6-2005

Verifying Completeness of Relational Query Results in Data Publishing

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

Arpit JAIN
Indian Institute of Technology, Bombay

Krithi RAMAMRITHAM
Indian Institute of Technology, Bombay

Kian-Lee TAN
National University of Singapore

DOI: https://doi.org/10.1145/1066157.1066204

Citation

PANG, Hwee Hwa; JAIN, Arpit; RAMAMRITHAM, Krithi; and TAN, Kian-Lee. Verifying Completeness of Relational Query Results in Data Publishing. (2005). *SIGMOD 2005: Proceedings of the ACM SIGMOD International Conference on Management of Data, Baltimore, Maryland, June 14-16, 2005*. 407-418. Research Collection School Of Information Systems.
Available at: https://ink.library.smu.edu.sg/sis_research/1136

Verifying Completeness of Relational Query Results in Data Publishing

HweeHwa Pang[*†] Arpit Jain[‡] Krithi Ramamritham[‡] Kian-Lee Tan[§]

ABSTRACT

In data publishing, the owner delegates the role of satisfying user queries to a third-party publisher. As the publisher may be untrusted or susceptible to attacks, it could produce incorrect query results. In this paper, we introduce a scheme for users to verify that their query results are complete (i.e., no qualifying tuples are omitted) and authentic (i.e., all the result values originated from the owner). The scheme supports range selection on key and non-key attributes, project as well as join queries on relational databases. Moreover, the proposed scheme complies with access control policies, is computationally secure, and can be implemented efficiently.

1. INTRODUCTION

In data publishing, a data owner delegates the role of satisfying user queries to a third-party publisher [10, 16]. This model is applicable to a wide range of computing platforms, including database caching [12], content delivery network [26], edge computing [14], P2P databases [11], etc.

The data publishing model offers a number of advantages over conventional client-server architecture where the owner also undertakes the processing of user queries. By pushing application logic and data processing from the owner out to multiple publisher servers situated near user clusters, network latency can be reduced. Adding publisher servers is also likely to be a cheaper way to achieve scalability than fortifying the owner's data center and provisioning more network bandwidth for every user. Finally, the data publishing model removes the single point of failure in the owner's data center, hence reducing the database's susceptibility to denial of service attacks and improving service availability.

To illustrate, a financial information provider could push historical stock prices, together with analytics software, to proxy servers operated by partner ISPs (Internet Service Provider). Such an arrangement enables users to run different pricing and risk models off the proxy servers directly instead of depending on a central data center that might be situated thousands of miles away, thus reducing communication latency and processing bottlenecks.

Since the publisher servers are outside of the administrative domain of the data owner, and in fact may reside on poorly secured platforms, the query results that they generate cannot be accepted at face value, especially where they are used as the basis for critical decisions. Instead, there must be provisions for the user to check the "correctness" of a query result, in terms of:

- *Authenticity*: All the values in the result originated from the data owner. For example, for the Employee table and query in Figure 1, the publisher indeed returns the result { [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 result; e.g., the result { [005, A, 2000, ..] , [001, D, 8010, ..] } for the query in Figure 1 is incomplete as [002, C, 3500, ..] is omitted.

The problem of checking the authenticity of query results has been studied recently in [10, 13, 17, 20]. Most of the proposed solutions [10, 20] require the owner to build and sign a hierarchy of digests over each data set, which the query processor subsequently uses to construct a correctness-proof for each query result; whereas [13] and [18] allow the publisher to aggregate the signatures of the result tuples into a single result signature to reduce transmission and verification overheads. To the best of our knowledge, only the proposal by Devanbu et al in [10] provides for the verification of query result completeness. Their scheme works by exposing the tuples immediately beyond the left and right boundaries of the query result for user inspection, and proving to the user that all the result tuples (including the two boundary tuples) are contiguous in the database.

For example, consider the Employee table in Figure 1. which is sorted on the Salary attribute. With Devanbu's

[*]Institute for Infocomm Research, Heng Mui Keng Terrace, Singapore 119613
[†]School of Information Systems, Singapore Management University, 469 Bukit Timah Road, Singapore 259756
[‡]Department of Computer Science and Engineering, Indian Institute of Technology, Bombay, Powai Mumbai 400076, India
[§]Department of Computer Science, National University of Singapore, 3 Science Dr 2, Singapore 117543

Emp:

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

Figure 1: Example Database

scheme, the query in Figure 1, issued by the HR manager, would retrieve { [005, A, 2000, ..], [002, C, 3500, ..], [001, D, 8010, ..], [004, B, 12100, ..] }. By proving those four records are contiguous in the database, the publisher can satisfy the HR manager that the result contains all the qualifying records, as the last record in the result has a salary that exceeds her query condition.

Now, suppose that the access policy on the database states that human resource (HR) executives can access only records of employees earning less than $9000, while records of (presumably more senior) employees earning more than that should be visible only to the HR manager. When the same query in Figure 1 is submitted by a HR executive, the access control mechanism would rewrite the query to: SELECT * FROM Emp WHERE Salary < 9000. Devanbu's scheme would then return the same set of records as for the HR manager, including the record with a salary of $12100, in an attempt to prove that all records with salary less than $9000 are included. Clearly, this contradicts the stipulated access policy for the HR executive.

Contributions: The objective of our work is to devise an authentication mechanism that enables a user to verify the *completeness* of a relational query result generated by an untrusted server, without compromising any access control rules on the database.

We first introduce a basic scheme that generates, for an ordered list $(r_1, r_2, .., r_n)$ and a query range $[a, b]$, cryptographic proof that the two entries bordering the answer $(r_i, .., r_j)$, $1 < i \leq j < n$ fall outside the query range, i.e., $r_{i-1} < a$ and $b < r_{j+1}$. This scheme can be adapted easily to support queries that retrieve up to all the data entries. Moreover, the scheme is secure in the sense that it is computationally infeasible for a rogue publisher to devise such a proof for an incorrect query answer.

Building upon the basic 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. Thus, referring to the example in Figure 1, our scheme would enable the publisher to return only { [005, A, 2000, ..], [002, C, 3500, ..], [001, D, 8010, ..] }, and provide proof that the next record has a higher salary than stated in the query condition, without revealing directly or indirectly what that salary amount is. Besides range query that retrieves all the tuples within a specified interval on the sorted attribute (e.g. "Salary < 10000" in Figure 1), our scheme can also handle range query on unsorted attribute that pulls back multiple partitions of tuples, such as "Dept = 1" in Figure 1.

We also demonstrate that, with just a simple enhancement, our scheme can concurrently verify the *authenticity* of a query result. Finally, to make our scheme amenable to efficient implementation, we propose an algorithm that generates the correctness proof in time complexity that is logarithmic in the size of the domain underlying the elements r_i's in the ordered list. The resulting query and update overheads of our scheme are significantly lower than existing solutions like [10] and [20] that are based on digest hierarchies (specifically, Merkle Hash Trees as explained in the next section).

The remainder of this paper is organized as follows. Section 2 describes some cryptographic primitives for our work, the data publishing model and associated threats, as well as related work. The basic approach of our completeness verification scheme is introduced in Section 3. Section 4 then extends the scheme to support select-project-join queries on relational databases. Following that, Section 5 presents optimization techniques for the proposed scheme, and Section 6 analyses the computation and transmission costs incurred by the scheme. Finally, Section 7 concludes the paper and discusses avenues for future work.

2. BACKGROUND

This section begins by defining some cryptographic primitives. Following that, we present the target system deployment model and the associated security threats, before summarizing related work.

2.1 Cryptographic Primitives

Our proposed solution and many 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 [22] and SHA [6]. 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 the 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 [24] and DSA [5] are two commonly-used signature algorithms.

