View metadata, citation and similar papers at core.ac.uk

brought to you by CORE

provided by Institutional Knowledge at Singapore Management University

Singapore Management University
Institutional Knowledge at Singapore Management University

Research Collection School Of Information Systems | School of Information Systems

5-2008

Verifying Completeness of Relational Query Answers from Online Servers

Hwee Hwa PANG
Singapore Management University, hhpang@smu.edu.sg

Kian-Lee TAN
National University of Singapore

DOI: https://doi.org/10.1145/1330332.1330337

Follow this and additional works at: https://ink.library.smu.edu.sg/sis_research


Part of the Databases and Information Systems Commons, and the Numerical Analysis and Scientific Computing Commons

Citation

PANG, Hwee Hwa and TAN, Kian-Lee. Verifying Completeness of Relational Query Answers from Online Servers. (2008). *ACM Transactions on Information and System Security*. 11, (2), 1-50. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/778

This Journal Article is brought to you for free and open access by the School of Information Systems at Institutional Knowledge at Singapore Management University. It has been accepted for inclusion in Research Collection School Of Information Systems by an authorized administrator of Institutional Knowledge at Singapore Management University. For more information, please email libIR@smu.edu.sg.

Verifying Completeness of Relational Query Answers from Online Servers

HWEEHWA PANG
Singapore Management University
and
KIAN-LEE TAN
National University of Singapore

The number of successful attacks on the Internet shows that it is very difficult to guarantee the security of online servers over extended periods of time. A breached server that is not detected in time may return incorrect query answers to users. In this article, we introduce authentication schemes for users to verify that their query answers from an online server are complete (i.e., no qualifying tuples are omitted) and authentic (i.e., all the result values are legitimate). We introduce a scheme that supports range selection, projection as well as primary key-foreign key join queries on relational databases. We also present authentication schemes for single- and multi-attribute range aggregate queries. The schemes complement access control mechanisms that rewrite queries dynamically, and are computationally secure. We have implemented the proposed schemes, and experiment results showed that they are practical and feasible schemes with low overheads.

Categories and Subject Descriptors: H.3 [**Information Systems**]: Information Storage and Retrieval; E.5 [**Data**]: Files

General Terms: Security

Additional Key Words and Phrases: query answer verification, secure database systems

H. Pang is partially supported by a research grant (C220/MSS5C002) from the Singapore Management University. K.-L. Tan is partially supported by a grant from the National University of Singapore.
Authors' addresses: H. Pang, School of Information Systems, Singapore Management University, 80 Stamford Road, Singapore 178902; email: hhpang@smu.edu.sg; K.-L. Tan, Department of Computer Science, National University of Singapore, 3 Science Dr 2, Singapore 117543.

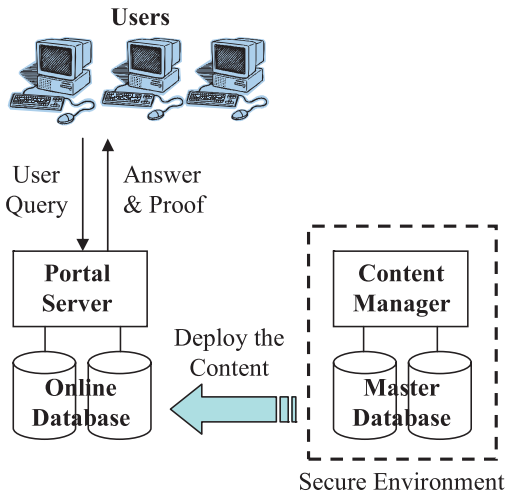


Fig. 1. System model.

1. INTRODUCTION

The amount of content on the Internet has grown tremendously over the last decade. So has the number of successful attacks on online servers. The types of attack ranged from tampering of data, to unauthorized access of sensitive information like credit card numbers and user passwords. The targets of the attacks included government, large corporations, and even e-business sites that we would expect to have been professionally administered and secured. This shows that it is very difficult to guarantee the security of online servers over extended periods of time.

In many organizations, data are created and maintained on a content management platform like Interwoven, before being deployed onto (remote) portal servers. This architecture, as depicted in Figure 1, can be exploited to improve data security as follows: the content manager can reside in a closed, secure network to ensure the integrity of the master database. In addition, authentication information is built into the database, before it is deployed onto online portal servers. With each query answer, the portal servers also generate an associated correctness proof from the authentication information. If any portal server gets compromised and returns a wrong query answer, the user would detect a mismatch with the accompanying correctness proof and be alerted.

In this paper, we consider a database that contains a table $\mathbf{R}\langle K_1, K_2, \ldots, K_d, S, A\rangle$ with key attributes $K_1, K_2, \ldots, K_d$, and A denotes the remaining attributes. We further consider two classes of queries performed by the portal servers: range selection

SELECT * FROM $\mathbf{R}$ WHERE P_1

and range aggregate queries such as

SELECT SUM(S) FROM $\mathbf{R}$ WHERE P_1 and P_2 and ... and P_c

Employee

ID	Name	Salary	Dept	Photo	...
005	A	2000	1	...	
002	C	3500	2	...	
001	D	8010	1	...	
004	B	12100	3	...	
003	E	25000	2	...	

Access Control Policies:
- Human Resource (HR) Manager sees all records
- HR Executive sees only records with Salary < 9000

Query: SELECT * FROM Emp WHERE Salary < 10000

Fig. 2. Example database.

Here P_i is a range predicate of the form $\alpha_i \leq K_i \leq \beta_i$, also denoted as $K_i[\alpha_i, \beta_i]$. Note that K_i's are the search keys used by the queries, and are not necessarily the primary keys of the table.

The objective is to devise authentication mechanisms that enable a user to verify the correctness of a relational query answer generated by an online server that is possibly compromised, without contravening any access control rules on the database. The correctness of a query answer involves two aspects:

—*Authenticity*. All the values in the answer originated from the master database; they have not been tampered with, nor have spurious records been introduced. For example, for the HR executive's query on the Employee table in Figure 2, the query server indeed returns the answer {[005, A, 2000, ..], [002, C, 3500, ..], [001, D, 8010, ..]}, and not {[005, C, 2000, ..], [002, A, 3500, ..], [001, D, 8010, ..]} (the names in the first two records have been swapped), nor {[005, A, 2000, ..], [002, C, 3500, ..], [001, D, 8010, ..], [009, X, 8050, ..]} (the last record is spurious).

—*Completeness*. Every record satisfying the query conditions is included in the answer; (e.g., the answer {[005, A, 2000, ..] , [001, D, 8010, ..]} for the HR executive's query in Figure 2 is incomplete as [002, C, 3500, ..] is omitted.)

In addition, the authentication mechanisms should satisfy the following requirements:

—*Precision*: Only records and attribute values that satisfy the conditions of each query are returned. The motivation is to avoid contravening access control rules on the database, which would become (part of) the query conditions through query rewriting.

—*Security*: It is computationally infeasible for a portal server to cheat by generating a valid proof for an incorrect query answer.

—*Efficiency*: The procedure for the portal server to generate the proof for a correct (i.e., complete and authentic) query answer has polynomial complexity. Likewise the procedure performed by the user to verify a query answer has polynomial complexity.

To the best of our knowledge, other than our earlier paper in Pang et al. [2005], only the proposals in Devanbu et al. [2000] and Narasimha and Tsudik [2006] address verification of query answer completeness for range selection queries (but without concurrently enforcing access control rules as we will explain in Section 3). Moreover, none of the existing authentication schemes can handle range aggregate queries.

In this paper, we first introduce a range selection scheme that generates, for an ordered list of records $[r_1, r_2, .., r_n]$ and a query range $[\alpha, \beta]$, cryptographic proof that the two records bordering the answer $[r_a, .., r_b]$, $1 < a \leq b < n$ indeed fall outside the query range, i.e., $r_{a-1}.K < \alpha$ and $\beta < r_{b+1}.K$. The scheme is secure in the sense that it is computationally infeasible for a breached portal server to devise such a proof for an incorrect query answer. Thus, referring to the example in Figure 2, our scheme would enable the portal server to return exactly {[005, A, 2000, ..], [002, C, 3500, ..], [001, D, 8010, ..]}, and provide proof that the next record in the table has a higher salary than stated in the query condition, without disclosing directly or indirectly what that salary amount is. Building upon the above scheme, we then present extensions for verifying general select-project queries, as well as an important class of select-project-join queries involving primary key-foreign key joins.

Following that, we present efficient authentication schemes for single- and multi-attribute range aggregate queries. Our schemes are based on a hierarchy of partial sums that satisfy range aggregate queries without requiring information from records beyond the query range. We build authentication capability in the partial sum hierarchy to facilitate verification that the answer to a range aggregate query covers all and only the records in the query range. The schemes are applicable to distributive and algebraic aggregates [Gray et al. 1997] in general.

We have implemented the proposed schemes non-intrusively in a middleware between the applications and the database server. Results from experiments conducted to evaluate the performance of our schemes show that the overhead is acceptable for the schemes to be of practical use.

Besides online servers that may become compromised over time, our authentication mechanisms are also applicable to systems in which queries are processed by untrusted servers, as in data publishing [Devanbu et al. 2000; Pang et al. 2005], outsourced databases [Hacigümüs et al. 2002, Damiani et al. 2005a; Damiani et al. 2005b], and edge computing [Pang and Tan 2004].

This article is an extension of our earlier work on authenticated range selection in Pang et al. [2005]. We have improved the scheme–the new scheme incurs $n/2$ times less computation where n is the number of records in the database table, and it ensures that brute-force attacks (by enumerating all possible attribute values) are now infeasible. Our new scheme also goes beyond the earlier work in handling null query answers. In addition, this article introduces new mechanisms for authenticating range aggregates and a performance study.

The rest of this article is organized as follows. The next section describes background on the system model and cryptographic primitives, while Section 3 covers related work. Section 4 introduces our scheme for authenticating range selection queries; the analysis of the scheme is given in Section 5. Following that, Sections 6 and 7 present our schemes for authenticating single- and multi-attribute range aggregate queries, respectively. Section 8 describes the implementation of the schemes, and presents results of a performance study. Finally, Section 9 concludes the article.

2. BACKGROUND

This section begins by defining some cryptographic primitives that are used in our solutions. Following that, we present the security threats in our system model.

2.1 Cryptographic Primitives

Our proposed solution and much of the related work are based on the following cryptographic primitives:

—One-way hash function. A one-way hash function, denoted as $h(.)$, works in one direction: it is easy to compute a hash value $h(m)$ from a pre-image m; however, it is hard to find a pre-image that hashes to a given hash value, Examples include MD5 [Rivest 1992] and SHA [2001]. We will use the terms hash, hash value, and message digest interchangeably.

—Digital signature. A digital signature algorithm is a cryptographic tool for authenticating the integrity of a signed message as well as its origin. In the algorithm, a signer keeps a private key secret and publishes the corresponding public key. The private key is used by the signer to generate digital signatures on messages, while the public key is used by anyone to verify the signatures on messages. RSA [Rivest et al. 1978] and DSA [DSS 1991] are two common signature algorithms.

—Signature aggregation. As introduced in Boneh et al. [2003], this is a multi-signer scheme that aggregates signatures generated by distinct signers on different messages into one signature. Signing a message m involves computing the message hash $h(m)$ and then the signature on the hash value. To aggregate t signatures, one simply multiplies the individual signatures so the aggregated signature has the same size as each individual signature. Verification of an aggregated signature involves computing the product of all message hashes and then matching with the aggregated signature.

—Merkle hash tree. We shall only explain the Merkle hash tree with the example in Figure 3, which is intended for authenticating data values d_1, .., d_4; a detailed definition can be found in Merkle [1989]. Each leaf node N_i is assigned a digest $h(d_i)$, where h is a one-way hash function. The value of each internal node is derived from its child nodes, (e.g., $N_{12} = h(N_1 \mid N_2)$ where $\mid$ denotes concatenation). In addition, the value of the 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 d_1, the

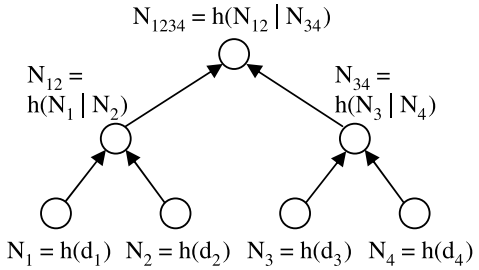


Fig. 3. Example of a Merkle hash tree.

VO contains N_2, N_{34} and the signed N_{1234}. The recipient first computes $h(d_1)$ and $h(h(h(d_1) \mid N_2) \mid N_{34})$, then checks if the latter matches the signed N_{1234}. If so, d_1 is accepted; otherwise, d_1 or some of the digests leading up to the root have been tampered with.

2.2 System and Threat Models

Figure 1 depicts our system model, which supports three distinct roles:

—The content manager maintains a master database, and deploys it with one or more associated digital signatures (e.g., RSA [Rivest et al. 1978]) onto the portal server.

—The portal server processes user queries, possibly after some query-rewriting to comply with the access control rules on the database. In general, there could be several portal servers, with some of them being situated remotely. Since the portal servers are open for external access, they could become compromised over time and a breach may not be detected immediately. Therefore the query answers that they supply to the user must be accompanied by some "correctness proof", derived from the database and signatures created by the content manager.

—The user issues queries to the portal server. To verify the query answers against their respective correctness proofs, the user obtains the public key of the organization through an authenticated channel, such as a public key certificate issued by a certificate authority.

There are several security considerations in the above model. Given that the portal server may become compromised, one concern is privacy of the data. Obviously, an adversary who gains access to the operating system or hardware of the portal server may be able to browse through the database, or make illegal copies of the data. Solutions to mitigate this concern include encryption (e.g., EFS, PGPdisk, Hacigümüs et al. [2002], Li and Omiecinski [2005]) and steganographic storage (e.g., Anderson et al. [1998], Pang et al. [2003], DriveCrypt), and are orthogonal to our work here.

Another concern relates to user authentication and access control, in specifying what actions each user is permitted to perform. Those issues have been studied extensively (e.g., Chokani [1992], Neuman and Tso [1994], Sandhu and

Samarati [1994]). Any query answer verification mechanism must not compromise the access control policy.

Our primary concern addressed in this article is the threat that a compromised portal server may return incorrect query answers to the users. An adversary who is cognizant of the data organization of the online database may attempt to make logical alterations to the data, thus inducing incorrect query answers; an example is to illegally effect fund transfers between two accounts. Even if the data organization is hidden, for example through data encryption or steganographic schemes [Anderson et al. 1998; Pang et al. 2003], the adversary may still sabotage the database by overwriting physical pages within the storage volume. In addition, a compromised portal server could be made to return incomplete query answers by withholding data intentionally. Therefore mechanisms for users to verify the correctness of their query answers are essential here.

3. RELATED WORK

Our primary concern in this article is to address the threat that a compromised portal server may return incorrect answers to relational queries. As such, we shall review only existing work on authenticating answers of SQL queries and techniques for processing range aggregate queries.

3.1 Range Selection

Authenticating query answers from untrusted databases has been an active area of research in the past few years. Here we shall review only those works that include verification of the completeness of answers produced by range selection queries. Many other works on query authentication like Mykletun et al. [2004], Pang and Tan [2004], and Ma et al. [2005] are skipped because they do not address the completeness concern, meaning they are unable to detect if answers have been dropped by the query server.

In Devanbu et al. [2000], Devanbu et. al. proposed an authentication scheme that builds a Merkle Hash Tree (MHT) over a sorted list of data. For a range selection $K[\alpha, \beta]$ whose answer is $[r_a, .., r_b]$, the server returns $[r_{a-1}, r_a, .., r_b, r_{b+1}]$ where $r_{a-1}.K < \alpha \leq r_a.K$, $r_b.K \leq \beta < r_{b+1}.K$, together with neighboring digests leading up to the root of the MHT. The MHT proves that the records in the answer set are also contiguous in the database. The user can verify that the answer is complete since the values bounding the answer are also disclosed, though this may violate access control rules. This work was extended in Devanbu et al. [2003] to multi-attribute range selections. The extended scheme builds an MHT over the first attribute, then expands each of its leaf nodes into an MHT over the second attribute, and so on. Martel et al. [2004] generalized the scheme to Search DAGs, which encompass multidimensional range trees, tries, and skip lists. All of these schemes do not support authentication of aggregates.

Li et al. [2006] extended the MHT approach to an Embedded Merkle B-tree (EMB-tree). The EMB-tree constructs an embedded MHT over the data within each page, then an upper MHT over the root digests of the embedded MHTs.

Only the upper MHT is materialized; the intra-page MHTs are reconstituted on-the-fly during VO computation. Experiments showed that EMB-trees support query processing very efficiently.

Deviating from the MHT approach, we developed a signature chain scheme that verifies the completeness of answers of range queries [Pang et al. 2005]. The scheme embodies two key ideas: first, each tuple is given a signature derived from its own digest and those of its left and right neighbors, so it is not possible to omit a tuple from the answer while still satisfying the neighbors' signatures. To reduce computation and traffic overheads, signatures of the result tuples can be aggregated. Second, a mechanism for verifying the boundary conditions was provided. In Cheng et al. [2006], we combined the signature chain with a spatial index structure for authenticating multi-attribute range selections. That authentication scheme is still unable to handle aggregates. In addition, the signature chain is computationally expensive and may be susceptible to brute-force attacks to reveal the tuples bounding the answers, as we will explain in the course of developing the solution in Section 4. The present paper offers an improved scheme that is secure and more efficient.

