Query Authentication and Processing on Outsourced databases

by

Weiwei Cheng
(*Bachelor of Computing, National University of Singapore*)

A thesis
submitted for the degree of Master of Science
in
Department of Computer Science
School of Computing

National University of Singapore
December 2010

Contents

Acknowledgment **vi**

Summary **vii**

1 Introduction **1**

1.1 Motivation 2

1.2 Contributions 5

1.3 Organization 7

2 Backgrounds **8**

2.1 Cryptographic Primitives 8

2.2 Related Work 9

3 Authenticating Window Query Results in Data Publishing **12**

3.1 System and Threat Model 12

3.2 Signature Chain in Multi-Dimensional Space 15

3.3 Verifying the Data Partitions 20

3.3.1 Space Partitioning 21

3.3.2 Data Partitioning 22

3.4 A Performance Study 24

3.4.1 Effect of Number of Dimensions 25

3.4.2 Effect of Different Data Distributions 25
3.4.3 Effect of Dataset Sizes 26
3.4.4 Effect of Node Capacity 27
3.4.5 Client Computation Cost 27
3.5 Summary 28

4 Authenticating KNN Query Results 29

4.1 Problem Definition 29
4.2 Enforcing Minimality: Hiding Non-answer Points 31
4.2.1 Collaborative Digest Computation 32
4.2.2 Hiding Non-Answer Points 32
4.3 Query Answer Verification 34
4.3.1 The Basic Solution 35
4.3.2 Generalizing to Other Query Types 37
4.4 kNN Authentication in Native Space 43
4.5 kNN Authentication in Metric Space: iDistance Based Scheme 46
4.6 Performance Study 50
4.6.1 Effect of Number of Dimensions 50
4.6.2 Effect of Different Dataset Size 51
4.6.3 Effect of Different Data Distributions 52
4.6.4 I/O Access Cost 53
4.7 Summary 54

5 Conclusion and Future Work 55

5.1 Conclusion 55
5.2 Future Work 56
5.2.1 Trust-Preserving Set Operations 56

5.2.2 Authenticating Aggregation Queries in Outsourced Database Systems 57

List of Figures

1.1 Data Publishing Model 3

3.1 Running Example 14

3.2 Partitioning Strategies 21

3.3 Chaining of Partitions. 22

3.4 The Verification R-tree. 24

3.5 Comparative Study 26

3.6 Client Computation Cost 27

4.1 Sample Queries on a 2-dimensional Dataset (A Running Example). . . . 30

4.2 Authentication Overhead on different Dataset Size 35

4.3 Illustration of the two-phase RNN algorithm in [17]. 39

4.4 Authentication of RNN point (Case (a)) 40

4.5 Authentication of RNN point (Case (b)) 42

4.6 iDistance based scheme 47

4.7 Authentication Overhead on Different Data Dimension 51

4.8 Authentication Overhead on different Dataset Size 52

4.9 Authentication Overhead on different Data Distribution 53

4.10 I/O Access Cost 53

Acknowledgment

I would like to express my sincerest gratitude to my supervisor, Professor Kian-Lee Tan, whose encouragement, guidance and support throughout my study period. I especially appreciate his kindness, generous and patient during the past two years, it would have been next to impossible to write this thesis without his help and guidance.

I also express my regards and blessings to all of those who supported me in any respect during the completion of this work. Moreover, I would like to thank my family members, especially my parents, and my husband, Xu Le, for their support and encouragement during the past few years.

Summary

In Outsourced Database model, data owners publish their data management requests through a number of remote, un-trusted external service providers. Service providers host owners' databases and offer seamless mechanisms to create, store, update and access (query) their databases. This model introduces several research issues related to data security. In this thesis, we introduce a mechanism for users to verify that their query answers on a *multi-dimensional dataset* are correct, in the sense of being complete and authentic. Two instantiations of the approach are studied:(1) Verifiable KD-tree (VKD-tree) that is based on space partitioning, and (2)Verifiable R-tree (VR-tree) that is based on data partitioning. The schemes are evaluated on window queries, and results show that VR-tree is highly precise, meaning that few data points outside of a query result are disclosed in the course of proving its correctness. Moreover, as an extension of the VR-tree, we proposed a mechanism that extend the signature-based mechanism for users to verify that their answers for k nearest neighbors queries on a multidimensional dataset are complete (i.e. no qualifying data points are omitted), authentic (i.e. no answer points are tampered) and minimal (i.e., no non-answer points are returned in the plain). Essentially, our scheme returns k answer points in the plain, and a set of $(\tilde{p}, q)$-pairs of points, where $\tilde{p}$ is the digest of a non-answer point p in the dataset to facilitate the signature chaining mechanism to verify the authenticity of the answer points, and q is a reference point (not in the dataset) used to verify that p is indeed further away from the query point than the kth nearest point. We study two instantiations of the approach - one based on the native data space using space partitioning method (a.k.a. R-tree) and the other based on the metric space using iDistance. We conducted an experimental study, and report our findings here.

Chapter 1

Introduction

Continued growths of the Internet and advances in networking technology have fuelled a trend toward outsourcing data management and information technology needs to external *Application Service Providers*. By outsourcing, organizations could operate their core task and other business applications via the Internet, while the involved maintenance of database could be operated in house (without connected to the Internet).

Database outsourcing [15] is an important manifestation of this trend. In this model, data owners engage third-party data servers (called publishers or service providers) to manage their data and process queries on their behalf [15, 23], and publishers are responsible for offering adequate software, hardware and network resources to host data owner's databases as well as mechanisms for the client to efficiently create, update and access the outsourced data.

This model is applicable to a wide range of computing platforms, including database caching [20], content delivery network [40], edge computing [21], P2P database [18], etc.

Comparing to the conventional client-server architecture where the owner also undertakes the processing of user queries, the Outsourced Database Model reduces Network

Latency by pushing application logic and data processing from the owner's data center out to multiple publisher servers situated near user clusters. Rather than fortifying the owners's data and provisioning more network bandwidth for every user, by adding publisher servers, scalability is much easier to be achieved. Moreover, the separation of business and maintaining tasks avoids the single point of failure in the data's own center, hence reducing the database's susceptibility to denial of service attacks and improves service availability.

The database outsourcing by Third-party Publisher poses numerous research challenges which influence the overall performance, usability and scalability. One of the foremost challenges is the security of stored data - it is essential to provide adequate security service measures to protect the stored data from both malicious outside attackers and the publisher itself. Security in this sense includes maintaining data integrity and guarding data privacy, moreover, how query processing can be efficiently performed over the secured data is closely relevant.

1.1 Motivation

High-value information, such as geophysical(or cartographic) data, pharmacological information, and business data, which are used in high-value decisions, are frequently made available for online-querying. Customers dependent upon highly reliable and efficient access to accurate information need assurance that their queries will be answered promptly, reliably, and accurately; incorrect information may lead to substantial losses.

Simple digital signature scheme and trusted-third party data publishing model are not suitable to solve this problem, both of them suffer from several problems.

With digital signature, the owner of the data operates an online database server, which processes queries and signs the results using a resident private signing key sk_{owner}. Users can verify the authenticity of the answers using the corresponding public key,pk_{owner}. Although this approach could provide both integrity and non-repudiation of the answers, it is impractical due to system vulnerability problem, as well as the expensive signing key protection mechanism. Moreover, the approach is generally too expensive to implement in the application domain.

A more scalable approach is to use a trusted third-party publishers of the data, in conjunction with a key management mechanism which allows certification of the signing keys of the publisher to speak for the author of the data. However, this approach also suffers from the problem and expense of maintaining a secure system accessible from the internet. Furthermore, to get a client to trust him to provide really valuable data, the publisher would have to adopt careful and stringent administrative policies, which might be more expensive for him (and thus also for the client).

In this work, we focus on query authentication and processing in an untrusted third-party data publishing model(in this thesis, we would only address the untrusted third-party data publishing model as Outsourced Database Model, and we will use these two terms exchangeably), especially concerned with data that is updated infrequently and queried much more often, such as financial histories, pharmacological data, cartography etc.

There are three main entities in the Outsourced Database Model: the data owner, the database service provider(publisher) and the client. Figure 1.1 depicts the model, in general, many instances of each entity may exist.

- The data owner maintains a master database, and distributes it with one or more associated signatures that prove 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 executes queries on behalf of the owner.

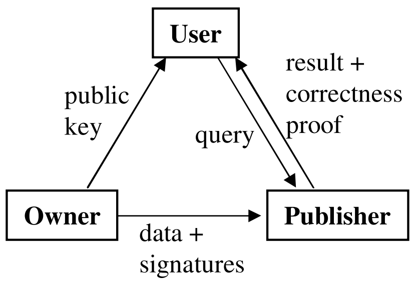


Figure 1.1: Data Publishing Model

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.

Moreover, as it is difficult for an attacker to successfully compromise multiple independent servers without being detected, security can be improved substantially when those servers are independent of each other in different part of the building or even belong to different data center.

- The user 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 signatures in 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. Query authentication is important for a client as it is necessary to ensure the results provided by the untrusted third party publisher is both inclusive and complete. Since the publishers 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 when they are used as basis for critical decisions.

Several existing works provide for checking the authenticity [25, 30] and completeness [15, 29] of query results. However, most of them only deal with one-dimensional datasets. Devanbu's scheme[15] handles multiple key attributes by essentially concatenating them in some preferred order $key_1|key_2|...|key_n$; this scheme is expected to be very inefficient for symmetric queries, such as window and nearest neighbor queries, that are typical in multi-dimensional context.

In this work, our primary concern 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 make logical alterations to the data, thus inducing incorrect query results. In addition, a compromised publisher server can be made to return incomplete query results by withholding data intentionally. Therefore mechanisms for users to verify the completeness as well as authenticity of their query results are essential for data publishing model. Moreover, it is highly desirable that only answers are returned in the plain to facilitate access control.

There are also other concerns that are not focused in our work. 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 steganography (e.g. [7, 32, 1]). Another concern relates to user access control, in specifying what actions each user is permitted to perform. Those issues have also been studied extensively (e.g. [13],[32], [26], [39]), and are orthogonal to our work here.

1.2 Contributions

In this work, we first propose a mechanism for users to verify that their window query results on a multi-dimensional dataset are *authentic* (i.e. no answer points are tampered) and *complete* (i.e. no qualifying data points are omitted). In addition, our approach guarantees *minimality* (i.e. no non-answer points are returned in the plain).

