Related Work

Most of the existing methods for query result verification fall under two categories – MHT-based and signature aggregation ones. The MHT-based approaches incorporate an MHT into the data index to facilitate verification (e.g. [12, 27, 4]). In [25], Nuckolls proposed a variation of the MHT that maintains a certified one-way accumulator over the digests of selected nodes; this allows a consolidated evidence to replace the neighboring digests along the path from those nodes to the root, thus reducing the size of the verification object (VO). That variation was extended to multiple hash tree levels in [15], where the authors also showed that replay attacks could be eliminated by periodically re-signing the timestamped accumulator. [22] proposed to split the authentication structure from the data index to provide architectural flexibility and better performance; this proposal is orthogonal to the issues studied here, and applies equally to MHT and signature aggregation schemes.

The most representative MHT scheme for disk-resident data is the Embedded Merkle B-tree (EMB$^-$ tree) in [18]. The idea is to index the data with a B$^+$-tree [11], and to embed into it an MHT with the same fanout. Similar to the original MHT, the root digest is signed by the owner. Posed a range query, the server returns, in addition to the qualifying tuples, two *boundary* ones, p^- and p^+, falling immediately to the left and to the right of the range. The VO contains all the left (right) sibling hashes to the path of p^- (p^+). Upon receipt of the result, the user calculates the hashes of the returned tuples, and combines them with the VO to reproduce the MHT root digest. If the latter matches the owner's signature, the result is deemed legitimate.

Signature aggregation schemes [26, 24] require a signature per tuple. With the database ordered on attribute A, the owner hashes and signs every triple of consecutive data tuples. Posed a range selection query on A, the server returns the qualifying data, along with hashes of the first tuple to the left and the first tuple to the right of the range. The signatures of all the result tuples are aggregated and placed into the VO. Finally, the user verifies that the "chained" result tuples and boundary hashes match the aggregate signature. This scheme, initially designed for one-dimensional data, was extended to multi-dimensional index structures in [9, 10].

A systematic comparison of MHT versus signature aggregation (using condensed RSA) was reported in [18]. The findings there overwhelmingly favored the MHT approach: (a) An RSA signature is typically 1024 bits in length, so signing all the data tuples requires substantially larger space than the MHT, in which each digest occupies just 160 bits. (b) A hashing operation in the MHT can be performed in roughly 3 μs. In contrast, signature aggregation involves modular multiplication, signing and verification operations that were 100, 10,000 and 1,000 times slower than hashing. The only advantage of signature aggregation is its smaller proof and, thus, its lower transmission overhead.

Nevertheless, in this paper we decide to revisit the signature aggregation approach, motivated by several factors:

- Newer signature schemes using Elliptic Curve Cryptography (ECC) allow much shorter signatures to be used without sacrificing security strength. As explained earlier, a 160-bit ECC signature provides comparable security to a 1024-bit RSA signature. Thus, an ECC signature has the same length as a hash digest.
- CPU speeds have improved tremendously. For example, the ECC performance timings in [6] and [24] were obtained on a 1GHz Intel Pentium 3 CPU. At present, a server can be equipped with a quad-core Xeon 3.4GHz processor at a reasonable price. As we show in Section 5, such processing power brings the timings of the ECC operations down to acceptable levels, even though they still lag significantly behind the hashing operations in MHT. In contrast, I/O and communication speeds (especially for wireless networks) have not improved by the same magnitude.
- More importantly, the single certified root digest (or accumulator) of the MHT reduces data freshness. If the MHT is recertified periodically, new data must be held back until the next MHT certification. On the other hand, if the MHT is eagerly renewed for individual updates, each update must contend for a lock on the root digest, thus incurring a locking delay. There is also a delay of $O(\log N)$ I/Os for updating the path from the affected leaf up to the root, where N is the number of database records. In addition, revoked signatures must be published by some trusted third party. This necessitates the revocation of an old signature to be synchronized with the availability of the replacement signature across all the query servers. It also requires users to check whether a received signature has been revoked. In contrast, signature aggregation is amenable to concurrent updates, which is critical for releasing fresh data to the users quickly.

3. AUTHENTICATION FOR RELATIONAL OPERATIONS

In this section, we begin with our protocol for checking data freshness. Next, we show how our authentication protocol supports the standard index structures and relational operators. We propose a novel scheme for equi-join verification that uses Bloom filters [5] to keep the proof size small. For completeness, we also briefly describe the verification of selection and projection operations.

3.1 Freshness Verification Protocol

Every ρ seconds, the data aggregator (DA) publishes a certified bitmap summary of the records that were updated (inserted, deleted, or modified) in the last period. In the signature of a record r, we include the timestamp ts that r was last certified at. Upon receipt of r at the user side, if its signature (i.e., ts) is older than the latest summary, its freshness can be confirmed by checking that r is excluded from all the summaries published since ts. If the signature is newer than the latest summary, r must be fresh. Therefore, whenever a record is updated, the DA can simply sign its content along with the update timestamp, and push the fresh record and signature out to the query server (QS) *immediately*.

We make an important decision here to decouple the dissemination of new records from the periodic release of bitmap summaries. Our rationale is that the QS is expected to operate reliably most of the time, so we ought to optimize information dissemination for normal operation, as long as any occasional lapses by the QS can be caught quickly. This decoupling allows data updates and summary generation to execute in parallel on separate processors, if available. It is also instrumental in avoiding the shortcomings of MHT described at the end of Section 2.2, and in providing tighter freshness guarantees. The details of our authentication protocol follow.

***Data Aggregator*:** Consider a relation R with schema $\langle rid, A_1, \ldots, A_M, ts\rangle$, where rid is the unique record identifier, A_i (for $1 \leq i \leq M$) are the attributes, and ts is the last record certification time. The signature of a record $r \in R$ is $r.sn = sign(h(r.rid \mid \ldots \mid r.ts))$, where $sign$ is a signature generation function and h is a one-way hash function. Whenever a record is updated, the new content and signature are transmitted *immediately* to the QS.

Every ρ seconds, the DA issues a certified bitmap summary of the records that have been updated. Each bit in the bitmap corresponds to a record in R, and is turned on if and only if the record has been modified in the current ρ-period. For inserted records, '1'-bits are appended to the bitmap. The bit corresponding to a deleted record is set to '1' in the current bitmap, and then to '0' in subsequent bitmaps. We expect the periods to be short, say one second in duration, so the bitmap is likely to be sparse and amenable to compression. The compressed bitmap is certified, along with the signing time, and released to the QS.

With compression techniques such as those in [14] and [30], the length of the compressed summary is only 2 to 3 times the number of '1'-bits in the original bitmap. Consequently, the size of the certified bitmaps in our scheme is proportional to the number of updates in a ρ-period, and insensitive to the database size.

***Query Server*:** Along with each query answer, the server returns the aggregate signature over the result records, as well as the certified summaries published after the oldest result record.

***User*:** The user verifies the authenticity of the result records by matching them with the aggregate signature. For checking whether each result record r is fresh, we consider the following cases:

- If r is newer than the latest bitmap b, i.e., $r.ts > b.ts$, r is either fresh or, at the very worst, out-of-date by $ct - r.ts < \rho$ seconds (where ct is the current time).
- Else, if $r.ts \leq b.ts$ and r is not marked in any of the bitmaps released since $r.ts$, then r is fresh or, at the very worst, out-of-date by $ct - b.ts < \rho$ seconds.

Multiple Updates to a Record within the Same ρ-Period: The above protocol is secure as long as any record is updated at most once within a ρ-period. If several versions of a record are released in some period t_i, however, the summary does not provide enough granularity to pinpoint the latest version among them. This shortcoming can be overcome by re-certifying the record in the subsequent period t_{i+1}, so that all the former versions are invalidated by the summary for t_{i+1}. With that, a record that is certified between the latest and the penultimate bitmaps could be up to 2ρ seconds out-of-date; the freshness in all other cases remains bounded by ρ.

Active Signature Renewal: Some records that change very infrequently may have old signatures that require many bitmaps to verify their freshness. To limit the verification overhead, the DA needs to refresh old signatures even if the associated records remain unchanged. When a record signature is refreshed, its associated bit is turned on in the bitmap summary for that period. Specifically:

- When a record r requires updating, the disk block containing r is fetched into the memory. The DA takes the opportunity to examine the other records in the disk block. Among them, those with a signature that is older than ρ' seconds are re-certified, and the new signatures are sent to the QS.
- In addition, a low-priority process utilizes idle resources at the DA to cycle through the records in R and refresh old signatures.

3.2 Indexing Structure

In the following we assume that the relational operations are facilitated by a disk-based B$^+$-tree. However, our techniques are applicable to other indexes and file organizations, including memory-based storage schemes. We first describe how to incorporate the signature aggregation technique into the B$^+$-tree, then compare the resulting index with the EMB$^-$ tree (presented in Section 2.2).