Signature aggregation: As introduced in [8], 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 2, which is intended for authenticating data values d_1, .., d_4; a detailed definition

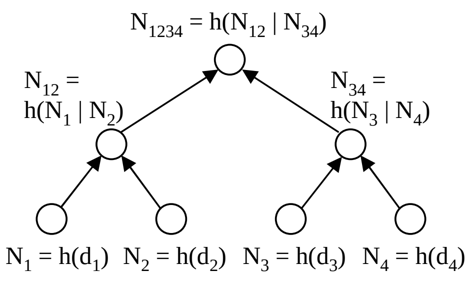


Figure 2: Example of a Merkle Hash Tree

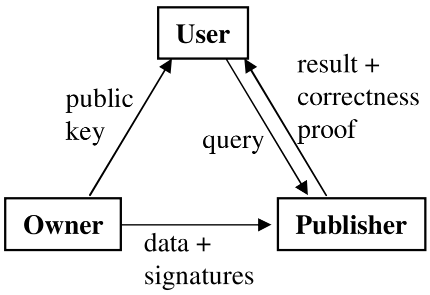


Figure 3: Data Publishing Model

can be found in [15]. 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 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 is the same as the signed N_{1234}. If so, d_1 is accepted; otherwise, d_1 has been tampered with.

2.2 System and Threat Models

Figure 3 depicts the data publishing model, which supports three distinct roles:

- The data owner maintains a master database, and distributes it with one or more associated signatures that proves the authenticity of the database. Any data that has a matching signature is accepted by the user to be trustworthy.

- The publisher hosts the database, and performs query processing on behalf of the owner, possibly after some query-rewriting to comply with the access control rules on the database. The access control model may be discretionary, mandatory or role-based (e.g., see [25]). Regardless of the exact model, the publisher should ensure that only data that satisfy the rewritten queries are returned, so as to avoid contradicting the access control rules. There could be several publisher servers that are situated at the edge of the network, near the user applications. The publisher is not required to be trusted, so the query results that it generates must be accompanied by some “correctness proof”, derived from the database and signatures issued by the owner.

- The user, who issues queries to the publisher explicitly, or else gets redirected to the publisher, e.g. by the owner or a directory service. To verify the query results, the user obtains the public key of the owner through an authenticated channel, such as a public key certificate issued by a certificate authority.

There are several security considerations in the data publishing model. Given that the publisher servers are not trusted, one concern is privacy of the data. Obviously, an adversary who gains access to the operating system or hardware of a publisher 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. [3], [2], [4]) and steganographic storage (e.g. [7], [21], [1]), 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. [9], [19], [25]) and are complementary to this work.

Our primary concern addressed in this paper is the threat that a dishonest publisher may return incorrect query results to the users, whether intentionally or under the influence of an adversary. An adversary who is cognizant of the data organization in the publisher server may attempt to make logical alterations to the data, thus inducing incorrect query results; 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 ([7, 21]), the adversary may still sabotage the database by overwriting physical pages within the storage volume. In addition, a compromised publisher server could be made to return incomplete query results by withholding data intentionally. Therefore mechanisms for users to verify the completeness of their query results are essential here.

2.3 Related Work

The existing work that is most relevant to this paper is [10], which describes a scheme for verifying the completeness and authenticity of query results produced by untrusted third-party publishers. The scheme requires the data owner to construct a Merkle hash tree (MHT) over each database table, and disseminate the signed root digest to users directly. To prove the completeness of the result for a range query $[a, b]$ over an ordered list $(r_1, r_2, .., r_n)$, the publisher needs to disclose to the user the two entries that are immediately below and above the query range, respectively. In other words, the query result becomes $(r_{i-1}, r_i, .., r_j, r_{j+1})$ where $r_{i-1} < a \leq r_i$, $r_j \leq b < r_{j+1}$, $1 < i \leq j < n$. The user can then check the expanded result, together with an associated verification object (VO), against the signed digest for the MHT.

This work by Devenbu et al [10] is among the first solutions for authenticating query results in data publishing. Their scheme exhibits the following characteristics:

1. A MHT is constructed for every sort-order on a table.

2. Each VO contains digests all the way to the root of the tree index, thus the VO grows linearly to the query

result and logarithmically to the base table.

3. Even attributes that are supposed to be filtered out through projection must be offered to users for verification. Besides potentially conflicting with column-based access control policies, this also could lead to wasteful data transfer especially if the filtered attributes are BLOBs, e.g. the photo attribute in Figure 1.

4. To check for completeness, tuples beyond the left and right boundaries of the query result must be exposed to the user; this could contradict row-based access control policies on the database, as explained in the Introduction.

5. The scheme works for range query on sorted attribute, but not range query on unsorted attribute that pulls back several segments of tuples.

[20] proposed a VB-tree that augments the B+-tree with a hierarchy of digests. The digests are computed using a cumulative and commutative hash function, thus overcoming limitation (1) above. To avoid limitation (2), each node digest in the VB-tree is signed, so the VO only needs to contain proofs for the smallest subtree that envelops the query result. Moreover, the VB-tree is built from the tuple attributes (instead of working at tuple granularity); this circumvents limitation (3). However, VB-tree does not check for completeness of query results.

More recently, Ma et al introduced an even more efficient authentication scheme in [13]: The owner constructs an MHT on the attribute values of individual tuples, then signs the root digest of each MHT. To prove the authenticity of a query result, the server only needs to supply a VO for those attributes that are projected out in each tuple, plus one single signature that combines the signed digest of the result tuples using the signature aggregation scheme in [8] or [18]. Again, query result completeness was not addressed.

3. BASIC APPROACH

The goal of our work is to devise an authentication mechanism that enables a user to verify the query results generated by an untrusted server. Our proposed mechanism is designed to satisfy all the following objectives, of which the first four are security requirements while the last is intended to ensure that the overheads incurred by the mechanism are acceptable in practice:

- Completeness – The user can verify that all the records that satisfy the conditions of a query are included in the result.
- Precision – Only records and attribute values that satisfy the conditions of each query are returned. The motivation is to avoid contradicting access control rules on the database, which would become (part of) the query conditions through query rewriting.
- Authenticity – The user can check that all the values in a query result originated from the owner; they have not been tampered with, nor have spurious records been introduced.
- Security – It is computationally infeasible for the publisher to cheat by generating a proof for an incorrect query result.
- Efficiency – The procedure for the publisher to generate the proof for a correct (i.e., complete and authentic) query result has polynomial complexity. Likewise the procedure performed by the user to verify a query result has polynomial complexity.

In Section 3.1, we first solve the problem of verifying completeness, while concurrently achieving high precision, of the output for a greater-than predicate on a sorted list. Following that, Section 3.2 gives an informal proof of the security of our proposed scheme, and Section 4 extends the scheme for relational Selection-Projection-Join queries. We also show in Section 4.1 that our scheme can ensure authenticity, while optimization techniques to achieve efficiency are presented in Section 5.

3.1 Greater-Than Predicate

Problem Definition: Consider the data publishing model in Section 2.2. Suppose the data owner creates a sorted list of distinct values, $\mathbf{R} = (r_1, r_2, .., r_n)$, $r_i \in (L, U)\ \forall\ 1 \le i \le n$ where L and U are the lower and upper bounds of the domain, respectively. (Duplicate values can be disambiguated by appending a replica number, so that the $r_i|repl\#$ entries are distinct.) Now a user submits a query for the r_i's that are larger than or equal to some constant $\alpha \in (L, U)$, i.e., $\sigma_{r \ge \alpha}(\mathbf{R})$. The publisher needs to prove to the user that the result $\mathbf{Q} = (r_a, r_{a+1}, .., r_b)$ is complete, namely,

- Contiguity – Each pair of successive entries r_i, r_{i+1} in $\mathbf{Q}$ also appear consecutively in $\mathbf{R}$.
- Terminal – The last element of $\mathbf{Q}$ is also the last element of $\mathbf{R}$, i.e., $r_b = r_n$.
- Origin – r_a is the first element in $\mathbf{R}$ that satisfies the query condition.

To simplify the solution, the owner inserts two fictitious entries, a left delimiter $r_0 \in (L, U)$ and a right delimiter $r_{n+1} \in (L, U)$[1], into $\mathbf{R}$, so the sorted list becomes $\mathbf{R} = (r_0, r_1, r_2, .., r_n, r_{n+1})$. The two fictitious entries are certified as such by the owner, so users would recognize them to be delimiters if they are encountered in any query result. We also assume that the lower and upper bounds, L and U, are known to everyone.