Our approach, which is described in chapter 3, builds authentication information into a spatial data structure, by constructing certified chains on the points within each partition, as well as on all the partitions in the data space. We introduce two schemes based on this approach. The first, the Verifiable KD-tree (VKDtree), is based on the space partitioning k-d tree. The second, the Verifiable R-tree (VRtree), employs data partitioning and is based on the R-tree. The schemes are evaluated on window queries, and results show that VRtree is highly precise, meaning that few data points outside of a query result are disclosed in the course of proving its correctness. Moreover, both schemes are computationally secure, and incur low processing and update overheads. To the best of our knowledge, the authentication mechanism introduced in this thesis is the first that enables a user to verify the *completeness* of a *multi-dimensional* query result generated by an untrusted server.

However, the mechanism above can only deal with hyper-rectangle window queries. While this scheme can be used for kNN queries, it will return more points in the plain than the answer points and thus is vulnerable to access control violation.

As an extention of the VR-tree mechanism, in chapter 4, we present the authentication scheme for kNN queries. Moreover, we further show that the entire framework can be nicely put together to support range, window, and RNN queries. While the extension to range and window queries is straightforward, that for RNN queries is non-trivial.

Like existing works [11, 29], our authentication mechanism for kNN query is based on the signature chain concept, and verifies that the k NN answers are complete (i.e. no qualifying data points are omitted), authentic (i.e. no answer points are tampered) and minimal (i.e. no non-answer points are returned in the plain). The core of the scheme is to return k answer points in the plain, and a set of $(\tilde{p}, q)$-pairs of points, where $\tilde{p}$ is the digest of a non-answer point p in the dataset to facilitate the signature chaining mechanism to verify the authenticity of the answer points, and q is a reference point (not in the dataset) used to verify that p is indeed further away from the query point than the kth nearest point. The scheme is minimal since only the k answer points are revealed.

We study two instantiations of the approach - one based on the native data space using space partitioning method (a.k.a. R-tree) and the other based on the metric space using iDistance. We have implemented both techniques, and our results show that the R-tree-based scheme has better performance when the number of dimensions is low $(d < 8)$, while iDistance-based scheme is superior in high-dimensional datasets $(d > 8)$. To our knowledge, this is the first reported work that addresses this problem.

We have implemented the proposed VR-tree and verification scheme, and conducted experiments on kNN queries. Our results show that we can verify kNN queries with low overheads.

1.3 Organization

The rest of the thesis proposal is organized as follows: In chapter 2, we discuss some backgrounds such as cryptographic primitives and related work. Next,we present our work on windows query authentication in data publishing model in chapter 3. Chapter 4 presents the authentication scheme for kNN queries. Finally, chapter 5 gives the conclusion and proposes some directions to pursue in the future work.

Chapter 2

Backgrounds

Before we present our solutions, in this chapter, we first describe some cryptographic primitives that our proposed solution based on, next we discuss some related works.

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 $h(.)$, is a hash function that works in one direction: it is easy to compute a fixed-length digest $h(m)$ from a variable-length pre-image m; however, it is hard to find a pre-image that hashes to a given hash value. Examples include MD5 [33] and SHA [6]. We will use the terms hash, hash value and digest interchangeably.

Digital signature: A digital signature algorithm is a cryptographic tool for authenticating the integrity and origin of a signed message. In the algorithm, the signer uses a private key to generate digital signatures on messages, while a corresponding public key is used by anyone to verify the signatures. RSA [34] and DSA [5] are two commonlyused signature algorithms.

Signature aggregation: As introduced in [10], 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.

Signature chain: In [29], a signature chain scheme is proposed that enables clients to verify the completeness of answers of range queries. A very nice property of the scheme is that only result values are returned, thus ensuring that there is no violation of access control. The scheme is based on two concepts: (a) The signature of a record is derived from its own digest as well as its left and right neighbors'. In this way, an attempt to drop any value from the answer of a range query will be detected since it would no longer be possible to derive the correct signature for the record that depends on the dropped value. (b) For the boundaries of the answer, a collaborative scheme that involves both the publisher and the client is proposed – the publisher performs partial computation based on but not revealing the two records bounding the answer and the query range, while the client completes the computation based on the two end points of the query range.

2.2 Related Work

Previous work on query authentication can be categorized to approaches that based on Merkle Hash Tree and approaches that based on Signature Chains.

Approaches [15, 14] utilize the Merkle Hash Tree to provide authentication. The owner builds a Merkle Hash Tree on the tuples in the database, based on the query attribute. Subsequently, the server answers the selection query by returning all tuples t covering the result as well as the minimum set of hashes necessary for the client to reconstruct the subtree of the Merkle Hash Tree corresponding to the query result.The scheme works for range queries, but not multi-point queries that pull back several segments of tuples.

The work by Roos et al [35] also employs the MHT to authenticate range queries. However, the focus is on encoding the VO in a compact form to minimize communication overhead; their scheme has the same limitations as [15].

In [16], Devanbu et. al. proposed a scheme that handles multiple key attributes by essentially concatenating them in some preferred order $key_1|key_2|...|key_n$. However, this scheme is expected to be very inefficient for symmetric queries, such as window and nearest neighbor queries, which are typical in multi-dimensional context.

The MB-tree concept proposed by Li et al. [19] combines concepts from the B+-tree and the MH-tree. The structure stores the actual records together with their digests into the leaves and associated each node a digest that computed on the concatenation of its children's digests. The data owner signs the root digest and send to the publisher along with the data. Range query results computed by the publisher are returned together with the two boundary records, digests of siblings along the path from the root to the boundary points are also returned. Upon receiving the results and VO, the client reconstructs the root digest and matches it against the signature. Unfortunately, the above schemes are applicable only for single dimensional data.

SearchDAG [22] transforms a wide class data structures into generalized authentication data structure. Authentication over peer-to-peer storage networks are proposed in [36]. Pang et al. [30]proposed the VB-tree structure, which is basically a B+-tree that incorporates hierarchically organized signed digest. This might be the first disk-resident authenticity data structure introduced; however, this structure doesn't ensure query completeness.

There are also approaches based on signature chains [29], a signature chain scheme is proposed that enables users to verify the completeness of answers of range queries. A very nice property of the scheme is that only result values are returned, thus ensuring that there is no violation of access control. The scheme is based on two concepts: (a) The signature of a record is derived from its own digest as well as its left and right neighbors'. In this way, an attempt to drop any value from the answer of a range query will be detected since it would no longer be possible to derive the correct signature for the record that depends on the dropped value. (b) For the boundaries of the answer, a collaborative scheme that involves both the publisher and the user is proposed – the publisher performs partial computation based on but not revealing the two records bounding the answer and the query range, while the user completes the computation based on the two end points of the query range.

Most of the above approaches only deal with one-dimensional datasets, and cannot handle queries over multiple attributes. Recently, an efficient authentication scheme for multi-attribute range aggregate queries was proposed in [31]. A multi-dimensional structure is used that maintains partial sums (or aggregates) at internal nodes of the structure. However, this work only deals with traditional relational aggregates such as count, sum and average, and is not designed for the more complex query types that we consider in this paper.

We note that there are other security issues that the data outsourcing model poses such as privacy, user authentication and access control. These have been studied extensively (e.g. [3], [32], [26], [39]), and are orthogonal to our work here.

Chapter 3

Authenticating Window Query Results in Data Publishing

In this chapter, we study the problem of authenticating window query results in data publishing. Section 2.1 describes the system and threat model by introducing a running example. Our authentication schemes are discussed in Sections 2.2 and 2.3, while Section 2.4 presents results from a performance study. Finally, Section 2.5 concludes the chapter.

3.1 System and Threat Model

Figure 1.1 in chapter One depicts the data publishing model, where we had described the three distinct roles of this model.

Our primary concern addressed in this work 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 make logical alterations to the data, thus inducing incorrect query results. Even if the data organization is hidden, for example through data encryption or steganographic schemes (e.g., [32]), 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 as well as authenticity of their query results are essential for the data publishing model.

In this work, we assume a d-dimensional data space. Let $L = (L_1, L_2, \ldots, L_d)$ and $U = (U_1, U_2, \ldots, U_d)$ be two points that bound the entire d-dimensional data space, where $L_r \leq U_r$ for all r. L and U are known to all users. Suppose the space contains N data points given by $\mathcal{DB} = \{p_1, p_2, \ldots, p_N\}$. We also denote $p_i = (x_{i1}, x_{i2}, \ldots, x_{id})$.

We would like design an authentication scheme for users to verify answers to the following queries:

- **Window query.** Let $p_l = (x_{l1}, x_{l2}, \ldots, x_{ld})$ and $p_u = (x_{u1}, x_{u2}, \ldots, x_{ud})$ be two points in the data space. A window query $Q_w = [p_l, p_u]$ returns all points within the hyper-rectangle determined by the two bounding points in Q_W In other words, a point $p_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ is in the answer if $x_{lj} \leq x_{ij} \leq x_{uj}$ for $1 \leq j \leq d$.

- **Range query.** Let $p_c = (x_{c1}, x_{c2}, \ldots, x_{cd})$. A range query $Q_r = [p_c, r]$ returns all points bounded by the hyper-sphere centered at p_c with radius r. In other words, a point $p_i = (x_{i1}, x_{i2}, \ldots, x_{id})$ is in the answer if $dist(p_c, p_i) \leq r$, where $dist(x, y)$ is a function that computes the Euclidean distance between two points x and y.

- **kNN query.** Let $p_c = (x_{c1}, x_{c2}, \ldots, x_{cd})$. A kNN query $Q_k = [p_c, k]$ returns k points $\mathcal{A} = \{q_1, q_2, \ldots, q_k\}$ such that

$$\forall q_i \in \mathcal{A}, \forall p_j \in \mathcal{DB} - \mathcal{A}, dist(p_c, q_i) < dist(p_c, p_j)$$

- **RNN query.** Let $p_c = (x_{c1}, x_{c2}, \ldots, x_{cd})$. An RNN query RNN($p_c$) returns all points that have p_c as their nearest neighbors, i.e.,

$$RNN(p_c) = \{p \in \mathcal{DB} | \forall p_j \in \mathcal{DB} - \{p\}, dist(p, p_c) < dist(p, p_j)\}$$

In this chapter, we discuss the authentication of window queries in a multi-dimensional dataset. The discussion of authenticating other query types are deferred to chapter 4.

A Running Example:

Consider a dataset containing 20 data points in two-dimensional space as shown in Figure 3.1. The figure also includes a window query **Q**, for which $\{r13, r14\}$ is the correct result. A rogue publisher may return a wrong result $\{r13, r14, r100\}$, which includes a spurious point $r100$, or $\{r13^*, r14\}$ in which some attribute values of $r13$ have been tampered with. To detect such incorrect values, the user should be able to verify the *authenticity* of query result.