We exemplify our indexing approach in Figure 2. A B$^+$-tree

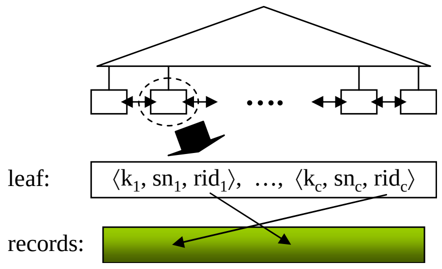


Figure 2: B$^+$-Tree with Authentication Information

N ($\times$1000)	10	100	1000	10000	100000
ASign	1	2	2	2	3
EMB$^-$ tree	2	2	3	3	4

Table 1: Height of Index Tree versus N

is built on the physical records, which are stored in an external file. The leaf nodes of the B$^+$-tree hold data entries of the form $\langle key, sn, rid\rangle$, representing the search key value, digital signature and identifier of the underlying record, respectively. We defer the discussion on what exactly sn is computed on, because it depends on the operations (queries) that we want to support. The internal nodes of the index have the same form (and functionality) as in a standard, non-authenticated B$^+$-tree.

To provide an intuition on the performance of our indexing approach, we compare it quantitatively with the EMB$^-$ tree. Assuming that the sizes of a key value, an ECC signature and a record identifier are 4 bytes, 20 bytes and 4 bytes, respectively, each data entry (i.e., each entry in the leaf level) in our method occupies 28 bytes. A 4-Kbyte page can therefore hold 146 data entries. This capacity is around 5 times larger than in the implementation of signature aggregation in [18], where equally secure but more voluminous (128-byte) RSA signatures were used.

Letting the size of a pointer be 4 bytes, the maximum fanout of each internal node is 512. Assuming an average utilization of 2/3, the effective fanout of each internal node is 341, and the height of the B$^+$-tree on a relation R of N records is $\lceil \log_{341} \frac{3}{2} \cdot \lceil \frac{N}{146} \rceil \rceil$.

Consider now the EMB$^-$ tree. Its leaf nodes store data entries of the form $\langle key, digest, rid\rangle$. Since each digest occupies 160 bits (i.e., an equivalent amount of space to an ECC signature), the EMB$^-$ tree has as many leaf nodes as our scheme. However, the internal nodes of the EMB$^-$ tree additionally store one digest per child entry. Thus, their effective fanout is only 97 (assuming again a 2/3 utilization), yielding an EMB$^-$ tree height of $\lceil \log_{97} \frac{3}{2} \cdot \lceil \frac{N}{146} \rceil \rceil$.

While both the EMB$^-$ tree and our scheme require changes to the B$^+$ tree code, the latter offers some performance advantages. Table 1 shows the tree height for different numbers of records N in our scheme (denoted by "ASign") and in the EMB$^-$ tree. The observed height difference translates to superior I/O performance for our technique, as we will see in Section 5. Another consideration is that the EMB$^-$ tree propagates every data update up to the root digest, so an update transaction must lock the entire index in exclusive mode and block all other updates and queries. In contrast, our scheme locks only the individual records (and their signatures) that are being updated; this allows transactions that operate on other records to proceed concurrently. Consequently, our scheme is expected to be much less susceptible to lock contention.

3.3 Selection

For a relation of records $\mathbf{R} = \{\langle rid, A_1, \ldots, A_M, ts\rangle\}$, a range selection operation produces $\sigma_C(\mathbf{R}) = \{r | r \in \mathbf{R}$ and $C(r)$ is true$\}$, where C is a constraint on the indexed attribute A_{ind} of $\mathbf{R}$.

To verify selection query answers, we simply apply the signature chaining technique of [26, 24]. The main idea is for each signature to "chain" the corresponding record to its immediate left and right neighbors in $\mathbf{R}$ in A_{ind} order. Specifically, the signature sn of a record $r \in \mathbf{R}$ is computed as $sign(h(r.rid \mid r.A_1 \mid \ldots \mid r.A_M \mid r.ts \mid r_{left}.A_{ind} \mid r_{right}.A_{ind}))$, where r_{left} and r_{right} are r's previous and next records in A_{ind} order, respectively. Posed a selection query, the server returns as VO the signatures of all result tuples, along with the index attribute value of the left and right boundary records. The user can confirm that the selection answer is authentic, because for each reported r, the corresponding signature sn is computed over r itself (among others). Completeness can be guaranteed if the boundary records enclose the selection range, and the records in the answer are contiguous as certified by the DA (meaning that no qualifying records between the boundaries are omitted). The VO size and therefore the communication cost are reduced by combining all the answer's signatures into a single aggregate signature, using the BAS scheme. Hence, the VO size pertaining to authenticity and completeness is independent of the query selectivity and minimal (since all existing schemes need to transmit at least one signature and two boundary values as proof).

3.4 Projection

Given a relation of records $\mathbf{R} = \{\langle rid, A_1, \ldots, A_M, ts\rangle\}$, a projection operation produces $\pi_{A_i,\ldots,A_j}(\mathbf{R}) = \{\langle rid, A_i, \ldots, A_j, ts\rangle\}$, where each of $A_i, \ldots, A_j$ is an attribute of $\mathbf{R}$.

To authenticate the output of a projection operation, one technique is to supply as part of the proof some compact proxy of the attribute values that are dropped from each record, such that the proxy combined with the returned attribute values will match the record signature. For example, [19] proposes to structure the attributes within each record in an MHT, whereas [24] concatenates the digest of the attribute values in each record to produce its signature. These techniques are computationally efficient, but are likely to require many digests in the VO, especially when the projected attributes are not contiguous in the schema. In particular, the VO size is in the order of $O(\log M)$ if record MHTs are used, and $O(M)$ if the attribute values within each record are concatenated.

We adopt the alternative of signing individual attribute values within each record, and setting the record signature to be the aggregation of its attribute signatures (this approach was also discussed in [24]). As signature aggregation is associative and commutative, we need to guard against any attempt by the server to swap attribute values between records. This can be achieved easily by making the signature of an attribute value dependent on its record identifier rid and attribute identifier. Thus, the signature of attribute value A_i in record r is computed as $sign(h(r.rid \mid i \mid r.A_i \mid r.ts))$. With this technique, the user can verify that all the attribute values in the query answer are authentic, and in the correct record and attribute positions. There are no additional computation or space overheads imposed by the attributes that are dropped by the projection. In other words, the VO contains just one aggregate signature, while the computation cost to produce and verify it is proportional to the number of projected attributes but independent of M.

3.5 Equi-Join

Here we focus on equi-join, the most common join operation. Let $\mathbf{R} \bowtie_{\mathbf{R}.A=\mathbf{S}.B} \mathbf{S}$ denote an equi-join between two relations $\mathbf{R}$ and $\mathbf{S}$ with join condition $\mathbf{R}.A = \mathbf{S}.B$ on their respective attributes A and B. Without loss of generality, we assume that the cardinality of $\mathbf{R}$ is smaller than or equal to that of $\mathbf{S}$, i.e., $|\mathbf{R}| \leq |\mathbf{S}|$.

To prove to the user that the join result is correct, our approach is to (a) apply any selection predicate on $\mathbf{R}$ and project out irrelevant attributes to produce a truncated version $\mathbf{R}'$, with a correctness

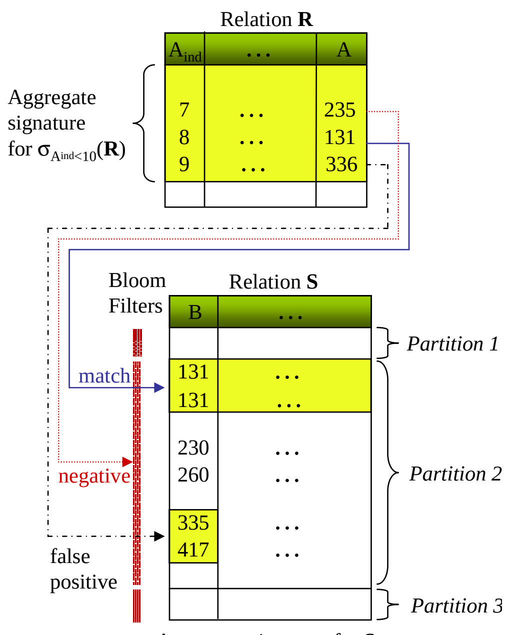


Figure 3: Authenticated Equi-Join with Bloom Filters

proof; (b) for each record $r \in \mathbf{R}'$ that has matches in $\mathbf{S}$, return r and the matching records in $\mathbf{S}$ along with their correctness proofs; (c) for each record $r \in \mathbf{R}$ that has no matches in $\mathbf{S}$, return the rid, A, and ts in r, and a proof that the value of $r.A$ does not exist in $\mathbf{S}.B$. Suppose that a fraction α of the records in $\mathbf{R}$ have matching records in $\mathbf{S}$, and that the remaining $1 - \alpha$ of them do not. The former class of records is handled like a selection $\sigma_{B=r.A}(\mathbf{S})$ (see Section 3.3). For the latter, we examine two authentication mechanisms. To simplify the presentation below, we assume $\mathbf{R} = \mathbf{R}'$.