Conceptually, the contiguity condition of the result can be proved by creating a digital signature (see Section 2.1) for each r_i, $1 \le i \le n$, based on r_i itself and the immediate left and right neighbors:

$$sig(r_i) \;=\; s(h(g(r_{i-1}) \mid g(r_i) \mid g(r_{i+1}))) \qquad (1)$$

where s is a signature function using the owner's private key, h is a collision-resistant hash function, $g(r_i)$ is a function to produce a digest for entry/record r_i and is defined simply as $g(r_i) = h(r_i)$ initially, and $|$ denotes concatenation. Moreover, for the two delimiters,

$$sig(r_0) \;=\; s(h(h(L) \mid g(r_0) \mid g(r_1)))$$
$$sig(r_{n+1}) \;=\; s(h(g(r_n) \mid g(r_{n+1}) \mid h(U)))$$

Together with the query result $\mathbf{Q}$, the publisher returns the associated signatures $sig(r_i)$, $a \le i \le n+1$, plus $g(r_{a-1})$

[1]For ease of presentation, we assume here that $r_0 > L$ and $r_{n+1} < U$. With careful implementation, it is possible to set $r_0 = L$ and $r_{n+1} = U$.

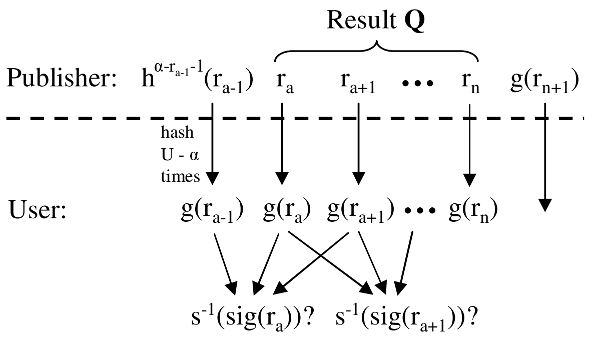


Figure 4: Completeness Check for Greater-Than Predicate

and $g(r_{n+1})$. The user then computes the digest for every entry in $\mathbf{Q}$, and matches with the signatures:

$$s^{-1}(sig(r_i)) \;?\!= h(g(r_{i-1}) \mid g(r_i) \mid g(r_{i+1}))$$

to verify that successive entries in $\mathbf{Q}$ are indeed neighboring entries in the original list $\mathbf{R}$. In particular, a successful test against the signature for the right delimiter r_{n+1} proves that the terminal of the result, r_n, is correct.

To prove that the result has the correct origin, i.e., $r_{a-1} < \alpha$, theoretically we could re-define g as $g(r_i) = h(U - r_i)$ where h is an additive hash function such that $h(x + y) = h(x) \circ h(y)$. The publisher would then return $h(\alpha - r_{a-1})$, and the user would compute $h(U - \alpha)$ followed by $g(r_{a-1}) = h(U - r_{a-1}) = h(U - \alpha) \circ h(\alpha - r_{a-1})$. Thus, the user could derive $g(r_{a-1})$ from just U and α, without knowledge of the actual value of r_{a-1}.

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 $x < 0$. The existence of the inverse function allows a cheating publisher server to return $h(\alpha - r_{a-1})$ even for $r_{a-1} > \alpha$, thus breaking the security of the above scheme.

In the absence of a suitable additive hash function, we need to devise another way to check the result for correct origin; we re-define the function g to:

$$g(r) = h^{U-r-1}(r) \tag{2}$$

where $h^i(r)$ hashes r iteratively such that $h^i(r) = h^{i-1}(h(r))$ and $h^0(r)$ applies a collision-resistant hash function h on r. To prove that $r_{a-1} < \alpha$, the publisher computes and returns an intermediate digest $h^{\alpha-r_{a-1}-1}(r_{a-1})$ (in place of $g(r_{a-1})$ above); the user then hashes it $(U-\alpha)$ more times to produce $g(r_{a-1}) = h^{(U-\alpha)+(\alpha-r_{a-1}-1)}(r_{a-1}) = h^{U-r_{a-1}-1}(r_{a-1})$ for verification with $sig(r_a)$. The scheme is summarized in Figure 4. The reason for choosing formula (2) which requires $h^i(r)$ to be defined for $i = \alpha - r_{a-1} - 1$, $i \geq 0$, instead of $g(r) = h^{U-r}(r)$ which requires $h^i(r)$ to be defined for $i = \alpha - r_{a-1}$, $i \geq 1$, will become clear when we discuss implementation optimizations in Section 5.1.

The above procedure is secure against cheating by the publisher provided it cannot produce $h^{\alpha-r_{a-1}-1}(r_{a-1})$ if $r_{a-1} \geq \alpha$. Therefore the inverse function $h^i(r)$ for $i < 0$ must be either undefined, or computationally infeasible to derive. This is why we use an iterative hash function here. In particular, to ensure that $h^{-1}(r) \neq r$, we just need to choose a hash function that outputs a different digest length from the length of r. To avoid cluttering the presentation of the paper, we will omit this consideration hereafter.

In addition, by picking a hash function h that is also one-way (see Section 2.1), the user would not be able to reverse-engineer the intermediate digest $h^{\alpha-r_{a-1}-1}(r_{a-1})$ to deduce the value of r_{a-1}.

Example: Suppose $\mathbf{R}$ = (2000, 3500, 8010, 12100, 25000), and the range for the entries is (0, 100000). According to our scheme, the owner inserts two fictitious entries, say 7 and 88888, so $\mathbf{R}$ becomes (7, 2000, 3500, 8010, 12100, 25000, 88888). Next, the owner derives $g(r) = h^{U-r-1}(r)$ for each $r \in \mathbf{R}$: $g(7) = h^{100000-7-1}(7) = h^{99992}(7)$, $g(2000) = h^{100000-2000-1}(2000) = h^{97999}(2000)$, and so on. Following that, a signature is generated for each entry: $sig(7) = s(h(h(0)|g(7)|g(2000))) = s(h(h(0)|h^{99992}(7)|h^{97999}(2000)))$, $sig(2000) = s(h(h^{99992}(7)|h^{97999}(2000)|h^{96499}(3500)))$, etc.

Now suppose the user submits a query for entries that are larger than or equal to 10000. Together with the result (12100, 25000), the publisher returns:

- $h^{10000-8010-1}(8010) = h^{1989}(8010)$,
- $g(88888) = h^{11111}(88888)$ for the right delimiter; and
- signatures for the result entries and the right delimiter.

Using these values, the user hashes $h^{1989}(8010)$ (100000 − 10000) more times to obtain $g(8010) = h^{91989}(8010)$, calculates $g(12100)$ and $g(25000)$ directly from their values, then checks the signatures for the result entries and right delimiter: $s^{-1}(sig(12100))?{=}h(g(8010)|g(12100)|g(25000))$, $s^{-1}(sig(25000))?{=}h(g(12100)|g(25000)|g(88888))$, $s^{-1}(sig(88888))?{=}h(g(25000)|g(88888)|h(100000))$. If all the signatures matched, the user accepts the result; otherwise the result is incorrect.

Finally, while the number of hashing operations in calculating $g(r)$ is linear in the size of the domain (L, U), the computation cost can still be high for large domains, e.g. long integers. Section 5.1 will explain how to reduce the number of hashing operations to logarithmic in the domain size.