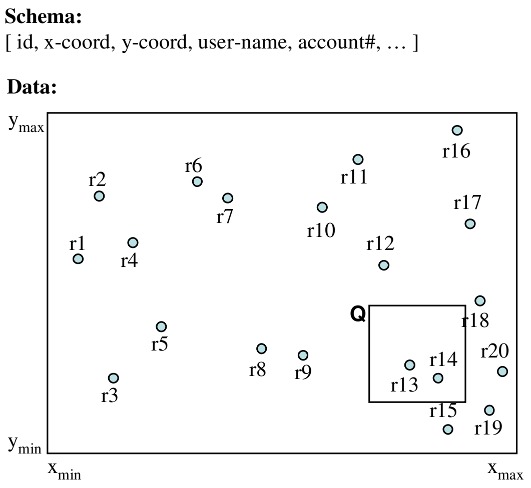


Figure 3.1: Running Example

A different threat is that the publisher may omit some result points, for example by returning only $\{r13\}$ for query **Q**. This threat relates to the *completeness* of query result.

3.2 Signature Chain in Multi-Dimensional Space

The goal of our work in this chapter is to devise a solution for checking the correctness of query answers on multi-dimensional datasets. The design objectives include:

- Completeness: The user can verify that all the data points that satisfy a window query are included in the answer.
- Authenticity: The user can check that all the values in a query answer originated from the data owner. They have not been tampered with, nor have spurious data points been introduced.
- Precision: Proving the correctness of a query answer entails minimal disclosure of data points that lie beyond the query window. We define precision as the ratio of the number of data points within the query window, to the number of data points returned to the user.
- Security: It is computationally infeasible for the publisher to cheat by generating a valid proof for an incorrect query answer.
- Efficiency: The procedure for the publisher to generate the proof for a query answer has polynomial complexity. Likewise the procedure for the user to check the proof has polynomial complexity.

Without loss of generality, we assume that the data in the multi-dimensional space are split into partitions – this can be done using a spatial data structure. To ensure that the answer for a window query is complete, two issues must be addressed. First, we need to prove that the answer covers all the partitions that overlap the query window. We refer to these partitions as candidate partitions. Second, we need to prove that all qualifying values within each candidate partition are returned. The first issue is dependent on the partitioning strategy adopted, and is deferred to Section 3.3. In the rest of this section, we shall focus on the second issue.

Assuming we have proven that the query answer covers all the candidate partitions, we now need to ensure that all the qualifying values in those partitions have not been dropped. Consider a candidate partition P for the window query $Q = [(q_{l1}, q_{l2}, \ldots, q_{ld}), (q_{u1}, q_{u2}, \ldots, q_{ud})]$. There are three possible cases: (a) Q contains P. Since the window query bounds the partition, we need to ensure that *all* the points in P are returned. (b) P contains Q. The query window is within the space covered by the partition. A naive solution is to return all the points in P. A better solution, which we advocate, is to return only those points that are necessary for users to check for completeness. In both cases, our concern is to ensure the secrecy of points that are outside Q. (c) P overlaps Q. This case can be handled by splitting P into two parts: the part of P that contains Q, and the part of P that does not overlap Q. The former is handled in case (b), while nothing needs to be done for the latter. Thus, we shall focus on cases (a) and (b), and not discuss case (c) any further.

Our solution extends the signature chain concept in [29] to multi-dimensional space. This is done by ordering the points within the partition, and then constructing the signature chain. In this chapter, we adopt a simple scheme of ordering the points based on increasing $(x_1, x_2, \ldots, x_d)$ value. In 2-d space, (x_1, y_1) is ordered before (x_2, y_2) if $x_1 < x_2$, or $x_1 = x_2$ and $y_1 < y_2$. Based on this ordering, we need to return all the points whose first dimension is within the range $[q_{l1}, q_{u1}]$, as well as the bounding points. Of course, some of these points may fall beyond the query window along the second dimension. *For such points that should not be part of the answer, we return only their digests rather than the actual values*, in order to protect their secrecy and achieve high precision.

We choose this simple ordering scheme over more sophisticated space filling curves [37] because: (a) A partition (corresponding to a 4K or 8K block/page) typically consists of a small number of points (100-200). Moreover, the actual number of points within a partition would be smaller than the maximum capacity (since the page is typically not full). As such, it may not be worthwhile to employ a complicated scheme. (b) None of the existing space filling curves perform well in all cases. Thus, they really offer no significant advantage over the simple scheme (especially given the small number of points).

For the example in figure 3.1, assuming that the entire space corresponds to one partition, the points would be ordered from r_1 to r_{20}. For case (a) where the query bounds the partition, r_1 to r_{20} would be returned; for case (b) where the query (i.e., the box that bounds r_{13} and r_{14}) is within the partition, we return the values of r_{13} and r_{14} and the digest of the various dimensions for r_{11}, r_{12}, r_{15}, r_{16} and r_{17}. We now present the details of our solution that extends the signature chain scheme to multi-dimensional setting.

Construction: Let $L = (L_1, L_2, \ldots, L_d)$ and $U = (U_1, U_2, \ldots, U_d)$ be two points that bound the entire data space, where $L_r \leq U_r$ for all r. L and U are known to all users. Consider a partition P bounded by two points $p_0 = (x_{01}, x_{02}, \ldots, x_{0d})$ and $p_{k+1} = (x_{(k+1),1}, x_{(k+1),2}, \ldots, x_{(k+1),d})$ where $x_{0r} \leq x_{(k+1),r}$ for all r. Suppose P contains k data points $p_1 = (x_{11}, x_{12}, \ldots, x_{1d}), \ldots p_k = (x_{k1}, x_{k2}, \ldots, x_{kd})$. Without loss of generality, we assume that p_i is ordered before p_j for $1 \leq i < j \leq k$. Clearly, p_0 is ordered before p_1 and p_{k+1} is ordered after p_k.

Our multi-dimensional signature chain constructs for each point within P an associated signature (based on [29]):

$$sig(p_i) = s(h(g(p_{i-1})|g(p_i)|g(p_{i+1}))) \tag{3.1}$$

where s is a signature function using the owner's private key, h is a one-way hash function, and | denotes concatenation. $g(p_i)$ is a function to produce a digest for point p_i:

$$g(p_i) = \sum_{r=1}^{d} h^{U_r - x_{ir} - 1}(x_{ir}) | h^{x_{ir} - L_r - 1}(x_{ir}) \tag{3.2}$$

where $h^j(x_{ir}) = h^{j-1}(h(x_{ir}))$ and $h^0(x_{ir})$ applies a one-way hash function on x.[1]

Moreover, for the two delimiters,

$$sig(p_0) = s(h(h(L_1| \dots |L_d)|g(p_0)|g(p_1))) \tag{3.3}$$

$$sig(p_{k+1}) = s(h(g(p_k)|g(p_{k+1})|h(U_1| \dots |U_d))) \tag{3.4}$$

In addition, each partition P has an associated signature:

$$sig(P) = s(h(g(p_0)|g(p_{k+1})|h(k))) \tag{3.5}$$

Query Processing: Assuming that a partition P is returned. We have to prove that all the data points within P that fall within the query window Q are returned.

Case (a): Q contains P. The verification process for this case is straightforward. The publisher server returns p_0 to p_{k+1}, and k, together with the respective signatures $sig(p_0)$ to $sig(p_{k+1})$ and $sig(P)$. (To reduce traffic overhead, we could send just one combined signature instead of the individual signatures, using the signature aggregation technique in [10].) The user first verifies that

$$s^{-1}(sig(P)) = h(g(p_0)|g(p_{k+1})|h(k))$$

Then, for each p_i, $1 \leq i \leq k$, the user verifies that p_i is indeed in P (by checking that P bounds p_i). Finally, for each p_i, $1 \leq i \leq k$, the user computes its digest and checks whether

$$s^{-1}(sig(p_i)) = h(g(p_{i-1})|g(p_i)|g(p_{i+1}))$$

[1]To achieve tighter security, $h^0(x_{ir})$ can be redefined as $h^0(x_{ir}|rand(p_i))$ where $rand(p_i)$ is a random number associated with p_i; in which case we will need to supply the corresponding $rand(p_i)$ with each returned record. For ease of presentation, we shall adopt the simpler definition of $h^0(x_{ir})$.

If all the above checks are successful, the answer contains all the data points in P.

Case (b): P contains Q. Let $p_i = (x_{i1}, x_{i2}, \ldots, x_{id})$. The data points in P can be separated into: (a) $p_\alpha, p_{\alpha+1}, \ldots, p_{\beta-1}, p_\beta$ such that $x_{i1} \in [q_{l1}, q_{u1}]$ for $\alpha \le i \le \beta$. These points can be further categorized into answer points ($\mathcal{A}$) and false positives ($\mathcal{F}$). For each answer point $p_i \in \mathcal{A}$, $\forall r\ x_{ir} \in [q_{lr}, q_{ur}]$, whereas for each false positive $p_i \in \mathcal{F}$, $\exists r\ x_{ir} \notin [q_{lr}, q_{ur}]$. (b) $p_1, \ldots, p_{\alpha-1}, p_{\beta+1}, \ldots, p_k$, which are clearly not answer points.

(i) For each point $p_i \in \mathcal{A}$, the server returns p_i and $sig(p_i)$.

(ii) For each point $p_i \in \mathcal{F} \cup \{p_{\alpha-1}, p_{\beta+1}\}$, the server returns several pieces of information: (i) if $x_{ir} \in [q_{lr}, q_{ur}]$, $h^{U_r-x_{ir}-1}(x_{ir})|h^{x_{ir}-L_r-1}(x_{ir})$ is returned; (ii) if $x_{ir} < q_{lr}$, $h^{q_{ur}-x_{ir}-1}(x_{ir})$ and $h^{x_{ir}-L_r-1}(x_{ir})$ are returned; (iii) if $x_{ir} > q_{ur}$, $h^{U_r-x_{ir}-1}(x_{ir})$ and $h^{x_{ir}-q_{lr}-1}(x_{ir})$ are returned.

(iii) The server also returns p_0, p_{k+1}, k, $sig(p_0)$, $sig(p_{k+1})$ and $sig(P)$.

With information from step (ii), the user can compute $g(p_i)$ without knowing the actual value of p_i:

- If $x_{ir} < q_{lr}$, the user applies h on $(h^{q_{ur}-x_{ir}-1}(x_{ir}))$ $(U_r - q_{ur})$ times to get $(h^{U_r-x_{ir}-1}(x_{ir}))$.
- If $x_{ir} > q_{ur}$, the user applies h on $(h^{x_{ir}-q_{lr}-1}(x_{ir}))$ $(q_{lr} - L_r)$ times to get $(h^{x_{ir}-L_r-1}(x_{ir}))$.
- The user computes $g(p_i)$ using Equation (3.2).