Authenticating with Boundary Values

For each record $r \in \mathbf{R}$ that has no matching $\mathbf{S}$ record, the existing method from [24] returns the boundary $\mathbf{S}.B$ values before and after $r.A$. We denote this method as BV. In the worst case, BV requires two boundary values per r. Where the records in $\mathbf{R}$ share the same boundaries, we can eliminate duplicate $\mathbf{S}.B$ values to reduce the transmission overhead. For example, for two consecutive records $r_1, r_2 \in \mathbf{R}$, $r_1.A$'s upper boundary in $\mathbf{S}$ may be $r_2.A$'s lower boundary. After duplicate elimination, the expected size of all the boundary values from $\mathbf{S}$ that are needed for the proof is

$$|VO|_{BV} = (1-\alpha)I_A \cdot \min(2, \frac{I_B}{I_A}) \cdot |\mathbf{S}.B| \qquad (2)$$

where I_A and I_B are the number of distinct values in $\mathbf{R}.A$ and $\mathbf{S}.B$ respectively, and $|\mathbf{S}.B|$ is the size (in bytes) of the $\mathbf{S}.B$ attribute. Note that this formula refers only to the part of the proof for records in $\mathbf{R}$ which have no matching counterparts in $\mathbf{S}$. Due to these records, the VO size is expected to be huge. This motivates our advanced method below.

Authenticating with Bloom Filters

Our second authentication mechanism, denoted by BF, returns a certified Bloom filter [5] on $\mathbf{S}.B$, for the user to test those unmatched $\mathbf{R}$ records in the query answer.

Suppose we construct an m-bit Bloom filter for the I_B distinct values in $\mathbf{S}.B$. The expected false positive rate is $FP = 0.6185^{m/I_B}$ (see Section 2.1). Where the Bloom filter gives a negative, the corresponding $\mathbf{R}$ record is certain not to have a match in $\mathbf{S}$, and no additional proof is needed. Where a false positive occurs for some $r.A$, the server needs to return the two corresponding boundary values from $\mathbf{S}.B$. Since the Bloom filter is unlikely to produce false positives for consecutive $\mathbf{R}$ records, there can be no significant improvement from duplicate elimination. Hence the expected size of the proof for the $(1-\alpha)$ unmatched fraction of $\mathbf{R}$ is $\frac{m}{8} + (1-\alpha) I_A \cdot FP \cdot 2 |\mathbf{S}.B|$ – the first term is the Bloom filter size, while the second term accounts for the boundary values for proving the unmatched $\mathbf{R}$ records that get a false positive on the filter.

Figure 3 illustrates how an equi-join $\sigma_{A_{ind}<10}(\mathbf{R}) \bowtie_{A=B} \mathbf{S}$ is authenticated with a Bloom filter. First, the range of $\mathbf{R}$ records that satisfy $\mathbf{R}.A_{ind} < 10$ are gathered as part of the query answer. To allow the user to verify this part of the answer, those $\mathbf{R}$ records' signatures are combined into an aggregate signature $ASign_R$. Some of those $\mathbf{R}$ records, e.g. $\langle 8, \ldots, 131 \rangle$, have matching records in $\mathbf{S}$; the matching $\mathbf{S}$ records are added to the query answer. To prove that the rest of the $\mathbf{R}$ records do not have matching $\mathbf{S}$ records, the certified Bloom filter (constructed by DA beforehand) is supplied to the user. $\mathbf{R}$ records like $\langle 7, \ldots, 235 \rangle$ test negative on the Bloom filter, which suffices to convince the user that there is no $\mathbf{S}$ record with $\mathbf{S}.B = 235$. The remaining $\mathbf{R}$ records (such as $\langle 9, \ldots, 336 \rangle$) give a false positive on the Bloom filter, and need to be proven by inspecting the adjoining boundary $\mathbf{S}$ records (with $\mathbf{S}.B = 335$ and $\mathbf{S}.B = 417$). The signatures for the matching $\mathbf{S}$ records, the boundary $\mathbf{S}$ records, and the Bloom filter are combined into an aggregate signature $ASign_S$. Finally, $ASign_R$ and $ASign_S$ are aggregated to produce the signature for the query answer.

Although the above mechanism is adequate for proving unmatched $\mathbf{R}$ records, there is a shortcoming. This is because new data can be added easily to a Bloom filter, but it is not possible to remove the effect of a record from the filter. Consequently, following every record deletion, the Bloom filter has to be reconstructed from the remaining records, which is very expensive for large datasets.

To limit the update overhead, we split $\mathbf{S}$ and create a Bloom filter per partition, rather than just a single filter for the entire $\mathbf{S}$. Continuing our illustration in Figure 3, $\mathbf{S}$ is sorted on $\mathbf{S}.B$, and partitioned horizontally into the ranges $[0, 120)$, $[120, 420)$ and $[420, 1000)$. The finer the partitions, the lower the update cost. However, there is an upper bound to the number of partitions beyond which the Bloom filter mechanism becomes more expensive than simply returning all the boundary values in $\mathbf{S}.B$.

Suppose we divide $\mathbf{S}.B$ into p partitions. For a given query, we return those partition filters that are probed by unmatched $\mathbf{R}$ records, along with the corresponding partition boundaries. Where adjoining partitions are returned, we can again avoid duplicating their common boundaries in the VO. This brings the proof size to

$$\begin{aligned} |VO|_{BF} = (1-\alpha)\frac{m}{8} \; &+ min(1, 2(1-\alpha)) \cdot p \, |\mathbf{S}.B| \\ &+ (1-\alpha) I_A \cdot FP \cdot 2 \, |\mathbf{S}.B| \end{aligned} \tag{3}$$

where m is the total size (in bits) of the partition filters. The first term above corresponds to the total size of the partition filters that are probed by unmatched $\mathbf{R}$ records. The second term accounts for the partition boundaries. If only a few partitions are probed, we send their lower and upper boundaries (thus $2(1-\alpha) \cdot p \, |\mathbf{S}.B|$); whereas if most of them are probed, it is cheaper to return all the boundaries (thus $p \, |\mathbf{S}.B|$). The third term is due to those unmatched $\mathbf{R}$ records that get a false positive, and hence need to be authenticated via boundary $\mathbf{S}$ records. We want $|VO|_{BF}$ to be lower than $|VO|_{BV}$, thus:

$$\frac{m}{8|\mathbf{S}.B|} < I_A \left[\min(2, \frac{I_B}{I_A}) - 2\, FP \right] - \frac{min(1, 2(1-\alpha))\, p}{1-\alpha} \tag{4}$$

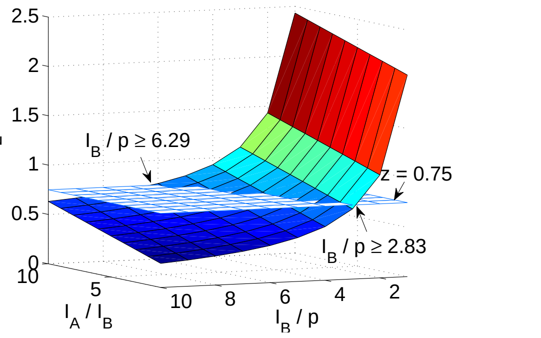


Figure 4: Configuration for Join Processing with Bloom Filters

To analyze the implications of the above condition, we first consider the case of a primary key $\mathbf{R}.A$ to foreign key $\mathbf{S}.B$ join between $\mathbf{R}$ and $\mathbf{S}$. The primary key-foreign key relationship requires every $\mathbf{S}.B$ value to exist in $\mathbf{R}.A$, so $I_A \geq I_B$. Assuming that $|\mathbf{S}.B|$ occupies 4 bytes, and setting $m = 8 I_B$ (meaning the partition filters are configured with 8 bits per distinct $\mathbf{S}.B$ value) so that $FP = 0.0216$, Formula 4 becomes:

$$0.75 \, I_B - 0.0432 \, I_A - \frac{min(1, 2(1-\alpha)) \cdot p}{1-\alpha} > 0 \tag{5}$$

A sufficient condition that satisfies the above inequality is $0.75 \, I_B > 0.0432 \, I_A + 2p$, or equivalently $z = 0.0432 \frac{I_A}{I_B} + 2 \frac{p}{I_B} < 0.75$. Figure 4 depicts the condition, with the blue surface under the white plane at $z = 0.75$ demarcating the viable $\frac{I_A}{I_B}$ and $\frac{I_B}{p}$ settings. According to the figure, we need $\frac{I_B}{p} \geq 2.83$ if $\frac{I_A}{I_B} = 1$, and $\frac{I_B}{p} \geq 6.29$ at $\frac{I_A}{I_B} = 10$. This means that a higher $\frac{I_A}{I_B}$ ratio requires each partition to contain proportionally more distinct $\mathbf{S}.B$ values. This requirement should not pose a difficulty in practice though. Given that a B$^+$-tree typically has a fanout of a few hundred, each leaf node can be a partition with its own Bloom filter, and the inequality in Formula 5 will still be satisfied for a wide spectrum of join queries. Such a granularity is also I/O-efficient, because the Bloom filter for an updated leaf node is written back to the disk along with the leaf node in a single I/O operation. For memory-resident indexes that typically have small fanout factors, a partition may span multiple leaf nodes in order to satisfy the constraint on p.

