Authenticating Location-based Services without Compromising Location Privacy

Haibo Hu, Jianliang Xu, Qian Chen, Ziwei Yang
Hong Kong Baptist University
Kowloon Tong, Hong Kong SAR, China
{haibo,xujl,qchen,zwyang}@comp.hkbu.edu.hk

ABSTRACT

The popularity of mobile social networking services (mSNSs) is propelling more and more businesses, especially those in retailing and marketing, into mobile and location-based forms. To address the trust issue, the service providers are expected to deliver their location-based services in an authenticatable manner, so that the correctness of the service results can be verified by the client. However, existing works on query authentication cannot preserve the privacy of the data being queried, which are sensitive user locations when it comes to location-based services and mSNSs. In this paper, we address this challenging problem by proposing a comprehensive solution that preserves unconditional location privacy when authenticating range queries. Three authentication schemes for R-tree and grid-file index, together with two optimization techniques, are developed. Cost models, security analysis, and experimental results consistently show the effectiveness, reliability and robustness of the proposed schemes under various system settings and query workloads.

Categories and Subject Descriptors

H.2.8 [**DATABASE MANAGEMENT**]: Database applications—*Spatial databases and GIS*

General Terms

Algorithms, Security

Keywords

query authentication, privacy-preserving, LBS

1. INTRODUCTION

Location-based services (LBSs) have been gaining tremendous popularity over the recent years, in particular since the emergence of mobile social networking services (mSNSs). Social networking giants such as Facebook and Twitter are all turning their services into mobile, along with specialized vendors like Foursquare, Gowalla and Loopt. Besides, major mobile carriers also strive to provide more value-added

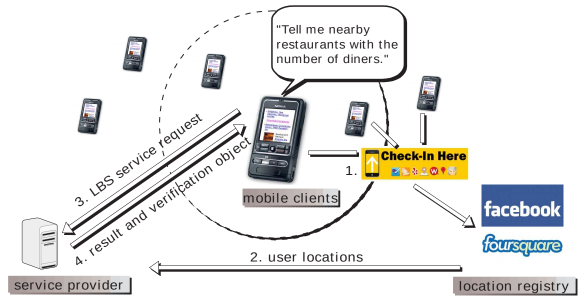


Figure 1: Authenticatable Location-Based Service

services to their subscribers, among which the most thrilling applications are LBSs such as location-aware advertisement ("check-in deals") and nearby-friend reminders.

A typical LBS business model consists of a location registry (typically a social network or a mobile carrier who accepts user location updates or "check-ins"), a service provider (SP, typically a third party application developed on the social network) that offers LBS applications based on user locations, and a client (typically a mobile user) who requests the service. In this model, the third-party application is authorized to access user locations but it is not trustworthy regarding its service returned to the client. For example in Fig. 1, an SP offers location-based restaurant browsing which tells the client not only the nearby restaurants, but also the numbers of diners as an indication of their popularity. Each of these numbers can be retrieved by the SP through a spatial range query on a user location dataset specified by the client. However, the client may not trust these numbers as the SP has the motive to manipulate them in favor of "sponsored restaurants". As another example in public services, the government may outsource the online traffic monitoring service to third-party vendors. For market profits, however, they may prioritize the services by sending updated and accurate congestion reports to paid users while sending delayed or inaccurate ones to free users. These trustworthy issues are extremely important as more day-to-day businesses and public services are turning mobile and location-based. It would be soon indispensable for service providers to deliver their services in an *authenticatable* manner, in which the correctness of service results — whether each result is genuine (soundness) and whether any result is missing (completeness) — can be verified by the client.

In the literature, there are a lot of works on the authentication of query results (e.g., [19, 17, 14, 26, 27]). In these works, the data owner (i.e., the location registry in this paper) publishes not only data (i.e., user locations) to the third-party SP, but also the endorsements of the data being published. These endorsements are signed by the data owner against tampering by the SP. Given a query, the SP returns both the query results and a proof, called *verification object* (VO), which can be used by the client to reconstruct the endorsements and thus verify the correctness of the results. As a location-based service usually concerns a spatial query, the authentication of such services can adopt the same paradigm as in query authentication. As Fig. 1 illustrates, after receiving a request, the SP evaluates the query based on the user locations obtained from the location registry, and delivers the result to the client. A VO, which includes endorsed values derived from user locations and ids, is also sent to the client to verify the correctness of the result.

However, while prior works address the query authentication issue, they fail to preserve the privacy of the data. In fact, they assume that during the verification process, the client can always be trusted and entitled to receive data values on the querying attribute(s). This assumption no longer holds in LBSs where the locations of mobile users are sensitive and should be protected against the clients. Therefore, the challenge of this work is how to design *privacy-preserving* query authentication schemes without disclosing any user location information to the client.

Unfortunately, the hiding of user locations from the client compounds the difficulty of authentication, and in fact, it brings out a new aspect of authentication. Traditional authentication verifies the soundness of a query by only checking whether the returned results are genuine because the *compliance* of the results, i.e., whether they comply with the query statement and are thus true results, is already implied by their returned values. However, without knowing these values, verifying the compliance is no longer trivial, which is indeed the challenge of privacy-preserving query authentication.

In this paper, we start with one-dimensional range queries on a B$^+$-tree and adopt a cryptographic construct that was originally proposed by Pang et al. in [17] for the value hiding of non-result objects. It is based on a proof for verifying $x \geq \alpha$ (α is the query bound) without disclosing x. The idea is to let the client and SP jointly compute a digest function $g()$ of value x. However, generalizing the one-dimensional solution to multi-dimensional indexes such as R-tree leads to significant performance overhead as the linearity in one-dimensional space no longer exists. As such, the authentication may involve more tree nodes so that the size of the VO outweighs the result itself, especially when the query is small. To cater for small queries, we propose to use grid-file as an alternative index and design the complete authentication scheme. Besides the R-tree and grid-file based schemes, we propose a third authentication scheme based on accumulative digests for static datasets to further reduce the authentication cost. Along the road of performance optimizations, we propose two directions, both of which are orthogonal to the underlying authentication schemes. First, we develop an enhanced digest function $g()$ that is not only faster than the original one by Pang et al., but also performance-independent of the input value, which is critical for a cryptographic function. Second, we propose linear ordering and embedding as the internal organization of each node (or cell). This optimization regains the linearity for multi-dimensional data and enables effective pruning techniques. To summarize, our contributions in this paper are as follows:

- To the best of our knowledge, this is the first work that addresses privacy-preserving query authentication for location-based services. The problem is critical for both mobile value-added service market and database research community.
- We develop three authentication schemes for R-tree and grid-file index, which are good for large queries, small queries, and queries on static datasets, respectively. Analytical models of computation and bandwidth costs are developed to justify these schemes. Security analysis shows they are secure by not disclosing any individual location information.
- We propose two optimization techniques that are orthogonal to the underlying authentication schemes.
- We conduct extensive experiments to evaluate the performance of the proposed schemes, with and without the two optimization techniques in effect. The results coincide with our analysis and further justify the efficiency of our approaches.

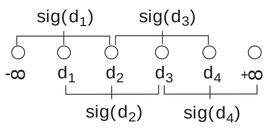
The rest of this paper is organized as follows. Section 2 introduces the research background and related work in query authentication. Section 3 formally defines the problem and privacy model, followed by the solution in single-dimensional space in Section 4. Section 5 presents three authentication schemes based on R-tree and grid-file index, followed by the security analysis. Section 6 analyzes their cost models and presents two optimization techniques, namely linear ordering and embedding. Section 7 shows the experimental results, followed by a conclusion in Section 8.

2. BACKGROUND AND RELATED WORKS

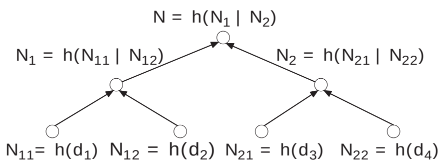
There is a large body of research works on query authentication for indexed data. These works originate from either digital signature chaining or Merkle hash tree.

Digital signature is a mathematical scheme for demonstrating the authenticity of a digital message. It is based on asymmetric cryptography. Given a message, the signer produces a signature with its private key. Then the verifier verifies the authenticity of the message by the message itself, the signer's public key and the signature. Based on this scheme, early works on query authentication impose a signature for every data value. The VB-tree [19] augments a conventional B^+-tree with a signature in each leaf entry. By verifying the signatures of all returned values, the client can guarantee the soundness of these results. To further reduce the number of signatures returned to the client, they can be aggregated into one signature of the same size as each individual signature [3]. However, the simple signature-based approach cannot guarantee the completeness, as the server can deliberately miss some results without being noticed. Therefore, Pang et al. proposed signature chaining [17], which connects a signature with adjacent data values to guarantee no result can be left out. Fig. 2(a) illustrates signature chaining for four sorted values d_1, d_2, d_3, d_4. The signature of each value depends not only on its own value but also on the immediate left and right values.[1] If the server returns d_2 and d_3 to the client, it will also send a verification object (VO) that contains: (1) the signatures of d_2 and d_3, and (2) the boundary values d_1 and d_4. Given the VO, the client can verify the results through the facts that: (1) the two boundary values fall outside the query range, and (2) all signatures

[1]For the first and the last values d_1 and d_4, two special objects $d_0 = -\infty$ and $d_5 = +\infty$ are appended.