Another recent work is Narasimha and Tsudik [2006], which addresses completeness of range selection queries, projections, joins, and set operation queries. The solution proposed there is also based on signature aggregation and chaining. That work also does not address the simultaneous enforcement of access control rules on the database, neither does it handle range aggregation.

3.2 Range Aggregates

There have been several studies on efficiently evaluating range aggregate queries, particularly in the context of OLAP systems. However, they are not directly applicable to our system model, as they either leak extra information or they do not support authentication of query answers.

The classic HAMS scheme in Ho et al. [1997] for OLAP range sum works as follows. Consider a data cube with d dimensions, and n_j elements along dimension j. $S[i_1, i_2, ..., i_d]$ denotes the value in cell $[i_1, i_2, ..., i_d]$, and Region($l_1 : h_1, l_2 : h_2, ..., l_d : h_d$) denotes the d-dimensional space bounded by $l_j \leq i_j \leq h_j$ along every dimension j. HAMS uses a d-dimensional array P of size $n_1 \times n_2 \times ... \times n_d$ to store precomputed prefix sums:

$$\begin{aligned} P[x_1, x_2, ..., x_d] &= \text{SUM}(0 : x_1, 0 : x_2, ..., 0 : x_d) \\ &= \sum_{i_1=0}^{x_1} \sum_{i_2=0}^{x_2} ... \sum_{i_d=0}^{x_d} S[i_1, i_2, ..., i_d] \end{aligned}$$

At runtime, any range sum can be computed from P as:

$$\text{SUM}(l_1 : h_1, l_2 : h_2, ..., l_d : h_d) = \sum_{\forall x_j \in \{l_j - 1, h_j\}} \left\{ \left(\prod_{i=1}^{d} s(i) \right) \times P[x_1, x_2, ..., x_d] \right\},$$

$\forall j \in \{1, 2, ..., d\}$

where

$$s(j) = \begin{cases} 1 & \text{if } x_j = h_j \\ -1 & \text{if } x_j = l_j - 1. \end{cases}$$

For example, when $d = 2$, the range sum is obtained as: $P[h_1, h_2] - P[h_1, l_2 - 1] - P[l_1 - 1, h_2] + P[l_1 - 1, l_2 - 1]$.

The HAMS scheme produces overheads and query answer sizes that are proportional to 2^d. Moreover, $2^d - 1$ of the prefix sums in the answer relate to records that are outside of the query scope, and thus constitute data leakage. Subsequent extensions to the prefix sum approach, (e.g., Chun et al. [2001] and Geffner et al. [2000]), have focused on reducing update overheads but not authentication considerations. Similarly, aggregate trees exist for spatial and spatio-temporal databases, (e.g., aR-tree [Papadias et al. 2001], aRB-tree [Papadias et al. 2002] and BA-tree [Zhang et al. 2002]); again, authentication of the aggregate result is not addressed.

The existing work that is most relevant to this paper is Przydatek et al. [2003], which describes an authentication scheme for sensor networks in which aggregators are entrusted with summarizing sensor data for a home server. In the specific case of an "average" operation, the aggregator A commits to the sensor data collection, then reports its average $\overline{x}$ to the home server H. In addition, A sorts the sensor data and commits to the sorted list as well. H then tests whether the two lists contain the same elements through sampling. If the test succeeds, A sends to H the frequency count of each sensor value (or each range of values). Finally, H computes the average directly from the frequency counts, and compares it with $\overline{x}$. This solution does not address our problem of aggregate authentication: (1) Our system model includes a secure content manager that can certify the dataset. There is no necessity, nor is it practical to maintain two lists of the dataset for counterchecking. (2) The frequency counts constitute a histogram of the data distribution. For a large dataset, the histogram has to be pregenerated, and that allows a breached query server to determine the maximum error it can introduce while eluding detection. (3) A large dataset is likely to reside on disk, in a series of physical blocks. Spot checks against the sorted list generate random I/Os and are expensive.

4. RANGE SELECTION QUERIES

We begin this section by elaborating on why access control imposes the *precision* requirement described in the Introduction. Following that, we introduce our signature chain scheme for authenticating range selection queries.

4.1 Access Control Requirement

Access control has been an integral mechanism in commercial database systems. At the most rudimentary level, access control may be defined on entire tables or columns. To achieve finer-grained control on which records can be accessed by which users, in theory it is possible to create views for specific user groups. However, this approach is not scalable to large numbers of user groups. Instead, modern database systems support fine-grained access control by dynamically modifying the user queries.

An example is the Virtual Private Database (VPD) feature of Oracle's *9i*R2 DBMS [Oracle VPD 2002]. VPD encodes the authorization policy into functions defined on each relation. Those functions, in conjunction with the user/application context, are used to generate *where* clause predicates to be appended to the user query before it is executed. The added predicates ensure that the user receives only those records in the table or view that are permitted by the authorization policy.

The above practice of dynamically rewriting user query points to the need for a solution that generates correctness proof for range queries, without divulging extra information like boundary records.

4.2 Basic Signature Chain Scheme

4.2.1 *Problem Definition.* Suppose the content manager has a sorted list of records $\mathbf{R} = [r_1, .., r_n]$, each record having the fields $\langle K, A \rangle$ where K is the sort key such that $L < K < U$ for some lower bound L and upper bound U, and A denotes the remaining record attributes. (If the key allows duplicate values, the records can be disambiguated by appending a replica number, so that the $r_i.K|repl\#$ values are distinct.) Now a user submits a query (to the portal server) for the records that have key values between α and β, i.e., $\sigma_{\alpha \leq r.K \leq \beta}(\mathbf{R})$. The server needs to prove to the user that the answer $\mathbf{Q} = [r_a, .., r_b]$ is correct.

In our proposed range selection authentication scheme, each record has an associated digital signature that is used to verify that the answer $\mathbf{Q}$ for a query is complete. The following two conditions together ensure completeness:

—*Contiguity*. Each pair of successive entries in $\mathbf{Q}$ also appear consecutively in $\mathbf{R}$. This can be checked by making the signature of each record in $\mathbf{R}$ dependent on the key value of its immediate left and right neighbors.

—*Correct boundaries*. r_a and r_b are the first and last records, respectively, in $\mathbf{R}$ that satisfy the query condition. Conceptually, we make the signature of r_a dependent on $h(r_a.K - r_{a-1}.K)$ where h is an additive hash function such that $h(x + y) = h(x) \circ h(y)$ for some operator $\circ$. To prove $r_{a-1}.K < \alpha$ without disclosing r_{a-1}, the portal server returns $h(\alpha - r_{a-1}.K)$, and the user then combines it with $h(r_a.K - \alpha)$ to obtain $h(r_a.K - r_{a-1}.K)$ for matching with the signature. As long as $h(x)$ for $x \leq 0$ is either undefined or computationally infeasible to derive, the portal server would not be able to cheat by returning a legitimate $h(\alpha - r_{a-1}.K)$ if in fact $r_{a-1}.K \geq \alpha$. A similar technique enables the verification of $r_{b+1}.K > \beta$.

Furthermore, to check the authenticity of the query answer, the attribute values of each record r_i in $\mathbf{R}$ also contribute to its signature. We detail below a scheme for integer keys; generalization to other attribute types is addressed in Section 4.5.

4.2.2 *Preparation.* To simplify the solution, the content manager inserts two boundary records r_0 and r_{n+1}, where $r_0.K = L$ and $r_{n+1}.K = U$,so the sorted list becomes $\mathbf{R} = [r_0, r_1, .., r_n, r_{n+1}]$. We assume that L and U are known to everyone.

As explained earlier, for checking boundaries $r_{a-1}.K < \alpha$ and $r_{b+1}.K > \beta$, the record signature has to incorporate the output of an additive hash function on the difference between $r_a.K$ and $r_{a-1}.K$, and between $r_b.K$ and $r_{b+1}.K$. Unfortunately, there is as yet no known algebraic function satisfying the additive property for which there is no simple way to derive the inverse function $h(x)$ for $x \leq 0$. The existence of the inverse function allows a breached server to return $h(\alpha - r_{a-1}.K)$ and $h(r_{b+1}.K - \beta)$ even for $r_{a-1}.K \geq \alpha$ and $r_{b+1}.K \leq \beta$, thus breaking the security of our scheme.

In the absence of a suitable algebraic function, we have to use an iterative hash function instead. Thus, the content manager creates the signature for record r_i, $1 \leq i \leq n$, as:

$$sig(r_i) = \; s(h(h^{r_i.K - r_{i-1}.K}(r_{i-1}) \mid h^{r_{i+1}.K - r_i.K}(r_{i+1}) \mid \text{MHT}(r_i))) \tag{1}$$

where s is a signature function using the portal's private key, $h^x(r_i)$ applies a collision-resistant hash function h on (the concatenation of) all the attribute values of record r_i iteratively for x times, and MHT(r_i) is the root digest of the Merkle hash tree over the attributes (including K) of r_i. The MHT enables the record to be authenticated, even when some attributes are omitted from the result.

By using all the record attributes in the input argument for the hash function, a brute-force enumeration of all possible arguments would incur a cost in the order of $2^{|r|}$, where $|r|$ is the record size in bits. At current hardware speed, $|r|$ = 1,024 bits would be sufficient to render the brute-force attack computationally infeasible. In case the actual attributes total less than 1,024 bits, we could simply pad each record with a randomly generated string, at the expense of some storage overhead.

Furthermore, for the boundary records r_0 and r_{n+1}:

$$sig(r_0) = s(h^{r_1.K - L}(r_1))$$
$$sig(r_{n+1}) = s(h^{U - r_n.K}(r_n))$$

Equation (1) has two important differences from our earlier formulation in Pang et al. [2005]. Previously, we had defined the leftmost component as $h^{U - r_{i-1}.K - 1}(.)$; on the average, that incurs $(U - L)/2$ iterative hashing operations. The current formulation, $h^{r_i.K - r_{i-1}.K}(.)$, requires only $(U - L)/n$ iterative hashes where n is the number of records in the database table. From the second component, $h^{r_{i+1}.K - r_i.K}(.)$, we realize a similar saving. Another difference is that in equation (1) the iterative hash functions are applied on r_{i-1} and r_{i+1}, rather than on r_i itself; the significance of this will become evident shortly after we describe the operations carried out in verifying $sig(r_a)$ and $sig(r_b)$.

4.2.3 *Query Processing.* Along with the query answer $\mathbf{Q} = [r_a, .., r_b]$, the server returns the associated record signatures, and intermediate digests $h^{\alpha - r_{a-1}.K}(r_{a-1})$ and $h^{r_{b+1}.K - \beta}(r_{b+1})$.

4.2.4 *Answer verification.* Finally, the user checks the signature of each record in $\mathbf{Q}$ as follows.

For $sig(r_a)$:

(1) Hash the intermediate digest $h^{\alpha - r_{a-1}.K}(r_{a-1})$, $(r_a.K - \alpha)$ more times to produce $h^{r_a.K - r_{a-1}.K}(r_{a-1})$.
(2) Compute $h^{r_{a+1}.K - r_a.K}(r_{a+1})$ from the value of r_a and r_{a+1} in $\mathbf{Q}$.
(3) Compute MHT(r_a) from the attribute values of r_a in $\mathbf{Q}$.
(4) Check whether:

$$s^{-1}(sig(r_a)) \text{ ?= } h(h^{r_a.K - r_{a-1}.K}(r_{a-1}) \mid h^{r_{a+1}.K - r_a.K}(r_{a+1}) \mid \text{MHT}(r_a))$$

where $s^{-1}(.)$ decrypts its argument with the portal's public key. If so, r_a is indeed the first record that satisfies $\alpha \leq r.K$, and r_a has not been tampered with; otherwise the answer $\mathbf{Q}$ is wrong.

For $sig(r_i)$, $a < i < b$:

(1) Compute $h^{r_i.K - r_{i-1}.K}(r_{i-1})$ from the value of r_{i-1} and r_i in $\mathbf{Q}$.
(2) Compute $h^{r_{i+1}.K - r_i.K}(r_{i+1})$ from the value of r_i and r_{i+1} in $\mathbf{Q}$.
(3) Compute MHT(r_i) from the attribute values of r_i in $\mathbf{Q}$.
(4) Check whether:

$$s^{-1}(sig(r_i)) \text{ ?= } h(h^{r_i.K - r_{i-1}.K}(r_{i-1}) \mid h^{r_{i+1}.K - r_i.K}(r_{i+1}) \mid \text{MHT}(r_i))$$

For $sig(r_b)$:

(1) Compute $h^{r_b.K - r_{b-1}.K}(r_{b-1})$ from the value of r_{b-1} and r_b in $\mathbf{Q}$.
(2) Hash the intermediate digest $h^{r_{b+1}.K - \beta}(r_{b+1})$, $(\beta - r_b.K)$ more times to produce $h^{r_{b+1}.K - r_b.K}(r_{b+1})$.
(3) Compute MHT(r_b) from the attribute values of r_b in $\mathbf{Q}$.
(4) Check whether:

$$s^{-1}(sig(r_b)) \text{ ?= } h(h^{r_b.K - r_{b-1}.K}(r_{b-1}) \mid h^{r_{b+1}.K - r_b.K}(r_{b+1}) \mid \text{MHT}(r_b))$$

We can now explain why the iterative hash functions are applied to r_{i-1} and r_{i+1} in equation (1). If the first iterative hash function there is applied on r_a instead, the server would have to return the intermediate digest $h^{\alpha - r_{a-1}.K}(r_a)$ to enable the user to check against $sig(r_a)$. As the user knows the value of α, and as r_a is returned to the user as part of the query answer, the user can simply see how many times he has to hash r_a to get a match with the intermediate digest $h^{\alpha - r_{a-1}.K}(r_a)$, and from there calculate $r_{a-1}.K$. Thus the key value of r_{a-1} is compromised. Similarly, the key value of r_{b+1} would be compromised if the iterative hash function is applied on r_b rather than r_{b+1} as in equation (1). This is exactly why our earlier scheme in Pang et al. [2005] is susceptible to brute-force attack; the new formulation in this paper eliminates that vulnerability.

4.3 Null Query Answer

As we will show in the completeness analysis shortly, with the basic signature scheme above the portal server cannot drop some valid values or introduce tampered values without being detected by the user. However, we still need extra provisions for checking null query answers. These provisions necessitate

a shadow table for each data table, and should be enabled only if users demand proof for null query answers.

Case 1: $\mathbf{Q} = \emptyset$ because $\beta < r_1.K$.

To prove that $\mathbf{Q}$ is correct, the portal server returns $sig(r_0)$ and the intermediate digest $h^{r_1.K-\beta}(r_1)$. The user hashes the intermediate digest $(\beta - L)$ more times to produce $h^{r_1.K-L}(r_1)$. If and only if it matches $s^{-1}(sig(r_0))$, β is smaller than $r_1.K$ and the null answer is correct.

Case 2: $\mathbf{Q} = \emptyset$ because $r_n.K < \alpha$.

To prove that $\mathbf{Q}$ is correct, the portal server returns $sig(r_{n+1})$ and the intermediate digest $h^{\alpha-r_n.K}(r_n)$. The user hashes the intermediate digest $(U - \alpha)$ more times to produce $h^{U-r_n.K}(r_n)$. If and only if it matches $s^{-1}(sig(r_{n+1}))$, α is larger than $r_n.K$ and the null answer is correct.

Case 3: $\mathbf{Q} = \emptyset$ because $r_i.K < \alpha \leq \beta < r_{i+1}.K$ for some $1 \leq i < n$.

A straightforward first attempt for proving $\mathbf{Q}$ in this case is for the portal server to return the signature $sig(r_i)$, the digest $h^{r_i.K-r_{i-1}.K}(r_{i-1})$, the intermediate digest $h^{r_{i+1}.K-\beta+\alpha-r_i.K}(r_{i+1})$, and the digest MHT$(r_i)$. The user will hash the intermediate digest $(\beta - \alpha)$ more times to derive $h^{r_{i+1}.K-r_i.K}(r_{i+1})$, then check against the signature $sig(r_i)$. Unfortunately, the portal server could just as well supply the signature and digests for any $1 \leq j < n$, $i \neq j$, such that $r_{j+1}.K - r_j.K > \beta - \alpha$, and the user verification would still succeed. Thus the first attempt is not secure.

To handle the present case securely, we need to introduce a shadow table $\mathbf{T} = [t_1, .., t_{n-1}]$, each shadow record t_i comprising a key K and a signature. $t_i.K$ is set to any value in between those of data records r_i and r_{i+1}, i.e., $r_i.K < t_i.K < r_{i+1}.K$ for $1 \leq i < n$. The signature of t_i is computed as:

$$sig(t_i) = s(h(h^{t_i.K-r_i.K}(r_i) \mid h^{r_{i+1}.K-t_i.K}(r_{i+1}) \mid h(t_i.K))) \tag{2}$$

For integer domain, we simply leave $sig(t_i)$ undefined whenever $r_i.K = r_{i+1}.K$ or $r_i.K + 1 = r_{i+1}.K$. The reason is that those conditions are incongruent with $r_i.K < \alpha \leq \beta < r_{i+1}.K$.