Now consider the case where $\mathbf{R}.A$ and $\mathbf{S}.B$ are not a primary key-foreign key pair; instead, $\mathbf{R} \bowtie_{\mathbf{R}.A=\mathbf{S}.B} \mathbf{S}$ is merely an arbitrary equi-join operation. If $I_A \geq I_B$, the earlier analysis is still applicable, so we shall focus on the situation where $I_A < I_B$. If $\frac{I_B}{I_A} > 2$, Formula 4 becomes $1.9568 \frac{I_A}{I_B} - \frac{min(1,2(1-\alpha))}{1-\alpha} \cdot \frac{p}{I_B} > 0.25$, again assuming that $|\mathbf{S}.B| = 4$, $m = 8 I_B$ and $FP = 0.0216$. A sufficient condition for the inequality is $0.9784 \frac{I_A}{I_B} - \frac{p}{I_B} > 0.125$. Since $\frac{p}{I_B} > 0$, the constraint indicates that the BF method is not beneficial to any equi-join where $I_B \geq 7.8272 \, I_A$. This is intuitive since the Bloom filters must then be configured with a large m in order to achieve the desired $\frac{m}{I_B}$ ratio and hence false positive rate, making the filters too bulky and costly to transmit to the user.

4. CACHING AGGREGATE SIGNATURES

While signature aggregation is fast enough to be practical on newer CPUs, it remains slower than hashing. This is a concern especially as we intend to push fresh data to the users quickly. In this section, we introduce the *SigCache* mechanism that significantly reduces the query server's proof construction cost, by caching only a small number of strategically chosen aggregate signatures.

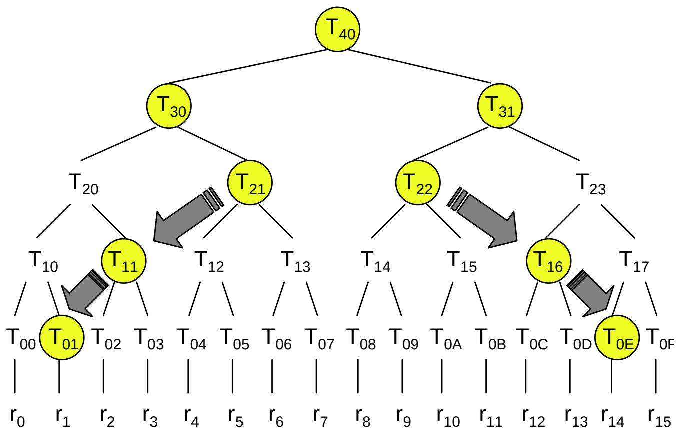


Figure 5: Example Aggregate Signature Tree

4.1 Selecting Aggregate Signatures for Caching

Consider a relation with N data items, $\mathbf{R} = \{r_1, r_2, \ldots, r_N\}$. For simplicity, assume that N is a power of 2. We compose a binary tree $\mathbf{T}$ of aggregate signatures over $\mathbf{R}$. Each leaf node in $\mathbf{T}$ corresponds to the signature of one record in $\mathbf{R}$, and each internal node aggregates the signatures in its two child nodes. We represent a node in $\mathbf{T}$ as $T_{i,j}$ where: (a) i is the tree level, with $i = 0$ for the leaf nodes, and $i = \log N$ for the root node; (b) j is the node's position in level i; $j = 0, 1, \ldots, \frac{N}{2^i} - 1$ from left to right.

This signature tree is *conceptual*, as only the nodes that are chosen for caching need to be materialized. Our decision on which nodes $T_{i,j}$ should be cached is based on an analysis of their corresponding benefits offered (i.e., computation savings achieved if each of them is cached). We will use Figure 5 as a running example to present our analysis and illustrate our caching mechanism.

A selection query $\mathbf{q}$ with cardinality q (i.e., whose result comprises q records) has a choice of $N - q + 1$ different ranges of data in $\mathbf{R}$. The aggregate signature for the answer $\mathbf{A}$ of $\mathbf{q}$ can be derived from the roots of the subtrees in $\mathbf{T}$ that together cover $\mathbf{A}$. Let $\xi(T_{i,j} \mid q)$ denote the number of queries with cardinality q that derive their aggregate signatures from $T_{i,j}$.

- If $2^i > q$, $\xi(T_{i,j} \mid q) = 0$.
  For example, nodes T_{30} and T_{31} are irrelevant to any query with a smaller cardinality than $2^3 = 8$.
- If $2^i \le q < 2^{i+1}$,

$$\xi(T_{i,j} \mid q) = \begin{cases} q - 2^i + 1 & \text{if } 0 < j < \frac{N}{2^i} - 1 \\ 1 & \text{otherwise} \end{cases}$$

  In our running example, only one query with cardinality $q = 7$ (involving r_0 to r_6) can make use of T_{20}. Likewise, T_{23} benefits only one query with cardinality $q = 7$ (involving r_9 to r_{15}). In contrast, T_{21} can be exploited by $7 - 2^2 + 1 = 4$ different queries with cardinality $q = 7$ (involving r_1 to r_7, r_2 to r_8, r_3 to r_9 and r_4 to r_{10} respectively). The same holds for T_{22}.
- If $2^{i+1} \le q$,
  (a) If j is odd,

$$\xi(T_{i,j} \mid q) = \begin{cases} 2^i & \text{if } \frac{N}{2^i} - j \ge \lceil \frac{q}{2^i} \rceil \\ 2^i - q + \lfloor \frac{q}{2^i} \rfloor \cdot 2^i & \text{if } \lfloor \frac{q}{2^i} \rfloor = \frac{N}{2^i} - j < \lceil \frac{q}{2^i} \rceil \\ 0 & \text{otherwise} \end{cases}$$

  Consider again queries with cardinality $q = 7$. T_{11} and T_{13} satisfy the first condition, and are relevant to 2^1 queries each; e.g., queries for r_5 to r_{11} and r_6 to r_{12} can use T_{13}. T_{15} satisfies the second condition, and is relevant to only $2^1 - 7 + \lfloor \frac{7}{2^1} \rfloor \cdot 2^1 = 1$ of the queries (the one for r_9 to r_{15}). The third condition applies to T_{0B}, T_{0D}, T_{0F} and T_{17}, none of which are relevant to queries with $q = 7$.

  (b) If j is even,

$$\xi(T_{i,j} \mid q) = \begin{cases} 2^i & \text{if } j + 1 \ge \lceil \frac{q}{2^i} \rceil \\ 2^i - q + \lfloor \frac{q}{2^i} \rfloor \cdot 2^i & \text{if } \lfloor \frac{q}{2^i} \rfloor = j + 1 < \lceil \frac{q}{2^i} \rceil \\ 0 & \text{otherwise} \end{cases}$$

  For queries with $q = 7$, the first condition applies to T_{14}, T_{16}, T_{08}, T_{0A}, T_{0C} and T_{0E}; T_{12} and T_{06} satisfy the second condition; whereas T_{00}, T_{02}, T_{04} and T_{10} make up the third category.

The probability that a node $T_{i,j}$ is used to produce the aggregate signature for a query with cardinality q is $P(T_{i,j} \mid q) = \frac{\xi(T_{i,j} \mid q)}{N-q+1}$. Furthermore, the overall probability of using $T_{i,j}$ for any query is $P(T_{i,j}) = \sum_{q=1}^{N} P(T_{i,j} \mid q) \cdot P(q)$, where $P(q)$ is the probability that a query has cardinality q.