(a) Signature Chaining



(b) Merkle Hash Tree

Figure 2: Basic Authentication Tools

are valid. The first condition ensures that no results are missing and the second guarantees no values are tampered with. Signature aggregation and chaining were adapted to multi-dimensional indexes by Cheng and Tan [5].

The Merkle hash tree (MHT) was introduced to authenticate a large set of data values [16]. Fig. 2(b) shows an MHT for the four data values in Fig. 2(a). It is a binary tree. Each leaf node with data value d_i is assigned a *digest* $h(d_i)$, where $h()$ is a one-way hash function. Each internal node N_i is assigned a digest which is derived from its child nodes, e.g., $N_1 = h(N_{11}|N_{12})$, where "|" denotes concatenation. In MHT, only the digest value of the root is signed by the data owner, and therefore it is more efficient than signature chaining schemes. An MHT can be used to authenticate any subset of data values. For example in Fig. 2(b), the server sends d_1 and d_2 to the client; and to prove their authenticity, the server also sends a VO to the client, which includes the digest of N_2 and the signed root digest N. The client computes $h(d_1)$ and $h(d_2)$, then $N_1 = h(h(d_1)|h(d_2))$, and finally $N = h(N_1|N_2)$. This computed root digest is then compared with the signed root digest in the VO. If they are the same, the client can verify that d_1 and d_2 are not tampered with by the server.

The notion of MHT has been generalized to an f-way tree and widely adapted to various index structures. Typical examples include Merkle B-tree and its variant Embedded Merkle B-tree (EMB-tree) [14]. The latter reduces the VO size by embedding a tiny EMB-tree in each node. For multi-dimensional datasets and queries, similar techniques were proposed by Yang et al., who integrated an R-tree with the MHT (which is called Merkle R-tree or MR-tree) for authenticating multi-dimensional range queries [26, 27].

Besides selection and range queries, recent studies focus on the authentication of more complex query types, including kNN queries [4, 28], join queries [27], and aggregation queries [13]. Besides relational and spatial datasets, authentication of semi-structured and non-structured datasets was studied for streaming data [15, 21] and text data [18].

Our work differs from all these works by being the first work on privacy-preserving query authentication. The lack of querying attribute values from the client makes the authentication problem significantly harder. This calls for a new design of the authentication data structures and procedures, together with optimization techniques, without which the authentication would be less practical.

As for location privacy, the literature of mobile computing and spatial databases extensively investigates this problem in various research domains, including query processing [12, 1, 8, 6, 23, 20, 11], message communication [7, 25], and location data publishing [24, 10]. In most works, location cloaking has been the predominant technique of privacy protection. However, it only protects privacy conditionally against certain privacy metrics, such as k-anonymity. Except for very few works [8, 23, 20, 11], *unconditionally* protecting user locations by disclosing nothing about them is an unprecedented task. Our work is the first of this kind on query authentication.

3. PROBLEM FORMULATION

Let us formally model the user locations as a spatial dataset $\mathcal{D}$ in an integer-domain d-dimensional space, and the location-based service as a range query Q in this space. Q can be represented by a hypercube $[\alpha, \beta]$ where $\alpha = (\alpha_1, \alpha_2, \cdots, \alpha_d)$ and $\beta = (\beta_1, \beta_2, \cdots, \beta_d)$, denoting the lower bound and upper bound of Q respectively. Without loss of generality, we assume the query results are the identifiers of users (denoted as ids) whose locations fall into the hypercube. In a real location-based service, Q may return specific contents to the querying client, such as the users' Facebook pages or the total number of matching users as in the motivating example. These contents can be derived faithfully from the identifiers.[2] The query Q is executed by the service provider (SP, or simply "server") on the dataset $\mathcal{D}$ that is authorized and signed by the location registry. The client needs to verify that the SP executes Q faithfully. As such, together with the query results, the SP also returns the authentication data structure (i.e., the verification object or VO) to the client. The challenge of this paper is to authenticate the range query results while preserving users' location privacy, or as stated in [2], "to prevent other parties from learning one's current or past locations." Obviously, cloaking user locations cannot fulfill this requirement completely, while simply pseudo-anonymizing user identifiers cannot work either as these ids are often needed by the client in many location-based services (e.g., in the Facebook page example above or, if the client is a business, for billing and service delivery to these users). Even if these ids are not needed, pseudo-identifiers are still vulnerable to association attacks that join the locations with background knowledge [12, 7, 8]. Therefore, we shall design new VO and associated authentication protocols which protect locations unconditionally. That is, when the client verifies the query results, it cannot infer any information about the locations of returned users, beyond what is implied from the results.[3]

3.1 Security Model

We assume that: (1) the location registry is trusted by the querying client and SP; (2) the SP has read access to the user locations; and (3) the location registry does not collude with the client or the SP. Therefore, the two security

[2]In the "total number" case, the identifiers are not needed as results — those endorsed values derived from them will suffice in the verification process. More details are in Footnote 4.

[3]Our problem is to prevent the client from knowing beyond what the query tells. A malicious client may attempt to narrow down or pinpoint the user locations by exhaustively sending range queries with extremely small extents. Depending on the business model, such threats can be prevented by access control, query parameters screening, or imposing penalties on heavy users; however, this topic is beyond the scope of this paper.

threats in this problem are: (1) the client may attempt to infer location information of returned users from the VO; and (2) the SP may dishonestly return wrong results of the query in favor of its own benefits.

For ease of presentation, we also assume all parties (the client, SP and location registry) follow a semi-honest model. That is, they follow the designated protocol properly except that they may record intermediate results and try everything they can to deduce about the private information of other parties. It has been shown that any protocol that is proven secure in the semi-honest model can be adapted to be secure in a malicious model, where the participants may not follow the protocol at all, by imposing the participants to follow the protocol [9].

Finally, we follow the common assumption in cryptography that any party may know the protocol and algorithms of other parties, except for the secret keys the other parties may own. Nonetheless, the capability of any adversary is bounded by its polynomial computational power and storage space.

4. PRELIMINARY: PRIVACY-PRESERVING AUTHENTICATION FOR SINGLE- DIMENSIONAL RANGE QUERIES

To start with, we first focus on the basic case where $d = 1$. That is, the user location is a single field x and is indexed by a B^+-tree. Since the x values of users are sorted and threaded in the leaf level of the index, the query is equivalent to finding user r_a, such that $r_a.x \geq \alpha$ and $r_{a-1}.x < \alpha$, and user r_b, such that $r_b.x \leq \beta$ and $r_{b+1}.x > \beta$. Then the result users are $\{r_a, r_{a+1}, \cdots, r_b\}$. The authentication should verify the following three conditions:

1. **compliance condition**: $r_a.x \geq \alpha$, $r_{a-1}.x < \alpha$, $r_b.x \leq \beta$ and $r_{b+1}.x > \beta$;
2. **genuineness condition**: no *id* attributes of $r_a, r_{a+1}, \cdots, r_b$ are tampered with;
3. **completeness condition**: no other user beyond the result set has such x that $\alpha \leq x \leq \beta$.

If the disclosure of location x were not a concern, condition (1) could be trivially verified by sending the x values of users r_{a-1}, r_a, r_b and r_{b+1} to the client, and conditions (2)(3) could be verified by a Merkle B-tree where the digest of each user is the joint hash on its x and *id* fields. However, as required in privacy-preserving authentication, verifying (1) without disclosing x values to the client needs some cryptographic constructs. In [17], Pang et al. designed a proof for verifying $x \geq \alpha$ without disclosing x (according to our problem definition, both x and α are integers). The idea is to let the client and server jointly compute the digest g of value $x - L$, where L is the lower bound of domain x. The server first computes $g(x-\alpha)$ and sends it to the client, who then computes $g(x-L) = g(x-\alpha) \otimes g(\alpha - L)$, where $\otimes$ is a well-defined operation on the digest. Note this equation is guaranteed by the homomorphic property of the digest function $g()$, and $g()$ has another property that accepts only non-negative numbers. As such, by sending $g(x - \alpha)$, the server proves $x \geq \alpha$. The client verifies $x \geq \alpha$ by comparing the computed $g(x-L)$ value with the $g(x-L)$ value signed by the data owner (i.e., the location registry in this paper). Similarly, by jointly computing $g(U-x)$, where U is the upper bound of domain x, the client can verify $x \leq \beta$ without disclosing x.