The above procedure is secure against cheating by the publisher provided $h^i(p)$ for $i < 0$ is either undefined or computationally infeasible to derive. We use an iterative hash function for $h^i(p)$, because there is no known algebraic function that satisfies the requirement. To ensure that $h^{-1}(p) \neq p$, a hash function is chosen that outputs a different digest length from the length of p.

Similar to case (a), the user verifies the completeness of the query answer as follows:

- Verify that the bounding box is correct using information from step (iii), and determine whether $s^{-1}(sig(P)) = h(g(p_0)|g(p_{k+1})|h(k))$.
- Verify that each point p in $\mathcal{A}$ is in P by checking that p is bounded by P.
- Verify that each point $p_i \in \mathcal{A}$ is authentic using information in step (ii) and the derived information to check $s^{-1}(sig(p_i)) = h(g(p_{i-1})|g(p_i)|g(p_{i+1}))$.

Again, any attempt by the publisher server to cheat would lead to an unsuccessful match in at least one of the above cases.

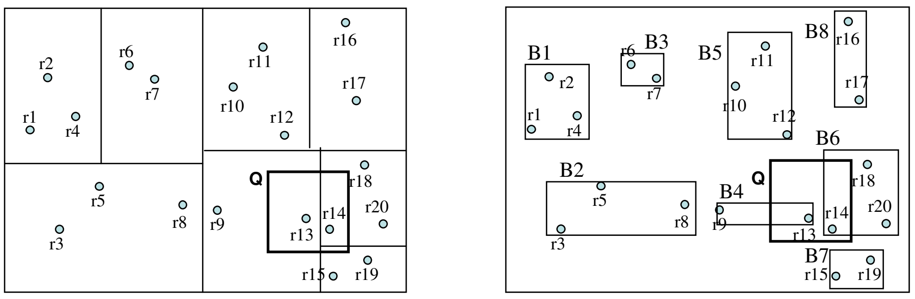
Finally, we emphasize that extra data points that are returned for proving completeness are in the form of digests. Thus only the existence of the data points are revealed, but not their actual content. If a non-answer $p_i \in \mathcal{F}$ has the same coordinate as an answer point $p_j \in \mathcal{A}$ along some dimension, both points will have the same digest for that dimension and p_i's coordinate will be revealed. This can be overcome by simply adopting $h^0(x_{ir}|rand(p_i))$ as explained previously.

3.3 Verifying the Data Partitions

Having shown how to prove that all qualifying data points in a candidate partition (that overlaps the query window) are returned correctly, we now look at the first issue of verifying that the query answer covers all the candidate partitions.

A naive solution is to treat the entire data space as a single large partition, so that the mechanism described in Section 3.2 alone suffices. However, we expect this solution to have poor precision.

To achieve high precision, we adopt partition-based strategies so that only those partitions that contain some qualifying data points need to be considered for a query. In this way, any potential information leakage is limited to only those partitions that contribute to the query answer, rather than across the entire data space. We present our



(a) Space Partitioning (b) Data Partitioning

Figure 3.2: Partitioning Strategies

solution based on two partitioning techniques (see Figure 3.2): space partitioning and data partitioning.

3.3.1 Space Partitioning

With space partitioning schemes, the partitions are disjoint but their union covers the entire data space. As such, all we need to do is to verify that the bounding boxes of the returned partitions are correct, and that the union of these partitions covers the query scope. The former has already been addressed in Section 3.2, while the latter is just a simple check on the partition boundaries.

To illustrate, Figure 3.2(a) shows the data space being partitioned through a k-d tree [9]. In the figure, the window of the query **Q** overlaps three partitions, so only data from these three partitions are returned in the answer.

Besides the k-d tree, other spatial indexing techniques like the grid file [27] and quadtree [38] can also be employed to help the publisher to locate the candidate partitions quickly. Our authentication mechanism entails no changes to the spatial data structures. (As we shall see shortly, this is not the case for data partitioning schemes.)

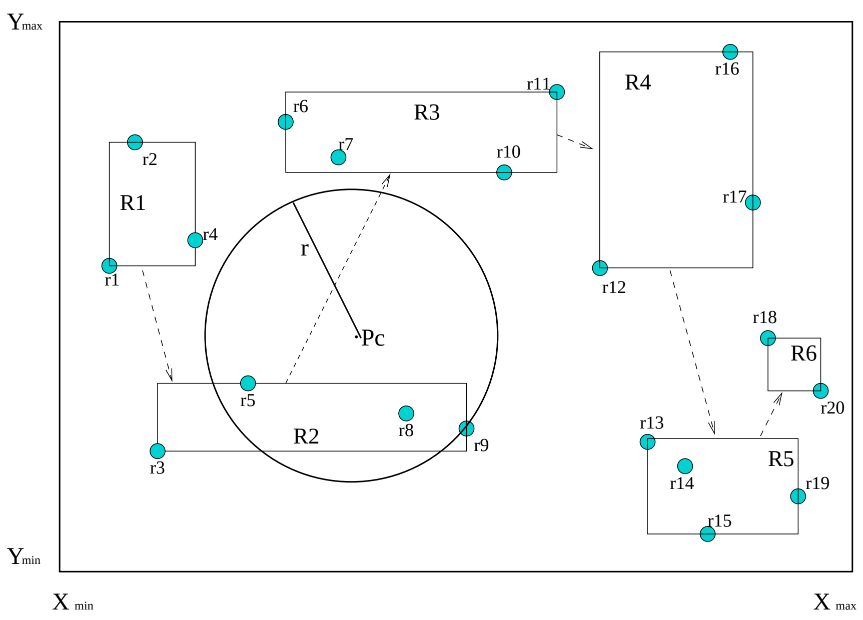


Figure 3.3: Chaining of Partitions.

3.3.2 Data Partitioning

With data partitioning approach (e.g., R-tree), the union of all the partitions may not cover the entire data space. Thus, space that contains no data points may not be covered by any partition, as illustrated in Figure 3.2(b). The existence of empty space poses a challenge to verifying the completeness of query answers: How does the user know that portions of a query window that are not covered by any returned partitions indeed are empty spaces, without physically examining all the partitions? Referring to Figure 3.2(b), how can the user be sure that Q only intersects boxes B4 and B6 and not the other partitions?

Our solution is to extend the signature chain concept to the partitions. Specifically, we order the partitions by their starting boundaries along a selected dimension (as is done for point data), then chain the partitions so that the signature of a partition is dependent on the neighboring partitions to its left and right.

Let the bounding box of the ith partition be demarcated by $[l, u]$ where $l = (l_{i1}, l_{i2}, \ldots, l_{id})$, and $u = (u_{i1}, u_{i2}, \ldots, u_{id})$. Each partition P_i has an associated signature (based on signature chaining):

$$sig(P_i) = s(h(g(P_{i-1})|g(P_i)|g(P_{i+1}))) \tag{3.6}$$

where P_{i-1} and P_{i+1} are the left and right sibling partitions of P_i, and $g(P_i)$ is defined as follows:

$$g(P_i) = h(h(l_{i1}|\ldots|l_{id})|h(u_{i1}|\ldots|u_{id})|h(k_i)) \tag{3.7}$$

where k_i is the number of points within P_i.

In addition, we define two fictitious partitions as delimiters. This is similar to what we did in building the signature chain for data points in Section 3.2, so we shall not elaborate further.

During query processing, all the partition information along with their signatures are returned as part of the query answer. The user can be certain that no partition is omitted, otherwise some signatures will not match. For those partitions that overlap the query window, the user then proceeds to check their data points using the mechanism in Section 3.2. The remaining partitions that do not intersect the query window are dropped from further consideration.

To minimize the extra partitions that are disclosed to the user, and to reduce performance overheads, we apply a hierarchical data partitioning indexing structure like the R-tree on the data. The partitions within each internal node of the R-tree are chained as described above. Given a window query, the publisher server iteratively expands the child nodes corresponding to those candidate partitions in the current node, starting from the root down to the leaf nodes. All the partition information and signatures along the path of traversal are added to the query answer for user verification.

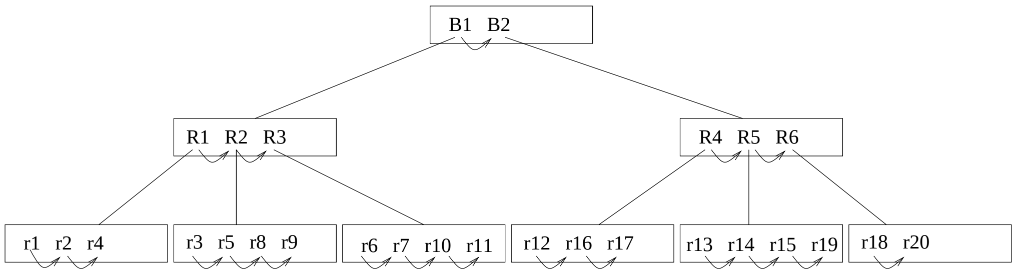


Figure 3.4: The Verification R-tree.

3.4 A Performance Study

In this section, we report results of an experimental study conducted to evaluate the effectiveness of our authentication mechanisms, which we have implemented in Java. We study three schemes: Verifiable KDtree (VKDtree) scheme that is based on space partitioning using the k-d tree; Verifiable Rtree (VRtree) scheme that is based on data partitioning using the R-tree; and Z-ordering scheme which employs Z-ordering [28] on the entire data space (as a single partition). The performance metric is the precision of query answers. Again, a low precision reveals the existence of extra data points and incurs traffic overhead, but not the actual content of those data points.

Unless stated otherwise, the following default parameter settings are used: the number of dimensions is 4, the data distribution is Gaussian, the number of data points is $1,000,000$. The domain of each dimension is [1, 10M]. The node capacity is 50 (i.e., each node holds up to 50 data points). Queries are generated by picking a point randomly from the dataset, then marking out the query window with the chosen point as center. The length of the query window along each dimension is $l \times domain\_size$; by default, l is set to 0.1. For each experiment, we run 500 queries, and take the average precision.

3.4.1 Effect of Number of Dimensions

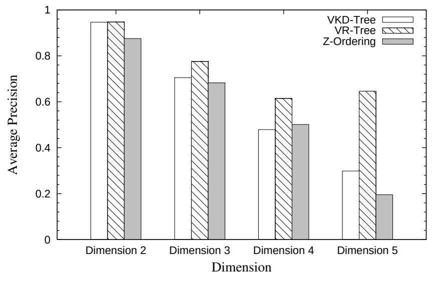
We first vary the number of dimensions from 2 to 5. The results are summarized in Figure 3.5(a). As expected, as the number of dimensions increases, all the schemes lose precision, because more non-answer points must be provided to verify the completeness of the query answers.