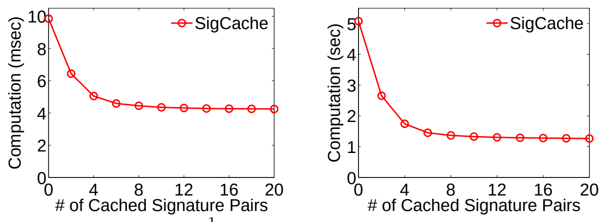
Let $s_{i,j}$ denote the savings in computation from the cached signature $T_{i,j}$, compared to deriving it on demand from any cached signatures and/or the data signatures underneath $T_{i,j}$. If none of the aggregate signatures under $T_{i,j}$ are cached, the savings over aggregating the data signatures is $s_{i,j} = (2^i - 1) \cdot c$, where c is the cost of an ECC addition operation. We can drop factor c since it is constant across all the nodes. The utility in caching $T_{i,j}$ is thus $u_{i,j} = P(T_{i,j}) \cdot s_{i,j} = P(T_{i,j}) \cdot (2^i - 1)$.

The nodes that are nearer to the leaf level are likely to have higher $P(T_{i,j})$ values because they are relevant whether the query has small or large cardinality. On the other hand, their corresponding aggregate signatures produce smaller savings $s_{i,j}$ each time. Conversely, nodes that are nearer to the root benefit only queries with large cardinalities, but lead to considerable reductions in computation whenever they are utilized.

Algorithm 1 *SigCache* Algorithm

```
1:  // Initialize the signature tree
2:  for all $T_{i,j} \in \mathbf{T}$ do
3:      compute $P(T_{i,j})$;
4:      set $s_{i,j} = 2^i - 1$;
5:      set $u_{i,j} = P(T_{i,j}) \cdot s_{i,j}$;
6:  totalCost = $\sum_{q=1}^{N} (q-1) \cdot P(q)$;
7:  // Evaluate the savings from caching each $T_{i,j}$
8:  order the $T_{i,j}$'s in decreasing utility values;
9:  prevCost = totalCost;
10: for the $T_{i,j}$ with the next highest utility do
11:     reduce the savings of $T_{i,j}$'s ancestors by $s_{i,j}$;
12:     add $T_{i,j}$ to the cache;
13:     currCost = totalCost - utility of all cached signatures;
14:     if currCost > prevCost then
15:         add back $s_{i,j}$ to the savings of $T_{i,j}$'s ancestors;
16:         remove $T_{i,j}$ from the cache;
17:     else
18:         prevCost = currCost;
```

Algorithm 1 presents the procedure for identifying the aggregate signatures to hold in the cache. The algorithm initializes the aggregate signature tree, with the probability, savings and utility of each node $T_{i,j}$ set as specified above. The average cost of computing the aggregate signature for each query is derived as the sum of the query cardinalities, weighted by their corresponding query probabilities. Next, we take a greedy approach in finding the next node that leads to the highest reduction in the average query cost. We repeatedly evaluate the node $T_{i,j}$ with the next highest utility. If $T_{i,j}$



(a) $P(q) = \frac{\frac{1}{q}}{\sum_{i=1}^{N} \frac{1}{i}}$ (b) $P(q) = \frac{1}{N}$

Figure 6: Reduction in VO Construction Cost

is added to the cache, its ancestors naturally should be derived from $T_{i,j}$ rather than the underlying data signatures themselves. Therefore, the ancestors' savings are reduced, which could result in a net increase in query cost if some of the ancestors are already cached. For example, initially $s_{3,0} = 2^3 - 1 = 7$, because it removes the need to aggregate nodes from $T_{0,0}$ to $T_{0,7}$. Once $T_{2,1}$ is cached, however, $T_{3,0}$ could be derived from $T_{0,0}$ to $T_{0,3}$ and $T_{2,1}$. Thus, caching $T_{3,0}$ saves only $s_{3,0} = 2^3 - 1 - s_{2,1} = 4$ signature aggregation operations now. If adding $T_{i,j}$ raises the query cost, then it is discarded; otherwise, $T_{i,j}$ is pinned in the cache.

We expect the average query cost to be reduced substantially through the aggregate signatures that are chosen early on for caching; as more signatures are cached, the incremental saving will diminish gradually. This provides an opportunity to terminate the $SigCache$ algorithm early, so that it does not have to evaluate all the nodes of the signature tree. Another optimization stems from the observation that nodes $T_{i,j}$ and $T_{i,N/2^i-j-1}$ mirror each other in probability, savings and utility. Thus, $SigCache$ can evaluate only the left half of the signature tree; whenever a node is chosen for caching, its mirror node is also cached automatically.

We conducted several experiments with different N values and $P(q)$ distributions to see whether there are consistent findings on which tree nodes have the highest utilities and should be cached. One of the distributions is:

$$P(q) = \frac{\frac{1}{q}}{\sum_{i=1}^{N} \frac{1}{i}}$$

for $1 \leq q \leq N$, which favors short queries (this is a truncated harmonic series). Another distribution is $P(q) = \frac{1}{N}$, where all query cardinalities are equally likely. The experiments consistently show that the most valuable aggregate signatures to cache are the *second* node from the left and right edges of the signature tree, starting from the third highest tree level and progressing down towards the leaves. Intuitively, this is because the second node from the edge is applicable to the widest range of query cardinalities, among all the nodes in the same tree level. It is also useful to cache the root node as well as its two immediate children. In our running example in Figure 5, the most beneficial aggregate signatures to cache are T_{21} and T_{22}, followed by T_{11} and T_{16}, then T_{01} and T_{0E}. The top three signatures, T_{40}, T_{30} and T_{31}, are also cached.

Figure 6 shows the saving that signature caching achieves, when applied on a dataset of one million randomly generated records. Without caching, the average computation cost per query is 9.85 milliseconds and 5.08 seconds for the skewed and uniform query length distributions, respectively. By caching just the top eight pairs of the aggregate signatures identified by $SigCache$, we are able to reduce the computation cost of proof construction by 57% and 75% for the two query distributions. The eight chosen pairs of signatures, in decreasing utility value, are:

- For the skewed query distribution, $\{T_{18,1}$, $T_{18,2}$, $T_{17,1}$, $T_{17,6}$, $T_{16,1}$, $T_{16,14}$, $T_{15,1}$, $T_{15,30}$, $T_{15,5}$, $T_{15,26}$, $T_{14,1}$, $T_{14,62}$, $T_{14,5}$, $T_{14,58}$, $T_{13,1}$, $T_{13,126}\}$;
- For the uniform query distribution, $\{T_{18,1}$, $T_{18,2}$, $T_{17,1}$, $T_{17,6}$, $T_{16,1}$, $T_{16,14}$, $T_{15,1}$, $T_{15,30}$, $T_{15,5}$, $T_{15,26}$, $T_{14,1}$, $T_{14,62}$, $T_{14,5}$, $T_{14,58}$, $T_{14,9}$, $T_{14,54}\}$.

In general, we find that the number of signatures to be cached is proportional to N/q, though it is possible to pick only some of the high-utility ones among them if the cache size is limited.

4.2 Adaptive Signature Caching

Clearly, the choice of aggregate signatures to cache hinges on the queries that the server receives. For example, Algorithm 1 will not pick $T_{i,j}$ in the signature tree if none of the queries has cardinality $q \geq 2^i$. In practice, the query cardinalities (i.e., answer sizes) may not span the entire range from one to N, and the query distribution may be skewed or even drift over time. To deploy $SigCache$ for general workloads, the query server first populates the cache by running Algorithm 1 with the $P(T_{i,j})$ values estimated from past queries as explained above. This initialization procedure needs to examine the utility of every node in the signature tree and is costly for large datasets, but it can be performed offline.

At runtime, each of the cached signatures $T_{i,j}$ can benefit any query with a scope that envelops $\{r_{j \cdot 2^i}, \ldots, r_{(j+1) \cdot 2^i - 1}\}$. In the course of processing user queries, additional aggregate signatures that are generated to prove the query answers are added to the cache, and an access count is kept on each cached signature. The server then periodically revises the list of cached signatures, by applying Algorithm 1 with minor changes: $\mathbf{T}$ now contains just the cached signatures, and the $P(T_{i,j})$'s are calculated from the respective access counts. Since only the cached signatures are involved here, the revision can be performed efficiently.

4.3 Updating the Cached Signatures

Having determined which signatures to cache, the query server needs to keep them updated with respect to the underlying records. This could be achieved through *eager* or *lazy* updates. Suppose that a record r_k (and, thus, its corresponding signature $T_{0,k}$) is updated. Any cached signature $T_{i,j}$ such that $\lfloor \frac{k}{2^i} \rfloor = j$ is an ancestor of $T_{0,k}$ in the signature tree and needs to be refreshed. The eager method would recalculate $T_{i,j}$ immediately, by adding to $T_{i,j}$ the inverse of the old $T_{0,k}$, followed by the new $T_{0,k}$. In contrast, the lazy method would simply invalidate $T_{i,j}$, and update it only when it is required for some subsequent query. We expect the eager strategy to shorten the query turnaround time as the aggregate signatures are updated beforehand, whereas the lazy method utilizes computing resources more efficiently in avoiding updates that do not benefit subsequent queries. We will systematically evaluate the eager and the lazy update approaches in Section 5.