3.2 Completeness Analysis

Theorem: Our proposed scheme can ensure query result completeness with respect to the threat model in Section 2.2.
Proof sketch: The various cases where the publisher may attempt to return incomplete results are as follows:

- Case 1: $r_{a-1} \geq \alpha$. Since $(\alpha - r_{a-1} - 1) < 0$, $h^{\alpha-r_{a-1}-1}(r_{a-1})$ is undefined, and it is computationally infeasible for the publisher to find a replacement to which the user can further hash $(U - \alpha)$ times to get $h^{U-r_{a-1}-1}(r_{a-1})$.
- Case 2: $\mathbf{Q} = \emptyset$ when, in fact, $\exists r_i \in \mathbf{R}$ such that $r_i \geq \alpha$. This implies that $r_n \geq \alpha$ since $\mathbf{R}$ is ordered. In this situation, the publisher is supposed to present $h^{\alpha-r_n-1}(r_n)$, which is then hashed $(U-\alpha)$ more times by the user to derive $g(r_n) = h^{U-r_n-1}(r_n)$ for matching with the right delimiter's $sig(r_{n+1})$. However, $(\alpha - r_n - 1) < 0$, so $h^{\alpha-r_n-1}(r_n)$ is undefined or computationally infeasible to derive.
- Case 3: $\mathbf{Q} = (r_a, .., r_b)$ where $b < n$; in other words, the terminal of $\mathbf{Q}$ is wrong, and one or more of the

largest values in $\mathbf{R}$ are omitted. To match $sig(r_b)$, the user requires $g(r_{b+1})$ according to formula (1). To deceive the user into accepting $g(r_{b+1})$ as the legitimate right delimiter, the publisher needs to fake the owner's signature, or else ensure $g(r_{b+1})$ tallies with the signature for the genuine right delimiter even though $r_{n+1} \neq r_{b+1}$, both of which are computationally infeasible. Another possibility is for the publisher to pass off the signed right delimiter $g(r_{n+1})$ in place of $g(r_{b+1})$ and make sure that formula (1) produces the same digest. However, this would require a collision in the collision-resistant hash function h.

- Case 4: $\mathbf{Q} = (r_a, .., r_i, r_j, .., r_n)$ 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_a and r_n are omitted. This query result causes the user to compare $h(g(r_{i-1})|g(r_i)|g(r_j))$ against $s^{-1}(h(g(r_{i-1})|g(r_i)|g(r_{i+1})))$. Since $\mathbf{R}$ is ordered, $(r_{i+1} < r_j)$ and the comparison succeeds only if there is a collision in the collision-resistant hash function h (because g is defined in terms of h in formula (2)).

- Case 5: $\mathbf{Q} = (r_a, .., r_i, r_j, r_k, .., r_n)$ but $r_j \notin \mathbf{R}$; in other words, the publisher introduces a spurious entry in the query result. To deceive the user, the publisher would need a valid signature for r_j, which can only be generated by the owner.

4. VERIFICATION OF RELATIONAL QUERY RESULTS

In this section, we progressly extend the basic approach in Section 3 for relational queries, i.e., range selection on sorted attribute, projection, join and range selection on unsorted attribute, in that order.

4.1 Selection Query

Selection: $\sigma_C(\mathbf{R}) = \{r \mid r \in \mathbf{R} \text{ and } C(r)\}$ where $\mathbf{R}$ is a relation sorted on attribute K, C is a condition $A_i \ \Theta \ c$, A_i is an attribute of $\mathbf{R}$, and $\Theta \in \{=, \neq, <, \leq, >, \geq\}$.

To verify completeness of the result of selection operations in general, it is necessary only to support range selection of the form $\alpha \leq K \leq \beta$, where K is an attribute of $\mathbf{R}$ and α, β are specified constants. This is because $K = \alpha$ is equivalent to $\alpha \leq K \leq \alpha$, $\alpha < K < \beta$ requires only a trivial adjustment in formulating the result $\mathbf{Q}$ but does not affect the verification procedure, and $K \neq \alpha$ can be mapped to $(L < K < \alpha) \cup (\alpha < K < U)$.

For now, we require the selection to be on the attribute K that $\mathbf{R}$ is sorted on, so that the result tuples occupy a contiguous range on K, and we can formulate the problem as follows. We will extend our scheme to support selection on unsorted attribute in Section 4.4.

Problem Definition: Given a relation $\mathbf{R}$ with the schema $[K, A_1, A_2, .., A_R]$ such that $\mathbf{R} = (r_1, r_2, .., r_n)$ is sorted on attribute K that has a range of (L, U), and A_i are the other attributes in $\mathbf{R}$. The publisher wishes to prove to the user that the result $\mathbf{Q} = (r_a, r_{a+1}, .., r_b)$ to the query condition $\alpha \leq K \leq \beta$ such that $r_a.K \geq \alpha$ and $r_b.K \leq \beta$ is complete, i.e., there is no record $r_i \in \mathbf{R}$ that satisfies $\alpha \leq r_i.K \leq \beta$, but $r_i \notin \mathbf{Q}$.

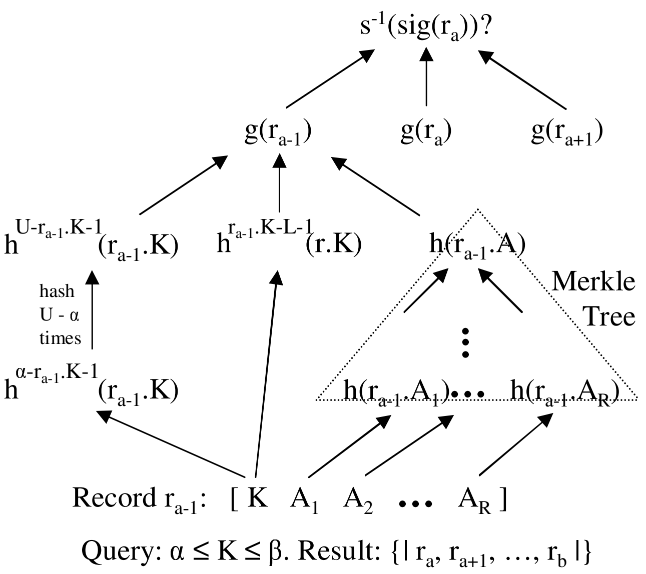

$s^{-1}(sig(r_a))$?
$g(r_{a-1})$
$g(r_a)$
$g(r_{a+1})$
$h^{U-r_{a-1}.K-1}(r_{a-1}.K)$
$h^{r_{a-1}.K-L-1}(r.K)$
$h(r_{a-1}.A)$
hash U - α times
Merkle Tree
$h^{\alpha-r_{a-1}.K-1}(r_{a-1}.K)$
$h(r_{a-1}.A_1)$ ••• $h(r_{a-1}.A_R)$
Record r_{a-1}: [K A_1 A_2 ••• A_R]
Query: $\alpha \leq K \leq \beta$. Result: $\{| r_a, r_{a+1}, \ldots, r_b |\}$

Figure 5: Completeness Verification for Select-Project-Join Query

The scheme presented in Section 3.1 can be extended for relational queries as shown in Figure 5. Again, the owner inserts two fictitious records r_0 and r_{n+1} so that $\mathbf{R}$ becomes $(r_0, r_1, .., r_n, r_{n+1})$. We re-define formula (2) to:

$$g(r) = h^{U-r.K-1}(r.K)|h^{r.K-L-1}(r.K)|\text{MHT}(r.A) \quad (3)$$

To ensure that $r_{a-1}.K < \alpha$, the publisher returns an intermediate digest $h^{\alpha-r_{a-1}.K-1}(r_{a-1}.K)$, which the user further hashes $(U-\alpha)$ times to produce $h^{U-r_{a-1}.K-1}(r_{a-1}.K)$, as before. In addition, the publisher sends back the digest $h^{r_{a-1}.K-L-1}(r_{a-1}.K)$ so the user can compute $g(r_{a-1})$ with formula (3), and finally verify with $sig(r_a)$ through formula (1).

Similar to the purpose that $h^{U-r.K-1}(r.K)$ serves, $h^{r.K-L-1}(r.K)$ is introduced in formula (3) for verifying that $r_{b+1}.K > \beta$. This is done by getting the publisher to return an intermediate digest $h^{r_{b+1}.K-\beta-1}(r_{b+1}.K)$, which the user then hashes a further $(\beta - L)$ times to obtain $h^{(r_{b+1}.K-\beta-1)+(\beta-L)}(r_{b+1}.K) = h^{(r_{b+1}.K-L-1)}(r_{b+1}.K)$. Again, the security of this verification requires that $h^{r_{b+1}.K-\beta-1}(r_{b+1}.K)$ for $r_{b+1}.K \leq \beta$ is either undefined or computationally infeasible to derive. Adaptation of the completeness analysis in Section 3.2 for the range selection here is straightforward.