Along with the query answer, the portal server returns $t_i.K$, $sig(t_i)$, and the following digests:

if $(\beta \leq t_i.K)$, then
  return $h^{\alpha-r_i.K}(r_i)$, $h^{\beta-r_i.K}(r_i)$ and $h^{r_{i+1}.K-t_i.K}(r_{i+1})$;
else if $(t_i.K \leq \alpha)$, then
  return $h^{t_i.K-r_i.K}(r_i)$, $h^{r_{i+1}.K-\alpha}(r_{i+1})$ and $h^{r_{i+1}.K-\beta}(r_{i+1})$;
else // $\alpha < t_i.K < \beta$
  return $h^{\alpha-r_i.K}(r_i)$ and $h^{r_{i+1}.K-\beta}(r_{i+1})$;

Finally, the user combines the digests and checks whether they match the signature:

$$s^{-1}(sig(t_i)) \;?=\; h(h^{t_i.K-r_i.K}(r_i) \mid h^{r_{i+1}.K-t_i.K}(r_{i+1}) \mid h(t_i.K)) \tag{3}$$

Specifically, the digests are combined as follows:

if $(\beta \leq t_i.K)$, then
  hash $h^{\alpha-r_i.K}(r_i)$ $(t_i.K - \alpha)$ times to get $h^{t_i.K-r_i.K}(r_i)$, then check equation (3);

hash $h^{\beta - r_i.K}(r_i)$ ($t_i.K - \beta$) times to get $h^{t_i.K - r_i.K}(r_i)$, then check equation (3);
else if ($t_i.K \leq \alpha$), then
hash $h^{r_{i+1}.K - \alpha}(r_{i+1})$ ($\alpha - t_i.K$) times to get $h^{r_{i+1}.K - t_i.K}(r_{i+1})$, then check equation (3);
hash $h^{r_{i+1}.K - \beta}(r_{i+1})$ ($\beta - t_i.K$) times to get $h^{r_{i+1}.K - t_i.K}(r_{i+1})$, then check equation (3);
else // $\alpha < t_i.K < \beta$
hash $h^{\alpha - r_i.K}(r_i)$ ($t_i.K - \alpha$) times to get $h^{t_i.K - r_i.K}(r_i)$, hash $h^{r_{i+1}.K - \beta}(r_{i+1})$ ($\beta - t_i.K$) times to get $h^{r_{i+1}.K - t_i.K}(r_{i+1})$, then check equation (3).

4.4 Generalization to SPJ Queries

Having presented the signature chain scheme for range selection queries, we now show that the scheme extends to other relational operations, specifically projection and some subclasses of join. Queries with range selections on multiple attributes can be supported by constructing a signature chain on the concatenated attribute values. An alternative is to combine the signature chain with a spatial index structure; we have studied this approach in Cheng et al. [2006].

4.4.1 *Multipoint Queries.* So far, we have considered range selection where the result tuples occupy a contiguous range within the table. In the case of a query that contains selection attribute(s) other than K, (e.g., "SELECT * FROM Emp WHERE Salary < 10000 AND Dept = 1" on the table in Figure 2), or a query with multiple WHERE clauses on the sort key, the answer could comprise several distinct intervals of records. We call this a multipoint query.

In general, the answer for a multipoint query can be treated as a range of contiguous records on K, some of which satisfy the query conditions while others should be filtered out. Consider a filtered record r_i within the result range, (e.g., [002, C, 3500, 2, ..]) in Figure 2.

—Case 1. The access control policy permits the user to see r_i. The portal server returns the attribute value(s) that fails the query condition, (i.e., "r_i.Dept = 2" in the above example, plus the key value "r_i.Salary = 3500") and digests for the remaining attribute values of r_i. This enables the user to match the signatures of r_{i-1}, r_i and r_{i+1} using equation (1). The overhead of this solution is proportional to the number of filtered records.

—Case 2. The access control policy prohibits the user from seeing r_i, so his query is rewritten to filter out r_i. Unlike Case 1, here the portal server cannot return any of the actual attributes of r_i. One solution is to introduce additional columns to the table, one column for each user group in the access model to indicate whether individual records are visible to that user group.
For example, consider an access model with user clearance levels of "secret," "confidential," and "unclassified." Three binary attributes would be added to the example table in Figure 2 to indicate whether each record can be seen by users with "secret," "confidential," and "unclassified" clearances, respectively. Furthermore, an artificial ordinal attribute is introduced as the

sort key in place of Salary; the ordinal key preserves the record ordering on Salary, but hides the actual Salary figures. Now, for a filtered record r_i that is shielded from a user with only "confidential" clearance, the portal server could return "r_i.confidential = No", plus the ordinal key value and digests for the other attributes of r_i. The user can then compute equation (1) for matching with the signatures of r_{i-1}, r_i and r_{i+1}. This solution reveals the total number of records that fall within the result range on K, but hides the actual attribute values of the filtered records.

4.4.2 *Selection-Projection Queries.* These queries involve the projection operation $\pi_{K,A_1,..,A_p}(\mathbf{R}) = \{\langle r.K, r.A_1, .., r.A_p\rangle \mid r \in \mathbf{R}\}$ where A_i's are attributes of relation $\mathbf{R}$, sorted on K.

The projection operation can filter out any or all of the attributes of $\mathbf{R}$, except for K which the user needs in order to test the query result for completeness. Unlike the scheme in Devanbu et al. [2000], we do not want the portal server to return to the user any attribute values in the result tuples that should be filtered out, so as to avoid disclosing sensitive columns and compromising access control rules. Another reason is that some of the omitted attribute values could be very large, (e.g., BLOBs), so sending them to the user would incur space and transmission overheads unnecessarily.

Our scheme allows unwanted attribute values to be removed at the portal server. Since MHT(r) in equation (1) is defined as the root digest of a Merkle Hash Tree on the attribute values of record r, the portal server can provide the digest in place of the actual value for those attributes that are projected out, so the user can still compute MHT(r) without the actual values.

Another issue to consider here is the handling of duplicates in the result $\mathbf{Q}$. For some queries, the user may want to retain the duplicates, (e.g., for the computation of SUM and AVG). For other queries, the user may require the portal server to perform duplicate elimination by specifying the keyword DISTINCT. In the former case, the MHT(r) component in equation (1) enables the user to uniquely identify each duplicate, so the portal server cannot omit some duplicates without being detected. In the latter case where duplicates are not needed, our scheme still requires the portal server to present MHT(r) and the signature $sig(r)$ for each eliminated duplicate r to enable all the signatures for $\mathbf{Q}$ to be checked. Admittedly, this discloses the number of data records underlying each result tuple, which may be undesirable in some situations.

4.4.3 *Selection-Projection-Join Queries.* These queries involve the join operation $\mathbf{R} \bowtie_C \mathbf{S}$ where C is a condition of the form $A_i \ \Theta \ A_j$, A_i, and A_j are attributes of relations $\mathbf{R}$ and $\mathbf{S}$ respectively, and $\Theta \in \{=, \neq, <, \leq, >, \geq\}$.

Our scheme uses signatures on the key attribute of a relation to generate proof of the completeness of query results from that relation. Therefore the scheme may not work for ad-hoc joins on arbitrary attributes in general. However, primary key-foreign key joins, an important class of join operations, can be supported as follows.

Consider $\mathbf{R}.A_i = \mathbf{S}.A_j$, where A_i is a foreign key attribute in $\mathbf{R}$ and A_j is the corresponding primary key in $\mathbf{S}$. Referential integrity constraint mandates

that every instance in $\mathbf{R}.A_i$ must have a matching entry in $\mathbf{S}.A_j$. Consequently, joining with $\mathbf{S}.A_j$ in itself does not cause any instance in $\mathbf{R}.A_i$ to drop out of the query result, so we need only deal with selection operations on $\mathbf{R}.A_i$ or $\mathbf{S}.A_j$. This can be achieved by ordering $\mathbf{R}$ on A_i at the content manager's master database, and constructing signatures for this sort order. After a join operation, the user checks the completeness of the result with respect to $\mathbf{R}.A_i$, possibly taking into account any selection conditions on $\mathbf{R}.A_i$ or $\mathbf{S}.A_j$, as with Select-Project queries.

Another class of joins that can be supported is $\mathbf{R}.A_i \leq \mathbf{S}.A_j$, where completeness of the join result can be checked using the techniques presented earlier:

—Let the first entry in the ordered $\mathbf{R}$ partition of the join result be $\min(\mathbf{R}.A_i)$, and the last entry in the ordered $\mathbf{S}$ partition be $\max(\mathbf{S}.A_j)$.

—Verify that the $\mathbf{R}$ partition contains all $r \in \mathbf{R}$ satisfying $L < r.A_i \leq \max(\mathbf{S}.A_j)$.

—Verify that the $\mathbf{S}$ partition contains all $s \in \mathbf{S}$ satisfying $\min(\mathbf{R}.A_i) \leq s.A_j < U$.

4.5 Generalization to Other Attribute Types

4.5.1 *Floating Point Attributes.* To extend our signature chain beyond the integer domain, we observe that a floating point attribute t is represented as a mantissa t_m and an exponent t_e ($t = t_m * 2^{t_e}$), where $0.5 \leq t_m < 1$ or $-1 < t_m \leq -0.5$. The inequality $t \geq c$ for some value $c = c_m * 2^{c_e}$ translates to one of the following conditions:

—$c_m < 0 \leq t_m$;

—$(c_m < 0)$ and $(t_m < 0)$ and $((t_e < c_e)$ or $((t_e = c_e)$ and $(t_m > c_m)))$; or

—$(c_m \geq 0)$ and $(t_m \geq 0)$ and $((t_e > c_e)$ or $((t_e = c_e)$ and $(t_m > c_m)))$.

Floating point attributes can thus be supported with two signature chains, on the integer mantissa and on the integer exponent.

4.5.2 *Character String Attributes.* Another common attribute type in relational databases is the character string. Equality match for character strings, e.g., name = "John Smith", can be supported easily, by hashing the string value to an integer digest before applying the signature chain on the digest. In contrast, ad-hoc prefix matches, e.g., name $\geq$ "John*", entail sorting the table on that string attribute, and building a signature chain on each character position; the attribute in each record can then be compared with the string value given in the query, character by character.

4.5.3 *When to Build Signature Chains.* In theory, a relation could have several signature chains, one on each sort order of the relation. As the signature chains incur storage and update overheads, they should be applied discreetly, on only the interesting sort orders (as defined by the WHERE clauses in expected queries for which correctness proofs are needed). This is analogous to building indexes only on interesting sort orders, rather than all attribute

permutations of a table. At a finer granularity, it is possible to design the portal server and client software to skip the proof generation and verification protocol for selected queries that do not require proofs, even when an underlying signature chain is available.

5. ANALYSIS OF SIGNATURE CHAIN SCHEME

5.1 Completeness Analysis

The inclusion of MHT(r_i) in the record signature $sig(r_i)$ ensures that the record content is authentic. Hence we need only consider whether the contiguity and boundary criteria are enforced properly by our authentication scheme. We now examine the various cases where a compromised server may attempt to violate those two criteria, and show in each case how the attempt at cheating cannot succeed:

—Case 1. The portal server omits a qualified record r_{a-1} that satisfies $r_{a-1}.K \geq \alpha$. Since $\alpha - r_{a-1}.K \leq 0$, $h^{\alpha - r_{a-1}.K}(r_{a-1})$ is undefined, and it is computationally infeasible for the portal server to find a replacement to which the user can further hash $(r_a.K - \alpha)$ times to get $h^{r_a.K - r_{a-1}.K}(r_{a-1})$.

—Case 2. The portal server omits a qualified record r_{b+1} that satisfies $r_{b+1}.K \leq \beta$. Similar to Case 1.

—Case 3. The portal server claims that $r_n.K < \alpha$ and returns $\mathbf{Q} = \emptyset$. Here the portal server is supposed to return $sig(r_{n+1})$ and $h^{\alpha - r_n.K}(r_n)$ so that the user can derive $h^{U - r_n.K}(r_n)$. If in fact $r_n.K \geq \alpha$, the portal server would not be able to produce $h^{\alpha - r_n.K}(r_n)$ as it is undefined or computationally infeasible to derive.

—Case 4. The portal server claims that $r_1.K > \beta$ and returns $\mathbf{Q} = \emptyset$. Similar to Case 3.

—Case 5. The portal server claims that $r_i.K < \alpha \leq \beta < r_{i+1}.K$ for some i and returns $\mathbf{Q} = \emptyset$. In this case, the portal server is supposed to return some digests corresponding to the shadow record t_i, along with its signature. If it is not true that $r_i.K < \alpha \leq \beta < r_{i+1}.K$, the required digests $h^{\alpha - r_i.K}(r_i)$/ $h^{r_{i+1}.K - \alpha}(r_{i+1})$ and $h^{\beta - r_i.K}(r_i)$/ $h^{r_{i+1}.K - \beta}(r_{i+1})$ would be undefined or computationally infeasible to produce.

—Case 6. The portal server returns $\mathbf{Q} = [r_a, .., r_i, r_j, .., r_b]$ where $i + 1 < j$; in other words, the entries in $\mathbf{Q}$ are not contiguous in $\mathbf{R}$ and one or more entries between r_i and r_j are omitted. The query answer causes the user to use $h^{r_j.K - r_i.K}(r_j)$, rather than the correct $h^{r_{i+1}.K - r_i.K}(r_{i+1})$, in checking $sig(r_i)$. This cannot succeed, otherwise there is a collision in the collision-resistant hash function h.

—Case 7. $\mathbf{Q} = [r_a, .., r_i, r_j, r_k, .., r_b]$ but $r_j \notin \mathbf{R}$; in other words, the portal server introduces a spurious entry in the query answer. To deceive the user, the portal server would need a valid signature for r_j, which can only be generated by the content manager.

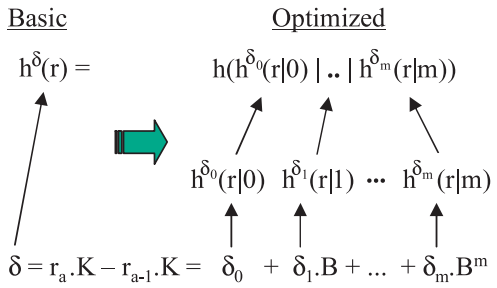


Fig. 4. Reduction in hash operations.

5.2 Complexity Analysis

The overheads for transmitting and verifying a signature for every $r_i \in \mathbf{Q}$ can be very large, especially if the communication bandwidth or processing capability of the user is limited. To lighten these overheads, the portal server can combine the signatures associated with individual entries in the answer $\mathbf{Q}$ into one aggregated signature, using the techniques proposed in Boneh et al. [2003]. This optimization also helps the user to cut down to just one signature verification operation per query answer. Thus the primary cost consideration now is the number of hashing operations during verification.

For each $sig(r_i)$ in the answer verification procedure in Section 4.2.1, step 3(a) incurs an average of $(U - L)/n$ iterative hash operations, where n is the number of records in the database table; likewise step 3(b) incurs $(U - L)/n$ iterative hash operations on the average. In fact, each $r_i.K$ in the query answer contributes to $sig(r_{i-1})$ as $h^{r_i.K-r_{i-1}.K}(r_i.K)$ and to $sig(r_{i+1})$ as $h^{r_{i+1}.K-r_i.K}(r_i.K)$, so the computation of the two digests can be shared. This cuts the combined number of hash operations for steps 3(a) and 3(b) for each $sig(r_i)$ to between $(U - L)/n$ and $2(U - L)/n$. Assuming a fan-out of 2 for the Merkle hash tree, step 3(c) requires another $2p - 1$ hash operations, where p is the number of record attributes. Therefore the expected total number of hash operations is at most $(b - a + 1)(2(U - L)/n + 2p - 1)$.

5.3 Reduction in Hashing Operations

For large key domains like LONG integers, $2(b - a + 1)(2(U - L)/n + 2p - 1)$ hash operations could amount to significant computation overheads. To reduce these overheads, we observe that in general any number $\delta \in [0, U - L)$ can be represented by a polynomial:

$$\delta = \delta_0 + \delta_1.B + \delta_2.B^2 + .. + \delta_m.B^m$$

with number base $B > 1$ and $m \geq \lceil log_B(U - L) \rceil$. For example, $B = 2$ yields the binary representation, and $B = 10$ gives the decimal representation. If $0 \leq \delta_i < B, \forall\ 0 \leq i \leq m$, we say the polynomial is the *canonical* representation of δ.

This observation can be exploited to optimize our authentication scheme, as depicted in Figure 4: The content manager replaces $h^{r_a.K-r_{a-1}.K}(r_{a-1})$ in

equation (1) with $h(h^{\delta_{t,0}}(r_{a-1}|0) \mid h^{\delta_{t,1}}(r_{a-1}|1) \mid .. \mid h^{\delta_{t,m}}(r_{a-1}|m))$, where $\delta_t = r_a.K - r_{a-1}.K = \delta_{t,0} + \delta_{t,1}.B + \delta_{t,2}.B^2 + .. + \delta_{t,m}.B^m$, $r_{a-1}|1$ concatenates record r_{a-1} with the number 1, and so on. After executing a query, the portal server returns $m+1$ intermediate digests $h^{\delta_{e,0}}(r_{a-1}|0)$, $h^{\delta_{e,1}}(r_{a-1}|1)$, .., $h^{\delta_{e,m}}(r_{a-1}|m)$ where $\delta_e = \alpha - r_{a-1}.K = \delta_{e,0} + \delta_{e,1}.B + \delta_{e,2}.B^2 + .. + \delta_{e,m}.B^m$. Let $\delta_c = r_a.K - \alpha = \delta_{c,0} + \delta_{c,1}.B + \delta_{c,2}.B^2 + .. + \delta_{c,m}.B^m$; the user then hashes $h^{\delta_{e,0}}(r_{a-1}|0)$ a further $\delta_{c,0}$ times, $h^{\delta_{e,1}}(r_{a-1}|1)$ another $\delta_{c,1}$ times, and so on, to produce $h^{\delta_{t,0}}(r_{a-1}|0), h^{\delta_{t,1}}(r_{a-1}|1), .., h^{\delta_{t,m}}(r_{a-1}|m)$.