5. EMPIRICAL EVALUATION

In this section, we empirically evaluate the performance of our authentication schemes. The key questions to be investigated include:

- Relative to Merkle hash tree methods, how fast is our scheme in making fresh data available to the users? How responsive is it in providing an authenticated query answer? How high a transaction throughput can it sustain?
- Is our equi-join authentication mechanism effective in reducing the VO size as the analysis in Section 3.5 indicates? Is the Bloom filter authentication technique robust enough to handle a wide range of workloads?
- What are the space-time trade-offs of signature caching?

Parameter	Description	Default
N	Number of records	1 million
$RecLen$	Record length	512 bytes
sf	Selectivity factor	0.1%
$\|sign\|$	Size of a signature	160 bits
$\|digest\|$	Size of a hash digest	160 bits
B_{WAN}	Bandwidth of wide area network connecting the data aggregator and the server	622 Mbps
B_{LAN}	Bandwidth of local network connecting the server and users	14.4 Mbps
$ArrRate$	Transaction arrival rate	–
$Upd\%$	Ratio of update transactions; the rest are queries	10%
ρ	Interval for update summaries	1 sec
ρ'	Signature renewal age	900 sec

Table 2: Experiment Parameters

5.1 Experiment Set-Up

Before investigating the above questions, we describe our experiment set-up. The experiment parameters and their default values are summarized in Table 2.

System model: The data aggregator (DA) and query server (QS) maintain identical copies of the database and authentication structure. When there is a data update, the DA forwards it to the QS, then refreshes its own authentication structure and sends the new record signature or MHT root signature to the QS. The QS updates its authentication structure accordingly[1]. Essentially, the DA and QS perform the same update operations, except for producing the new signature which is carried out by the DA. As the QS has to service both user queries and data updates, it is the primary performance determinant.

Workload: As in [18], we create a relation $\mathbf{R}$ containing N randomly/uniformly generated records. Each record contains $RecLen$ bytes, including a 4-byte integer key. Transaction arrivals at the QS follow a Poisson process at a rate of $ArrRate$. Among them, $Upd\%$ are data updates forwarded from the DA, while user queries make up the rest of the workload. Selection queries are distributed uniformly within $\mathbf{R}$, with selectivity factor between $\frac{1}{2}sf$ and $\frac{3}{2}sf$. By default, both queries and updates are processed in real time, rather than in batches, to maximize data freshness. All the transactions at the QS follow the two-phase locking protocol to ensure that the database and authentication information are consistent.

Algorithms: We shall benchmark our proposed scheme against the EMB$^-$ tree, the most representative MHT design for disk-resident data. The EMB$^-$ implementation used in our experiments is the original code from [18]. We label our scheme as BAS, as it builds on the Bilinear Aggregate Signature technique. In order to provide a consistent comparison with past studies, our implementations are compiled with the same cryptographic libraries. Specifically, we use the MIRACL library [1] for BAS, and the OpenSSL library [2] for the condensed RSA protocol.

System configuration: The DA and QS machines are identical; each runs Windows Server 2003 and is equipped with Intel Core 2 Quad 3GHz CPU, 3GB memory and two Hitachi HTS541616J9SA00 160GB hard disks. The disks are formatted with 4-Kbyte blocks, the default in NTFS. The user machine has an Intel Core Duo 2GHz CPU and 1GB memory[2]. To study the authentication mechanisms under different system configurations, we model the wide area network connection between the DA and each server as a queue with a capacity of B_{WAN} = 622 Mbps, the bandwidth of OC12. The local network between the users and QS is modeled as a queue with default bandwidth of B_{LAN} = 14.4 Mbps, corresponding to the highest HSDPA (also known as 3.5G) data rate. The networks are the only simulated components in our system; the server processing and user verification components are actual implementations.

Performance factors: Unless otherwise specified, our evaluation centers around the update time, query time, VO size, and user verification time. The first measures how soon fresh data can be made available at the QS, while the next three factors determine the overall time required for a user to receive an answer and confirm that it is correct. As the server and the network are shared resources, the system's scalability (with respect to the number of users and transactions supported) depends on the first three performance factors, which receive most of our attention.

5.2 Choice of Cryptographic Primitives

In Table 3 we compare the costs of the basic cryptographic operations in 160-bit BAS which were obtained in Year 2006 on a Pentium 3 800 MHz CPU with 1 GB memory as reported in [23], versus the current timings on our test machine. The measurements were made with the default settings of Table 2. We observe an impressive speed-up of BAS, with one order of magnitude faster ECC signing and almost 40 times faster BAS verification.

Table 3 also presents measurements for an RSA-based implementation of signature aggregation; the results correspond to 1024-bit signatures, offering an equivalent level of security to the 160-bit ECC/BAS scheme. Additionally, Table 3 includes hash computation costs for different message sizes (using SHA); this operation is important to all schemes, but especially so for MHT-based ones. The results show that signature aggregation through condensed RSA or BAS is now viable in terms of computation time. Between the two, RSA is faster while BAS has the advantage of shorter signature length, which translates to lower space requirements. We adopt BAS in our study because its signature length is the same as that of a digest in the MHT (both at 160 bits), leading to comparable storage overheads[3] and allowing us to focus on the runtime trade-offs between MHT and BAS.

5.3 Comparison with MHT Approach

In Table 4, we set N to 1 million, and execute point queries/updates (i.e., selectivity $sf = 10^{-6}$ = 1 record) and range queries/updates with selectivity $sf = 10^{-3}$ (1,000 records), *one transaction at a time*. The results show that BAS outperforms EMB$^-$ in query processing, update time, and VO size for both point and range queries.

Figure 7(a) presents the overall response time for point queries (the solid blue lines with 'Q' in the labels) at various arrival rates. Here we measure the elapsed time between the arrival of a query at the QS and the verification of its answer by the user. EMB$^-$ is only

[1]In MHT schemes an alternative update approach is possible, where the DA applies the updates locally and sends (the updated parts of) the MHT to the QS in order to avoid repeating the involved digest computations at the QS. This approach worsens performance, as the CPU time savings on digest computations are offset (by several orders) by the extra communication overhead incurred. This technique also does not circumvent the lock contention issues of the MHT, since again the MHT root must be exclusively locked (among others) by each and every update.

[2]The user machine could have been a lower-end one, or even a mobile device. However, since we are targeting high-value applications like online trading for which data freshness is important, the user terminals are more likely to be adequately equipped.

[3]Actually, as explained in Section 3.2, BAS takes up slightly smaller space, because MHT approaches store additional digests in the internal nodes of their indexes.

Operation	Year 2006	Current
Bilinear Aggregate Signature		
(a) Individual signature		
• Signing	12.0 ms	1.5 ms
• Verification	77.4 ms	40.22 ms
(b) 1000-signature aggregate		
• Aggregation	N.A.	9.06 ms
• Verification	12085.4 ms	331.349 ms
Condensed RSA		
(a) Individual signature		
• Signing	6.82 ms	6.06 ms
• Verification	0.16 ms	0.087 ms
(b) 1000-signature aggregate		
• Aggregation	N.A.	0.078 ms
• Verification	44.12 ms	0.094 ms
Secure Hashing Algorithm (SHA)		
• 256-byte message	–	1.35 μs
• 512-byte message	–	2.28 μs
• 1024-byte message	–	4.2 μs

Table 3: Costs of Cryptographic Primitives

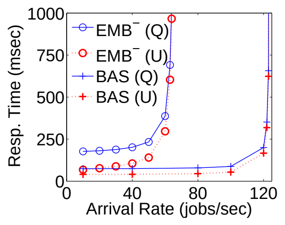
Selectivity	Operation	EMB$^-$	BAS
$sf = 10^{-6}$	Query (msec)	35.316	31.433
(1 record)	Update (msec)	60.206	40.246
	VO Size (bytes)	440	20
	Verification (msec)	139	42.92
$sf = 10^{-3}$	Query (msec)	129.782	61.502
(1000 records)	Update (msec)	248.89	237.4
	VO Size (bytes)	720	20
	Verification (msec)	171	375

Table 4: Performance of Standalone Queries & Updates

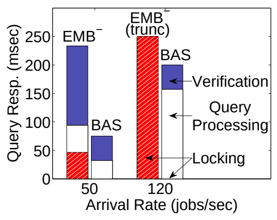
able to handle up to 50 jobs/second before lock contention becomes a bottleneck, whereas BAS scales all the way to 120 jobs/second. Figure 7(b) shows the breakdown of the response time, which highlights the concurrency deficiencies of EMB$^-$.