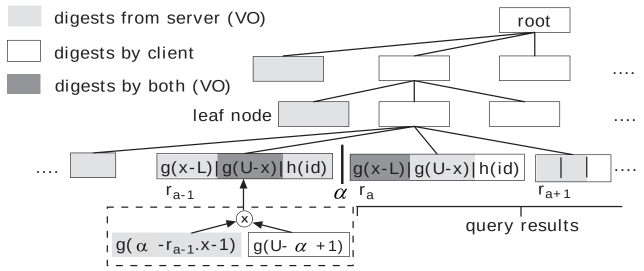


Figure 3: Verification Object for 1D Range Query

With the digest function $g()$, we design the verification for single-dimensional range queries on a Merkle B-tree as follows. The digest of each leaf entry (i.e., user) e in a leaf node is defined as:

$$dig(e) = h(g(e.x - L)|g(U - e.x)|h(e.id)), \qquad (1)$$

where "|" is concatenation and $h()$ is a one-way hash function. The digest has three components — the first can be used to verify $e.x \geq$ some value α as $e.x$ has a positive sign, the second component can be used to verify if $e.x \leq$ some value β as $e.x$ has a negative sign, and the last component can be used to verify if the *id* attribute is tampered with.

Recursively, the digest of a leaf node is computed from the digests of all its leaf entries; the digest of a non-leaf node N (including the root node) is computed from the digests of all its child nodes $N_1, N_2, \cdots, N_m$. It is noteworthy that, by convention MHT uses a concatenation-based recursive digest definition (as shown in Fig. 2(b)), which may disclose the order of child nodes. To avoid this, we propose an order-insensitive recursive definition as below:

$$dig(N) = h^2(dig(N_1)) \cdot h^2(dig(N_2)) \cdots h^2(dig(N_m)) \mod n$$

where $n = pq$ and p, q are two large primes. Obviously, this definition is order-insensitive as modular multiplication is a commutative operator. Further, this definition guarantees the authenticity of $dig(N)$ in a stronger fashion than the concatenation-based definition. In fact, if the server attempts to accommodate a modified $dig(N_i)$, to retain the same $dig(N)$ value, it has to forge some $dig(N_j)$ such that $h^2(dig(N_i)) \cdot h^2(dig(N_j)) \mod n$ is intact. The hardness of this task is guaranteed by two levels of security constructs. The outer level is the modular square — due to the computational hardness of the quadratic residuosity problem [9], given $h^2(dig(N_j)) \mod n$, to find its modular square root $h(dig(N_j))$ is asymptotically as hard as to factorize n. The inner level is the one-way hash function $h()$ — even if $h(dig(N_j))$ could be found, the server would yet have to find $dig(N_j)$ and send it to the client. This is referred to as the "first preimage attack" on $h()$, which alone is as difficult as (if not more difficult than) forging in the original concatenation-based definition, which is a constrained "second preimage attack" [9].

Now that the digests are defined, for query $Q = [\alpha, \beta]$ whose result user set $Q = \{r_a, r_{a+1}, \cdots, r_b\}$, the VO should include the following digests or components of digests:

1. $g(\alpha - r_{a-1}.x - 1)$ to verify $r_{a-1}.x < \alpha$;
2. $g(r_a.x - \alpha)$ to verify $r_a.x \geq \alpha$;
3. $g(\beta - r_b.x)$ to verify $r_b.x \leq \beta$;
4. $g(r_{b+1}.x - \beta - 1)$ to verify $r_{b+1}.x > \beta$;
5. all digests or digest components that are necessary for the client to compute the digest of the root node;
6. the signed digest of the root node;

Fig. 3 illustrates the VO and protocol, where only the "α" side is depicted for simplicity. The client uses item (1) $g(\alpha - r_{a-1}.x - 1)$ to compute one of the components of digest r_{a-1}, $g(U - r_{a-1}.x) = g(\alpha - r_{a-1}.x - 1) \otimes g(U - \alpha + 1)$. A similar approach applies to items (2)(3)(4). These digest components, illustrated by dark-grey boxes in Fig. 3, are computed collaboratively by the client and the server. The digest components in item (5), illustrated by light-grey boxes, are directly returned by the server. These digest components are used for the client to compute the digests of (i) boundary non-result object r_{a-1} (using digest components $g(r_{a-1}.x - L)$ and $h(r_{a-1}.id)$ from (5) and computing $g(U - r_{a-1}.x)$ from (1)); (ii) boundary result object r_a (using digest component $g(U - r_a.x)$ from (5), computing $g(r_a.x - L)$ from (2), and computing $h(r_a.id)$ from returned result $r_a.id$); (iii) internal result objects such as r_{a+1} (using digest components $g(r_{a+1}.x - L)$ and $g(U - r_{a+1}.x)$ from (5) and computing $h(r_{a+1}.id)$ from return result $r_{a+1}.id$ [4]); and (iv) intermediate nodes that are needed to compute the root node digest. To summarize, the VO includes all dark-grey and light-grey boxes, which can be either digests or digest components. With this VO, the client authenticates the results by computing the digest of the root node in a bottom-up fashion along the tree path. In Fig. 3, all these client-computed digests are shown by white boxes.

5. AUTHENTICATION FOR MULTI- DIMENSIONAL RANGE QUERIES

In this section, we study the $d > 1$ case for our problem. To support location-based services, we are particularly interested in 2D datasets and queries. In what follows, we propose three schemes for privacy-preserving authentication on two common multi-dimensional indexes, namely, the R-tree and grid-file, respectively.

5.1 Authentication on R-tree Index

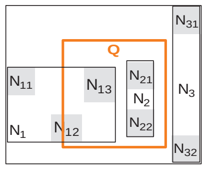
In Section 4, only users at the boundaries of the result set (i.e., $r_{a-1}, r_a, r_b, r_{b+1}$) need to be verified with the query range $[\alpha, \beta]$ because all of them are in linear order, i.e., they are sorted by their x values at the leaf level of the B^+-tree index. However, this trick no longer works in a multi-dimensional range query as the leaf level of the index is not sorted. To verify that the server traverses the index correctly and visits nodes no more and no less, the boundary verification in Section 4 needs to be applied on *every node where the query stops branching*. In an R-tree index, this requires the digest of any node N to comprise the minimum bounding boxes (MBB) of its child entries. Let us start with the definition of the digest for a leaf entry (i.e., user) e as:

$$dig(e) = h(dig(e.mbb)|h(e.id)), \tag{2}$$

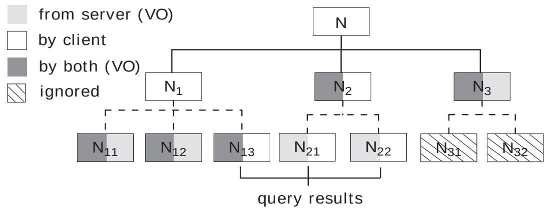
where $dig(e.mbb)$ is defined similarly to Eqn. 1 for privacy-preserving purposes as:

$$dig(e.mbb) = h(g(e.mbb.\mathbf{l} - \mathbf{L})|g(e.mbb.\mathbf{u} - \mathbf{L}) \\ |g(\mathbf{U} - e.mbb.\mathbf{l})|g(\mathbf{U} - e.mbb.\mathbf{u})). \tag{3}$$

[4] $r_{a+1}.x \geq \alpha$ is guaranteed by the Merkle B-tree and thus does not need to be verified. For aggregation queries (e.g., returning the "total number"), since ids are not returned as results, digest components $h(r_a.id)$, $h(r_{a+1}.id)$, ..., $h(r_b.id)$ are returned directly from the server for the client to verify the aggregate result.



(a) Nodes and Objects



(b) R-tree Index and Verification Object

Figure 4: Query Authentication on R-tree Index

Here $\mathbf{u}, \mathbf{l}, \mathbf{U}$ and $\mathbf{L}$ are all vectors, denoting the upper and lower bounds of the multi-dimensional MBB and the entire domain, respectively. Then $dig(N)$, the digest of an R-tree node N is defined as:

$$h^2(dig(N.mbb)) \cdot h^2(dig(N_1)) \cdots h^2(dig(N_m)) \mod n \tag{4}$$

The definition above effectively splits $dig(N)$ into two parts: the left part $h^2(dig(N.mbb))$ depends on the node itself and the right part $h^2(dig(N_1)) \cdots h^2(dig(N_m))$ depends on its child nodes. Conceptually, for $dig(e)$, the left part is $dig(e.mbb)$ while the right part is $h(e.id)$.