The third component in the computation of $g(r)$, MHT$(r.A)$, is the root digest of the Merkle Hash Tree on the attribute values of record r, $r.A_1, .., r.A_R$. This digest is needed to uniquely identify each record r, as K is not necessarily a primary key of $\mathbf{R}$. To illustrate, suppose $\mathbf{Q}$ should contain three records with the same K value "xyz", $r_1 = \langle \overline{\text{xyz}}, 1\rangle$, $r_2 = \langle \underline{\text{xyz}}, 2\rangle$ and $r_3 = \langle \underline{\text{xyz}}, 3\rangle$. With just $g(r) = h^{U-r.K-1}(r.K)|h^{r.K-L-1}(r.K)$, $g(r_1) = g(r_2) = g(r_3)$. If the publisher omits r_2 from the query result, the user would derive $h(g(r_0)|g(r_1)|g(r_3))$ which matches $s^{-1}(sig(r_1)) = h(g(r_0)|g(r_1)|g(r_2))$, and $h(g(r_1)|g(r_3)|g(r_4))$ which matches $s^{-1}(sig(r_3)) = h(g(r_2)|g(r_3)|g(r_4))$; so the omission of r_2 would go undetected.

Authenticity: One interesting consequence of incorporating MHT($r.A$) in formula (3) is that $sig(r)$ now depends on *all* the attribute values of r. This means that any attempt at tampering with the content of r will be detected. Therefore the latest formulation enables the user to check the query results for completeness as well as authenticity.

4.2 Selection-Projection Query

Projection: $\pi_{K,A_1,..,A_p}(\mathbf{R}) = \{[r.K, r.A_1, .., r.A_p] \mid r \in \mathbf{R}\}$ 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 [10], we do not want the publisher to return to the user any attribute values in the result tuples that should be filtered out, so as to avoid compromising access control rules by disclosing sensitive columns. 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 publisher. Since MHT($r.A$) in formula (3) is defined as the root digest of a Merkle Hash Tree on the attribute values of record r, the publisher can provide the digest in place of the actual value for those attributes that are projected out, so the user can still compute MHT($r.A$) 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 publisher to perform duplicate elimination by specifying the keyword DISTINCT. In the former case, the MHT($r.A$) component in formula (3) enables the user to uniquely identify each duplicate, so the publisher cannot omit some duplicates without being detected. In the latter case where duplicates are not needed, our scheme requires the publisher to present $g(r_i)$ and the signature $sig(r_i)$ for each eliminated duplicate r_i to enable all the signatures for $\mathbf{Q}$ to be checked.

4.3 Selection-Projection-Join Query

Join: $\mathbf{R} \bowtie_C \mathbf{S}$ 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 proposed scheme (with g as defined in formula (3)) 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 owner'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 $\min(\mathbf{R}.A_i) \leq s.A_j < U$.

4.4 Multipoint Query

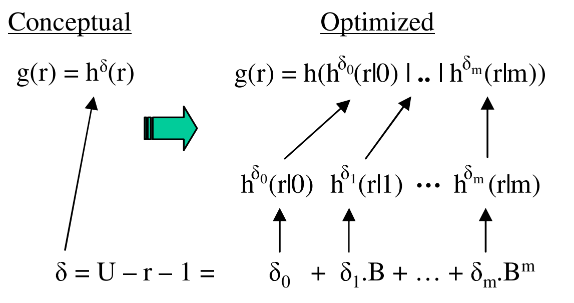
In Section 4.1, we have considered range query where the result tuples occupy a contiguous range on the attribute K that the relation is sorted on. In the case of a query that involves selection attribute(s) other than K, e.g. "SELECT * FROM Emp WHERE Salary < 10000 AND Dept = 1" on the table in Figure 1, the result could comprise multiple points or ranges on K. We call this a multi-point query.

In general, the result for a multi-point query can be treated as a range of contiguous tuples on K, some of which satisfy the query condition while others should be filtered out. Consider a filtered record r_i within the result range, e.g. [002, C, 3500, 2, ..] in the above example.

- Case 1: The access control policy permits the user to see r_i. The publisher server returns the attribute value(s) that fails the query condition, i.e., "r_i.Dept = 2" in the above example, plus the digests for the remaining attribute values of r_i. This enables the user to compute $g(r_i)$ for matching with the signatures of r_{i-1}, r_i and r_{i+1}.
- Case 2: The access control policy prohibits the user from seeing r_i, so his query is rewritten to filter out r_i. Here, the publisher server cannot return any of the actual attributes of r_i unlike Case 1. Our solution is to introduce additional columns to the relation, one column for each user group in the access control model to indicate whether individual records in the relation are visible to that user group.
  For example, consider an access model that differentiates between users with security clearance levels of "secret", "confidential" and "unclassified". The owner would add three binary attributes to the example table in Figure 1 to indicate whether each record can be seen by users with "secret", "confidential" and "unclassified" clearances, respectively. Thus, for a filtered record r_i that is shielded from a user with only "confidential" clearance, the publisher could return "r_i.confidential = No", plus the digests for the other attributes of r_i. The user can then compute $g(r_i)$ 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.

5. OPTIMIZATION

Having introduced our extended scheme for verifying the completeness of relational query results, we now discuss how



Conceptual
Optimized
g(r) = h^δ(r)
g(r) = h(h^δ0(r|0) | .. | h^δm(r|m))
h^δ0(r|0) h^δ1(r|1) ··· h^δm(r|m)
δ = U – r – 1 = δ0 + δ1.B + … + δm.B^m

Figure 6: Optimized $g(r)$

the scheme can be implemented efficiently. For clarity, we focus on the Greater-Than predicate here, though the techniques apply to relational queries too.

5.1 Reduction in Hashing Operations

First, we note that $g(r) = h^{U-r-1}(r)$ is very expensive to compute. For example, for a four-byte integer field, $g(r)$ entails 2^{32} hashes in the worst case, which requires almost 60 hours at 50 μsec per hash! To reduce the processing overhead, 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 for δ.

This observation can be exploited to optimize the derivation of $g(r)$, as depicted in Figure 6: The owner replaces $h^{U-r-1}(r)$ in formula (2) with $h^{\delta_{t,0}}(r|0) \mid h^{\delta_{t,1}}(r|1) \mid ..$ $\mid h^{\delta_{t,m}}(r|m)$, where $\delta_t = U - r - 1$. After executing a query, the publisher returns $m+1$ intermediate digests $h^{\delta_{e,0}}(r|0)$, $h^{\delta_{e,1}}(r|1)$, .., $h^{\delta_{e,m}}(r|m)$ where $\delta_{e,i}$ are the coefficients in the polynomial representation for $\delta_e = \alpha - r_{a-1} - 1$. Let $\delta_{c,i}$ be the coefficients in the canonical representation for $\delta_c = U - \alpha$; the user then hashes $h^{\delta_{e,0}}(r|0)$ a further $\delta_{c,0}$ times, $h^{\delta_{e,1}}(r|1)$ another $\delta_{c,1}$ times, and so on, to produce $h^{\delta_{t,0}}(r|0), h^{\delta_{t,1}}(r|1), .., h^{\delta_{t,m}}(r|m)$. As it is possible to have some $\delta_i = 0$, $h^j(.)$ must be defined for $j = 0$.