To illustrate, suppose $\delta_t = r_a.K - r_{a-1}.K = 5555$, $\delta_c = r_a.K - \alpha = 1 + 2 \times 10 + 3 \times 10^2 + 4 \times 10^3$; so, $\delta_e = \delta_t - \delta_c = \alpha - r_{a-1} = 4 + 3 \times 10 + 2 \times 10^2 + 1 \times 10^3$. The portal server returns the digests $h^4(r_{a-1}|0)$, $h^3(r_{a-1}|1)$, $h^2(r_{a-1}|2)$, $h^1(r_{a-1}|3)$. Upon receiving them, the user further hashes $h^4(r_{a-1}|0)$ once to produce $h^5(r_{a-1}|0)$, $h^3(r_{a-1}|1)$, twice to produce $h^5(r_{a-1}|1)$, etc. With that, the user computes $h(h^5(r_{a-1}|0) \mid h^5(r_{a-1}|1) \mid h^5(r_{a-1}|2) \mid h^5(r_{a-1}|3))$ in place of $h^{5555}(r_{a-1})$, and from there confirms that $r_{a-1} < \alpha$ as before.

There is a complication, though. Suppose the query condition is such that $\delta_c = r_a.K - \alpha = 2828$. The portal server now gets $\delta_e = \alpha - r_{a-1}.K = 2727$, and the above procedure produces $h^{15}(r_{a-1}|0)$, $h^4(r_{a-1}|1)$, $h^{15}(r_{a-1}|2)$, $h^4(r_{a-1}|3)$, corresponding to the noncanonical representation $5555 = 15+4\times10+15\times10^2+4\times10^3$. In general, this complication arises if $\exists\ 0 \le i \le m$ such that $\delta_{t,i} < \delta_{c,i}$. To enable the user to succeed with the verification, the content manager would have to produce digests corresponding to the noncanonical representations too. Unfortunately, there are up to 2^m noncanonical representations in the worst case. Clearly, this overhead is unacceptable.

To limit the number of noncanonical representations that must be supported, we observe that while the user knows only the value of δ_c, the portal server has access to both δ_t and δ_c. Thus the portal server can return digests corresponding to certain preferred noncanonical representations of δ_e in order to influence the representation of δ_t that the user derives. Referring to our running example, if the portal server returns digests corresponding to the representation $\delta_e = 7 + 12 \times 10 + 6 \times 10^2 + 2 \times 10^3$ instead, the user would derive final digests corresponding to $5555 = 15 + 14 \times 10 + 14 \times 10^2 + 4 \times 10^3$.

Definition 1. For any $\delta \ge 0$, a representation $\delta = \delta_0 + \delta_1.B + \delta_2.B^2 + .. + \delta_m.B^m$ is valid if $\delta_i \ge 0$, $\forall\ 0 \le i \le m$.

Definition 2. Given $\delta_t \ge 0$, let its canonical representation be $\delta_t = \delta_{t,0} + \delta_{t,1}.B + \delta_{t,2}.B^2 + .. + \delta_{t,m}.B^m$, $0 \le \delta_{t,i} < B$. We define m preferred noncanonical representations:

$$
{}^{i}\delta_t = \begin{cases} (\delta_{t,0} + B) + (\delta_{t,1} + B - 1).B + .. + (\delta_{t,i} + B - 1).B^i \\ \qquad + (\delta_{t,i+1} - 1).B^{i+1} + \delta_{t,i+2}.B^{i+2} + .. + \delta_{t,m}.B^m & \text{for } 0 < i < m \\ \\ (\delta_{t,0} + B) + (\delta_{t,1} - 1).B + \delta_{t,2}.B^2 + .. + \delta_{t,m}.B^m & \text{for } i = 0 \end{cases}
$$

Note that some of the m representations may not be valid. For example, for the canonical representation $\delta_t = 3 + 2 \times B + 0 \times B^2 + 3 \times B^3$, ${}^{1}\delta_t$ is not valid because $\delta_{t,2} - 1 < 0$.

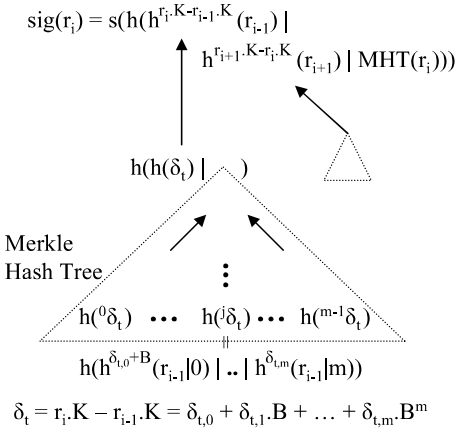


Fig. 5. Signature construction.

LEMMA 1. *For any* $0 \le \delta_c \le \delta_t$ *with canonical representation* $\delta_c = \delta_{c,0} + \delta_{c,1}.B + \delta_{c,2}.B^2 + .. + \delta_{c,m}.B^m$, $0 \le \delta_{c,i} < B$, *there exists a valid representation* ${}^{i_{max}}\delta_t$, i_{max} *is the largest* i *where* $\delta_{t,0} + .. + \delta_{t,i}.B^i < \delta_{c,0} + .. + \delta_{c,i}.B^i$ *holds, such that* $\delta_e = {}^{i_{max}}\delta_t - \delta_c$ *has a valid representation* $\delta_e = \delta_{e,0} + \delta_{e,1}.B + \delta_{e,2}.B^2 + .. + \delta_{e,m}.B^m$ *with* $\delta_{e,i} \ge 0$.

PROOF. Since the coefficients of the canonical representations satisfy $0 \le \delta_{t,i} < B$ and $0 \le \delta_{c,i} < B$, we must have:

- $0 \le \delta_{e,i} = (\delta_{t,i} + B) - \delta_{c,i} < 2B$ *for* $i = 0$,
- $0 \le \delta_{e,i} = (\delta_{t,i} + B - 1) - \delta_{c,i} < 2B - 1$ *for* $1 \le i \le i_{max}$,
- $0 \le \delta_{e,i} = (\delta_{t,i} - 1) - \delta_{c,i} < B - 1$ *for* $i = i_{max} + 1$,
- $0 \le \delta_{e,i} = \delta_{t,i} - \delta_{c,i} < B$ *for* $i_{max} + 1 < i \le m$

Therefore δ_e is a valid representation with $0 \le \delta_{e,i} < 2B$.

The lemma allows the system to support only the canonical representation of δ_t, plus m preferred noncanonical representations ${}^i\delta_t$, $0 \le i < m$, as defined above. We thus arrive at the following implementation scheme. □

5.3.1 *Signature Construction by the Content Manager.* (Figure 5): During creation and update of the sorted table $\mathbf{R} = [r_0, r_1, .., r_n, r_{n+1}]$ (recall r_0 and r_{n+1} are fictitious records), the content manager derives the signature for each new entry r_i as follows:

—Starting with the m noncanonical representations ${}^j\delta_t$ and the canonical representation $\delta_t = \delta_{t,0} + \delta_{t,1}.B + \delta_{t,2}.B^2 + .. + \delta_{t,m}.B^m$ of $\delta_t = r_i.K - r_{i-1}.K$, the content manager computes a digest for each valid noncanonical representation:

$$\begin{aligned} h(\delta_t) &= h(h^{\delta_{t,0}}(r_{i-1}|0) \mid .. \mid h^{\delta_{t,m}}(r_{i-1}|m)) \\ h({}^j\delta_t) &= h(h^{\delta_{t,0}+B}(r_{i-1}|0) \mid h^{\delta_{t,1}+B-1}(r_{i-1}|1) \mid .. \mid h^{\delta_{t,j}+B-1}(r_{i-1}|j) \mid \\ &\quad h^{\delta_{t,j+1}-1}(r_{i-1}|j+1) \mid h^{\delta_{t,j+2}}(r_{i-1}|j+2) \mid .. \mid h^{\delta_{t,m}}(r_{i-1}|m)) \end{aligned}$$

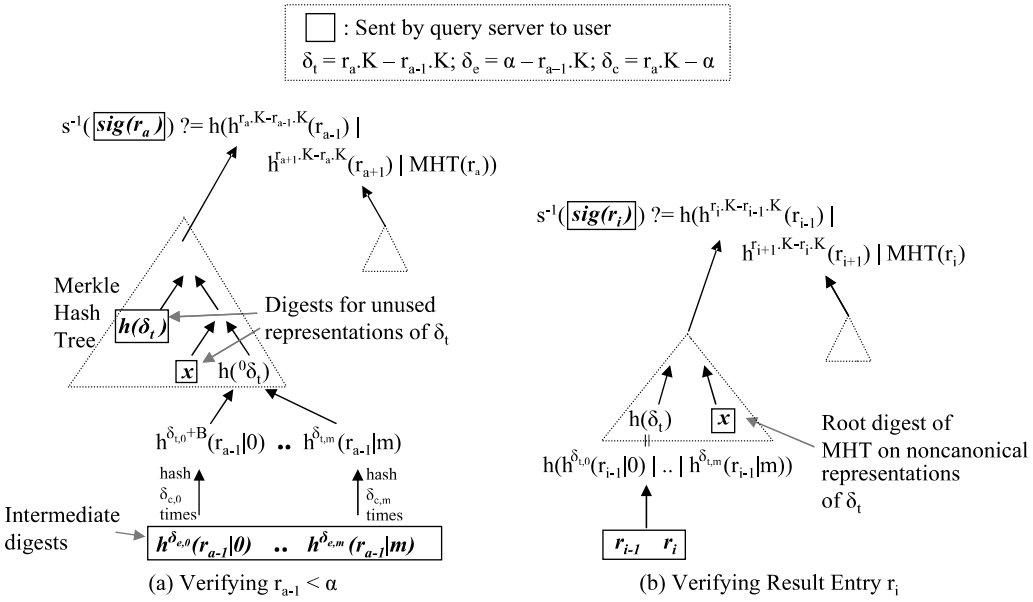


Fig. 6. Completeness verification.

For an invalid representation ${}^{j}\delta_t$ where $\delta_{t,j+1} - 1 < 0$, $h^{\delta_{t,j+1}-1}$ is undefined, so we drop it from the computation of the digest:

$$h({}^{j}\delta_t) = h(h^{\delta_{t,0}+B}(r_{i-1}|0) \mid h^{\delta_{t,1}+B-1}(r_{i-1}|1) \mid .. \mid h^{\delta_{t,j}+B-1}(r_{i-1}|j) \mid h^{\delta_{t,j+2}}(r_{i-1}|j+2) \mid .. \mid h^{\delta_{t,m}}(r_{i-1}|m))$$

—Next, a Merkle Hash Tree (MHT) is built over the m noncanonical representations of δ_t. The root digest of the MHT is concatenated with the digest of the canonical representation, then hashed to produce a digest in place of $h^{r_i.K-r_{i-1}.K}(r_{i-1})$.

—$h^{r_{i+1}.K-r_i.K}(r_{i+1})$ is derived using a similar procedure, while MHT(r_i) is computed from the content of r_i.

—The signature of r_i is now generated using equation (1).

5.3.2 *Completeness Verification between Portal Server and User.* (Figure 6, in which the items in italicized, bold font are transmitted by the portal server to the user): To verify the result $\mathbf{Q} = [r_a, r_{a+1}, .., r_b]$ for query condition $\alpha \le r \le \beta$, the user checks the signature $sig(r_i)$ for each $r_i \in \mathbf{Q}$, which in turn requires $h^{r_i.K-r_{i-1}.K}(r_{i-1})$ and $h^{r_{i+1}.K-r_i.K}(r_{i+1})$ for $a \le i \le b$. To compute $h^{r_a.K-r_{a-1}.K}(r_{a-1})$ (see Figure 6(a)):

—The portal server utilizes its knowledge of α, r_{a-1} and r_a to compute the canonical representations of $\delta_t = r_a.K - r_{a-1}.K$ and $\delta_c = r_a.K - \alpha$.

—If $\delta_{t,i} \ge \delta_{c,i}\ \forall i$,

  —Δ_t is equated with the canonical representation of δ_t.

—Return the root digest of the MHT over the noncanonical representations of δ_t.
otherwise,
—Starting from the largest i where $\delta_{t,0} + .. + \delta_{t,i}.B^i < \delta_{c,0} + .. + \delta_{c,i}.B^i$ holds, increment i_{max} until the noncanonical representation ${}^{i_{max}}\delta_t$ is valid. Δ_t is equated with ${}^{i_{max}}\delta_t$. A valid ${}^{i_{max}}\delta_t$ must exist because $\delta_t \geq \delta_c$.
—Return the digest of the canonical representation of δ_t, as well as the digests in the MHT covering those representations of δ_t that are not used as Δ_t. There are $\lceil log_2 m \rceil$ such digests.

—The coefficients in the polynomial representation for $\delta_e = \alpha - r_{a-1}.K$ is then calculated as: $\delta_{e,i} = \Delta_{t,i} - \delta_{c,i}$. The portal server computes $m+1$ intermediate digests for $h^{\delta_{e,i}}(r_{a-1}|i)$, $0 \leq i \leq m$, and returns them to the user.
—Upon receiving the digests, the user first determines the canonical representation of $\delta_c = r_a.K - \alpha$, then hashes each of the $h^{\delta_{e,i}}(r_{a-1}|i)$ digests $\delta_{c,i}$ more times to derive $h^{\Delta_{t,i}}(r_{a-1}|i)$.
—Next, the user concatenates them and derives the digest $h(\Delta_t)$. If Δ_t is the canonical representation, $h(\Delta_t)$ is then combined with the MHT root digest from the portal server to produce $h^{r_a.K - r_{a-1}.K}(r_{a-1})$. If not, $h(\Delta_t)$ is combined with the MHT digests to derive the root digest, and then with the digest for the canonical representation to produce $h^{r_a.K - r_{a-1}.K}(r_{a-1})$.

Computation of $h^{r_{b+1}.K - r_b.K}(r_{b+1})$ is carried out similarly. All the remaining digests are derived as illustrated in Figure 6(b). For example, for $h^{r_i.K - r_{i-1}.K}(r_{i-1})$, $a < i \leq b$:

—The portal server returns the root digest of the MHT over the noncanonical representations ${}^i\delta_t$ of $\delta_t = r_i.K - r_{i-1}.K$.
—With the query results r_{i-1} and r_i, the user generates the $m + 1$ digests $h^{\delta_{t,j}}(r_{i-1}|j)$, and combines them into a single digest for the canonical representation of δ_t. This digest is then concatenated with the root digest of the MHT from the portal server to produce $h^{r_i.K - r_{i-1}.K}(r_{i-1})$.

With this optimization, the number of hash operations required by the user verification procedure is reduced to $(b - a + 1)[log_B(2(U - L)/n) + 2p - 1]$, where p is the number of attributes.

5.4 Cost Analysis

Having presented our authentication mechanism, we now analyze the overheads that it introduces for range selection queries. We begin by quantifying the communication overhead, before looking at the incremental computation cost. We shall focus on the costs involving the user, who shoulders most of the runtime authentication load and is likely to be the resource-constrained party in the system.

The parameters used in the analysis are summarized in Table I, in which the values for C_{hash} and C_{sign} are obtained from Rivest and Shamir [2001]. For computation costs, we model only hashing and signature verification; other

Table I. Cost Parameters

Parameter	Meaning	Default
C_{hash}	Computation cost of a hash operation	50 μsec
C_{sign}	Computation cost for verifying a signature	5 msec
C_{user}	Total computation cost incurred by the user	–
M_{digest}	Size of a hash/digest (bits)	128
M_{sign}	Size of a signature (bits)	1024
M_{user}	Total size of authentication information sent to the user	–
M_r	Size of a result record (bytes)	–
B	$\delta = \delta_0 + \delta_1.B + .. + \delta_m.B^m$	–
m	$m = log_B(U - L)$	–

operations like concatenation are assumed to be negligible relative to C_{hash} and C_{sign}. (The hashing operations here involve one-way hash functions like SHA [2001], which are much costlier than simple hash functions used in, say, typical database hash indices.)

5.4.1 *Communication from Portal Server to User.* The authentication information that the portal server transmits to the user includes the following components:

—Digests for computing $h^{r_a.K-r_{a-1}.K}(r_{a-1})$ in equation (1) include the $m+1$ intermediate digests corresponding to the polynomial representation for h^{δ_e}, $\lceil log_2 m \rceil$ digests in the MHT covering the representations of $\delta_t = r_a.K - r_{a-1}.K$ that are not selected, plus one digest for the canonical representation in the worst case (if a noncanonical representation ${}^i\delta_t$ is used). The traffic amounts to $[m + 2 + \lceil log_2 m \rceil] \times M_{digest}$.