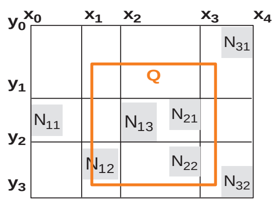
Fig. 4 illustrates the query processing and VO construction procedure, where $N_1, N_2, \cdots$ are the leaf nodes and $N_{11}, N_{12}, \cdots$ are the leaf entries (i.e., users). The query processing starts from the root node N. Since N_1 intersects with query Q, it will be branched, i.e., its subtree is further explored. N_2 is totally inside Q, so N_2 will not be branched for verification; but all leaf entries in its subtree will be accessed and returned as results. N_3 is totally outside of Q, so it will not be branched, either; and there are no results from N_3. As such, among the child nodes of N, N_2 and N_3 require boundary verification as they stop branching. Then N_1 is branched, since it is already a leaf node, its entries N_{11}, N_{12} and N_{13} will stop branching anyway, which means all of them require boundary verification. The final result users are $\{N_{13}, N_{21}, N_{22}\}$ and Fig. 4(b) shows the VO, which includes:

1. the digest components for boundary verification, including $g(\alpha - N_{11}.mbb.u)$, $g(\alpha - N_{12}.mbb.l)$, $g(N_{12}.mbb.u - \alpha)$, $g(N_{13}.mbb.l - \alpha)$, $g(\beta - N_{13}.mbb.u)$, $g(N_2.mbb.l - \alpha)$, $g(\beta - N_2.mbb.u)$, $g(N_3.mbb.l - \beta)$, which are shown in dark-grey boxes;
2. all digests or digest components that are necessary for the client to compute the root digest, shown in light-grey boxes;
3. the signed digest of the root node.

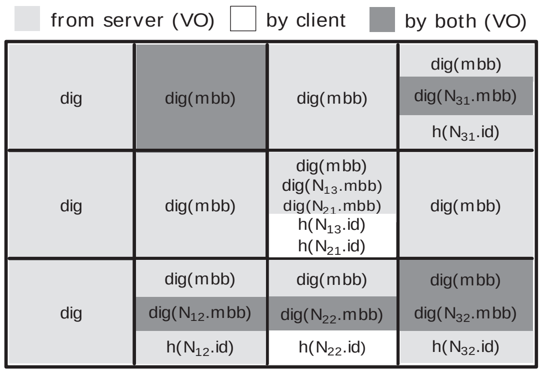
In Fig. 4(b), the right parts of the digests of result users (i.e., $h(e.id)$) are shown in white boxes, i.e., they are computed by the client, because these id values are sent to the client as results.

5.2 Authentication on Grid-File Index

The R-tree index may not be favorable for privacy-preserving authentication due to the following two reasons. First, it loses the linearity of leaf-level entries. Consequently, the



(a) Grid and Object Placement



(b) Grid and Verification Object

Figure 5: Query Authentication on Grid-File Index

boundary verification must be conducted on every single node that stops branching, instead of only on the four boundary leaf entries in the B$^+$-tree. This could be very costly as each boundary verification requires a computation-intensive digest function $g()$. Second, the R-tree index does not favor queries with small ranges. Since only the root digest is signed, the verification of any query must go all the way up to the root, which requires a significant number of necessary digests in VO and high computational overhead. As an extreme case, even if the query is contained in only one leaf node, the total number of boundary verifications is fh, where f is the average node fanout and h is the tree's depth. To address these two issues, in this subsection we turn to an alternative index — grid-file — and force the index nodes to regain linearity by imposing an order on the grid cells.

Fig. 5(a) shows the grid partition on the same dataset in Fig. 4. The grid is formed by horizontal partition lines $x_0, x_1, \cdots, x_4$, and vertical partition lines $y_0, y_1, \cdots, y_3$. To protect location privacy, the numbers of these lines, i.e., 5 and 4, and their coordinates/values are unknown to the client. Given the same query Q as in Fig. 4, it overlaps with 9 cells in Fig. 5(a). Since the cells are sorted in their x and y values, to verify $Q = [\alpha, \beta]$ the client only needs to verify the four boundary lines of Q with respect to the grid partition lines. In this example, $x_1 \le \alpha_x < x_2$, $x_3 < \beta_x \le x_4$, $y_0 \le \alpha_y < y_1$ and $y_2 < \beta_y \le y_3$. Then users who are in those cells that are completely inside the boundary lines must be result users, e.g., users N_{13} and N_{21} in the cell $(3, 2)$ (i.e., the cell whose right and bottom bounds are x_3 and y_2); users who are in those cells that intersect with the boundary lines of Q need to be further verified with these bound lines, e.g., user N_{12} in the cell $(2, 3)$. To support boundary verification, the digest of each cell C that contains users $e_1, e_2, \cdots, e_m$ is defined as follows:

$$dig(C) = h(dig(C.mbb)|dig(e_1)|dig(e_2)|\cdots|dig(e_m)), \quad (5)$$

where the digest of its MBB, $dig(C.mbb)$, is defined as:

$$dig(C.mbb) = h(g(C.mbb.\mathbf{u} - \mathbf{L})|g(\mathbf{U} - C.mbb.\mathbf{l})). \quad (6)$$

Note that the digest definition of $C.mbb$ is simpler than that of $N.mbb$ for an R-tree node in Eqn. 3 because the boundary verification is always in the form of $C_{i,j}.mbb.l < Q < C_{i,j}.mbb.u$ for some cell (i, j) and a boundary line of Q. On the other hand, the digest of user e, $dig(e)$, has the same definition as in Eqn. 2.

$$dig(e) = h(dig(e.mbb)|h(e.id)), \quad (7)$$

where $dig(e.mbb)$ has the same definition as in Eqn. 3.

$$dig(e.mbb) = h(g(e.mbb.\mathbf{l} - \mathbf{L})|g(e.mbb.\mathbf{u} - \mathbf{L}) \\ |g(\mathbf{U} - e.mbb.\mathbf{l})|g(\mathbf{U} - e.mbb.\mathbf{u})). \quad (8)$$

The distinguishing difference between R-tree and grid-file index on query authentication lies in their signatures. For the R-tree index, the common practice (as in MR-tree and in Section 5.1) is to sign the root digest only. The advantage is to limit the number of signature computations (which are believed to be costly) to 1. The disadvantage, however, is that the computation of the root digest requires a lot of necessary digests included in the VO and hash computations. While the hierarchy in the R-tree makes such costs less significant, the grid-file, unfortunately, is not given this edge if the same single-signature strategy is applied. Furthermore, in privacy-preserving authentication, the computational cost of signature operations is less dominant than the digest function $g()$. With these design factors, we argue that the digest of each cell in a grid-file should be signed. In addition to this, the lack of a hierarchy also leaves the completeness of the query result in jeopardy, as the server may omit in the result some cells together with their signatures. To remedy this, besides the digest of itself, the signature of a cell also chains up the digests of 4 neighboring cells. That is, the signature of cell (x, y) is defined as:

$$sig(C_{x,y}) = signature(dig(C_{x-1,y})|dig(C_{x,y}) \\ |dig(C_{x+1,y})|dig(C_{x,y-1})|dig(C_{x,y+1})) \quad (9)$$

For completeness, for the cells on the boundary of the entire space, we define some artificial cells (e.g., $C_{0,y}$ and $C_{x,0}$) with their digests set to 0.

It is noteworthy that the above definitions of cell digests and signatures require only local information, as opposed to the digest of an R-tree node which depends recursively on its descendant nodes. Therefore, upon a simple user insertion or deletion that does not change the topology of the index, the grid-file requires up to 5 cell digest reads, 1 cell digest write and 5 cell signature writes, whereas the R-tree index requires up to fh node digest reads, h node digest writes and 1 signature write (f is the average node fanout and h is the tree's depth). With this said, the grid-file has the advantage of handling more frequent user updates than the R-tree index.

Let us reexamine the query Q in Fig. 5(a). Fig. 5(b) illustrates the VO of this query and the client verification procedure. Since Q overlaps with 9 cells, to verify the results, the client needs the signatures of all these cells from the SP, which are included first in the VO. Note that since the signature of each cell has chained up the 4 neighboring cells, the VO should also include the digests of the cells that are immediately adjacent to these overlapping cells (i.e., the first column of cells in Fig. 5(b)). All rest work of the client is to compute the digests of all overlapping cells and check if they match the signatures. As with the R-tree index authentication, these digests or their components can be obtained in three ways (shown in different colors in Fig. 5(b)). First, if a boundary of an MBB needs to be verified with Q, then the digest component is computed collaboratively with the server and shown in dark-grey boxes. Note that the MBB may belong to a cell or a user, and therefore there are cell-level and user-level boundary verifications. In the cell level, since cells share the same and sorted partition lines, only the two corner cells (top-left and bottom-right) need to verify their boundaries with Q. As such, the digests of these two cells' MBBs (denoted by $dig(mbb)$) are marked in dark-grey boxes. In the user level, only those users who are in the intersecting cells need to verify their MBBs with Q. As such, the digests of the following users' MBBs are marked as dark-grey: $N_{12}, N_{22}, N_{31}, N_{32}$. Second, the query returns the id values of the result users, so the digest component $h(N_{ij}.id)$ of these users are computed by the client and are shown in white boxes. Third, all rest digests or digest components are retrieved from the server and are shown in light-grey boxes. Algorithm 1 summarizes the pseudo-code of the server query processing and VO construction procedure.