We also observe that the VKDtree scheme performs well for two-dimensional space, but its precision drops dramatically at higher dimensions. This is because more partitions are returned as a result of their overlapping the query window. The result for Z-ordering is, surprisingly, similar to the VKDtree scheme. In fact, it even performs better than VKDtree in some cases. Investigation shows that this is because the coverage of the partitions returned under VKDtree may be larger than the region covered by the Z-ordering scheme. Finally, the VRtree scheme achieves precisions of at least 60%, is least affected by dimensionality, and appears to perform the best overall. This is because the data partitioning scheme is able to effectively limit the number of candidate partitions returned in the query answers.

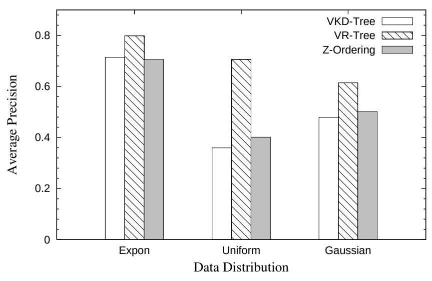
3.4.2 Effect of Different Data Distributions

In the second experiment, we study the effect of different data distributions. Figure 3.5(b) shows the precisions of the various schemes under three different distributions: Exponential, Uniform and Gaussian. The precisions of all the schemes are better with the exponential dataset, because the data generated under the exponential distribution are clustered toward one corner (the origin) of the data space, whereas they are more spread out under the other two distributions.

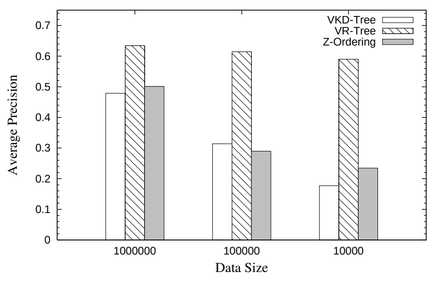
The relative performance of the three schemes remain largely the same as before: with VRtree performing the best, while VKDtree and Z-ordering exhibit similar performance. We also note that VRtree is much more effective than VKDtree and Z-ordering



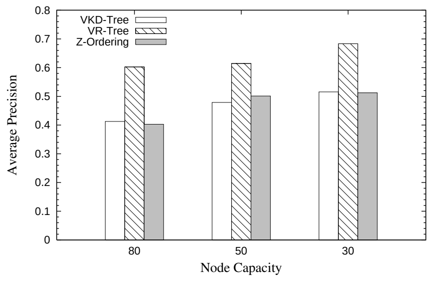
(a) Dimension



(b) Data Distribution



(c) Database Size



(d) Node Capacity

Figure 3.5: Comparative Study

under uniform data distribution.

3.4.3 Effect of Dataset Sizes

With a fixed data space, the size of the dataset will have an effect on the performance of the schemes. In particular, for large datasets, the data space becomes more densely populated. For a fixed-size query, this means that the precision will, with high probability, be higher (compared to one with small dataset size). This intuition is confirmed in our study, as shown in Figure 3.5(c) which presents the results for dataset sizes of 1,000,000, 100,000, and 10,000. The relative performance of the various schemes remain largely the same as in the earlier experiments, though VRtree is less affected by the size of the datasets compared to VKDtree and Z-ordering.

3.4.4 Effect of Node Capacity

In this study, we examine the effect of node capacity, which determines the maximum number of points allowed per partition. Obviously, a larger node capacity means that it is more likely that more non-answer points are returned (compared to a smaller node capacity), thus yielding lower precisions. Figure 3.5(d) shows the results for node capacities of 30, 50 and 80. From the figure, we notice that the precision of all the schemes improve as the node capacity reduces from 80 to 50 and then to 30.

3.4.5 Client Computation Cost

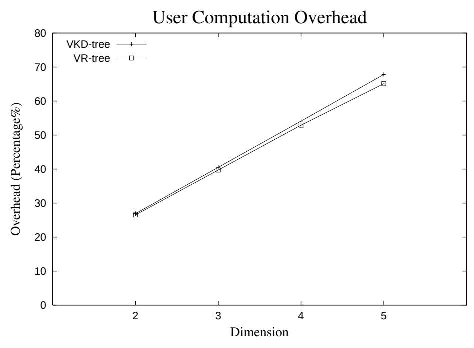


Figure 3.6: Client Computation Cost

In this section, we evaluate the overhead of computation cost at the client side in authenticating the query results. For both VKDtree and VRtree, the client computation cost includes result entry verification cost (C_{RV}), boundary verification cost(C_{BV}) and signature verification cost (C_{SV}). Figure 3.6 shows the authentication overhead of VKD-tree and VR-tree conducted in our experiment, where the overhead is measured as

$$\frac{client\ computation\ cost\ \ - processing\ cost}{processing\ cost}$$

where the processing cost refers to the cost for verifying only answer tuples. It turns out that there is no significant differences between the two schemes - while VRtree incurs lower cost to verify the answers (lower false drops), it incurs additional cost to verify the chaining of partitions; whereas VKDtree does not need to deal with partition chaining but it returns more false drops and hence incur larger cost to verify the answers.

3.5 Summary

In this chapter, we introduce a mechanism for users to verify that their windows query answers on a *multi-dimensional dataset* are correct. The mechanism follows a partition-based strategy, and comprises two steps: (a) verify that all partitions relevant to the query are returned, and (b) verify that all qualifying data points within each relevant partition are returned. The *signature chain* technique from [29] is used to chain up points and partitions so that any malicious omissions can be detected by the user. We study two schemes: Verifiable KD-tree (VKDtree) that is based on space partitioning, and Verifiable R-tree (VRtree) that is based on data partitioning. The schemes are evaluated on window queries, and results show that the VRtree is highly precise, meaning that few data points outside of a query answer are disclosed in the course of proving its correctness.

Chapter 4

Authenticating KNN Query Results

In this chapter, we first introduce the problem definition of authenticating kNN Query results in section 4.1. Section 4.2 describes the method of hiding non-answer points to enforce minimality of Verification Objects. Section 4.3 presents an overview of the query verification scheme. In section 4.4 and 4.5, we present how to handle kNN queries under the native and metric space respectively. Section 4.6 shows results from a performance study. Finally, section 4.7 concludes this chapter.

4.1 Problem Definition

The general setting of our KNN Query authentication problem is as follows. A data owner of a multi-dimensional dataset $\mathcal{DB}$ outsourced the management of $\mathcal{DB}$ to a third-party publisher. Besides $\mathcal{DB}$, (s)he also created one or several associated signatures of $\mathcal{DB}$ that are outsourced together with it. Users are also made aware of certain meta-data, as well as the public key of the owner. During query processing, the publisher returns the answers and the associated *verification objects* (VOs) for the users to verify the correctness of the answers.

Consider the example in previous chapter: a dataset containing 20 data points, r_1 to

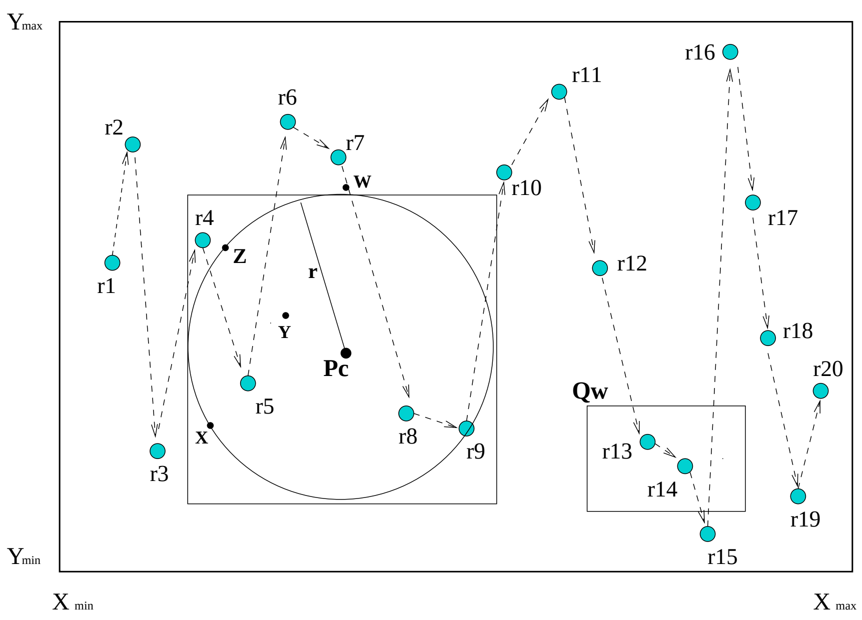


Figure 4.1: Sample Queries on a 2-dimensional Dataset (A Running Example).

r_{20}, in a 2-dimensional space. Figure 4.1 shows a window query Q_w for which $\{r_{13}, r_{14}\}$ is the correct result. A rogue publisher may return a wrong result $\{r_{13}, r_{14}, r_{100}\}$, which includes a spurious point r_{100}, or $\{r_{13^*}, r_{14}\}$ in which some attribute values of r_{13} have been tampered with. To detect such incorrect values, the user should be able to verify the *authenticity* of the query result. A different threat is that the publisher may omit some result points, for example by returning only $\{r13\}$ for query **Q**. This threat relates to the *completeness* of query result.

Similarly, the figure also shows a range query $[p_c, r]$ whose correct answers are $\{r_5, r_8, r_9\}$. Here, an adversary may choose to return $\{r_5, r_9\}$ (i.e., an incomplete answer). As another example, the figure also illustrates a 3NN query (i.e., $k = 3$) centered at p_c. The correct answers for this 3NN query are $\{r_5, r_8, r_9\}$. Now, a compromised publisher may return $\{r_4, r_8, r_9\}$ (i.e., an incorrect answer). Likewise, the RNN of r_{14} is $\{r_{13}, r_{15}\}$, and an adversary may simply return $\{r_{13}\}$ (i.e., an incomplete answer).

As shown in the above examples, there is a need to design mechanisms for users to verify the authenticity and completeness of their query answers. In addition, we aim to design mechanisms that return only the answer points in the plain (and no other data points will be returned in the plain). We refer to this as the *minimality* property. The minimality property is highly desirable as it facilitates confidentiality without violating access control. So, referring to our example, our proposed mechanism will return exactly the answers - $\{r_{13}, r_{14}\}$ for the window query, $\{r_5, r_8, r_9\}$ for the range and 3NN queries, and $\{r_{13}, r_{15}\}$ for RNN(r_{14}) - as well as additional verification objects which will not contain any data points in the plain.