Figure 8 shows statistics on the update summaries. As the signature renewal age ρ' is relaxed, the number of forced record recertifications in each period declines, leading to shorter compressed bitmaps; at the same time, the average age of the record signatures increases. The total summary information needed to perform a freshness check is a function of the per-bitmap size and the signature age. As shown in Figure 8(b), the total summary size bottoms out at 171 KB for $\rho = 1$ second and $\rho' = 900$ seconds. On a 14.4 Mbps link, the summaries require around 95 msec to transmit. In our design, the server sends the summaries to the user upon log-in, from the current one back to the one for the average signature age; this takes place in conjunction with other start-up tasks like verifying the public key certificate of the DA. Thereafter, the size of each periodic update summary is only 375 bytes on the average, and compressing/decompressing it takes 6 to 7 msec. An alternative design is to start pushing the summaries (from the latest backwards) to the user immediately upon receipt of a query, and continue transmission while the query is being executed. We note (figure omitted) that the average bitmap size and signature age are not sensitive to the update rate. The reason is that when updates are few, more idle resources are channeled to the background signature renewal process and vice versa, so that their combined activities remain stable.

In Figure 9 we perform the same test as Figure 7(a) for range queries. At very light loads, EMB$^-$ is faster than BAS, because the latter incurs longer user verification time (see Figure 9(b)). However, EMB$^-$ degrades quickly as the increasing workload raises

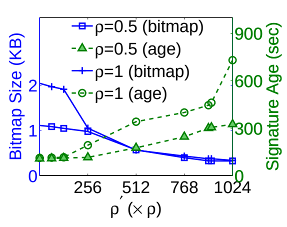


(a) Response Time

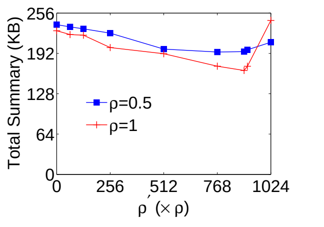


(b) Breakdown

Figure 7: EMB$^-$ versus BAS ($sf = 10^{-6}$)

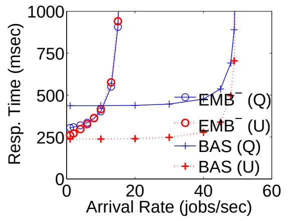


(a) Bitmap Size vs. Sign. Age

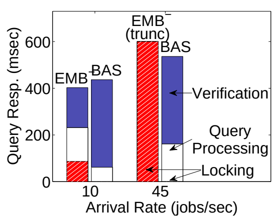


(b) Total Summary Size

Figure 8: Compressed Update Summaries

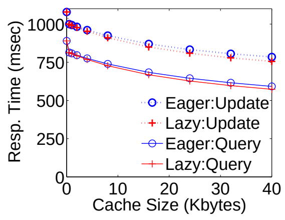


(a) Response Time

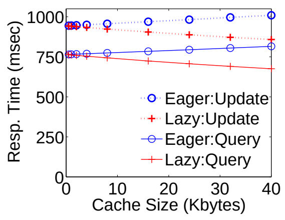


(b) Breakdown

Figure 9: EMB$^-$ versus BAS ($sf = 10^{-3}$)



(a) $Upd\% = 10\%$



(b) $Upd\% = 40\%$

Figure 10: $SigCache$ effectiveness ($N = 1$M records)

lock contention at the QS; the results show that EMB$^-$ reaches saturation at just 10 jobs/second. In contrast, we are able to push BAS beyond 45 jobs/second on the same workload composition.

We now turn to the update times in Figures 7(a) and 9(a), in dotted red lines with 'U' in the labels. The update time includes the cost of modifying the EMB$^-$ tree/B$^+$-tree at the QS, plus the time for the DA to generate and send over the new signature. BAS is consistently faster than EMB$^-$ in making fresh data available to the users. BAS also scales to much higher update rates.

5.4 Signature Caching

In the previous experiments we used no signature caching for BAS. In this section, we investigate the impact of the $SigCache$ mechanism on the performance of BAS for varying cache sizes. Figure 10(a) focuses on range queries and updates over a disk-resident database with 1 million records, with the transaction arrival rate set to 50 jobs/second (where the system is heavily loaded for BAS). The results show that $SigCache$ enhances considerably the performance of BAS with just a modest amount of cache. For example, with just a 40-Kbyte cache, $SigCache$ manages to reduce the overall response time by 30% for both updates and queries.

Having demonstrated the effectiveness of $SigCache$, we now focus on the alternative Lazy and Eager strategies for updating the cached signatures. In Figure 10(a) we plot the running time of both Eager and Lazy $SigCache$ implementations for the default $Upd\% = 10\%$. Their timings are similar, with Lazy being slightly faster. This is because a cached signature is usually needed by one or more queries in between updates. The amount of computations required to refresh a cached signature that has been invalidated by a data update is the same, whether it is carried out as part of the update transaction or by the first query that requests for the invalidated signature. By shifting the cache refresh from the update transactions (which lock their data in exclusive mode) to the queries (which hold only shared locks), the Lazy strategy achieves higher concurrency and hence a slight advantage over the Eager method. For these reasons, Eager does not outperform Lazy even for smaller $Upd\%$ (charts are omitted due to lack of space).

For larger $Upd\%$, Lazy has a more significant advantage over Eager. In Figure 10(b) we set $Upd\%$ to 40%. With the higher update ratio, there is an increased likelihood that a cached signature is invalidated multiple times before it is required for a query. Under these circumstances, refreshing the cached signatures eagerly wastes computations, especially with a large cache. In contrast, the Lazy strategy remains effective, and improves with the cache size.

5.5 Equi-Join Operation

Next, we investigate the performance of the two equi-join verification mechanisms described in Section 3.5. The existing method [24] that always returns boundary values to prove unmatched records is denoted by BV, whereas our proposed scheme that utilizes Bloom filters is denoted by BF. We focus on the most common join operation – the primary key-foreign key equi-join of the form $\sigma(\mathbf{R}) \bowtie_{\mathbf{R}.A=\mathbf{S}.B} \mathbf{S}$. The two relations $\mathbf{R}$ and $\mathbf{S}$ for this experiment are extracted from the TPC-E benchmark. $\mathbf{R}$ corresponds to the 'Security' table, and contains N_R = 6,850 records with an equal number of distinct $\mathbf{R}.A$ values (i.e., I_A = 6,850). Each $\mathbf{R}$ record is 18 bytes in size. $\mathbf{S}$ is a subset of the 'Holding' table, comprising N_S = 894,000 records with I_B = 3,425 distinct $\mathbf{S}.B$ values, and an average record size of 62.95 bytes. Our evaluation centers on the VO size, the most critical factor in join verification; the remaining costs are similar for both techniques.

First, in Figure 11(a), we compare the VO sizes of BV and BF for different α values. (α is the ratio of $\mathbf{R}$ records that have matching records in $\mathbf{S}$.) To be able to control α, we set the selectivity on $\mathbf{R}$ to 20% and use records that yield the desired α value. The default setting for the number of Bloom filter bits per distinct I_B value, m/I_B, is 8. The partition size I_B/p, in terms of the number of distinct $\mathbf{S}.B$ values in each partition, is set to 4.

We observe that for small α values, BV generates very large VOs (of size close to the entire $\mathbf{S}$), because numerous boundary values are needed in order to prove the unmatched $\mathbf{R}$ records. As α grows, the unmatched records become fewer and the VO size decreases. As for BF, its VO is dominated by the boundary values for proving the records that get a false positive from the Bloom filters. The number of false positives is proportional to the number of unmatched $\mathbf{R}$ records that are tested against the Bloom filters. This explains the concise VOs in BF when α is small. BF is beneficial for the entire range of α values, consistently generating VOs that are around 60% smaller than BV.

For Figure 11(b), we set α to 0.5 and vary m/I_B, i.e., the number of Bloom filter bits per distinct $\mathbf{S}.B$ value. BF outperforms BV in all tested settings. With more Bloom filter bits, the VO size of BF drops initially because the lower false positive rate reduces the number of boundary values in the VO. This improvement however diminishes gradually, and eventually reverses as gains from the lower false positive rates are offset by the larger Bloom filter sizes. The results suggest that a range between 8 and 12 for m/I_B is adequate, with BF achieving more than 60% reduction in VO size compared to BV.