Algorithm 1 Query Processing and VO Construction

```
Input:  $Q$: the query and $\cup C_{i,j}$: the grid-file
Output: $\mathcal{C}$: the result set and $VO$: the verification object
Procedure:
1: $C_{l_x,l_y}$ and $C_{u_x,u_y}$ are the two corner cells;
2: for each $C_{i,j}$ that overlaps $Q$ do
3:     if $C_{i,j}$ is not a boundary cell then
4:         insert all users in $C_{i,j}$ to $\mathcal{C}$;
5:         insert digests of all user and cell MBBs to $VO$;
6:     else
7:         for each user $u$ do
8:             if $u$ is contained in $Q$ then
9:                 insert $u.id$ to $\mathcal{C}$;
10:            else
11:                insert $h(u.id)$ to $VO$;
12:            insert part of $dig(u.mbb)$ to $VO$;
13:        if $C_{i,j}$ is not a corner cell then
14:            insert $dig(C_{i,j}.mbb)$ to $VO$;
15:        else
16:            insert part of $dig(C_{i,j}.mbb)$ to $VO$;
```

5.3 Accumulative Digest for Grid-File Index

The grid-file index overcomes two drawbacks of the R-tree index on query authentication: overhead for small queries and frequent user location updates. However, as the number of signatures to be sent and verified by the client is proportional to the number of overlapping cells, the above scheme cannot scale well to large queries. Although signature aggregation techniques [3] can be applied on these signatures by the server to reduce bandwidth costs, it cannot reduce the computational costs, as verifying the aggregate signature is as computation-intensive as verifying all individual signatures. In this subsection, we propose *accumulative digest* as a remedy for authentication on large queries, by assuming infrequent user location updates.

The basic idea of accumulative digest is to associate the digest of each cell (which is then signed by the data owner, i.e., the location registry) with the digests of all cells spanned from the origin. Fig. 6 illustrates how the accumulative digests are derived, where the origin is on the top-left corner. The accumulative digest of cell $C_{x,y}$, denoted by $DIG(C_{x,y})$, is recursively defined as the hash value of the DIG of its immediate left and top cells, concatenated with its own cell

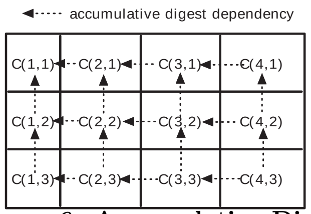


Figure 6: Accumulative Digest

digest dig. Formally,

$$DIG(C_{x,y}) = h(DIG(C_{x-1,y})|DIG(C_{x,y-1})|dig(C_{x,y})), \tag{10}$$

where $x > 1$ and $y > 1$. For the margin cases,

$$\begin{aligned} DIG(C_{1,1}) &= dig(C_{1,1}), \\ DIG(C_{x,1}) &= h(DIG(C_{x-1,1})|dig(C_{x,1})), \\ DIG(C_{1,y}) &= h(DIG(C_{1,y-1})|dig(C_{1,y})). \end{aligned}$$

As the accumulative digest of a cell already chains up neighboring cells, signature chaining is no longer necessary. As such, the signature of this cell is simply defined on its own accumulative digest: $sig(C_{x,y}) = signature(DIG(C_{x,y}))$.

Given these definitions, the VO of query Q in Fig. 5(b) will be revised as follows. Instead of sending the signatures of all overlapping cells, only the signature of the bottom-right cell among them (i.e., cell $C_{4,3}$) needs to be included in the VO. In addition, the client also needs some accumulative digests to start with when computing the accumulative digest of this bottom-right cell. Therefore, the DIG values of the two cells that are immediate left and top to the top-left overlapping cell (i.e., cell $C_{2,1}$) need to be included in the VO. In this example, since there is no immediate-top cell to $C_{2,1}$, only $DIG(C_{1,1})$ is included in the VO.

5.4 Security Analysis

In this subsection, we analyze the security of the proposed schemes. Recall the two threats in this paper are: (1) the client inferring the locations of returned users from the VO, and (2) the SP dishonestly returning wrong results. As our schemes follow the general Merkle hash tree or signature chaining paradigm for query authentication, the second threat is resolved as long as the digest function $g()$ holds the designed properties as discussed in Section 4. Therefore, in what follows we focus on the analysis of the first threat.

To demonstrate that the SP does not leak location information of any returned user to the client, we adopt *security proof by simulation* originated from zero-knowledge proof [9]. This is achieved by "simulating the view" of the client, i.e., while the client has a-priori knowledge of any user u being at position a with $P(u = a)$ probability, after receiving the VO, its posterior probability $P(u = a|VO)$ is the same as $P(u = a)$. In what follows, we assume a is a point for ease of presentation.

Thanks to the one-way property of the digest function $g()$, the only information disclosed by the VO to the client in all our schemes can be summarized by the following three types: (1) an MBB A is fully contained in Q (denoted by $A \subseteq Q$); (2) an MBB A (of a node or a cell) overlaps with but is not fully contained in the query range Q (denoted by $A \bigcap Q \neq \emptyset$); (3) an MBB A is to the left (right, top, bottom) of another MBB B. In the following lemmas, we show types (1) and (2) have the posterior probability equal to the a-priori probability.

LEMMA 5.1. *Let $u \in Q$ and $u \in A$, $\forall a \in A$, $P(u = a) = P(u = a|A \subseteq Q)$.*

PROOF.

$$P(u = a|A \subseteq Q) = \frac{P(A \subseteq Q|u = a) \cdot P(u = a)}{P(A \subseteq Q)}$$
$$= \frac{P(A \subseteq Q \bigwedge u = a)}{P(A \subseteq Q)} = P(u = a) \quad (11)$$

The first equality is due to Bayes' Theorem and the third equality is due to the fact that $A \subseteq Q$ is independent of $u = a$ in our privacy-preserving boundary verification. In fact, knowing $u = a$ does not limit the size or placement of the uncertain A because as a known point a, $a \in A$ and $u \in Q$ are known conditions. □

LEMMA 5.2. *Let $u \in Q$ and $u \in A$, $\forall a \in A$, $P(u = a) = P(u = a|A \bigcap Q \neq \emptyset)$.*

PROOF. Proof follows that of Lemma 5.1. □

As the R-tree based scheme only discloses types (1) and (2) information, the following theorem shows its security.

THEOREM 5.3. *The R-tree based scheme does not leak the location of any user u, given any VO.*

PROOF. Equivalently, we show there is a polynomial-time simulator SIM that can simulate the view of the client without knowing the data of SP. Specifically, it reproduces the VO of the client with the same probability distribution as if it were sent from the real SP.

According to Lemmas 5.1 and 5.2, without changing the distribution $P(u = a)$, SIM is allowed to know (1) if $A \subseteq Q$ and (2) if $A \bigcap Q \neq \emptyset$, for any MBB A. As such, SIM can reproduce the VO according to Section 5.1 as follows. If $A \subseteq Q$, SIM adds A's digest components for boundary verification to VO; else if $A \bigcap Q \neq \emptyset$, SIM adds to VO only necessary digest components of A to compute the root digest; otherwise, SIM adds only A's digest itself to VO. This VO has the same probability distribution as generated by the real SP. Also SIM runs in polynomial time. □

Unfortunately, it is hard to show type (3) information holds the same property as types (1) and (2). Since the two grid-based schemes disclose this type of information, we cannot reach a similar theorem for them as above directly. In fact, type (3) information adds complexity by possibly disclosing the relative positions of users. For example, from the VO in Fig. 5(b), the client can infer that user N_{22} is to the south of users N_{13} and N_{21} because the cell of the former is to the south of the cell of the latter two. Fortunately, there is an immediate remedy for grid-based schemes — instead of a strict grid where the upper bound of a cell, e.g., x_i^u, must coincide with x_{i+1}^l, the lower bound of the cell next to it, we adopt a loose grid where this requirement is eliminated. For example, in the x-axis in Fig. 5(a), instead of cell (2,1) having its lower bound coincide with the upper bound of cell (1,1) at line x_1, it can use another line x_1' as its lower bound; and x_1' can be either to the left or right of x_1. The former leads to overlapping cells while the latter leads to gaps between cells, both of which are valid as long as each object is assigned to one and only one cell. In a loose grid, while all lower bounds or upper bounds are still sorted, i.e., $x_i^l < x_j^l$ and $x_i^u < x_j^u$ if $i < j$, there is no direct relation between a lower bound x_i^l and an upper bound x_j^u anymore. As such, we effectively replace type (3) information with "next (or prior) to" information for grid-based schemes. The following lemma shows the latter has the posterior probability equal to the a-priori probability.

Sym.	Definition	Sym.	Definition
N	# of users	u	user rectangle size
N_A	# of node accesses	q	query length
N_l	# of nodes in level l	f	avg. node fanout
N_A^l	# of level-l node accesses	h	R-tree height
s_l	extent of level-l node MBB	R	result cardinality
D_l	density of level-l nodes	c	avg. cell length

Table 1: List of Symbols

LEMMA 5.4. *In a loose grid, let $u \in Q$ and $u \in A$, $\forall a \in A$, $P(u = a) = P(u = a|A \rightharpoonup B)$, where $\rightharpoonup$ stands for "next (or prior) to" in dimension x (or y).*

PROOF. Proof follows that of Lemmas 5.1 and 5.2. □

Now we reach the following theorem on the security of grid-based scheme .