To illustrate, suppose $\delta_t = U - r_{a-1} - 1 = 5555$, $\delta_c = U - \alpha = 1 + 2 \times 10 + 3 \times 10^2 + 4 \times 10^3$; so, $\delta_e = \delta_t - \delta_c = \alpha - r_{a-1} - 1 = 4 + 3 \times 10 + 2 \times 10^2 + 1 \times 10^3$. The publisher 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:

$$g(r_{a-1}) = h(h^5(r_{a-1}|0) \mid h^5(r_{a-1}|1) \mid h^5(r_{a-1}|2) \mid h^5(r_{a-1}|3))$$

and from there confirms that $r_{a-1} < \alpha$ as before.

There is a complication, though. Suppose the query condition is such that $\delta_c = U - \alpha = 2828$. The publisher now gets $\delta_e = \alpha - r_{a-1} - 1 = 2727$, 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 non-canonical representation $5555 = 15 + 4 \times 10 + 15 \times 10^2 + 4 \times 10^3$. In general, this complication arises if $\exists\ 0 \leq i \leq m$ such that $\delta_{t,i} < \delta_{c,i}$. To enable the user to succeed with the verification, the owner would have to produce digests corresponding to the non-canonical representations too. Unfortunately, there are up to 2^m non-canonical representations in the worst case. Clearly, this overhead is unacceptable.

To limit the number of non-canonical representations that must be supported, we observe that while the user knows only the value of δ_c, the publisher has access to both δ_t and δ_c. Thus the publisher can return digests corresponding to certain preferred non-canonical representations for δ_e in order to influence the representation for δ_t that the user derives. Referring to our running example, if the publisher 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: For any $\delta \geq 0$, a representation $\delta = \delta_0 + \delta_1.B + \delta_2.B^2 + .. + \delta_m.B^m$ is *valid* if $\delta_{t,i} \geq 0$, $0 \leq i \leq m$.

Definition: Given $\delta_t \geq 0$, let its canonical representation be $\delta_t = \delta_{t,0} + \delta_{t,1}.B + \delta_{t,2}.B^2 + .. + \delta_{t,m}.B^m$, $0 \leq \delta_{t,i} < B$. We define m *preferred non-canonical representations*:

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

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 $\delta_{t,2} - 1 < 0$.

Lemma: For any $0 \leq \delta_c \leq \delta_t$ with canonical representation $\delta_c = \delta_{c,0} + \delta_{c,1}.B + \delta_{c,2}.B^2 + .. + \delta_{c,m}.B^m$, $0 \leq \delta_{c,i} < B$, 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 $\delta_{e,i} \geq 0$.

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

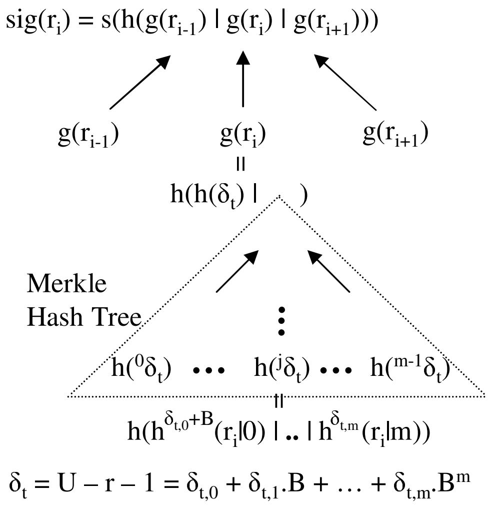
- $0 \leq \delta_{e,i} = (\delta_{t,i} + B) - \delta_{c,i} < 2B$ for $i = 0$,
- $0 \leq \delta_{e,i} = (\delta_{t,i} + B - 1) - \delta_{c,i} < 2B - 1$ for $1 \leq i \leq i_{max}$,
- $0 \leq \delta_{e,i} = (\delta_{t,i} - 1) - \delta_{c,i} < B - 1$ for $i = i_{max} + 1$,
- $0 \leq \delta_{e,i} = \delta_{t,i}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

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

The lemma allows the system to support only the canonical representation for δ_t, plus m preferred non-canonical representations $^i\delta_t$, $0 \leq i < m$ as defined above. We thus arrive at the following implementation scheme.

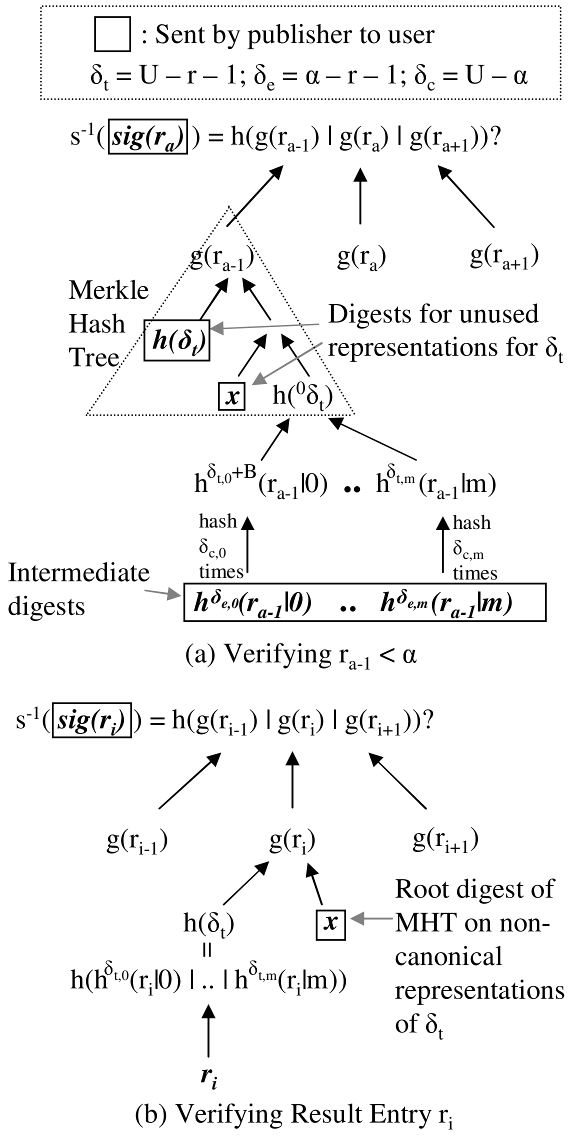
Signature Construction by Owner (Figure 7): During creation and update of the sorted list $\mathbf{R} = (r_0, r_1, .., r_n, r_{n+1})$ (recall r_0 and r_{n+1} are fictitious entries), the owner derives the signature for each new entry r_i as follows:

- Starting with the m non-canonical representations $^j\delta_t$



sig(r_i) = s(h(g(r_{i-1}) | g(r_i) | g(r_{i+1})))
g(r_{i-1})
g(r_i)
g(r_{i+1})
h(h(δ_t) |)
Merkle Hash Tree
h(^0δ_t) ••• h(^jδ_t) ••• h(^{m-1}δ_t)
h(h^{δ_{t,0}+B}(r_i|0) | .. | h^{δ_{t,m}}(r_i|m))
δ_t = U – r – 1 = δ_{t,0} + δ_{t,1}.B + … + δ_{t,m}.B^m

Figure 7: Signature Construction



: Sent by publisher to user
δ_t = U – r – 1; δ_e = α – r – 1; δ_c = U – α
s^{-1}(sig(r_a)) = h(g(r_{a-1}) | g(r_a) | g(r_{a+1}))?
g(r_{a-1})
g(r_a)
g(r_{a+1})
Merkle Hash Tree
h(δ_t)
Digests for unused representations for δ_t
x
h(^0δ_t)
h^{δ_{t,0}+B}(r_{a-1}|0) .. h^{δ_{t,m}}(r_{a-1}|m)
hash δ_{c,0} times
hash δ_{c,m} times
Intermediate digests
h^{δ_{e,0}}(r_{a-1}|0) .. h^{δ_{e,m}}(r_{a-1}|m)
(a) Verifying r_{a-1} < α
s^{-1}(sig(r_i)) = h(g(r_{i-1}) | g(r_i) | g(r_{i+1}))?
g(r_{i-1})
g(r_i)
g(r_{i+1})
h(δ_t)
x
Root digest of MHT on non-canonical representations of δ_t
h(h^{δ_{t,0}}(r_i|0) | .. | h^{δ_{t,m}}(r_i|m))
r_i
(b) Verifying Result Entry r_i

Figure 8: Completeness Verification

and the canonical representation $\delta_t = \delta_{t,0} + \delta_{t,1}.B + \delta_{t,2}.B^2 + .. + \delta_{t,m}.B^m$ for $\delta_t = U - r_i - 1$, the owner computes a digest for each valid non-canonical representation:

$$h(\delta_t) = h(h^{\delta_{t,0}}(r_i|0) \mid .. \mid h^{\delta_{t,m}}(r_i|m))$$

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

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:

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

- Next, a Merkle Hash Tree (MHT) is built over the m non-canonical representations for δ_t. The root digest of the MHT is concatenated with the digest of the canonical representation, then hashed to produce a digest for $g(r_i)$.
- Similarly, $g(r_{i-1})$ and $g(r_{i+1})$ are derived for the left and right neighbors, respectively.
- The signature of r_i is now generated using formula (1).