—Likewise, digests for computing $h^{r_{b+1}.K-r_b.K}(r_{b+1})$ amount to $[m+2+\lceil log_2 m \rceil] \times M_{digest}$.

—Digests for computing $h^{r_i.K-r_{i-1}.K}(r_{i-1})$, $a < i \leq b$, and $h^{r_{i+1}.K-r_i.K}(r_{i+1})$, $a \leq i < b$. For each of them, the portal server sends the root digest of the MHT over the noncanonical representations of δ_t. This amounts to $2(b-a) \times M_{digest}$.

—The digests MHT($r_i.A$), $a \leq i \leq b$, are also returned. The traffic for all the result entries amounts to $(b - a + 1) \times M_{digest}$.

—The aggregated signature, derived from the individual signatures for the result entries. The size of this signature is M_{sign}.

The total traffic to the user is, therefore:

$$M_{user} = [2m + 5 + 3(b - a) + 2\lceil log_2 m \rceil] \times M_{digest} + M_{sign} \quad (4)$$

Figure 7 plots the user traffic overhead, defined as M_{user}/ ResultSize where ResultSize = $q \times M_r$, against the result record size M_r for various number of result entries q. The parameters M_{digest} and M_{sign} are set to their default values of 128 bits and 1,024 bits, respectively. The figure shows that the traffic overhead reduces very quickly as q grows beyond one, as the cost of the aggregated signature is amortized over more result entries. This reduction stabilizes at around $q = 5$, at which point the per-entry overhead falls within 25% for $M_r \geq$ 512 bytes.

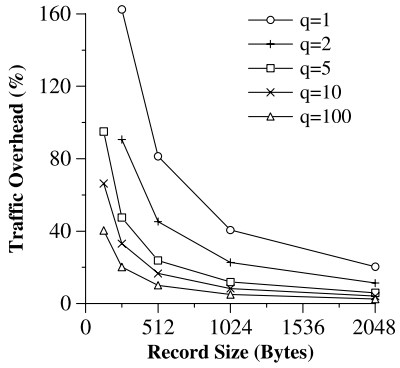


Fig. 7. User traffic overhead.

From equation (4), we also observe that the space overhead incurred by our proposed solution is linear in the result size (i.e., $b - a$). This compares favorably with the previous scheme in Devanbu et al. [2000] that enables verification of result completeness for range selection; the space overhead of their scheme grows linearly to the query result, as well as logarithmically to the underlying database.

5.4.2 *Computation Overhead on the User.* The computations performed by the user in authenticating the query results include:

—Derivation of $h^{r_a.K-r_{a-1}.K}(r_{a-1})$. This entails hashing the $m+1$ intermediate digests corresponding to the selected representation for $\delta_t = r_a.K - r_{a-1}.K$, each of which requires up to B additional hashes. In the worst case (where the selected representation is noncanonical), the resulting digest is then combined with the digests in the MHT over the noncanonical representations, and the digest for the canonical representation to get the first component in equation (1), which requires $\lceil log_2 m \rceil + 1$ hashes. Evaluation of equation (1) incurs another hash. The computation cost amounts to $[B(m+1)+\lceil log_2 m \rceil + 2] \times C_{hash}$.

—Similarly, derivation of $h^{r_{b+1}.K-r_b.K}(r_{b+1})$ incurs $[B(m+1) + \lceil log_2 m \rceil + 2] \times C_{hash}$.

—Derivation of $h^{r_i.K-r_{i-1}.K}(r_{i-1})$, $a < i \leq b$, and $h^{r_{i+1}.K-r_i.K}(r_{i+1})$, $a \leq i < b$. For each of them, the user first computes the $m + 1$ digests corresponding to the canonical representation for δ_t, each requiring up to B hashes. Next, combining the $m + 1$ digests incurs another hash. This incurs $2(b - a)(B(m + 1) + 1) \times C_{hash}$ in all.

—Evaluation of equation (1) incurs a hash for each result record. The computation for all the result entries thus amounts to $(b - a + 1) \times C_{hash}$.

—Derivation of aggregated digest, then matching with the aggregated signature, costing $C_{hash} + C_{sign}$.

The total computation cost incurred by the user is:

$$C_{user} = [(b - a)(2B(m + 1) + 3) + 2B(m + 1) + 2\lceil log_2 m \rceil + 6] \times C_{hash} + C_{sign} \quad (5)$$

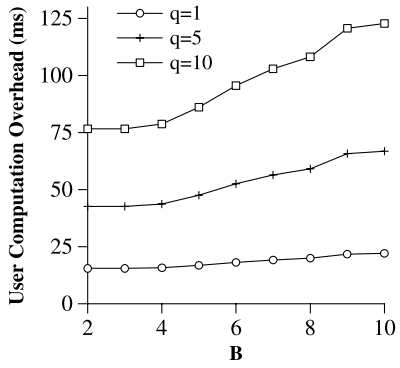


Fig. 8. User computation overhead.

Since $m = \lceil log_B(U - L) \rceil$ is a tunable parameter, C_{user} can be minimized by choosing a B (or, equivalently, an m) value such that $y = (b - a)(2B(m + 1) + 3) + 2B(m + 1) + 2\lceil log_2 m \rceil + 6$ is minimum. It can be shown that this occurs at $2 < B < 3$ where $\frac{dy}{dB} = 0$. To illustrate, Figure 8 plots B against C_{user} for different result sizes $q = b - a + 1$. Therefore, if computation overhead at the user is the performance bottleneck in a deployed system, B can be set to 2 or 3 depending on the actual range $[L, U]$.

With $B = 2$, $m = log_B 2^{32} = 32$ if the key is an integer, for example. Using the default values for C_{hash} and C_{sign} from Rivest and Shamir [2001], equation (5) reduces to $C_{user} = 6.75(b - a) + 12.4$ milliseconds. Thus, C_{user} is roughly 12.4 milliseconds, 681 milliseconds and 6.76 seconds for result sizes of 1,100, and 1,000 records, respectively, which is not excessive especially as the user device can start to verify the initial result tuples while the remaining result values are being transmitted.

5.4.3 *Database Updates.* Having evaluated the overheads for verifying query answers, we now consider the impact of our proposed scheme on update operations.

To enable the portal server to produce completeness proof for query answers at runtime, the content manager has to pregenerate signatures on each attribute or group of attributes that are expected to participate in the query conditions. This is analogous to creating B+-trees on those attributes to facilitate efficient query processing. In fact, for performance reasons our scheme *may* be incorporated into the B+-tree, by storing the signatures for each record along with its pointer in the leaf node of the B+-tree.

Suppose the proposed scheme is indeed implemented as a data structure like the B+-tree. According to equation (1), each record update affects the signature of the record itself, and its left and right neighbors. This is conceptually similar to updating a doubly-linked list. Since a B+-tree node typically contains hundreds of entries, most of the time the three affected signatures would reside within the same node, so there is no additional I/O or (page) locking overhead. In the worst case, the affected signatures would span only two adjoining leaf nodes. Hence the update overheads incurred by our scheme are significantly

Table II. Notation

Parameter	Description
n	# records in the database table
q	# records in the query range
f	Fan-out factor of partial sum hierarchy

less than Merkle Hash Tree schemes (e.g., Devanbu et al. [2000], Pang and Tan [2004]) that need to propagate every update up to the digest of the root node, which becomes a locking contention hot spot. Therefore, the scheme in this paper is more appropriate for databases that experience non-negligible update activities.

6. SINGLE-ATTRIBUTE RANGE AGGREGATE

In this section, we extend the signature chain scheme in the previous sections to authenticate single-attribute range aggregate queries of the form SELECT SUM(S) FROM $\mathbf{R}$ WHERE $\alpha \leq K \leq \beta$. We first present our solution for the range sum query, and discuss extension to other aggregation functions at the end of the section. Support for GROUP BY is discussed in Section 6.3.

We assume that a user who has access rights to pose an aggregate query over a given range of records, $\alpha \leq K \leq \beta$, is also allowed to aggregate over any sub-interval within $[\alpha, \beta]$. By extension, the user can access the underlying record values that contribute to the aggregate. However, the user chooses not to retrieve the underlying data directly due to resource considerations, e.g., because the user device is resource-constrained or to minimize network traffic. Instead, the user relies on the portal server to compute the aggregate. Obviously, this approach makes sense only if the cost of verifying the aggregate answer is much lower than the cost of retrieving the underlying data for user inspection. In the case where the user is not authorized to inspect the underlying data, a trusted intermediary has to be enlisted to produce the correct aggregate result, or to verify the correctness of the answer produced by the portal server.

6.1 Partial Sum Hierarchy

At one extreme, a range aggregate query could be satisfied by returning all the underlying record values. After verifying that the returned values are authentic and complete, using a range selection authentication mechanism (e.g., Devanbu et al. [2000], Pang et al. [2005]), the user then computes the aggregate herself. With this approach, the network transmission and client computation are proportional to q, the number of records in the query range. These overheads are very high, considering that the user desires just a single aggregate value.

From the user's perspective, ideally the portal should return a single certified value for the aggregate query. This is possible if the content manager precomputes and signs every range aggregate. Without prior knowledge of the query range, the content manager has to materialize all $n!$ possible range aggregates for a table of n records. Clearly, the computation and storage overheads are prohibitive. As a compromise, we propose for the portal server

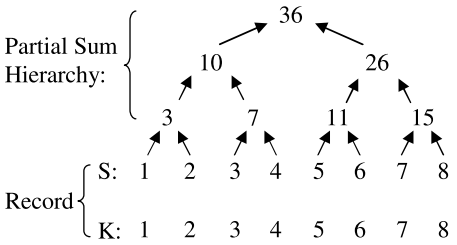


Fig. 9. Partial sum hierarchy with $f = 2$.

to return a list of certified partial sums corresponding to partitions of the query range.

Our solution requires the content manager to materialize a hierarchy of partial sums over the records, as Figure 9 demonstrates. The size of the query answer can be reduced by returning partial sums corresponding to the largest partitions that together cover the query range. The partial sum hierarchy that has to be maintained by the content manager and the portal server has approximately $2n - 1$ nodes.

Let us now consider the number of values that are returned to the user. We shall call the smallest subtree in the partial sum hierarchy that encompasses the query range the *covering subtree*. We first consider the case where the left boundary of the query range is aligned with the leftmost record of the covering subtree, (e.g., the range selection $1 \leq K \leq 6$ on the data set in Figure 9). Here the first 2^2 records are aggregated into a partial sum 10, and the next 2^1 records are aggregated into another partial sum 11. Denote the binary representation of the query answer size q as $q = \sum_{i=0}^{\lfloor log_2 q \rfloor} b_i \cdot 2^i$, $b_i = 0$ or 1. Starting from $i = \lfloor log_2 q \rfloor$ down to 0, from the left boundary of the query range towards the right, 2^i records in the query range are aggregated into a partial sum if $b_i = 1$. Thus the number of returned partial sums is $\sum_{i=0}^{\lfloor log_2 q \rfloor} b_i$, and is at most $\lfloor log_2 q \rfloor + 1$. The case where the right boundary of the query range is aligned with the rightmost record of the covering subtree (e.g., the range selection $3 \leq K \leq 8$ on the data set), is similar to the first case. If both the left and right boundaries of the query range are not aligned with the edges of the covering subtree, the query range is decomposed into up to $2(\lfloor log_2 \frac{q}{2} \rfloor + 1)$ partitions and hence partial sums. For example, for the range selection $2 \leq K \leq 7$, the query answer comprises the datum 2, the partial sums 7 and 11, followed by the datum 7.

The partial sum hierarchy can be generalized to have a fan-out of $f > 1$ (i.e., each node in the hierarchy aggregates f child nodes). With this general scheme, the partial sum hierarchy contains approximately $\frac{fn-1}{f-1}$ nodes. Figure 10 illustrates a partial sum hierarchy with n=16 and f=4. Consider again the case where the left boundary of the query range is aligned with the leftmost record of the covering subtree (e.g., the range selection $2 \leq K \leq 13$ on the data set in Figure 10). Denote the query answer size q with the polynomial $q = \sum_{i=0}^{\lfloor log_f q \rfloor} b_i \cdot f^i$, $0 \leq b_i < f$. Starting from $i = \lfloor log_f q \rfloor$ down to 0, from the left boundary of the query range towards the right, $b_i \cdot f^i$ records in the query

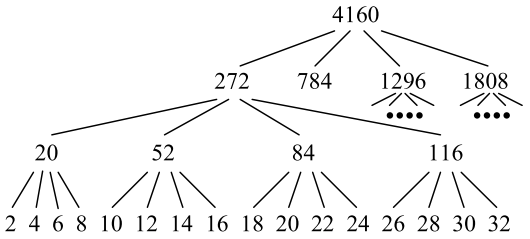


Fig. 10. Partial sum hierarchy with $f = 4$.

range are aggregated into b_i partial sums. Hence the number of partial sums returned is

$$\#\text{ partial sums} = \sum_{i=0}^{\lfloor log_f q \rfloor} b_i \tag{6}$$

and at the most there are

$$\max \#\text{ partial sums} = (f-1) \cdot (\lfloor log_f q \rfloor + 1) \tag{7}$$

partial sums in the query answer. If both the left and right boundaries of the query range are not aligned with the edges of the covering subtree, the query range is decomposed into up to $2(f-1)(\lfloor log_f \frac{q}{2} \rfloor + 1)$ partial sums. For example, for the range selection $8 \leq K \leq 29$, the query answer size comprises the record value 8, the partial sums 52 and 84, followed by the record values 26 and 28.

Equation (7) confirms that our approach of producing partial sums scales better than returning the underlying record values to the user directly, which would have generated query answer sizes that are proportional to q.

In terms of query answer size, it would appear that a smaller f is better. For example, for q=10 and assuming both boundaries of the query range are not aligned with the edges of the covering subtree, the worst case answer size is 6 and 12, for f=2 and f=4 respectively. Besides query answer size which determines transmission cost and user computation cost, I/O cost at the portal server also plays a role in the choice of f. We will study the trade-offs in the selection of f through experiments in Section 8.

For single-attribute aggregates, the classic prefix sum approach [Ho et al. 1997] requires only two prefix sums to produce an aggregate result. For example, an aggregate of record a to record b is derived by subtracting prefix sum $a-1$ from prefix sum b. This is obviously much cheaper than our partial sum hierarchy. The drawback is that the prefix sum $a-1$ pertains to data that are beyond the query range, and hence violates our *precision* requirement.

6.2 Certifi ed Partial Sum Hierarchy

Having introduced the partial sum hierarchy for computing range aggregates, we now address how to build authentication information into the scheme. There are two primary considerations for proving the correctness of a range aggregate result–each returned partial sum is an accurate aggregation of the

Table III. Attributes of Child Pointer p_i in the Partial Sum Hierarchy

Parameter	Description	Size
ptr	Points to a sub-partition within p_i (Bytes)	4
K_l	Key value of the leftmost record covered by p_i (Bytes)	4
K_h	Key value of the rightmost record covered by p_i (Bytes)	4
D_l	Digest from applying a one-way hash function h on the first record in partition p_i (Bytes)	16
D_h	Digest from applying h on the last record in p_i (Bytes)	16
psum	Partial sum of the records covered by p_i (Bytes)	8
sign	Signature for p_i (Bytes)	128
	Total pointer size (Bytes)	180

records in its scope, and together the returned partial sums cover all the records in the query range.

Since the content manager is secure in our system model, the content manager can generate a signature for each partial sum based on its value. Whenever a partial sum is returned to the user, the portal server produces the associated signature to prove that it has not been tampered with. Besides the value of the partial sum, we also include in the signature computation the record immediately below and the record immediately above the scope of the partial sum (i.e., using the signature chain concept introduced in Section 4). This enables the user to confirm that the partial sums in a query answer provide continuous coverage of the query range, by simply checking that the record just above the scope of one partial sum is indeed the first record in the next partial sum. The detailed formulation is as follows.

Given a set of records, each having the attributes $\langle K, S, A\rangle$ where K is the key attribute for the range selection, S is the attribute to be aggregated, and A denotes the remaining record values. Sort the records on K, then construct a partial sum hierarchy over the records. (The construction algorithm will be given shortly.) Every partition in the hierarchy has up to f child pointers p_i, each with the attributes listed in Table III.

The signature for each partition, p_i.sign, is precomputed by the content manager for distribution to the portal server along with the data, using the formula:

$$p_i.\text{sign} = s(h(h^{p_i.K_l - p_i.r_l.K}(h(p_i.r_l)) \mid p_i.K_l \mid p_i.K_h \mid p_i.\text{psum} \mid h^{p_i.r_h.K - p_i.K_h}(h(p_i.r_h)))) \quad (8)$$

where s signs its argument with the organization's private key, and $h^x(y)$ hashes y iteratively for x times. Furthermore, $p_i.r_l$ and $p_i.r_h$ are the records immediately below and above the coverage of p_i, respectively. For example, for the partition anchored at node 52 in Figure 10, the records below and above its coverage are 8 and 18 respectively, so its signature is derived as $s(h(h^{10-8}(h(8)) \mid 10 \mid 16 \mid 52 \mid h^{18-16}(h(18))))$ (assuming for simplicity that each record contains only its key value, i.e., $r_i = r_i.K$).

For a range aggregate query, the returned answer contains a list of partitions $\{p_a, p_{a+1}, .., p_b\}$, each having the format $\langle K_l, K_h, D_l, D_h, \text{psum}, \text{sign}\rangle$. We choose to return D_l and D_h in place of the actual records to minimize transmission and storage overheads.