4.2 Enforcing Minimality: Hiding Non-answer Points

In the last chapter, we have examined how points can be signature-chained together. We have shown how the authenticated structure can ensure authenticity and completeness. Authenticity is realized through the signature computation scheme. Completeness is realized by returning a chain of points that contains a superset of the answer points and verifying that they are correct - this is because dropping any point along the chain can be easily detected as it would not lead to correct signatures for the point's neighbors. Before we look at the proposed query verification schemes, let us examine how we can enforce minimality so that all non-answer points that are needed in query verification are not returned in the plain. **We note that we cannot simply return the digests of non-answer points because we do not have a guarantee that the digests correspond to non-answer points.** Referring to our running example in Figure 4.1, for the range query $[p_c, r]$, suppose the adversary returns only r_5 and r_9 in the plain together with the digests for r_3, r_4, r_6, r_7, r_8 and r_{10}. Clearly, we can determine that the chain is correct. However, we cannot be sure that any of these non-answer points are truly non-answer points. In fact, in this example, the adversary has dropped r_8. Thus, we need a scheme that allows us to hide non-answer points while guaranteeing that they are indeed outside of the query region.

Our solution is to associate with each non-answer point p a *reference* point q determined by the publisher which is typically not a data point (unless it so happen that the data point is also in the answer set). With q, the publisher returns $(\tilde{p}, q)$-pairs to the user instead of p, where $\tilde{p}$ is a partial computation of the digest of p. The user can then determine the digest of p from $\tilde{p}$ and q. Moreover, with q, the user can determine that p is outside of the query region. We will discuss this process in the rest of this section.

4.2.1 Collaborative Digest Computation

In our authentication scheme, the signature of a point is dependent on the one-way hash function g (i.e., Equation 3.2) used to compute the digest of a point. We note that g is an iterative hash function that can facilitate the user and publisher to collaboratively determine the digest of a point p. The basic idea is that given a *reference* point q known to both the user and the publisher, the publisher can partially compute the digest of p wrt q and then the user completes the computation wrt q. To illustrate, let a point $p = \{x_1, x_2, ..., x_d\}$ and another point $q = \{y_1, y_2, ..., y_d\}$, such that $x_i < y_i \ \forall i$. Then, instead of returning the digest of p directly, the server can compute $h^{y_i - x_i - 1}(x_i)$ and $h^{x_i - L_i - 1}(x_i)$. The user will then derive $g(p)$ using Equation 3.2 after applying h on $(h^{y_i - x_i - 1}(x_i))$ an additional of $(U_i - y_i)$ times to get $(h^{U_i - x_i - 1}(x_i)) \ \forall i$. Now, similar computation can be derived for different relations between x_i and y_i. Thus, we can determine the digest of p collaboratively without revealing p.

4.2.2 Hiding Non-Answer Points

The combination of signature chain and collaborative computation turns out to provide a very powerful mechanism to hide non-answer points while guaranteeing that they are indeed not in the query regions.

We illustrate this important concept using three examples. In Figure 4.2(a), we have a window query. Here, along a signature chain of 5 points (p_1 to p_5), only p_2 and p_4 are answer points. Let each point p_i be represented as (x_{i1}, x_{i2}). Now, let $X(l_1, l_2)$ and $Y(u_1, u_2)$ be the two bounding points of the window query. Let $L(L_1, L_2)$ and $U(U_1, U_2)$ be the lower and upper bounding points of the entire data space. Note that the user needs the digest of p_1 and p_3 in order to verify that p_2 is authentic. On one hand, we do not want to return p_1 in the plain since that may violate confidentiality. On the other hand, we cannot simply return the digest of p_1. Our collaborative scheme described above hides p_1 by using X as a reference point. Instead of returning p_1 in the plain, the publisher computes $h^{l_1-x_{11}-1}(x_{11})$, $h^{x_{11}-L_1-1}(x_{11})$ and $(h^{U_2-x_{12}-1}(x_{12})|h^{x_{12}-L_2-1}(x_{12}))$. The user will then derive $g(p)$ using Equation 3.2 after applying h on $h^{l_1-x_{11}-1}(x_{11})$ an additional of $(U_1 - l_1)$ times to get $h^{U_1-x_{11}-1}(x_{11})$. Now, X is an appropriate reference point as we actually use its x-dimension value to assure us that p_1 is outside/to-the-left of the query window (i.e,. $x_{11} < l_1$). Similarly, we can hide p_3 and p_5 using Y as the reference point. From the example, we can also see that reference points for window queries are essentially the bounding points of the query.

In Figure 4.2(b), we see how non-answer points can be hidden from a range query (centered at q with radius r). Here, we can use the bounding hyper-cube of the range query to hide points p_1 and p_5 (as described above using the hyper-cube as a window). However, for point p_4, the publisher introduces and returns a reference point $X(x_1, x_2)$ in addition to $h^{U_1-x_{41}-1}(x_{41})$, $h^{x_{41}-x_1-1}(x_{41})$ and $h^{x_2-x_{42}-1}(x_{42})$, $h^{x_{42}-L_2-1}(x_{42})$. The user will then derive $g(p)$ using Equation 3.2 after applying h on $h^{x_{41}-x_1-1}(x_{41})$ an additional of $(x_1 - L_1)$ times to get $h^{x_{41}-L_1-1}(x_{41})$, and applying h on $h^{x_2-x_{42}-1}(x_{42})$ an additional of $(U_2 - x_2)$ times to get $h^{U_2-x_{42}-1}(x_{42})$. More importantly, with X, we know that p_4 is outside of the range query region: from the computation of the digest, we know that

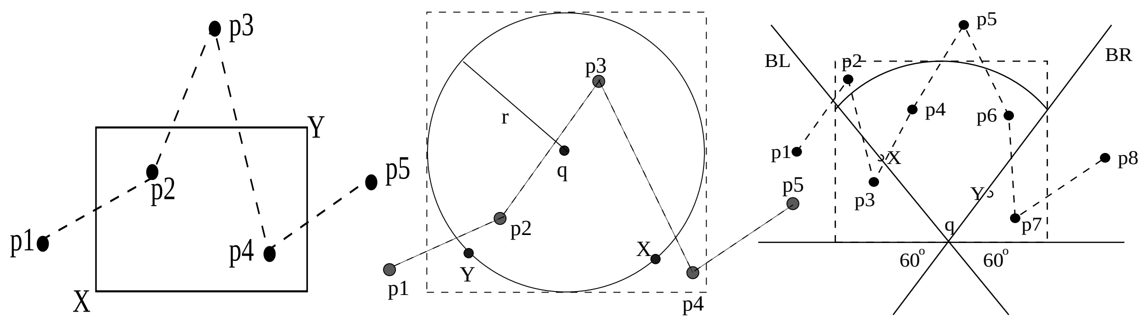
$x_{41} > x_1$ and $x_2 > x_{42}$ (but we do not know the actual values), otherwise the digest will not be defined; therefore, as long as $r \leq dist(X, q)$, we know that p_4 is outside of the query range. In a similar way, reference point Y can be used to hide p_1 (though we have chosen to use the hyper-cube bounding point).

Finally, in Figure 4.2(c), the data space is split into 6 equal regions. A constrained range query centered at q and radius r is one that is restricted to one region (e.g., the region bounded by the two lines BL and BR). As we shall see later, such a query is useful when we process RNN queries. For a constrained range query, certain points can be hidden in a similar way as we handle window queries (e.g., p_1, p_5 and p_8) and range queries (e.g., p_2). For points like p_3 and p_7 it becomes more challenging. However, the same concept of reference points can be used. In our example, for p_3, we can pick a reference point X on the line BL. We note that the user needs to verify that the reference point is on the line BL. (Alternatively, the reference point can be outside of the line BL. In this case, to verify that the point is a valid point that is outside of the line BL, the user can compute the angle between the line formed by q and X, and the horizontal line passing through q, and compare this against that of the angle formed by BL and the horizontal line passing through q.) Now, we can use the collaborative approach for the user to compute the digest of p_3. Using the same logic, a reference point Y can be used to facilitate the collaborative computation of the digest of p_7 without returning p_7 in the plain.

Thus, as we can see, non-answer points can be hidden!

4.3 Query Answer Verification

In this section, we present an overview of the query verification scheme. First, we give the basic solution to verify kNN queries. Then, we generalize the scheme for authenti-



(a) Window query (b) Range query (c) Constrained range query

Figure 4.2: Authentication Overhead on different Dataset Size

cating window, range and RNN queries.

4.3.1 The Basic Solution

Our proposed solution, in its most basic form, ensures authenticity, completeness, and minimality, and works as follows. WLOG, let us consider a kNN query $[p_c, k]$ (see Figure 4.1). Once the publisher computes the k answers, it returns only the k answers in plaintext. In addition, it also returns the following verification objects:

- It returns the k signatures of the answer points. These are used to verify that the data have not been tampered with.

- The k points returned may not fall into a consecutive sequence along the signature chain. For example, in Figure 4.1, there is a gap between r_5 and r_8 (i.e., there are points between r_5 and r_8 which are not answer points). Thus, the publisher will also need to return the partial computation of the digests of a number of points that form a chain. Referring to our example again, we need to return the partial digests of points r_3, r_4, r_6, r_7 and r_{10}. We will defer the discussion on how these points are determined to the later sections. It suffices at this moment to note that we must return r_3 to be certain that there is no point within the hyper-sphere that is chained between r_3 and r_4. The user will then derive the digests of these points to verify the authenticity of the answer points. For example, by computing the digests of r_4 and r_6, we can verify if r_5 is authentic. Similarly, with the digest of r_7, we can verify if r_8 is authentic. Similarly, the digest of r_{10} is needed to verify the authenticity of r_9.