THEOREM 5.5. *The grid-based scheme does not leak the location of any user u, given any VO and loose grid.*

PROOF. According to Lemmas 5.1, 5.2 and 5.4, without changing the distribution $P(u = a)$, SIM is further allowed to know (1) if $A \subseteq Q$, (2) if $A \bigcap Q \neq \emptyset$, and (3) if $A \rightharpoonup B$, for any MBBs A and B. As such, SIM can reproduce the VO according to Section 5.2 as follows. If $A \subseteq Q$, SIM adds A's digest components for cell-level boundary verification to VO; else if $A \bigcap Q \neq \emptyset$, SIM adds to VO A's digest components for user-level boundary verification; else if $A \rightharpoonup B$, SIM only adds to VO necessary components for partial (lower or upper) boundary verification, thanks to the loose grid. This VO has the same probability distribution as generated by the real SP. Also SIM runs in polynomial time. □

The loose grid does not change much to the VO construction and authentication. In fact, the only major change is that, during the cell-level boundary verification on the two corner (top-left and bottom-right) cells, there are possibly a set of "top-left" and "bottom-right" cells for verification. Nonetheless, since a strict grid is also a loose grid, we consistently use strict grid throughout the paper for crisp presentation, unless strict security is required.

6. PERFORMANCE ANALYSIS AND OPTIMIZATIONS

In this section, we analyze the performance of the proposed authentication schemes and propose optimizations that are orthogonal to the underlying schemes used.

6.1 Cost Models of Authentication Schemes

In this subsection, we derive the cost models of client verification computation (CPU) and VO size for the proposed authentication schemes on R-tree and grid-file index. For simplicity, CPU is in terms of the total number of $g()$ digest function calls and signature verifications, while VO is in terms of the number of digests, digest components and signatures. We also assume a 2D unit space and query Q is a square with length q. Table 1 summarizes the symbols used in this subsection. In the R-tree, boundary verifications occur on all accessed nodes or entries that stop branching, and they can be categorized into three cases (1) the leaf entries (results and non-results); or (2) the nodes that do not overlap with Q; or (3) the nodes that are totally contained in Q. Let K_1, K_2 and K_3 respectively denote their numbers. For the nodes in (3) and result entries in (1) (whose number is denoted by R), all 4 boundary lines need to be verified with both Q's lower and upper bounds, so each boundary verification requires 8 $g()$ calls. For the nodes in (2) and non-result entries in (1) (i.e., $K_1 - R$), only one of the boundary lines needs to be verified with either Q's lower or upper bound, so the boundary verification only needs one $g()$ call. In addition, only the root signature needs to be verified. Therefore, the client verification computation is:

$$CPU_{rtree} = 8(R + K_3) + (K_1 - R) + K_2 + 1. \qquad (12)$$

By definition, $K_1 = fN_A^1$, where N_A^1 is the number of level-1 node (i.e., leaf node) accesses for Q. $K_2 = \sum_{l=1}^{h-1} fN_A^{l+1} - N_A^l$. And $K_3 = \sum_{l=1}^{h-1} fN_A^{l+1} \cdot (q - s_l)^2$, where s_l is the average extent of node rectangles in level l, and $q \geq s_l$. Substituting these equations in Eqn. 12, we have:

$$CPU_{rtree} = 7R + (f-1)N_A + 8\sum_{l=1}^{h-1} fN_A^{l+1}(q - s_l)^2 + 1, \quad (13)$$

where N_A is the total number of node accesses. This equation shows that when q is small, the computation cost is dominated by $(f-1)N_A$, which coincides with our earlier discussion in Section 5.2. As q becomes larger, the third item will increase quadratically and dominates the others. To get the numeric value of CPU, Theodoridis et al. [22] developed a cost model of N_A^l, s_l and N_A for uniformly distributed objects as follows.

$$s_l = \sqrt{D_l \cdot \frac{f^l}{N_l}}, \qquad (14)$$

$$N_A^l = N_l \cdot (s_l + q)^2, N_A = \sum_{l=1}^{h} N_A^l, \qquad (15)$$

where N_l is the number of nodes in level l, i.e., $N_l = N/f^l$, and D_l is the density of level-l nodes, i.e., the number of nodes that cover an average point. $D_l = (1 + \frac{\sqrt{D_{l-1}} - 1}{\sqrt{f}})^2$, and $D_0 = N \cdot u$.

As for the VO, new items will be included in three cases: (1) when a node N is accessed, $f + 1$ digests will replace the node digest in VO according to Eqn. 4; (2) when a result entry is found, 2 digests will replace the entry digest in VO according to Eqn. 2; (3) when a boundary verification is required, 4 digests will replace the digest of MBB in VO according to Eqn. 3. Adding 1 for the root signature, the total VO size is:

$$\begin{aligned} VO_{rtree} &= fN_A + R + 3(K_1 + K_2 + K_3) + 1 \\ &= (4f - 3)N_A + R + 3\sum_{l=1}^{h-1} fN_A^{l+1} \cdot (q - s_l)^2 + 1. \end{aligned} \qquad (16)$$

Similar to $COMP$, the VO size is dominated by $(4f-3)N_A$ when q is small and by the third item when q is large.

In the grid-file index, let c denote the average cell length, then each cell has Nc^2 users. Cell-level boundary verification only occurs on two corner cells, each of which requires 4 calls of $g()$. User-level boundary verification occurs on the $4\frac{q}{c}$ boundary cells, each of which needs only one $g()$ call. As for the signature verification, there are $(\frac{q}{c})^2$ overlapping cells, each of which has a signature to verify. So the client verification computation is:

$$CPU_{grid} = 8 + 4Ncq + (\frac{q}{c})^2. \qquad (17)$$

The accumulative digest scheme reduces the number of signature verifications to 1, and therefore

$$CPU_{accu\_grid} = 9 + 4Ncq. \qquad (18)$$

The above two equations show that the accumulative digest scheme reduces the computation from quadratic to q to linear to q.

As for the VO, by default the digest of each overlapping cell is included in VO. It is then replaced with digests of its MBB and users according to Eqn. 5 in three cases: (1) the two corner cells whose MBB's digest is further replaced according to Eqn. 6; (2) the non-empty boundary cells whose user's digest is further replaced according to Eqns. 7 and 8; (3) any other non-empty cells whose user's digest is further replaced according to Eqn. 7. In addition, the signatures of overlapping cells, together with the digests of their immediate-adjacent cells, are also included in VO. Therefore,

$$VO_{grid} = 2(\frac{q}{c})^2 + 12Ncq + 2q^2N + 4\frac{q}{c} + 2. \qquad (19)$$

The accumulative digest scheme reduces the number of signatures in VO to 1, but adds two accumulative digests. Therefore,

$$VO_{accu\_grid} = (\frac{q}{c})^2 + 12Ncq + 2q^2N + 4\frac{q}{c} + 5. \qquad (20)$$

While the accumulative scheme has a smaller VO size, both schemes have their VO sizes dominated by $2q^2N$.

6.2 Linear Ordering and Embedding

In this subsection, we propose an optimization technique that addresses the non-linearity issue in multi-dimensional space. Whatever indexes we use for privacy-preserving authentication, R-tree or grid-file, at certain point we have to conduct boundary verification with almost every child entry. For R-tree, this occurs every time a node is branched; for grid-file, this occurs in the cells that partially overlap with (but is not totally contained in) the query range.

To filter out those entries that are faraway and hence reduce the number of entries for boundary verification, a baseline approach is to impose a linear order on these entries. It has the advantage of incurring no change on digest definition and no additional cost — the entries require an ordering anyway when they are serialized to external storage. Fig. 7 illustrates a linear order on their x values. Specifically, every entry is sorted by the x values of their rightmost boundaries. For example, in node N_1, users (i.e., child entries) are sorted as N_{11}, N_{12} and N_{13}. Given query Q', when N_1 is branched, since its leftmost boundary already exceeds the rightmost boundary of N_{12}, there is no need to verify the boundary with N_{11}. Nonetheless, this optimization is not at no cost: during new entry insertion, rather than appending in the end, it requires the new entry to respect the order, and therefore this insertion could cause rearrangement of the entries in the node.