In verifying each partition p_i in the query answer, the user checks whether:

$$s^{-1}(p_i.\text{sign}) \stackrel{?}{=} h(h^{p_i.K_l - p_{i-1}.K_h}(p_{i-1}.D_h) \mid p_i.K_l \mid p_i.K_h \mid p_i.\text{psum} \mid h^{p_{i+1}.K_l - p_i.K_h}(p_{i+1}.D_l)) \quad (9)$$

where s^{-1} decrypts the given signature with the organization's public key.

The first term in equation (8), $h^{p_i.K_l - p_i.r_l.K}(h(p_i.r_l))$, matches the first term in equation (9), $h^{p_i.K_l - p_{i-1}.K_h}(p_{i-1}.D_h)$, only if the record just below p_i's coverage ($p_i.r_l$) is the same record that hashes to $p_{i-1}.D_h$, otherwise there is a collision in the collision-resistant hash function h. This proves that partition p_i's coverage begins right after the preceding partition p_{i-1} in the query answer. Likewise, the last term $h^{p_{i+1}.K_l - p_i.K_h}(p_{i+1}.D_l)$ in equation (9) is for verifying that p_i's coverage ends just before the subsequent partition p_{i+1} in the query answer.

Besides ensuring contiguous coverage of the query range by the partitions in the answer, the user will also want to ascertain that the first partition p_a in the answer starts from the first qualifying record. Equivalently, the record immediately below p_a should have a smaller key value than the query range, (i.e., $p_a.r_l.K < \alpha$). This is achieved by having the portal server return an intermediate digest $h^{\alpha - p_a.r_l.K}(h(p_a.r_l))$, which the user then hashes $p_a.K_l - \alpha$ more times to produce the first term in equation (9). Since the intermediate digest is defined only for a positive number of iterations of h, p_a's signature can be matched only if indeed $p_a.r_l.K < \alpha$. Similarly, the portal server can prove that the record immediately after the last partition p_b satisfies $p_b.r_h.K > \beta$.

In proving that the query answer covers the query range completely, we have to be careful that the proof does not leak information about records beyond the query range. In particular, the intermediate digest $h^{\alpha - p_a.r_l.K}(h(p_a.r_l))$ should not compromise $p_a.r_l$. Since α originated from the user, $h(p_a.r_l)$ should not be released also to the user, otherwise the user can deduce $p_a.r_l.K$ by counting the number of times that $h(p_a.r_l)$ needs to be hashed in order to match the intermediate digest.

6.3 Query Processing using the Partial Sum Hierarchy

6.3.1 *Aggregate with range predicate.* A single-attribute range sum has the form SELECT SUM(S) FROM $\mathbf{R}$ WHERE $\alpha \leq K \leq \beta$. This query is processed by traversing down the root of the partial sum hierarchy, along progressively finer partitions that contain the query range, till we reach the covering subtree. Recall that the covering subtree is the smallest subtree in the partial sum hierarchy that encompasses the query range $K[\alpha, \beta]$. For example, for the range selection $8 \leq K \leq 29$ on the data set in Figure 10, we traverse from the root (4160) to reach the partial sum (272) that anchors the covering subtree. The partial sums (52 and 84) within the covering subtree aggregate the records in the middle of the query range. To complete the query answer, we iteratively retrieve record values or partial sums, on the next level down the hierarchy, that aggregate the records toward the left and right boundaries of the query range. In our current example, this step elicits the record values 8 on the left, and 26 and 28 on the right. Together, the retrieved records values and partial sums effectively form a "canopy" over the query range.

6.3.2 *Aggregate with GROUP BY.* This query has the form SELECT SUM(S) FROM $\mathbf{R}$ GROUP BY K. The result comprises a list of sums, one for each $k_i \in K$. Each sum, in turn, is made up of a list of record values and partial sums, as with a range sum query over $[k_i, k_i]$. The user needs to verify that the last record in the scope of a sum is the immediate neighbor of the first record in the scope of the following sum, using equation (9). Queries with GROUP BY or range selection on more than one attribute are treated as multi-attribute aggregates, to be discussed in Section 7.

6.3.3 *Signature Aggregation.* The overheads of transmitting and verifying a signature for every partial sum in the answer for a range aggregation can be very large. Instead, the portal server can combine the signatures for the individual partial sums into one aggregated signature, using techniques in Boneh et al. [2003], so that there is only one signature verification per query answer. Our performance study implements this optimization.

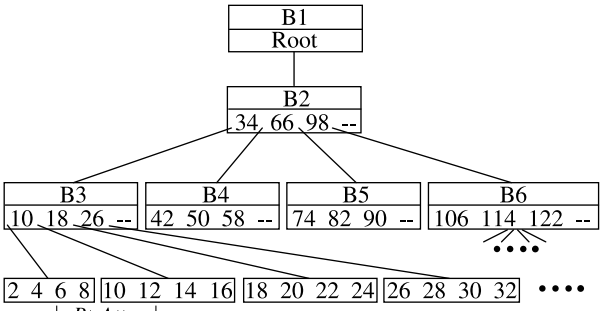
6.4 Data Organization

As shown in Table III, each child pointer in the partial sum hierarchy has the attributes ⟨ptr, K_l, K_h, D_l, D_h, psum, sign⟩, with a total length of 180 bytes. If these attributes are stored within the partial sum hierarchy, each node can only hold at most 22 child pointers, assuming a typical block size of 4 Kbytes. A table containing just 10 million records would thus generate a partial sum hierarchy with a height of 6.

An alternate data organization is to store all the pointer attributes outside of the partial sum hierarchy, in a separate *PtrAttr* file. With this organization, the partial sum hierarchy can be simplified to a conventional B+-tree, in which each node is augmented with an offset into the *PtrAttr* file where the attribute values of the node's child pointers are located. Assuming 4 bytes each for the file offset, child pointers and key values in a tree node, a 4-Kbyte block allows a fan-out f of around 510. Thus a table with 10 million records requires only a B+-tree height of 3, half of the height for the previous organization where the attribute values are stored within the hierarchy. In actual deployment, the realizable I/O savings could be even higher, as the top two layers of the tree/hierarchy are usually buffered. In addition, we can exploit the following observations to optimize the structure of the *PtrAttr* file, with the aim of reducing the number of random I/Os in favor of sequential I/Os:

(1) If the partial sum for a subtree in the hierarchy is utilized, no descendant partial sums/record values within that subtree will be retrieved by the same query. The converse is also true. For example, in the answer {8, 52, 84, 26, 28} for the range aggregate over $8 \leq K \leq 29$ in Figure 10, there is no ancestor-descendant relationship between the result entries. It is therefore not beneficial to cluster together attribute values belonging to ancestor and descendant nodes.

(2) Within any node along the canopy over the query range, several of the partial sums are likely to be required for the query answer for example, {52, 84} and {26, 28}. For this reason, the attribute values of child pointers



PtrAttr:

Offset	K_l	K_h	D_l	D_h	psum	sign
B1	2	128	..	..	4160	..
			(blank)			
B2	2	32	..	..	272	..
	34	64	..	..	784	..
	66	96	..	..	1296	..
	98	128	..	..	1808	..
B3	2	8	..	..	20	..
	10	16	..	..	52	..
	18	24	..	..	84	..
	26	32	..	..	116	..
	....					

Fig. 11. Data organization.

within the same node should be stored contiguously so that they can be retrieved through sequential I/Os, and the *PtrAttr* file can be structured as a series of buckets that are $176f$ bytes each. (Excluding the child ptr which remains in the partial sum hierarchy, the remaining attributes in Table III have a combined size of 176 bytes.) With this layout, insertions and deletions to a node propagate only to the corresponding bucket in the *PtrAttr* file; there is no need for sophisticated file management like compaction and shifting of file content. There is also the additional advantage that we only need to allocate one file offset for each tree node, so the concomitant reduction in fan-out f is negligible.

(3) The canopy over a query range contains nodes along the boundaries of adjacent subtrees. For example, {... 32, 784 ...}, {... 116, 784 ...} and {... 272, 784 ...} are all possible canopies in Figure 10. It may therefore be tempting to try to store adjacent nodes consecutively. However, without prior knowledge of the workload, there is no objective basis to choose between alternative pairings, (e.g., whether 32, 116, or 272 should be selected to precede 784).

Figure 11 shows an order-2 B+-tree and the associated *PtrAttr* file for the example in Figure 10.

6.5 Extension to Other Aggregation Functions

Our partial sum hierarchy has application beyond the SUM function. Gray et al. [1997] defined three classes of aggregation functions: Distributive

aggregates can be computed by aggregating the partial aggregates on partitions of the dataset, and include COUNT, SUM, MIN, and MAX. Algebraic aggregates are derived from a scalar function of distributive aggregates. For example, AVG can be expressed as SUM / COUNT. In contrast, holistic aggregates like MEDIAN cannot be computed through a divide-and-conquer strategy on the underlying dataset.

Our authenticated partial sum hierarchy applies to distributive aggregates and algebraic aggregates in general, by simply substituting the SUM function with the desired aggregation function. The solution does not extend to holistic aggregates, however.

7. MULTI-ATTRIBUTE RANGE AGGREGATE

Having presented the authentication scheme for single-attribute aggregate operations, we now extend the solution to multi-attribute aggregate queries.

Suppose the master database has a sorted list of records $\mathbf{R} = [r_1, ..., r_n]$, each record having the fields $\langle K_1, ..., K_d, S, A \rangle$ where K_1 to K_d are the attributes involved in the range selection or GROUP BY clauses, S is the attribute to aggregate over, and A denotes the remaining record values.

Now a user submits a query of the form SELECT SUM(S) FROM $\mathbf{R}$ WHERE P_1 and P_2 and ... and P_d, or SELECT SUM(S) FROM $\mathbf{R}$ GROUP BY K_1, K_2, ..., K_d. The portal server needs to prove that its answer is authentic and complete, while keeping the answer precise. For simplicity, we focus on the SUM operator and range selections $\sigma_{\alpha_i \le r.K_i \le \beta_i}(\mathbf{R})$, $1 \le i \le d$. The treatment of GROUP BY clauses can be extended from range selection, as discussed in Section 6.3. Generalization to other aggregation functions is deferred to the end of the section.

7.1 Authentication Scheme

Like the single-attribute context, our multi-attribute solution builds a partial sum hierarchy into a multi-dimensional index structure. In this paper, we adopt the KDB-tree [Robinson 1981]. The KDB-tree partitions the data space recursively, such that after a split the subregions contain roughly the same number of data points. Moreover, each split is perpendicular to one of the dimensions. Regions associated with nodes at the same level in the tree are disjoint, and together they cover the entire data space. At the lowest level, data points are grouped into leaf nodes. The KDB-tree is balanced like the B-tree, so it has the advantage of being insensitive to the data distribution. Figure 12 illustrates a KDB-tree over two-dimensional data, together with the partial sum associated with each node.

With our KDB-tree cum partial sum hierarchy (KDB+PS), we intend to construct an answer for a multi-attribute aggregate, by assembling partial sums over the largest possible hypercubes inside the query scope, plus partial sums for smaller hypercubes and records at the fringe of the query scope. In general, the larger hypercubes would tend to correspond to nodes that are higher up the KDB+PS tree. The detailed construction is as follows.

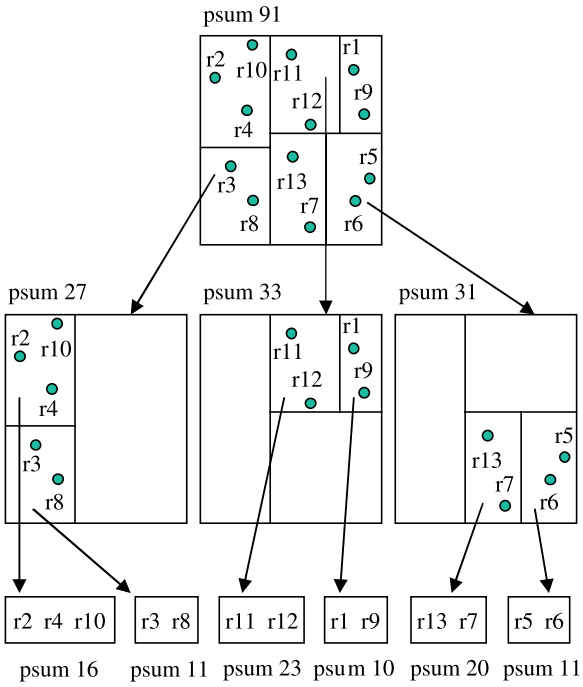


Fig. 12. KDB-Tree cum Partial Sum Hierarchy over a 2-dimension data set.

7.1.1 *Construction.* Each node in the KDB+PS tree has the structure $\langle scope, psum, sign, numChild, \{child\}^*\rangle$, where *scope* demarcates the hypercube covered by that node and consists of d sets of lower and upper bounds, *psum* aggregates the S values of all the records within the scope, *sign* is the node signature, *numChild* indicates the number of child nodes/records leading from that node, and $\{\text{child}\}^*$ are pointers to the child nodes/records. In the case of an internal node,

$$sign = s(h(scope \mid psum)) \tag{10}$$

whereas for a leaf node,

$$sign = s(h(scope \mid h(psum) \mid numChild)) \tag{11}$$

Moreover, each record $r = \langle k_1, .., k_d, s, a\rangle$ within a leaf node with the scope $\langle [l_1, u_1], ..., [l_d, u_d]\rangle$ has the signature:

$$sign = s(h(h^{k_1-l_1}(r') \mid h^{u_1-k_1}(r') \mid \ \ldots \ \mid h^{k_d-l_d}(r') \mid h^{u_d-k_d}(r'))) \tag{12}$$

where $r' = k_1 \mid ... \mid k_d \mid s \mid h(a)$.

7.1.2 *Query Processing.* Given an aggregate query, the portal server searches the KDB+PS tree from the root node towards the records in the leaf nodes, comparing each node with the query scope:

(1a) If the scope of the node is contained within the query scope, its $\langle scope, psum\rangle$ are pushed into the query answer, its signature is "added" to

an aggregated signature [Boneh et al. 2003], and the search along that path ends.

(1b) If the scope of the node is disjoint from the query scope, the search along that path terminates.

(1c) If the node is an internal node and its scope partially overlaps the query scope, the search goes down one level to each child of the node in turn.

(1d) If the node is a leaf and its scope partially overlaps the query scope, its $\langle scope, h(psum), numChild\rangle$ are pushed into the query answer, and its signature is "added" to the aggregated signature. For each record r in that leaf node,

—if r falls within the query scope, the record values $\langle k_1, ..., k_d, s, h(a)\rangle$ are pushed into the query answer.

—if r is beyond the query scope because one of its key values is below the corresponding lower bound of the query scope, i.e., $k_i < \alpha_i$ for some $1 \le i \le d$, the portal server pushes the intermediate digest $h^{\alpha_i - k_i}(r')$, and the rest of the $h^{u_j - k_j}(r')$ and $h^{k_j - l_j}(r')$ digests for this record, into the query answer. Since the intermediate digest $h^{\alpha_i - k_i}(r')$ is defined only for $\alpha_i - k_i > 0$, it is computationally infeasible to generate this intermediate digest if in fact $k_i \ge \alpha_i$.

—if one of the key values of r is above the corresponding upper bound of the query scope, i.e., $k_i > \beta_i$ for some $1 \le i \le d$, the portal server pushes $h^{k_i - \beta_i}(r')$, and the rest of the $h^{u_j - k_j}(r')$ and $h^{k_j - l_j}(r')$ digests for this record, into the query answer.

At the end of this process, the query answer contains a series of $\langle scope, psum\rangle$ from the internal nodes, $\langle scope, h(psum), numChild\rangle$ from the leaf nodes, digests for records in those leaf nodes, and an aggregated signature.

Upon receiving the query answer, the user tests whether the scopes in the answer cumulatively cover the query scope, and whether the components of the answer match the aggregated signature:

(2a) For each record r in the answer that has a key value below the corresponding lower bound of the query scope, i.e., $k_i < \alpha_i$ for some $1 \le i \le d$, the user hashes the intermediate digest, $h^{\alpha_i - k_i}(r')$, $(u_i - \alpha_i)$ more times to derive $h^{u_i - k_i}(r')$; u_i is the upper bound along dimension i of the leaf node that contains r. The computed $h^{u_i - k_i}(r')$ is then combined with the rest of the $h^{u_j - k_j}(r')$ and $h^{k_j - l_j}(r')$ digests received from the portal server, to obtain the overall record digest.

(2b) For each record r in the answer that has a key value above the corresponding upper bound of the query scope, i.e., $k_i > \beta_i$ for some $1 \le i \le d$, the user hashes the intermediate digest, $h^{k_i - \beta_i}(r')$, $(\beta_i - l_i)$ more times to derive $h^{k_i - l_i}(r')$; l_i is the lower bound along dimension i of the leaf node that contains r. The computed $h^{k_i - l_i}(r')$ is then combined with the rest of the $h^{u_j - k_j}(r')$ and $h^{k_j - l_j}(r')$ digests received from the portal server, to obtain the overall record digest.

(2c) For each record r that is within the query scope, compute the overall record digest from its values.

(2d) The above record digests, the ⟨scope, psum⟩ pairs for the internal nodes, and the ⟨*scope*, h(*psum*), *numChild*⟩ triplets for the leaf nodes together should match the aggregated signature.