Figure 11(c) investigates the effect of the partition size. We now vary I_B/p from 2 to 2048, while keeping the other parameters at their default values. Interestingly, as I_B/p increases, the VO size in BF initially grows (for $I_B/p \leq 4$) before dropping (for $I_B/p > 4$). Behind this behavior lie two factors. On one hand, with very small partitions, a large proportion of them do not cover any unmatched $\mathbf{R}$ records and can, thus, be excluded from the VO. On the other hand, coarser partitions translate to fewer partition boundaries included in the VO. Overall, BF should be configured with the largest possible I_B/p that keeps the partitions within one disk block each for disk-resident data, so that the partition filters can be updated without necessitating extra I/Os. If $\mathbf{S}$ is memory-resident, the results suggest that one Bloom filter over the entire table is best as far as VO size is concerned. However, that worsens update performance because (i) the entire filter must be locked for each update (which lowers concurrency), and (ii) the CPU cost to recompute the filter is higher, as indicated by the dashed line in the figure giving the computation time to update a partition filter. As a guide, a partition granularity similar to that for a disk-based setting achieves a good trade-off.

Figure 11(d) plots the VO size as we vary the selectivity on $\mathbf{R}$. The figure shows that as the selectivity increases, there are more unmatched $\mathbf{R}$ records that need to be proven. This pushes up the VO size of both BV and BF, though the increase for BV is steeper. As a result, the VO size in BF is 45% to 75% smaller (for selectivities of 0.5% and 95%, respectively).

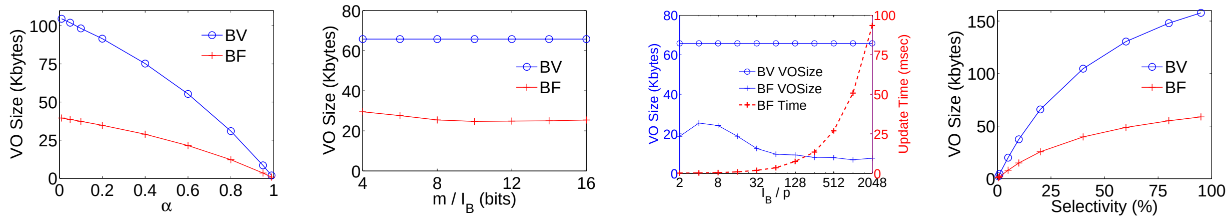
5.6 Discussion on Experiment Results

In summary, our experiments confirm the following:

- Between the two signature aggregation protocols, condensed RSA is faster than BAS in proof construction and user verification. However, BAS has benefited enough from recent advances in CPU speed to be practical for query answer authentication.
- On the server, the MHT imposes severe concurrency constraints for dynamic datasets, whereas BAS is able to take advantage of concurrent transaction execution. In our experiments, BAS consistently outperforms MHT, both in its promptness in making available fresh data to the users, and in its responsiveness to user queries. Moreover, BAS scales to much heavier workloads.
- For equi-join queries, our authentication mechanism based on Bloom filters is superior to the straightforward alternative of returning boundary values to prove all unmatched records. Particularly, the Bloom filters help to surmount the main problem in join verification, namely, the huge VO size.
- Our signature aggregation method is very effective in exploiting a modest cache to significantly reduce proof construction time. As for the choice of cache maintenance strategy, Lazy has an edge over Eager regardless of the query-update composition of the workload.

6. CONCLUSION

In this paper, we study the problem of verifying the authenticity, completeness and freshness of query answers from frequently updated databases that are hosted on untrusted servers. We introduce a protocol, built upon signature aggregation, for checking the correctness of query answers. Our approach has the important property



(a) Match Ratio (b) Filter Size (c) Partition Size (d) Selectivity

Figure 11: Primary Key-Foreign Key Equi-Join

of allowing new data to be disseminated *immediately*, while ensuring that outdated values beyond a pre-set age can be detected. We also construct authentication mechanisms for the B^+-tree and standard relational operators that are suitable for dynamic databases. Additionally, we propose an efficient verification technique for ad-hoc equi-joins, for which no practical solution existed. Finally, for servers that need to process heavy query workloads, we introduce a mechanism that significantly reduces the proof construction time by caching just a small number of strategically chosen aggregate signatures. Extensive experiments confirm that our solution performs significantly better on the freshness requirement than existing MHT schemes, while achieving considerably higher transaction throughput.

7. REFERENCES

[1] MIRACL: Multiprecision Integer and Rational Arithmetic C/C++ Library. http://www.shamus.ie.

[2] OpenSSL Project. http://www.openssl.org.

[3] M. Aschbacher. *Finite Group Theory, Second Edition*. Cambridge University Press, 2000.

[4] E. Bertino, B. Carminati, E. Ferrari, B. M. 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.

[5] B. Bloom. Space/Time Trade-Offs in Hash Coding with Allowable Errors. *Communications of the ACM*, 13(7):422–426, July 1970.

[6] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. A Survey of Two Signature Aggregation Techniques. *CryptoBytes*, 6(2), 2003.

[7] D. Boneh, B. Lynn, and H. Shacham. Short Signatures from the Weil Pairing. In *AsiaCrypt*, pages 514–532, 2001.

[8] Certicom. SEC2: Recommended Elliptic Curve Domain Parameters, Version 1.0. *Standards for Efficient Cryptography*, September 2000. http://www.secg.org/download/aid-386/sec2_final.pdf.

[9] W. Cheng, H. Pang, and K.-L. Tan. Authenticating Multi-Dimensional Query Results in Data Publishing. In *DBSec*, pages 60–73, July 2006.

[10] W. Cheng and K.-L. Tan. Query Assurance Verification for Outsourced Multi-dimensional Databases. *Journal of Computer Security*, 2008.

[11] D. Comer. Ubiquitous B-Tree. *ACM Computing Surveys*, 11(2):121–137, 1979.

[12] P. T. Devanbu, M. Gertz, C. U. Martel, and S. G. Stubblebine. Authentic Data Publication over the Internet. *Journal of Computer Security*, 11(3):291–314, 2003.

[13] Fact Sheet NSA Suite B Cryptography. *National Security Agency*. http://www.nsa.gov/ia/industry/crypto_suite_b.cfm.

[14] A. S. Frenkel and S. T. Klein. Novel Compression of Sparse Bit-Strings – Preliminary Report. *Combinatorial Algorithms on Words, NATO ASI Series F*, 12:169–183, 1985.

[15] M. T. Goodrich, R. Tamassia, and N. Triandopoulos. Super-Efficient Verification of Dynamic Outsourced Databases. In *CT-RSA*, pages 407–424, 2008.

[16] A. Joux and K. Nguyen. Separating Decision Diffie-Hellman from Computational Diffie-Hellman in Cryptographic Groups. *Journal of Cryptology*, 16(4):239–247, 2003.

[17] A. K. Lenstra and E. R. Verheul. Selecting Cryptographic Key Sizes. *Journal of Cryptology*, 14:255–293, 2001.

[18] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic Authenticated Index Structures for Outsourced Databases. In *SIGMOD*, pages 121–132, 2006.

[19] D. Ma, R. H. Deng, H. Pang, and J. Zhou. Authenticating Query Results in Data Publishing. In *ICICS*, pages 376–388, 2005.

[20] 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.

[21] R. C. Merkle. A Certified Digital Signature. In *Crypto*, pages 218–238, 1989.

[22] K. Mouratidis, D. Sacharidis, and H. Pang. Partially Materialized Digest Scheme: An Efficient Verification Method for Outsourced Databases. *International Journal on Very Large Data Bases*, 18(1):363–381, 2009.

[23] M. Narasimha, E. Mykletun, and G. Tsudik. Authentication and Integrity in Outsourced Databases. *ACM Transactions on Storage*, 2(2):107–138, May 2006.

[24] M. Narasimha and G. Tsudik. Authentication of Outsourced Databases using Signature Aggregation and Chaining. In *DASFAA*, pages 420–436, 2006.

[25] G. Nuckolls. Verified Query Results from Hybrid Authentication Trees. In *DBSec*, pages 84–98, 2005.

[26] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *ACM SIGMOD*, pages 407–418, 2005.

[27] H. Pang and K.-L. Tan. Authenticating Query Results in Edge Computing. In *IEEE ICDE*, pages 560–571, 2004.

[28] S. Papadopoulos, Y. Yang, and D. Papadias. CADS: Continuous Authentication on Data Streams. In *VLDB*, pages 135–146, 2007.

[29] R. L. Rivest, A. Shamir, and L. M. Adleman. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. *Communications of the ACM*, 21(2):120–126, 1978.

[30] D. Salomon. Prefix Compression of Sparse Binary Strings. *Crossroads*, 6(3):22–25, 2000.

[31] SHA. *Secure Hashing Algorithm*. NIST. FIPS 180-2, 2001.

[32] The Case for Elliptic Curve Cryptography. *National Security Agency*. http://www.nsa.gov/ia/industry/crypto_elliptic_curve.cfm.