Completeness Verification between Publisher and User (Figure 8, in which the items in italized, bold font are transmitted by the publisher to the user): To verify the result $\mathbf{Q} = (r_a, r_{a+1}, .., r_n)$ for query condition $r \geq \alpha$, the user checks the signature $sig(r_i)$ for each $r_i \in \mathbf{Q}$, which in turn requires $g(r_i)$ for $a-1 \leq i \leq n+1$. To compute $g(r_{a-1})$ (see Figure 8(a)):

- The publisher utilizes its knowledge of U, α and r_{a-1} to compute the canonical representations for $\delta_t = U - r_{a-1} - 1$ and $\delta_c = U - \alpha$.
- If $\delta_{t,i} \geq \delta_{c,i}\ \forall i$,
  - Δ_t is equated with the canonical representation for δ_t.
  - Return the root digest of the MHT over the non-canonical 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 non-canonical 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 for δ_t that are not used as Δ_t (there are $\lceil log_2 m \rceil$ such digests as explained in Section 2.1).

- The coefficients in the polynomial representation for $\delta_e = \alpha - r_{a-1} - 1$ is then calculated as: $\delta_{e,i} = \Delta_{t,i} - \delta_{c,i}$. The publisher then 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 for $\delta_c = U - \alpha$, 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 publisher to produce $g(r_{a-1})$. If not, $h(\Delta_t)$ is combined with the MHT digests to derive the root digest, and then the digest for the canonical representation to produce $g(r_{a-1})$.

As shown in Figure 8(b), for each r_i where $a \leq i \leq n+1$, $g(r_i)$ is produced by:

- The publisher returns the root digest of the MHT over the non-canonical representations ${}^i\delta_t$ for $\delta_t = U - r_i - 1$.

- With the query result r_i, the user generates the $m+1$ digests $h^{\delta_{t,j}}(r_i|j)$, and combines them into a single digest for the canonical representation for δ_t. This digest is then concatenated with the root digest of the MHT from the publisher to produce $g(r_i)$.

5.2 Reduction in Signatures

Another implementation issue to consider is that the overheads for transmitting and verifying a signature for each $r_i \in \mathbf{Q}$ can be very large, especially if the communication bandwidth or processing capability of the user is limited. To lighten this overhead, the publisher can combine the signatures associated with individual entries in the result **Q** into one aggregated signature, using the techniques proposed in [8] or [18]. This optimization also helps the user to cut down to just one signature verification operation per query result. The resulting savings in computation can be substantial as signature verification is around 100 times slower than hashing operation [23], and **Q** can potentially contain thousands of entries.

However, signature aggregation must be implemented carefully, as explained in [18]. To ensure aggregated signatures are secure, the aggregation scheme should possess the immutability property, i.e., it is difficult for an unauthorized party to generate new verifiable aggregated signatures after amassing enough aggregated signatures from past query results. To achieve immutability, any of the several practical aggregation schemes proposed in [18] can be adopted.

6. COST ANALYSIS

Having presented our authentication mechanism, we now analyze the overheads that it introduces for relational queries with greater-than selection; extension to range selection is straightforward. We begin by quantifying the communication overhead, before looking at the incremental computation cost. Due to space constraint, 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 1, while the values for C_{hash} and C_{sign} are obtained from [23]. For computation costs, we model only hashing and signature verification; other 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 [6], which are much costlier than simple hash functions used in, say, conventional hash index.)

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 data entry (bytes)	–
B	$\delta = \delta_0 + \delta_1.B + .. + \delta_m.B^m$	–
m	$m = log_B(U - L)$	–

Table 1: Cost Parameters

6.1 Communication from Publisher to User

The authentication information that the publisher transmits to the user includes the following components:

- Digests for computing $g(r_{a-1})$. The $h^{U-r.K-1}(r.K)$ component in formula (3) requires the $m+1$ intermediate digests corresponding to the polynomial representation for $h^{\delta_e}(r_{a-1})$, $\lceil log_2 m \rceil$ digests in the MHT covering the representations for $\delta_t = U - r_{a-1} - 1$ that are not selected, plus one digest for the canonical representation in the worst case (if a non-canonical representation ${}^i\delta_t$ is used). The other two components in formula (3) require one digest each. The traffic amounts to $[m + 4 + \lceil log_2 m \rceil] \times M_{digest}$.

- Digests for computing $g(r_i)$, $a \leq i \leq n$. For each such r_i, the publisher sends the root digest of the MHT over the non-canonical representations for $\delta_t = U - r_i - 1$, the root digest of the MHT over the non-canonical representations for $\delta_t = r_i - L - 1$, as well as a digest for $MHT(r.A)$. The traffic for all the result entries amounts to $3 \times (n - a + 1) \times M_{digest}$.

- Digest for the right delimiter, $g(r_{n+1})$; the size is 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:

$$M_{user} = [m + 4 + 3(n - a + 1) + \lceil log_2 m \rceil] \times M_{digest} + M_{sign} \tag{4}$$

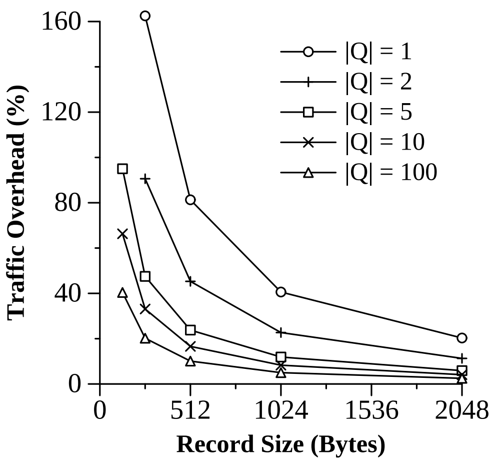


Figure 9: User Traffic Overhead

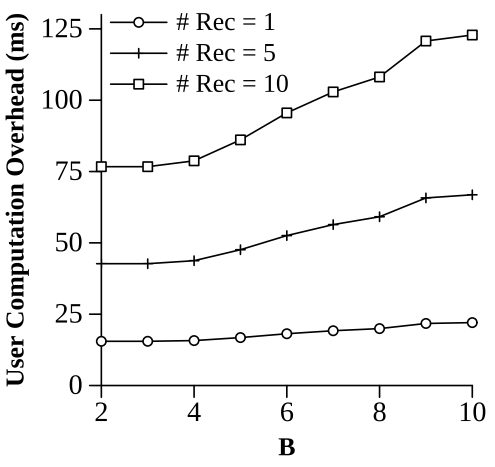


Figure 10: User Computation Overhead

Figure 9 plots the user traffic overhead, defined as $M_{user}/$ ResultSize where ResultSize $= |\mathbf{Q}| * M_r$, against the data entry size M_r for various number of result entries $|\mathbf{Q}|$. The parameters M_{digest} and M_{sign} are set to their default values of 128 bits and 1024 bits, respectively. The figure shows that the traffic overhead reduces very quickly as $|\mathbf{Q}|$ grows beyond one, as the cost of the aggregated signature is amortized over more result entries. This reduction stabilizes at around $|\mathbf{Q}| = 5$, at which point the per-entry overhead falls within 25% for $M_r \geq 512$ Bytes.