The disadvantage above inspires us to use global ordering instead of local ordering within an index node. Specifically, each entry e can be embedded (i.e., mapped) to a

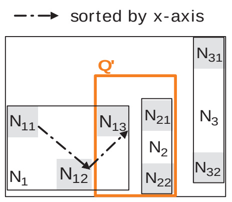
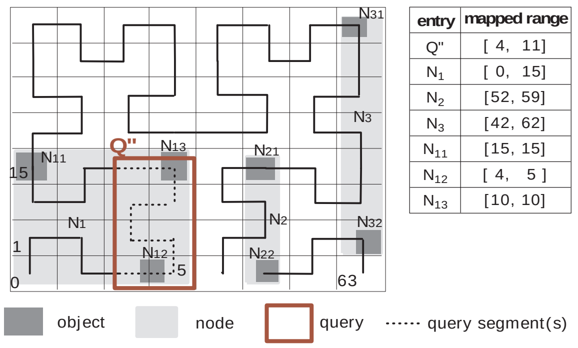


Figure 7: Linear Ordering



entry	mapped range
Q"	[4, 11]
N_1	[0, 15]
N_2	[52, 59]
N_3	[42, 62]
N_{11}	[15, 15]
N_{12}	[4, 5]
N_{13}	[10, 10]

Figure 8: Linear Embedding

value range according to its MBB, and thus denoted by $map(e.mbb)$. This mapping is public, which means it can be calculated on-the-fly by the data owner, the SP and the client, and therefore does not need to be stored in the node. The mapped value $map(e.mbb)$ is then included in the entry e's digest, in the same way as $e.mbb$ in Eqn. 2. That is,

$$dig'(e) = h(dig(e)|dig(map(e.mbb)))$$

To enable the same filtering mechanism as in linear ordering, this mapping should preserve most of the locality. There are a lot of mature techniques on dimension-reduction mapping, most famous of which are space filling curves. Fig. 8 shows a Hilbert curve of order 3 that partitions the space of Fig. 4(a) into 2^3 by 2^3 grid cells. The curve labels each cell with a Hilbert value from 0 to 63. The mapped value range of an MBB is the lower and upper bound of the cell values with which this MBB overlaps. For example, N_1's MBB overlaps the lower-left 16 cells, so $map(N_1.mbb) = [0, 15]$. The query Q" is also mapped to a value range, which is $[4, 11]$ in this example. With these ranges, the boundary verifications of N_2 and N_3 can be carried out on their 1D mapped values, instead of their 2D MBBs. Specifically, the client only needs to verify (in privacy-preserving manner) that the upper bound of Q", 11, is larger than the lower bounds of N_2 and N_3, which are 52 and 42, respectively, and unknown to the client. Note that linear ordering can be applied on top of linear embedding. Instead of sorting entries by the x values of their rightmost boundaries, the entries are sorted by the lower bounds of their mapped ranges. In this example, entries are sorted as N_1, N_2 and N_3. Then since N_2's lower bound already exceeds the upper bound of Q", there is no need to verify N_3.

It is noteworthy that both linear ordering and linear embedding are orthogonal to the index. As such, they can be applied to both R-tree and grid-file index.

7. EMPIRICAL RESULTS

In this section, we evaluate the experimental results of the proposed three authentication schemes, namely, R-tree based, grid-file based (*grid* for short) and accumulative digest for grid-file (*accu_grid* or *a.grid* for short). To simulate a real-life and yet sufficiently large location registry, we assume users are distributed on a road network and thus use the California Roads dataset from Census Bureau's MAF/TIGER database. The dataset contains 2,249,727 streets of California, from which all user location coordinates in our experiment are extracted and converted to their closest integers. Both an R-tree index and a grid-file index are built on user locations, with the page size set to 4KB. As such, the fanout of an R-tree node f and the capacity of a cell cap are both 200.

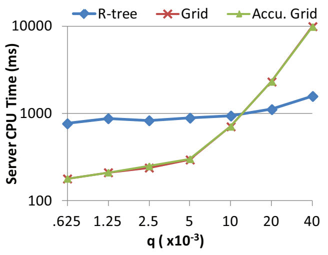
Parameter	Symbol	Value
dataset size	N	$2,249,727$
page size	–	4KB
query length	q	$[6.25 \times 10^{-4}, 4 \times 10^{-2}]$
R-tree node capacity	f	200
grid cell capacity	cap	200

Table 2: Parameter Settings for Experiments

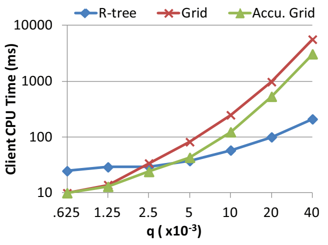
The client is set up on a desktop computer with Intel Core 2 Quad processor and 4GB RAM, running Windows XP SP3, and the server is set up on an IBM server xSeries 335, with Dual 4-core Intel Xeon X5570 2.93GHz CPU and 32GB RAM, running GNU/Linux. The code of our experiments is implemented and executed in OpenJDK 1.6 64-bit. The hash function $h()$ is 160-bit SHA-1, accordingly to which we set the length of n in the commutative digest definition. The signature function is 1024-bit RSA. We use the same digest function $g()$ as in [17] with the base of the canonical representation set to 16. For performance evaluation, we measure the computational cost (in terms of the server and client CPU time, for query processing and verification, respectively), the communication overhead (in terms of the size of the VO) and the query response time (as the total CPU time plus the communication time over a typical 3G network at 2Mbps download rate and 1Mbps upload rate). The query ranges are squares whose centroids are randomly generated and whose side lengths are from 6.25×10^{-4} to 4×10^{-2} of the total space length, as controlled by parameter q. For each measurement, 1,000 queries are executed and their average value is reported. Table 2 summarizes the parameter settings used in the experiments.

7.1 Basic Query Authentication Performance

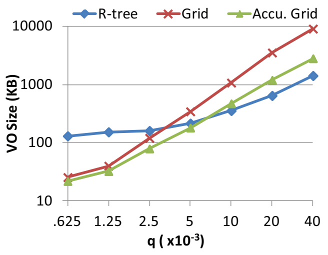
In this subsection, we evaluate the authentication performance of the three schemes without introducing any optimization. For visualization purpose, we normalize the dataset to a unit space. We repeatedly double the query length q from 6.25×10^{-4} to 4×10^{-2} and plot the server CPU time, client CPU time, VO size and query response time (together with the result size) in Fig. 9. These figures show that *grid* and *grid_accu* outperform R-tree in small and medium-sized queries, until at $q = 10 \times 10^{-3}$ where the query result size reaches 388. Furthermore, *grid_accu* consistently outperforms *grid*, in terms of the client CPU time and VO size. For example, at $q = 5 \times 10^{-3}$, *grid_accu* v.s. *grid* is 43 ms v.s. 82 ms (client CPU time) and 180 KB v.s. 344 KB (VO size). As q increases, this performance gap becomes even larger. This coincides with our analysis in Section 6.1 that *grid_accu* reduces the computation of *grid* from quadratic to linear and halves the VO size for large q. Nonetheless, for very large queries (e.g., $q = 20 \times 10^{-3}$ whose result size is $1,912$), R-tree is the best in all metrics, thanks to the hierarchy imposed in



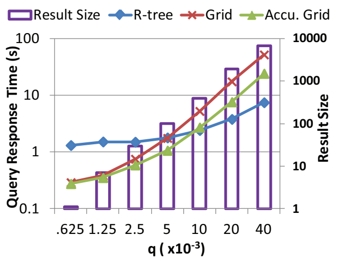
(a) Server CPU Time



(b) Client CPU Time



(c) VO Size



(d) Query Response Time

Figure 9: Basic Query Authentication Performance

	CPU Time (s)			Index Size (MB)		
	Rtree	**Grid**	**A.Grid**	**Rtree**	**Grid**	**A.Grid**
original	2311	11520	11619	124	2441	2441
x ordering	2453	11728	11770	124	2441	2441
Hil. embed.	2526	11859	11842	124	2441	2441

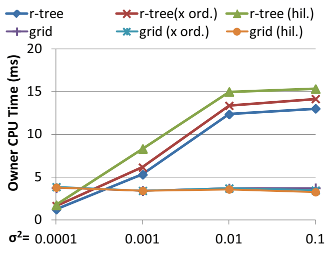
Table 3: Construction Cost

space. All these metrics of R-tree also have similar trends, which can be explained by the cost model in Section 6.1 that they are all dominated by $\sum_{l=1}^{h-1} f N_A^{l+1} \cdot (q - s_l)^2$ when q is large. As a summary, the query response time in Fig. 9(d) indicates the winner is *accu_grid* for queries of small and medium size, and is R-tree for extremely large queries.

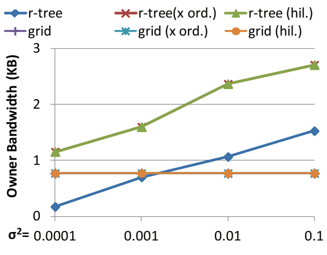
7.2 Performance with Optimizations

In this subsection, we evaluate the performance of the three schemes with the linear ordering and linear embedding introduced in Section 6.2. In particular, we implement the ordering by x-value (labeled by "x ordering") and the embedding by Hilbert values with curve order set to 10 (labeled by "Hilbert embedding" and imposed linear order on top of it). Table 3 shows the construction time and index size for different schemes. We observe that by introducing the accumulative digest, linear ordering or embedding does not have noticeable effect on the construction time, nor does they have any effect on the index size. The grid-based schemes take more time to construct than R-tree, simply because a grid file has more cells than R-tree nodes to compute digests or signatures. Nonetheless, the longest construction time is just about 3 hours, which is acceptable as the construction is an offline operation.