If both of the above tests succeed, the user can proceed to add up the partial sums and record values in the answer to derive the desired query aggregate.

7.2 Information Leakage

With our multi-attribute range aggregate scheme, the only extra information returned in a query answer are (1) the scope of the leaf nodes in the answer, which may extend beyond the query scope; and (2) the number of records in each leaf node in the answer. For records in those leaf nodes that are outside of the query scope, only digests of the form $h^a(r')$ are disclosed. It is not feasible for the user to perform a brute-force attack to recover any record content from those digests. Therefore our scheme preserves the confidentiality of records that are external to the query scope.

7.3 Data Organization

As explained above, whenever the scope of an internal node of the KDB+PS tree partially overlaps the query scope, the search proceeds to all the child nodes of that internal node. We decide to keep the scope, partial sum, and *numChild* attributes with each child pointer within the parent node, so that child nodes that do not overlap the query scope can be skipped over directly without generating separate random I/Os. Therefore, in our implementation, each node of the KDB+PS tree corresponds to a physical block that stores the node signature as well as pointers for several child nodes, each child pointer having the format ⟨*scope*, *psum*, *numChild*, *childPtr*⟩ where *childPtr* indicates the location of the physical block for that child node.

For a d-dimension space, a scope is represented as a pair of lower and upper bounds per dimension, with a total size of $2d$ integers or $8d$ bytes. In addition, *psum* requires double integer space of 8 bytes assuming that the S attribute of the underlying records is an integer, while *numChild* and *childPtr* take up 4 bytes each. The total size needed for each child pointer within a node/block is thus $8d + 16$ bytes. With 4-KByte blocks, the fan-out of the KDB+PS tree can be up to $\lfloor \frac{4096 - sign}{8d+16} \rfloor$, where *sign* is the signature size.

7.4 Extensions

Our approach of incorporating partial sums into a KDB-tree has been employed before, for example in the BA-tree in Zhang et al. [2002]. The authentication provisions in our KDB+PS scheme can therefore be applied easily to the BA-tree, as well as other space partitioning index structures like the Quadtree [Samet 1984]. Extension to data partitioning index structures, such as aR-tree [Papadias et al. 2001] and aRB-tree [Papadias et al. 2002], is not so straightforward though. Since data partitioning structures capture only

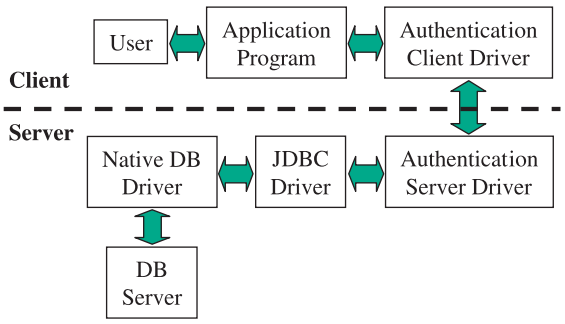


Fig. 13. Application architecture.

occupied regions in the index space, proving completeness is a challenge–how to prove whether an aggregate result accounts for all the data partitions within the query range. Our earlier work in Cheng et al. [2006] addressed this challenge for range selection using R-trees, but that scheme does not extend to range aggregates. We intend to do a thorough study in the future on how to add aggregate authentication capability to different multidimensional index structures.

Finally, our authenticated KDB+PS tree applies to distributive and algebraic aggregation functions [Gray et al. 1997] in general, by simply substituting the SUM function with the desired aggregation function (e.g., COUNT, MIN, and MAX).

8. EXPERIMENTS

This section describes one way to implement the authentication mechanisms introduced earlier. Our implementation targets applications that use JDBC. Obviously, other application architectures may necessitate different implementation techniques, but that is beyond the scope of this paper. Following that, we present experiment results obtained with our implementation.

8.1 Application Architecture

A common database application architecture contains the following: (1) The user interacts with an application program (possibly with a GUI). (2) The application program invokes APIs of a JDBC driver. (3) The JDBC driver calls the native (proprietary) database driver. (4) The database driver passes the query to the database server. Our implementation uses Oracle Database 10g as the server.

To authenticate the query answer generated by the portal server, we need to effect some minor changes to the database architecture. The modified architecture is depicted in Figure 13:

(1) Assuming that the source of the application program is available, we modify it to pass its query to our authentication client driver instead of invoking the JDBC driver. Alternatively, we can override the JDBC classes if the application program cannot be modified directly.

Table IV. Resource and Workload Parameters

Parameter	Description	Default
CPUSpeed	Clock speed of Pentium-4 CPU (GHz)	3
MemSize	DDR-400 memory (Gbytes)	2
BuffSize	Buffer allocation per partial sum hierarchy (Mbytes)	2
DiskSize	Size of EIDE disk – 7200 rpm, 8 MB cache (Gbytes)	160
BlkSize	Physical block size (Kbytes)	4
NetSpeed	Network speed (Mbps)	100
n	# records in database table	10 mil
RecSize	Record size (bytes)	512
q	# records in query range	100000
f	Fan-out factor of partial sum hierarchy	500

(2) The authentication client driver passes the query to the authentication server running on the portal server. The authentication server then formulates an accompanying query to pull data for composing verification information for the user query, and passes both queries to the JDBC driver.
(3) When the JDBC driver returns the answers, the authentication server creates the verification information, and sends it with the query answer to the authentication client.
(4) The authentication client driver verifies the query answer and returns it to the application program, or signals an error if the verification fails.

Moreover, we rewrite the application program to offer users an option to enable/disable authentication. If the application program cannot be modified, authentication can be turned on by default in the authentication client for all queries and databases.

Since our authentication driver sits on top of the JDBC driver, there is no need for any database vendor to modify the DBMS (except to optimize performance); there is also no need for a new query standard. During deployment, the authentication client could be rolled out alongside the application program, possibly using one of the standard enterprise application deployment platforms like Tivoli.

8.2 On Single-Attribute Aggregates

8.2.1 *Experiment Set-up.* For the experiments, we install the modified application architecture in Section 8.1 on a pair of PCs. The resource parameters of the PCs are listed in Table IV.

Algorithms: One objective of the experiments is to understand the trade-offs between various configurations of our partial sum hierarchy scheme. In particular, we wish to evaluate the integrated partial sum hierarchy with binary branching (iPSH-2) versus the maximum fan-out of 22 (iPSH-22) allowed by the default parameter settings in Table IV, and also the integrated partial sum hierarchy versus the partial sum hierarchy with separate *PtrAttr* file (PSH-PA).

Another objective that we wish to achieve is to profile the cost of proving completeness and maintaining precision in the range aggregate results, (i.e., of avoiding disclosure of record values or partial sums that are beyond the query range). For this reason, we select the HAMS scheme in Ho et al. [1997] as baseline. Recall that HAMS reveals the prefix sum of the entries below the query range. Moreover, HAMS does not provide any proof that the correct prefix sums are returned (e.g., that the lower prefix sum indeed aggregates right up to the record immediately below the query range).

Detailed configuration of the various schemes, with the default parameter settings in Table IV, are as follows:

(1) iPSH-2: For f=2, the partial sum hierarchy has a height of 24. Each node has a size of $180f$=360 bytes (see Table III), so 2 Mbytes of allocated buffer space is sufficient to cache 5825 nodes, or the top 12 layers of the partial sum hierarchy.
(2) iPSH-22: For f=22, the partial sum hierarchy has a height of 6. Each node has a size of $180f$=3,960 bytes, so 2 Mbytes of buffer space is sufficient to cache 529 nodes, or the top two layers of the partial sum hierarchy.
(3) PSH-PA: With the pointer attributes in a separate *PtrAttr* file, the partial sum hierarchy can support f=510 as explained in Section 6.4. This leads to a tree height of only 3. Moreover, 2 Mbytes of buffer can hold the top two layers of the partial sum hierarchy, so physical I/Os are generated only for queries that require one or more partial sums in the leaf nodes. Each bucket in the *PtrAttr* file has a size of $176f$=89760 bytes, or 22 $\times$ 4-Kbyte blocks. Retrieving all the partial sums in a node of the partial sum hierarchy thus generates one random I/O followed by 21 sequential I/Os to the associated bucket. To realize the benefits of sequential I/Os, the *PtrAttr* file is created in a raw disk partition.
(4) HAMS: For one-attribute aggregates, we implement the HAMS scheme as a B+-tree on the sort keys, constructed over a data file that holds pairs of record key and prefix sum. The prefix sum of record i aggregates the values of record 1 up to record i. With 4 bytes per record key and per pointer, the B+-tree can support approximately f=510. Thus 10 million records lead to a tree height of 3, with the top two layers fitting into 2 Mbytes of buffer.

Performance Metrics. The primary metrics that we use to measure the various range aggregate schemes are: (1) server retrieval time, and (2) query answer size, which determines network transmission and user computation costs.

8.2.2 *Baseline Experiment.* At the end of Section 6.1, we deduced that a partial sum hierarchy with a small fan-out produces smaller query answers than a corresponding partial sum hierarchy with a large fan-out. Our baseline experiment is designed to study the trade-offs in selecting the fan-out. We run 10,000 range aggregate queries over randomly selected intervals of the database, according to the default experiment settings in Table IV. Figures 14 and 15 show the average query answer size and server I/O time, respectively, for the various schemes.

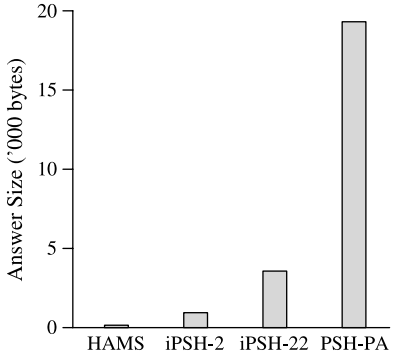


Fig. 14. Baseline – answer size.

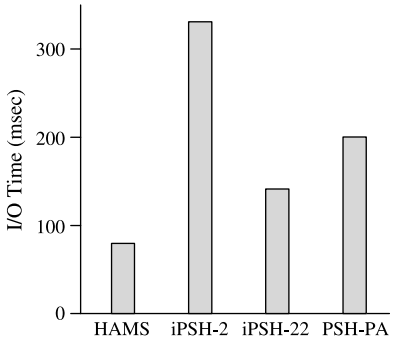


Fig. 15. Baseline – I/O time.

HAMS incurs at most 4 random I/Os per query–one leaf node retrieval plus one data file access for the prefix sum from the first record up to the record just below the query range, and another two I/Os for the prefix sum from the first record up to the last record in the query range. The query answer contains two prefix sums (8 bytes each) with their corresponding key values (4 bytes each), and a 128-byte signature.

Results for the 3 PSH schemes confirm that a smaller fan-out in the partial sum hierarchy indeed leads to smaller query answers. However, a larger fan-out means there are fewer nodes in the partial sum hierarchy, and thus fewer random I/Os during query processing. This is evident in the lower retrieval time of iPSH-22 relative to iPSH-2.

As for PSH-PA, detailed measurements show that its strategy of clustering neighboring partial sums indeed produces fewer random I/Os and more sequential I/Os than iPSH-22. Unfortunately, PSH-PA's large fan-out leads to a much larger number of partial sums being returned in the query answer, to the extent that it overwhelms the savings from sequential I/Os.

Overall, the best-performing scheme for this experiment is between iPSH-2 and iPSH-22, depending on the relative significance of network transfer time (as determined by the query answer size) versus server retrieval time. With a

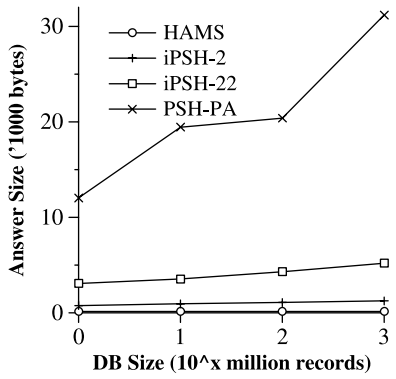


Fig. 16. DB size – answer size.

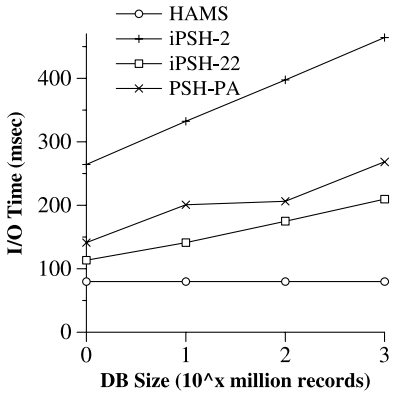


Fig. 17. DB Size – I/O time.

fast network speed of 100 Mbps, server retrieval time dominates, so iPSH-22 is superior. iPSH-2 becomes the preferred choice only for network speed below 100 kbps.

8.2.3 *Sensitivity to Database Size.* In the second experiment, we investigate the sensitivity of the schemes to the database size. Here we set n = 1, 10, 100 and 1000 million records, and keep the other experiment parameters at their default values in Table IV. The results are summarized in Figures 16 and 17.

Referring to equation (7), the number of partial sums in the query answer is bounded by $(f-1)\cdot(\lfloor\frac{log_2 q}{log_2 f}\rfloor+1)$. Any increase in the query range q is thus magnified by a factor of $\frac{f-1}{log_2 f}$, meaning that a larger fan-out is more sensitive to q. With a fixed query selectivity, q is proportional to n. Thus iPSH-2 demonstrates the smallest growth among the 3 PSH schemes, while PSH-PA with the largest fan-out deteriorates the fastest. This is confirmed in Figure 16. Again, iPSH-22 is the best for network speed higher than about 120 Kbps, and iPSH-2 is preferable for slower networks.

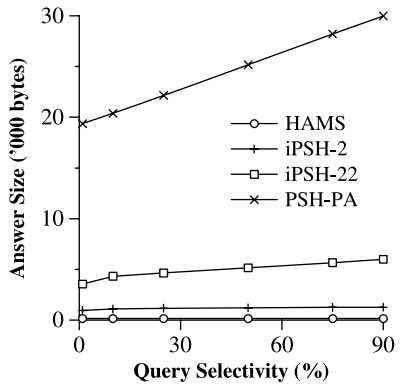


Fig. 18. Query selectivity – answer size.

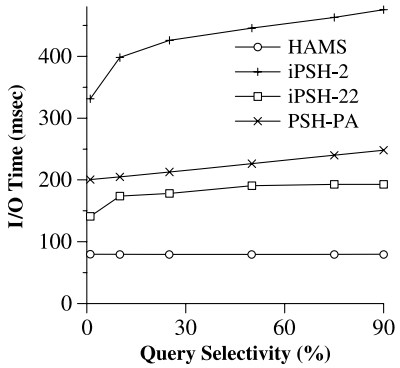


Fig. 19. Query selectivity – I/O time.

8.2.4 *Sensitivity to Query Selectivity.* Our next experiment aims to investigate the sensitivity of the various schemes to the query selectivity. We fix all the experiment parameters at their default settings in Table IV, except for the query size q which is varied from 1% to 90% of the database. Figures 18–20 show the resulting average query answer size, server I/O time, and client processing cost respectively.

For the same reasons as in the previous experiment, PSH-PA suffers the worst increase in query answer size as query selectivity grows, followed by iPSH-22. iPSH-2's answer size is almost unchanged, while HAMS' is constant. However, all 4 schemes are still cheaper than returning the underlying record values directly, which would have generated a query answer size in excess of 400KB even for q = 1% of the records.

In terms of server retrieval time, iPSH-2 and iPSH-22 incur significant penalties initially. This is because their hierarchies contain more levels and nodes in the lower levels cannot be cached in the buffer, so a larger query range induces more random I/Os. The increase in penalty tapers off gradually, however, as larger query ranges reach the upper levels of the partial sum

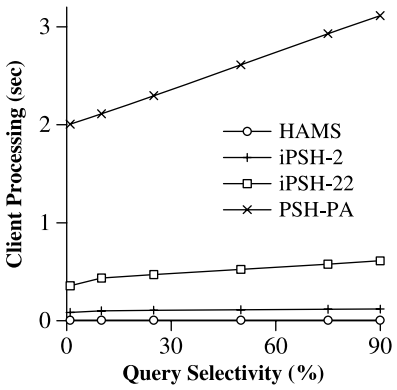


Fig. 20. Query selectivity – client processing time.

hierarchies that are buffered. In contrast, PSH-PA slows down very gradually, as the increase in the number of partial sums in the query answer introduces sequential (rather than random) I/Os.

As expected, the client processing cost is proportional to the query size. Among the proposed schemes, iPSH-2's smaller answer size results in the lowest client processing, while PSH-PA is the worst because of the larger answer size. We also note that the client processing time for iPSH-2 and iPSH-22 is below 1 sec, even when more than 90% of the database is aggregated over. This low processing cost shows the practicality of our authentication scheme.

Overall, iPSH-22 is still the best performer, with iPSH-2 starting to outperform the rest only when the network speed falls below 100-150 Kbps.

8.2.5 *Observations.* The experiments highlight that, on one hand, a higher fan-out in the partial sum hierarchy produces larger query answers, and is more susceptible to variations in the query scope and database size. On the other hand, small fan-outs suffer from poor server retrieval time. Overall, iPSH-22 strikes the best balance at high network speeds, while slow networks favor iPSH-2.