From formula (4), we also observe that the space overhead incurred by our proposed solution is linear in the result size (i.e., $n - a + 1$). This compares favorably with the only existing scheme by Devanbu et al in [10] that enables verification of query result completeness; the space overhead of their scheme grows linearly to the query result, as well as logarithmically to the underlying database.

6.2 Computation Overhead on the User

The computations performed by the user in authenticating the query results include:

- Derivation of $g(r_{a-1})$. This entails hashing the $m+1$ intermediate digests corresponding to the selected representation for $\delta = U - r_{a-1} - 1$, each of which requires up to B additional hashes. In the worst case (where the selected representation is non-canonical), the resulting digest is then combined with the digests in the MHT over the non-canonical representations, and the digest for the canonical representation to get the first component in formula (3), which require $\lceil log_2 m \rceil +$ 1 hashes. Evaluation of formula (3) incurs another hash. The computation cost amounts to $[B(m+1) + \lceil log_2 m \rceil + 2] \times C_{hash}$.

- Derivation of $g(r_i)$, $a \leq i \leq n$. For each such r_i, the user first computes the $m+1$ digests corresponding to the canonical representation for $\delta_t = U - r_i - 1$, each of which requires up to B hashes. Next, combining the $m+1$ digests incurs another hash. This is repeated for $\delta_t = r_i - L - 1$. Evaluation of formula (3) incurs another hash. The computation for all the result entries thus amounts to $(n-a+1) \times [2B(m+1)+3] \times C_{hash}$.

- Derivation of $h(g(r_{i-1})|g(r_i)|g(r_{i+1}))$ for each $a \leq i \leq n$, amounting to $(n-a+1) \times C_{hash}$.

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

The total computation cost incurred by the user is:

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

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 $y = 2(n-a+1)(B(m+1)+2) + B(m+1) + \lceil log_2 m \rceil + 3$ is minimum. It can be shown that this occurs at $2 < B < 3$ where $\frac{dy}{dB} = 0$. To illustrate, Figure 10 plots B against C_{user} for different result sizes (i.e., $n-a+1$). 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} (obtained from [23]), formula (5) reduces to $C_{user} = 6.8(n-a+1) + 8.7$ msec. Thus, C_{user} is roughly 15.5 msec, 689 msec and 6.81 sec for result size of 1, 100 and 1000 records, respectively, which is not significant.

6.3 Database Updates

Having evaluated the overheads for verifying query results, we now consider the impact of our proposed scheme on update operations.

To produce completeness proof for query results at runtime, the owner has to pre-generate 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, our extended scheme (with g as defined in formula (3)) can 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.

According to formula (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 less than Merkle Hash Tree schemes (e.g. [10], [20]) 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.

7. CONCLUSION

In this paper, we present a scheme for authenticating results generated by untrusted (relational) query processors. Our scheme enables the query processor to produce proof that each result is complete (i.e., no qualifying tuples are omitted) and authentic (i.e., all the result values originated from the owner). The scheme does not disclose more data than necessitated by the query conditions, hence it does not contradict the access control policies on the database. Moreover, the scheme is computationally secure, and introduces low query processing and update overheads compared to existing alternatives.

We are currently extending this work in a number of ways. One extension is to support multi-dimensional indices, to avoid having to generate a set of signatures for every interesting sort order on a table. We are also looking into generalizing the proposed scheme for non-relational structures, e.g. directed acyclic graphs. Another extension is to implement the scheme in an open-source database system.

8. ACKNOWLEDGEMENTS

We would like to thank Robert H. Deng, H.V. Jagadish, Wang-Chiew Tan and the reviewers for their helpful suggestions.

9. REFERENCES

[1] DriveCrypt Secure Hard Disk Encryption. http://www.drivecrypt.com.

[2] E4M Disk Encryption. http://www.e4m.net.

[3] Encrypting File System (EFS) for Windows 2000. http://www.microsoft.com/windows2000/techinfo/howit works/security/encrypt.asp.

[4] PGPdisk. http://www.pgpi.org/products/pgpdisk/.

[5] Proposed Federal Information Processing Standard for Digital Signature Standard (DSS). *Federal Register*, 56(169):42980–42982, 1991.

[6] *Secure Hashing Algorithm*. National Institute of Science and Technology. FIPS 180-2, 2001.

[7] R. Anderson, R. Needham, and A. Shamir. The Steganographic File System. In *Information Hiding, 2nd International Workshop*, D. Aucsmith, Ed., Portland, Oregon, USA, April 1998.

[8] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In *Proceedings of Advances in Cryptology – EUROCRYPT'03, E. Biham, Ed., LNCS, Springer-Verlag*, 2003.

[9] S. Chokani. Trusted Products Evaluation. *Communications of the ACM*, 35(7):64–76, 1992.

[10] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic Data Publication over the Internet. In *14th IFIP 11.3 Working Conference in Database Security*, pages 102–112, 2000.

[11] R. Huebsch, J. Hellerstein, N. Lanham, B. Loo, S. Shenker, and I. Stoica. Querying the Internet with PIER. In *Proceedings of the 29th International Conference on Very Large Databases*, pages 321–332, 2003.

[12] Q. Luo, S. Krishnamurthy, C. Mohan, H. Pirahesh, H. Woo, B. Lindsay, and J. Naughton. Middle-Tier Database Caching for E-Business. In *Proceedings of the 2002 ACM SIGMOD International Conference on Management of Data*, pages 600–611, 2002.

[13] D. Ma, R. Deng, and H. Pang. Authenticating Query Results From Untrusted Servers Over Open Networks. In *Submitted for Publication*, 2004.

[14] D. Margulius. Apps on the Edge. *InfoWorld*, 24(21), May 2002. http://www.infoworld.com/article/02/05/23/020527feedgetci_1.html.

[15] R. Merkle. A Certified Digital Signature. In *Proceedings of Advances in Cryptology-Crypto '89, Lecture Notes in Computer Science*, volume 0435, pages 218–238, 1999.

[16] G. Miklau and D. Suciu. Controlling Access to Published Data Using Cryptography. In *Proceedings of the 29th International Conference on Very Large Data Bases*, pages 898–909, 2003.

[17] E. Mykletun, M. Narasimha, and G. Tsudik. Authentication and Integrity in Outsourced Databases. In *Proceedings of the Network and Distributed System Security Symposium*, February 2004.

[18] E. Mykletun, M. Narasimha, and G. Tsudik. Signature Bouquets: Immutability for Aggregated/Condensed Signatures. In *Proceedings of the European Symposium on Research in Computer Security*, September 2004.

[19] B. Neuman and T. Tso. Kerberos: An Authentication Service for Computer Networks. *IEEE Communications Magazine*, 32(9):33–38, 1994.

[20] H. Pang and K. Tan. Authenticating Query Results in Edge Computing. In *IEEE International Conference on Data Engineering*, pages 560–571, March 2004.

[21] H. Pang, K. Tan, and X. Zhou. StegFS: A Steganographic File System. In *Proceedings of the 19th International Conference on Data Engineering*, pages 657–668, Bangalore, India, March 2003.

[22] R. Rivest. *RFC 1321: The MD5 Message-Digest Algorithm*. Internet Activities Board, 1992.

[23] R. Rivest and A. Shamir. PayWord and MicroMint: Two Simple Micropayment Schemes. In *http://theory.lcs.mit.edu/ rivest/RivestShamir-mpay.pdf (This version is dated 2001). An earlier version appears in Security Protocols, Lecture Notes in Computer Science, LNCS 1189, pp. 69-87*, 2001.

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

[25] R. Sandhu and P. Samarati. Access Control: Principles and Practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.

[26] S. Saroiu, K. Gummadi, R. Dunn, S. Gribble, and H. Levy. An Analysis of Internet Content Delivery Systems. In *Proceedings of the 5th Symposium on Operating Systems Design and Implementation*, pages 315–327, 2002.