- Now, for the user to verify that the answers are indeed the k answer points, he/she need to show that all other points in the chain are outside of the hyper-sphere centered at P_c with radius r. We note that the $r = dist(P_c, kth\ answer\ point)$. Using our example, the user need to verify that r_3, r_4, r_6, r_7 and r_{10} are outside of the hyper-sphere. To do this, the publisher also returns a set of reference points. Let the number of non-answer points returned be M. Then, the number of reference points needed is (at most) M, one for each of the non-answer points. These reference points are points in the space but not from the dataset. Moreover, they are points on or outside of the hyper-sphere surface so that the distance between these points and P_c is larger than or equal to r, but shorter than the distance between their corresponding non-answer points and P_c. Note that the publisher can easily determine these points since it knows all the points in the dataset. Using our running example again, r_3 has a reference point X, r_4 has a reference point Z, and r_6 and r_7 have the same reference point W. For each (non-answer point, reference point) pair, the partial digest of the non-answer point is computed by the publisher (as described earlier), and the user can complete the computation and derive the actual digest of the non-answer point. As long as the digest is valid, the user will know that the non-answer point is outside of the hyper-sphere (since it knows that the distance between P_c and the reference point is larger than the radius of the hyper-sphere). We will discuss how the reference points are selected in subsequent sections (since not any arbitrary reference point works). In addition, we note that we can optimize the number of reference points returned since it is possible that a number of non-answer points can use the same reference point. Referring to our example, one reference point W can be used for both points r_6 and r_7.

Taking our running example again, the query answer for this 3NN query Q is $\{r_5, r_8, r_9\}$. Besides the plaintext for these+ 3 answers, the publisher also returns the following verification objects:

- Signatures of the 3 answer points, which are $sig(r_5)$, $sig(r_8)$ and $sig(r_9)$.
- For the two boundary points r_3 and r_{10} of the answer's signature chain returned, the publisher returns two pairs $(\tilde{r_3}, B_1)$ and $(\tilde{r_{10}}, B_2)$, where $\tilde{r_3}$ and $\tilde{r_{10}}$ are the partial computation of the digests of r_3 and r_{10} respectively. Points B_1 and B_2 are the leftmost and rightmost point of the hyper-sphere query respectively, where $B_1.x = P_c.x - dist(P_c, r_9)$ and $B_2.x = P_c.x + dist(P_c, r_9)$.
- For points r_4, r_6, and r_7 that fall into the gap of the answer points along the consecutive signature chain sequence, the publisher returns pairs $(\tilde{r_4}, Z)$, $(\tilde{r_6}, W)$, and $(\tilde{r_7}, W)$ respectively, where $\tilde{r_i}$ is the partial digest of point r_i, Z and W are the corresponding reference points selected for each r_i.

Clearly, the proposed method is minimal since only the k answer points are returned in the plain!

4.3.2 Generalizing to Other Query Types

The above scheme can be easily generalized to handle window and range queries. We also describe how it can authenticate the more complicated reverse NN queries.

Window Query

For window query [p_l, p_u], all objects outside of the window can use either one of these two bounding points as a reference point (recall the discussion in Section 4.2). For example, consider the window query (hyper-cube centered at P_c) in Figure 4.1. Now, r_3, r_6, r_7, and r_{10} are not part of the answer points that need to be returned. For r_3, we can see that the x_1 value of p_l would suggest r_3 is outside of the window. Similarly, the x_2 value of p_u would suggest that r_6, r_7 and r_{10} are outside the window. Thus, for window queries, as we have described in chapter 3. the query's bounding points themselves provide the reference points. Which means there is no need for the publisher to provide any reference points.

Range Query

A range query $[P_c, r]$ can be easily handled in the same way as a kNN query - it needs to verify that the answer points are in the hyper-sphere centered at P_c with radius r, and that all points outside of the hyper-sphere are indeed outside (as is done in the verification for kNN query).

Reverse NN Queries

In [17], a two phase algorithm is proposed to retrieve the RNN of a query point q in a 2-dimensional data space. In the first phase, the data space around the query point q is divided into six equal regions S_1 to S_6. For each region S_i ($1 \leq i \leq 6$), a constrained NN query is processed to retrieve the nearest neighbors of q in that region. Let the point for S_i be p_i. It turns out that these six points constitute the candidate result set. In other words, either $p_i \in \ RNN(q)$ or (ii) there is no RNN of q in S_i. Thus, in the second phase, a NN query is applied to find the NN of each candidate p_i. We denote the NN of p_i as p'_i. If $dist(p_i, q) < dist(p_i, p'_i)$, then p_i belongs to the actual result; otherwise, it is

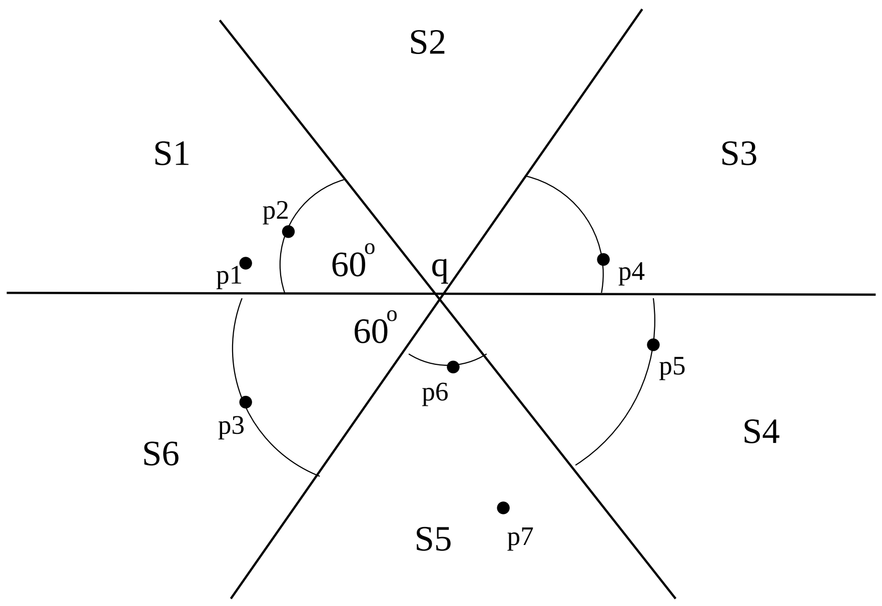


Figure 4.3: Illustration of the two-phase RNN algorithm in [17].

a false hit and discarded.

As an example, consider Figure 4.3 which divides the 2-dimensional space around a query point q into six equal regions S_1 to S_6. In Figure 4.3, the NN of q in S_1 is point p_2. However, the NN of p_2 is p_1. Consequently, there is no RNN of q in S_1 and we do not need to search further in this region. The same is true for S_2 (no data points), S_3, S_4 $(p_4, p_5$ are NNs of each other) and S_6 (the NN of p_3 is p_1). There is only one answer for RNN(q) which is p_6 in region S_5.

Now, since both phases of the above scheme consists of a series of NN queries, we can adapt our kNN authentication scheme here. The authentication scheme comprises two cases: (a) The point p_i in region S_i is indeed the RNN of q; and (b) The point p_i in region S_i is not the RNN of q. **Case (b) is much more challenging because we need to hide p_i as well as its NN in order to show that its NN is not q.** We present our solution to these two cases below.

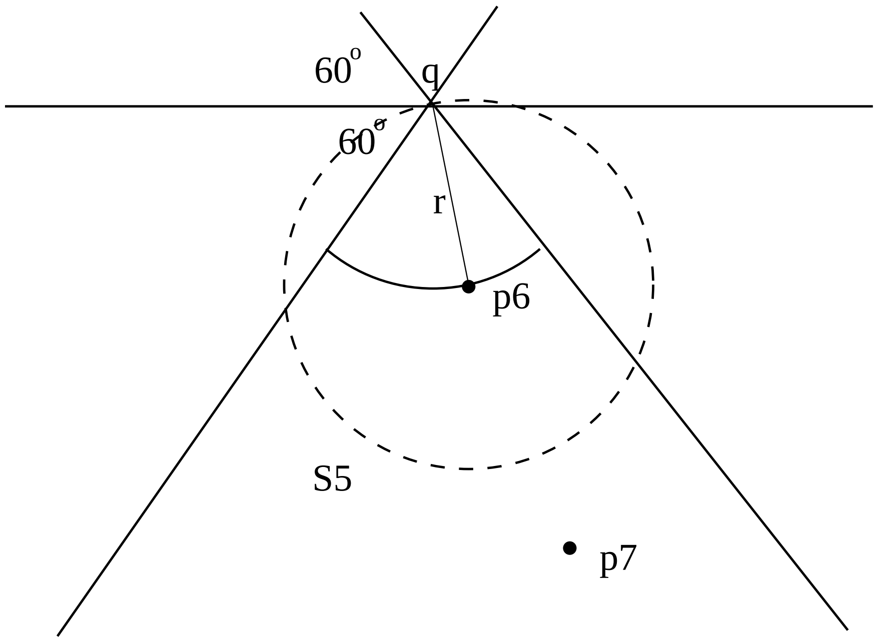


Figure 4.4: Authentication of RNN point (Case (a))

Case (a): p_i in region S_i is the RNN of q

When the publisher returns p_i in region S_i as the answer (in the plain), the user need to do the following to verify that it is indeed an answer (we also describe the verification objects that the publisher need to return):

- Verify that p_i is the NN of q. To do this, the publisher returns the results of the constrained range query with q as the center and $r = dist(p_i, q)$ as the radius. A constrained range query refers to the query being bounded by the splitting plane of the region (as discussed in Section 4.2). We note that the results consist of p_i, the partial digests of points that are along the signature chain, and the associated reference points. As shown in Section 4.2, we can then verify if p_i is indeed the only point, and if so, it is the NN of q. Otherwise, we know that the publisher has cheated.

- Verify that q is the NN of p_i. To do this, the publisher returns the results of a range query centered at p_i with radius r (together with the associated signature chain, and reference points). Clearly, as long as there is no answer point for this query (q is a query point), we know that q is the NN of p_i. We can thus conclude that p_i is a RNN of q.

Figure 4.4 illustrates an example. Here, region S_5 has two points p_6 and p_7. Since p_6 is the answer, it will be returned in the plain. The first constrained range query centered at q with radius $r = dist(q, p_6)$ would allow us to know that p_6 is indeed the NN of q. The second range query centered at p_6 with radius r would confirm that no points are within this query region, and hence p_6 is the correct answer. From the figure, it is clear that p_7 is further away to p_6 than q.

Case (b): p_i in region S_i is not the RNN of q

In this case, since p_i is not an RNN of q, we cannot return p_i in the plain. However, we need to (1) verify that p_i is an NN of q, and (2) verify that there exists another point t such that $dist(p_i, t) < dist(p_i, q)$. Note that these have to be done without revealing p_i and t.