8.3 On Multi-Attribute Aggregates

To evaluate the performance of our proposed KDB+PS scheme, we run it on the same system and experiment parameters as in Section 8.1 and Table IV, except that here the records are mapped into multidimensional space. The results are summarized in Figures 21 to 27.

8.3.1 *Experiment Results.* For the first experiment, we fix n=10 million, d=3, and study our scheme's sensitivity to the query scope. We vary the query scope size from 1 record to 20,000 records. Here we observe an interesting cyclicality in the results in Figures 21 and 22. As the query scope increases, it overlaps more leaf nodes and hence the number of records in those leaf nodes that have to be proven to be irrelevant, as well as the I/Os in retrieving them, both go up. Beyond a certain threshold, however, the query scope becomes large enough to envelop an internal node of the KDB+PS tree. When this

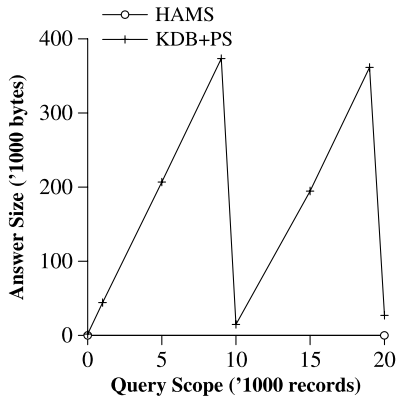


Fig. 21. Query scope – answer size.

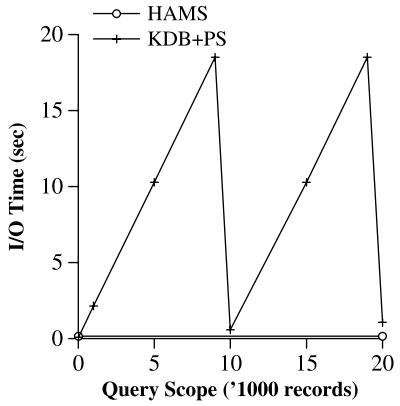


Fig. 22. Query scope – I/O time.

happens, a large portion of the query answer is captured by the partial sum of that one internal node, thus reducing the answer size and I/O time at once. This pattern is repeated at regular intervals as the query scope grows further.

Next, we fix d=3 and the query scope to contain ten records, while varying the database size. Figures 23 and 24 show that increasing database size has no observable impact on the query answer size. In contrast, the I/O time of KDB+PS increases gradually because the tree grows deeper with larger databases.

Turning our attention to our scheme's sensitivity to the number of dimensions, we fix n=10 million and the query scope to contain ten records, while varying the number of dimensions. The results, depicted in Figure 25, show that the answer size increases exponentially. This is due to two factors: (1) The number of hypercubes corresponding to leaf nodes that the query scope overlaps grows exponentially with the number of dimensions. (2) The size of the "scope" attribute in the child pointer is proportional to the dimensionality. With a higher number of dimensions and hence larger child pointers, the

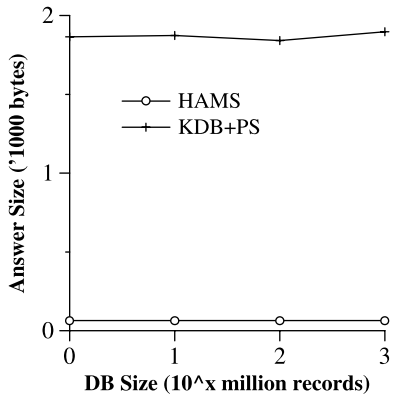


Fig. 23. DB size – answer size.

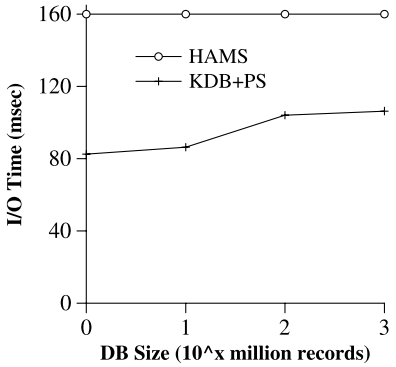


Fig. 24. DB size – I/O time.

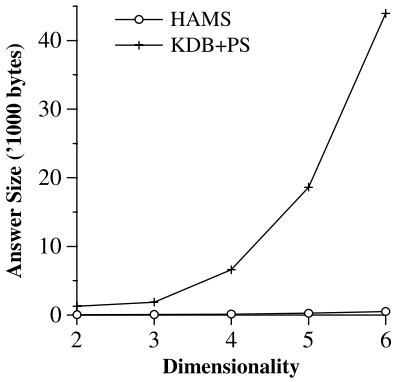


Fig. 25. Dimensionality – answer size.

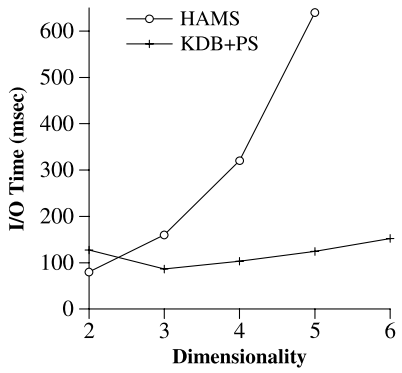


Fig. 26. Dimensionality – I/O time.

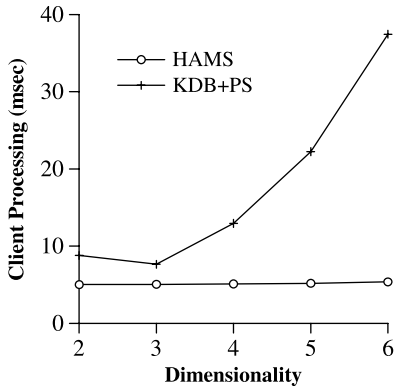


Fig. 27. Dimensionality – client processing time.

fan-out of the KDB+PS tree shrinks, so each leaf node contains more records. Consequently, there are more records in the hypercubes overlapping the query scope, for which the portal server needs to provide digests to prove that they are not part of the query answer.

Next, we examine Figure 26. For d=2, the average number of records per leaf node is less than the query answer size, and we observe that the query scope often overlaps 3 hypercubes along a dimension. For d=3, each leaf node contains more records than the query answer size, so the query scope overlaps only 1 or 2 hypercubes along each dimension almost all the time. Thus the portal server incurs significantly fewer I/Os to retrieve the overlapped hypercubes. Beyond this point, the number of overlapped hypercubes per dimension cannot lower any further, and the overall number of overlapped hypercubes again increases with the dimensionality.

From Figure 27, we observe that (1) the client computation cost is negligible–at 6 dimensions, the authentication costs no more than 40 milliseconds. Thus, our scheme is practical. (2) As in the single-attribute case, the client processing time grows with the returned answer size. (3) For d = 3, the

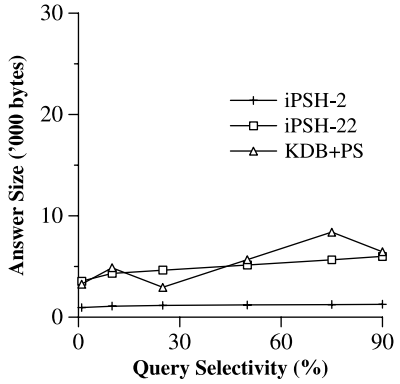


Fig. 28. 1-D versus multi-D – answer size.

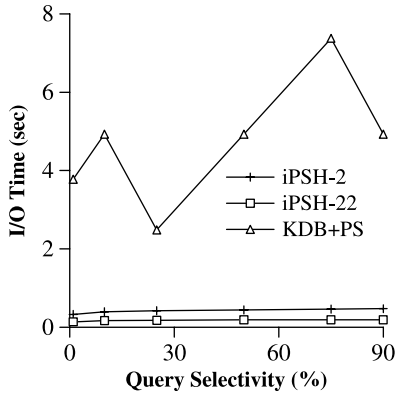


Fig. 29. 1-D versus multi-D – I/O time.

lower cost (relative to $d = 2$) follows from the low overlap between the query scope and the hypercubes.

Across the three experiments, HAMS consistently produces very compact query answers. However, its I/O time grows exponentially with the dimensionality. We also have had to provision huge storage spaces as HAMS' storage overhead is roughly proportional to n^d. Again, we emphasize that HAMS does not support authentication of query answers.

Finally, Figures 28 and 29 compare KDB+PS with the single-dimensional iPSH-2 and iPSH-22. (We leave out PSH-PA which has been shown to be inferior to iPSH-2 and iPSH-22, and HAMS which does not support authentication.) The figures indicate that, while KDB+PS produces comparable answer sizes as iPSH-22, the former incurs significantly higher I/O costs. Therefore, the iPSH schemes are superior for single-attribute aggregate queries.

8.3.2 *Observations.* The experiments show that, in the course of enabling authentication, KDB+PS incurs larger transmission and verification overheads compared to HAMS. While KDB+PS is less I/O intensive than

HAMS, it is still more expensive than the iPSH schemes for single-attribute aggregates.

9. CONCLUSION

In this article, we present schemes for authenticating relational range selection, as well as single- and multi-attribute range aggregate query answers generated by online servers that may become compromised over time. The schemes enable each query answer to be checked for completeness (i.e., all records within the query range contribute to the answer) and authenticity (i.e., the correct values are returned). The schemes do not disclose more data than necessitated by the query conditions, hence they do not contravene access control mechanisms that rewrite queries dynamically. Moreover, the schemes are computationally secure and introduce low query processing and update overheads.

ACKNOWLEDGMENT

We thank the reviewers of this paper for their valuable comments and suggestions. We would like to acknowledge Arpit Jain and Krithi Ramamritham who codeveloped an early version of the authentication scheme for range selection.

REFERENCES

ANDERSON, R., NEEDHAM, R., AND SHAMIR, A. 1998. The Steganographic file system. In *Information Hiding, 2nd International Workshop*. Portland, OR. D. Aucsmith Ed.

BONEH, D., GENTRY, C., LYNN, B., AND SHACHAM, H. 2003. Aggregate and verifiably encrypted signatures from bilinear maps. In *Proceedings of Advances in Cryptology (EUROCRYPT'03)*, E. Biham Ed., *Lecture Notes in Computer Science*, Springer-Verlag. 416–432.

CHENG, W., PANG, H., AND TAN, K.-L. 2006. Authenticating multi-dimensional query results in data publishing. In *Proceedings of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*.

CHOKANI, S. 1992. Trusted Products Evaluation. *Comm. ACM 35,* 7, 64–76.

CHUN, S.-J., CHUNG, C.-W., LEE, J.-H., AND LEE, S.-L. 2001. Dynamic update cube for range-sum queries. In *Proceedings of the International Conference on Very Large Data Bases (VLDB'01)*. 521–530.

DAMIANI, E., DI VIMERCATI, S. D. C., FORESTI, S., JAJODIA, S., PARABOSCHI, S., AND SAMARATI, P. 2005a. Metadata management in outsourced encrypted databases. In *Proceedings of the 2nd VLDB Workshop on Secure Data Management (SDM'05)*.

DAMIANI, E., DI VIMERCATI, S. D. C., FORESTI, S., SAMARATI, P., AND VIVIANI, M. 2005b. Measuring Inference Exposure in Outsourced Encrypted Databases. In *Proceedings of the 1st Workshop on Quality of Protection*.

DEVANBU, P., GERTZ, M., MARTEL, C., AND STUBBLEBINE, S. 2000. Authentic data publication over the Internet. In *14th IFIP Working Conference in Database Security*. 102–112.

DEVANBU, P., GERTZ, M., MARTEL, C., AND STUBBLEBINE, S. 2003. Authentic data publication over the Internet. *J. Comput. Secur. 11*, 291–314.

DRIVECRYPT. Secure hard disk encryption. http://www.drivecrypt.com.

DSS. 1991. Proposed federal information processing standard for digital signature standard (DSS). *Federal Register 56, 169*, 42980–42982.

EFS. Encrypting file system for windows 2000. http://www.microsoft.com/windows2000/techinfo/howitworks/security/encrypt.asp.

GEFFNER, S., AGRAWAL, D., AND ABBADI, A. E. 2000. The dynamic data cube. In *Proceedings of the International Conference on Extending Database Technology* (*EDBT'00*). 237–253.

GRAY, J., CHAUDHURI, S., BOSWORTH, A., LAYMAN, A., REICHART, D., VENKATRAO, M., PELLOW, F., AND PIRAHESH, H. 1997. Data Cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. *J. Data Min. Knowl. Discov. 1,* 1, 29–53.

HACIGÜMÜS, H., IYER, B. R., LI, C., AND MEHROTRA, S. 2002. Executing SQL over encrypted data in the database-service-provider model. In *Proceedings of the ACM International Conference on Management of Data* (*SIGMOD'02*). 216–227.

HO, C.-T., AGRAWAL, R., MEGIDDO, N., AND SRIKANT, R. 1997. Range Queries in OLAP data cubes. In *Proceedings of the ACM International Conference on Management of Data* (*SIGMOD'97*). 73–88.

LI, F., HADJIELEFTHERIOU, M., KOLLIOS, G., AND REYZIN, L. 2006. Dynamic authenticated index structures for outsourced databases. In *Proceedings of the 25th ACM International Conference on Management of Data* (*SIGMOD'06*). 121–132.

LI, J. AND OMIECINSKI, E. R. 2005. Efficiency and security trade-off in supporting range queries on encrypted databases. In *Proceedings of the 19th Annual IFIP WG 11.3 Working Conference on Data and Applications Security*.

MA, D., DENG, R. H., PANG, H., AND ZHOU, J. 2005. Authenticating query results from untrusted servers. In *Proceedings of the 7th International Conference on Information and Communications Security*.

MARTEL, C., NUCKOLLS, G., DEVANBU, P., GERTZ, M., KWONG, A., AND STUBBLEBINE, S. 2004. A general model for authenticated data structures. *Algorithmica 39,* 1, 21–41.

MERKLE, R. 1989. A certified digital signature. In *Proceedings of Advances in Cryptology (Crypto'89),* Lecture Notes in Computer Science. vol. 0435. 218–238.

MYKLETUN, E., NARASIMHA, M., AND TSUDIK, G. 2004. Authentication and integrity in outsourced databases. In *Proceedings of the Network and Distributed System Security Symposium*.

NARASIMHA, M. AND TSUDIK, G. 2006. Authentication of outsourced databases using signature aggregation and chaining. In *Proceedings of the 11th International Conference on Database Systems for Advanced Applications*, (*DASFAA'06*), 420–436.

NEUMAN, B. AND TSO, T. 1994. Kerberos: An authentication service for computer networks. *IEEE Comm. 32,* 9, 33–38.

ORACLE VPD. 2002. The virtual private database in Oracle9ir2: An Oracle technical white paper. http://otn.oracle.com/deploy/security/oracle9ir2/pdf/vpd9ir2twp.pdf.

PANG, H., JAIN, A., RAMAMRITHAM, K., AND TAN, K.-L. 2005. Verifying completeness of relational query results in data publishing. In *Proceedings of the ACM International Conference on Management of Data* (*SIGMOD'05*). 407–418.

PANG, H. AND TAN, K. 2004. Authenticating query results in edge computing. In *IEEE International Conference on Data Engineering*. 560–571.

PANG, H., TAN, K., AND ZHOU, X. 2003. StegFS: A Steganographic file system. In *Proceedings of the 19th International Conference on Data Engineering*. Bangalore, India, 657–668.

PAPADIAS, D., KALNIS, P., ZHANG, J., AND TAO, Y. 2001. Efficient OLAP operations in spatial data warehouses. In *Proceedings of the 7th International Symposium on Spatial and Temporal Databases*. 443–459.

PAPADIAS, D., TAO, Y., KALNIS, P., AND ZHANG, J. 2002. Indexing spatio-temporal data warehouses. In *IEEE International Conference on Data Engineering*. 166–175.

PGPDISK. http://www.pgpi.org/products/pgpdisk/.

PRZYDATEK, B., SONG, D., AND PERRIG, A. 2003. SIA: Secure information aggregation in sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, (SenSys'03)*. 255–265.

RIVEST, R. 1992. RFC 1321: The MD5 Message-Digest Algorithm. Internet Activities Board.

RIVEST, R. AND SHAMIR, A. 2001. PayWord and MicroMint: Two simple micropayment schemes. In http://theory.lcs.mit.edu/ rivest/RivestShamir-mpay.pdf.

RIVEST, R., SHAMIR, A., AND ADLEMAN, L. 1978. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM 21,* 2, 120–126.

ROBINSON, J. 1981. The K-D-B-Tree: A search structure for large multidimensional dynamic indexes. In *Proceedings of the ACM International Conference on Management of Data* (*SIGMOD'81*). 10–18.

SAMET, H. 1984. The Quadtree and Related Hierarchical Data Structures. *ACM Comput. Surv. 16,* 2, 187–260.

SANDHU, R. AND SAMARATI, P. 1994. Access Control: Principles and Practice. *IEEE Comm. 32,* 9, 40–48.

SHA. 2001. Secure hashing algorithm. National Institute of Science and Technology. FIPS 180-2.

ZHANG, D., TSOTRAS, V. J., AND GUNOPULOS, D. 2002. Efficient aggregation over objects with extent. In *Symposium on Principles of Database Systems (PODS ACM SIGACT-SIGMOD-SIGART'02)*. 121–132.

Received June 2006; revised February 2007; accepted August 2007