To evaluate the effects of optimizations for various queries, we plot the same metrics as above in Fig. 10 for $q = 6.25 \times 10^{-4}$ and $q = 40 \times 10^{-3}$, respectively. Except for the server CPU time, in all occasions the optimizations enhance the query authentication performance. This justifies our claim that the optimizations are transparent and orthogonal to the authentication schemes employed. The performance gain is particularly significant for *grid* and *accu_grid* schemes in small queries, which is up to 40% reduction. For example, in *A.Grid*(.625), the performance of x-ordering v.s. no optimization is: 8ms v.s. 10 ms (client CPU time), and 13 KB v.s. 21 KB (VO size). This corresponds to our discussion in Section 6.2 that imposing a linear order can prune unnecessary boundary verifications of faraway entries, although larger queries may make this pruning less beneficial. The server CPU time of optimized schemes is worse than the basic ones because we implement the VO construction with no cache, that is, the server computes the digests of entries on-the-fly. As such, pruning unnecessary boundary verifications essentially ships some $g()$ calls from the client back to the server. With the caching of digests in effect, we expect the optimized schemes will also outperform the basic ones in terms of server CPU time.



(a) CPU Time



(b) Bandwidth

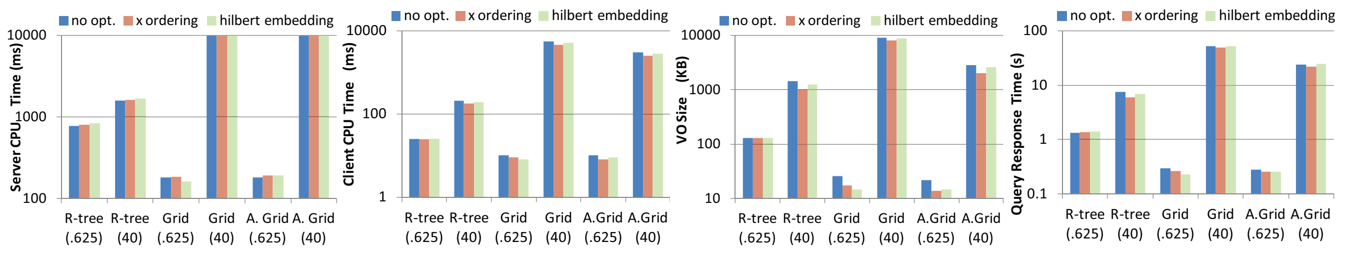
Figure 11: Data Owner Update Cost

7.3 Update Costs of User Locations

In this subsection, we evaluate the cost of dataset updates for the authentication schemes, with and without the optimizations. Since the *accu_grid* scheme is designed mainly for static datasets, we omit it in the comparison. We simulate a user random walk by moving *dev* distance away from his/her current position in each dimension, where *dev* follows a Gaussian distribution with $\mu = 0$ and σ^2 as the scaling factor that controls how faraway the user's new location is from the old one. The larger the σ, the farther away the new location is. Each location update is a deletion immediately followed by an insertion in the dataset. We simulate 5,000 location updates and plot the average CPU time and bandwidth (to update the server's copy) of the data owner for each update in Figs. 11(a) and 11(b), respectively. We observe that *grid* is more efficient than R-tree for location updates as only the user-residing cell and other 4 adjacent cells need to be updated, as opposed to R-tree where the update needs to be propagated along the tree path all the way to the root. Furthermore, as the deviation factor σ^2 increases, the cost of R-tree increases, which can be explained as follows. As σ^2 becomes larger, the new location shares less common tree path with the old location, and is also more likely to cause an upper node overflow or underflow, both of which lead to more digests of nodes to be updated. Another observation is that the proposed optimizations do incur overhead in the R-tree scheme, when the entries are resorted or even their embedded values (and hence their digests) recomputed. In Fig. 11, this overhead is about 1-2ms CPU time and 1KB bandwidth per update.

8. CONCLUSION

In this paper, we study the problem of privacy-preserving query authentication for location-based services. With the



(a) Server CPU Time (b) Client CPU Time (c) VO Size (d) Query Response Time

Figure 10: Optimized Query Authentication Performance

single-dimension building block of authentication on B^+-tree, we propose three authentication schemes for multi-dimensional indexes, including *R*-tree and grid-file index. We further enhance the efficiency of the schemes by two optimization techniques, namely, the linear ordering and embedding. The performance of all schemes is evaluated both analytically and empirically, which consistently shows their effectiveness and robustness under various system settings. The security perspective of these schemes is also studied.

As for future work, we plan to investigate on more query types for location-based services. In particular, we are interested in privacy-preserving authentication on k-nearest-neighbor queries. As neither the user locations nor their distances to the query point can be disclosed to the client, the authentication is even more challenging than range queries.

9. ACKNOWLEDGEMENT

The authors would like to thank the anonymous reviewers for their constructive comments and suggestions. This work was supported by Research Grants Council, Hong Kong SAR, China, under projects 210811 and 211510.

References

[1] B. Bamba, L. Liu, P. Pesti, and T. Wang. Supporting anonymous location queries in mobile environments with privacygrid. In *Proc. WWW*, 2008.

[2] A. Beresford and F. Stajano. Location privacy in pervasive computing. *IEEE Perv. Computing*, 2(1), 2003.

[3] D. Boneh, C. Gentry, H. Shacham, and B. Lynn. Aggregate and verifiably encrypted signatures from bilinear maps. In *EUROCRPYT*, pages 416–432, 2003.

[4] W. Cheng and K. Tan. Authenticating knn query results in data publishing. In *SDM*, 2007.

[5] W. Cheng and K. Tan. Query assurance verification for outsourced multi-dimensional databases. *Journal of Computer Security*, 2009.

[6] C. Chow, M. Mokbel, and W. Aref. Casper*: Query processing for location services without compromising privacy. *ACM TODS*, 2009.

[7] B. Gedik and L. Liu. Protecting location privacy with personalized k-anonymity: Architecture and algorithms. *IEEE TMC*, 7(1):1–18, 2008.

[8] G. Ghinita, P. Kalnis, A. Khoshgozaran, C. Shahabi, and K. Tan. Private queries in location based services: Anonymizers are not necessary. In *SIGMOD*, 2008.

[9] Oded Goldreich. *The Foundations of Cryptography – Volume 2*. Cambridge University Press, 2004.

[10] H. Hu, J. Xu, S. T. On, J. Du, and K. Ng. Privacy-aware location data publishing. *TODS*, 35(3), 2010.

[11] H. Hu, J. Xu, C. Ren, and B. Choi. Processing private queries over untrusted data cloud through privacy homomorphism. In *Proc. of ICDE*, 2011.

[12] P. Kalnis, G. Ghinita, K. Mouratidis, and D. Papadias. Preventing location-based identity inference in anonymous spatial queries. *TKDE*, 19(12):1719–1733, 2007.

[13] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Authenticated index structures for aggregation queries. *ACM TISSEC*, 13(32):1–35, 2010.

[14] F. Li, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proc. SIGMOD*, pages 121–132, 2006.

[15] F. Li, K. Yi, M. Hadjieleftheriou, and G. Kollios. Proof-infused streams: Enabling authentication of sliding window queries on streams. In *VLDB*, 2007.

[16] R. C. Merkle. A certified digital signature. In *Proc. Crypto*, pages 218–238, 1989.

[17] H. Pang, A. Jain, K. Ramamritham, and K.-L. Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD*, pages 407–418, 2005.

[18] H. Pang and K. Mouratidis. Authenticating the query results of text search engines. In *VLDB*, 2008.

[19] H. Pang and K.-L. Tan. Authenticating query results in edge computing. In *Proc. ICDE*, 2004.

[20] S. Papadopoulos, S. Bakiras, and D. Papadias. Nearest neighbor search with strong location privacy. In *VLDB*, 2010.

[21] S. Papadopoulos, Y. Yang, and D. Papadias. Continuous authentication on relational streams. *Very Large Data Bases Journal (VLDBJ)*, 19:161–180, 2010.

[22] Y. Theodoridis, E. Stefanakis, and T. Sellis. Efficient cost models for spatial queries using r-trees. *TKDE*, 12(1):19–32, 2000.

[23] W. Wong, W. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *Proc. SIGMOD*, 2009.

[24] T. Xu and Y. Cai. Exploring historical location data for anonymity preservation in location-based services. In *IEEE Infocom, Phoenix, Arizona*, 2008.

[25] T. Xu and Y. Cai. Location cloaking for safety protection of ad hoc networks. In *IEEE Infocom*, 2009.

[26] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Spatial outsoucing for location-based services. In *Proc. ICDE*, pages 1082–1091, 2008.

[27] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios. Authenticated indexing for outsourced spatial databases. *The VLDB Journal*, 18(3):631–648, 2009.

[28] M. L. Yiu, E. Lo, and D. Yung. Authentication of moving knn queries. In *Proc. ICDE*, pages 565–576, 2011.