Our approach works as follows:

- We note that to verify that a point (without revealing it in the plain) is in a query region, we need two reference points. For example, consider Figure 4.2(a), to verify that p_2 is in the window query, we basically need to say that p_2 is on the right of and above X as well as on the left of and below Y. Clearly, with only one of X or Y, we would not be able to guarantee that p_2 is in the window query. Thus, the publisher returns two reference points X and Y such that: (a) $r_l = dist(q, X) < r_u = dist(q, Y)$, (b) p_i is the only answer of a constrained range query centered at q with radius r_u, (c) there are no answer points of a constrained range query centered at q with radius r_l. Now, since the user knows X and Y,

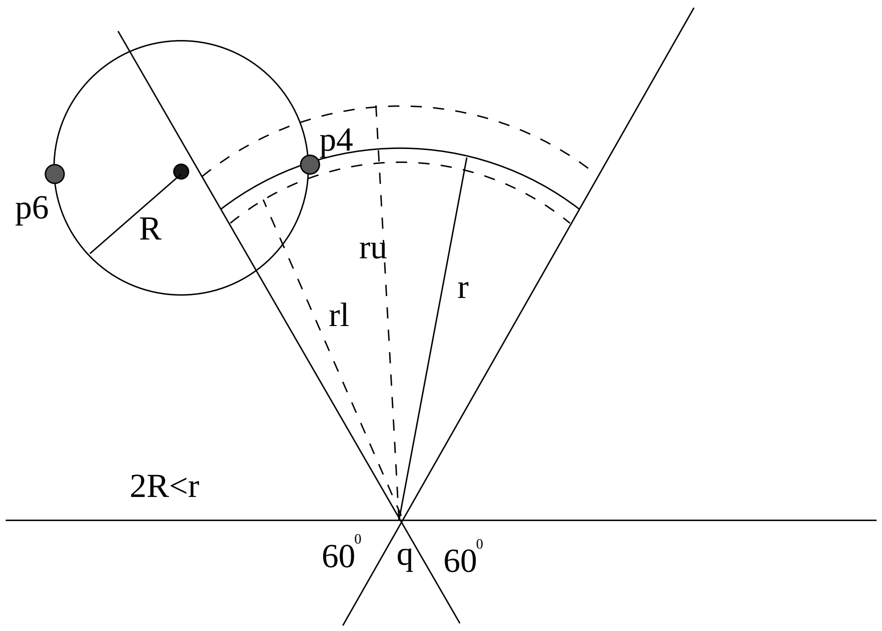


Figure 4.5: Authentication of RNN point (Case (b))

he/she can easily verify that no points are in the constrained range query $[q, r_l]$ and only p_i is in $[q, r_u]$.

- Verify that $dist(p_i, t) < r_l$. To do this, the publisher need to also return a reference point v such that a range query centered at v with radius, R, contains p_i and t. We note that $R \geq dist(p_i, t)/2$. As long as $dist(p_i, t) < 2R < r_l$, we are done. Thus, the publisher need to choose 4 additional reference points - 2 to verify that p_i is in the range query, and 2 to verify that t is in the range query. With the four reference points, the publisher is able to determine R.

Figure 4.5 illustrates an example. Here, the restricted region has one point p_4 which is the NN of q. However, p_4's NN is p_6. We first find reference points and r_l and r_u that allow us to verify the boundaries of p_4. Similary, we can find reference points to allow us to find the boundaries that encloses p_6 and p_4. As shown, in this figure, $2R < r_l < r$ and hence we can determine that p_4 is not an RNN(q) without revealing p_4 and p_6.

4.4 kNN Authentication in Native Space

In this section, we present our solution for authenticating kNN queries in the native space. With the VR-tree, the verification process involves two steps: a) we need to verify that none of the valid partitions have been missed; b) for partitions that should be checked, none of the valid points have been missed.

Verifying that the query answer covers all the candidate partitions is straightforward for a known hyper-sphere. It comprises the following two phases:

- In the first phase, we need to identify the list of candidate partitions. A partition is a candidate if the range of the ordering dimension of its MBR overlaps the range of the ordering dimension of the tightest hyper-cube that bounds the hyper-sphere. In our example, only partitions R1, R2 and R3 are candidate partitions.

- In the second phase, some of these candidate partitions can be further pruned. The ones to be pruned are those whose minimum distance to P_c is larger than the hyper-sphere radius r (since no points inside these partitions will ever be nearer than the kth NN answer point. This can be easily verified since the bounding points of each partition are known. In our example, R1 and R3 are further pruned.

For each remaining candidate partition P, there are 3 possible relationships between P and the hyper-sphere (denoted $H(P_c, r)$) centered at P_c with radius r.

1. $H(P_c, r)$ contains P. In this case, we return all the points in P, i.e., the publisher returns p_0 to p_{n+1} and n, together with the respective signatures $sig(P_0)$ to $sig(P_{n+1})$ and $sig(P)$. The user would compute the digests for both the points and the partition to verify the result.

2. P contains $H(P_c, r)$. Let $P_i = (x_{i1}, x_{i2}, ..., x_{id})$, and $P_c = (o_1, o_2, ..., o_d)$. Let $H'(P_c, r)$ be the most tightly bounded hyper-cube of $H(P_c, r)$, thus $H'(P_c, r)$ is

also centered at point P_c, and the length of each edge $l = 2r$. Let $H'(P_c, r)$'s bounding points be $(h_{l1}, h_{l2}, ..., h_{ld})$ and $(h_{u1}, h_{u2}, ..., h_{ud})$. Thus, $h_{ui} = o_i + r$ and $h_{li} = o_i - r$ for all $i \in [1, d]$. The data points in P can be separated into

(a) $p_\alpha, p_{\alpha+1}, ..., p_{\beta-1}, p_\beta$, such that $x_{i1} \in [h_{l1}, h_{u1}]$ for $\alpha \leq i \leq \beta$.

These points can be further categorized into answer points $(\mathcal{A})$ and false positives $(\mathcal{F})$. For each answer point $p_i \in \mathcal{A}$, $dist(P_c, P_i) \leq r$, and for each false positive $p_i \in \mathcal{F}$, $dist(P_c, P_i) > r$. Furthermore, there are two types of false positive points. In the first type, denoted $\mathcal{F}_a$, for each $p_i \in \mathcal{F}_a$, $\exists z, x_{iz} \notin [h_{lz}, h_{uz}]$. In the second type, denoted $\mathcal{F}_b$, for each $p_i \in \mathcal{F}_b$, $\forall z, x_{iz} \in [h_{lz}, h_{uz}]$. Note that $\mathcal{F}_a$ corresponds to points outside the hyper-cube, while $\mathcal{F}_b$ are points inside the hyper-cube but outside the hyper-sphere. Let us use the data space in Figure 4.1 as an example of a partition containing the hyper-sphere. Here, we have $\mathcal{A} = \{r_5, r_8, r_9\}$, $\mathcal{F}_a = \{r_6, r_7\}$ and $\mathcal{F}_b = \{r_4\}$.

(b) $p_1, ...p_{\alpha-1}, p_{\beta+1}, ...p_k$, which are clearly not answer points. Referring to Figure 4.1, these points are r_1 to r_3 and r_{10} to r_{20}.

For data points from different categories, the publisher returns different sets of verification objects.

(a) For each point $p_i \in \mathcal{A}$, the publisher returns p_i and $sig(p_i)$.

(b) The publisher also returns p_0, p_{n+1}, $sig(p_0)$ and $sig(p_{n+1})$, and $sig(P)$.

(c) For each point $p_i \in \mathcal{F}_a \cup \mathcal{F}_b \cup \{p_{\alpha-1}, p_{\beta+1}\}$, the publisher finds a reference point $S = (S_1, S_2, ..., S_d)$ on the surface of the hyper-sphere[1], such that, if $x_{iz} < o_z$, $S_z \in (x_{iz}, o_z)$, else if $x_{iz} > o_z$, $S_z \in (o_z, x_{iz})$.

[1]We do not require the point to be on the surface. All that is needed is to find a point that is outside of the hypersphere that is closer to the query point than the point to be hidden. However, for ease of presentation, we shall refer to the reference point as a point on the surface.

We note that the same S point could be used as a reference point for multiple p_is as long as the above conditions hold. For simplicity, we pick the point closest to the sphere's surface on the line joining P_c and p_i. Among these points, we then eliminate "redundant" reference points.

After an S point is chosen for each $p_i \in \mathcal{F}_b$, we could simply verify that $dist(P_c, p_i) > dist(P_c, S) \geq r$.

The publisher then returns several pieces of information together with the detailed information of point S:

i. if $x_{iz} < S_z$, $h^{S_z - x_{iz} - 1}(x_{iz})$ and $h^{x_{iz} - L_z - 1}(x_{iz})$ are returned.

ii. if $x_{iz} > S_z$, $h^{U_z - x_{iz} - 1}(x_{iz})$ and $h^{x_{iz} - S_z - 1}(x_{iz})$ are returned.

With the above information, the user can compute $g(p_i)$ without knowing the actual value of p_i.

- if $x_{iz} < S_z$, the user applies h on $h^{S_z - x_{iz} - 1}(x_{iz})$ an additional $(U_z - S_z)$ times to get $h^{U_z - x_{iz} - 1}(x_{iz})$.
- if $x_{iz} > S_z$, the user applies h on $h^{x_{iz} - S_z - 1}(x_{iz})$ an additional $(S_z - L_z)$ times to get $h^{x_{iz} - L_z - 1}(x_{iz})$.
- The user computes $g(p_i)$ using Equation 3.2.

Consider Figure 4.1 again as our example where P contains $H(P_c, r)$. We could see that the point r_7 is outside the hyper-cube, which means that r_7 is not an answer. Instead of just returning the value of r_7, the publisher picks a reference point W near the circle, where $W.x > r_7.x$ and $W.y < r_7.y$. Then (part of the information) the server returns: for query answers $\{r_8, r_9\}$, it returns r_8, r_9, $sig(r_8)$, and $sig(r_9)$; for r_7, it returns (1) $h^{W.x - r_7.x - 1}(r_7.x)$ and $h^{r_7.x - L.x - 1}(r_7.x)$;(2) $h^{U.y - r_7.y - 1}(r_7.y)$

and $h^{r_7.y-W.y-1}(r_7.y)$. Here, L and U denote the two bounding points of the partition. With these, the user can determine $h^{U.x-r_7.x-1}(r_7.x)$ and $h^{r_7.y-L.y-1}(r_7.y)$, and compute the digest of r_7. (S)he can then further verify that r_8 is an answer point.

3. P overlaps $H(P_c, r)$. This case can be handled by splitting P into two parts: one overlaps $H'(P_c, r)$ (the hyper-cube of $H(P_c, r)$), and the other does not overlap $H'(P_c, r)$ (which means it does not overlap $H(P_c, r)$). For the first part, we handle it in the same manner as case (2) above. For the second part, it can be dropped (except to verify that its points are outside $H'(P_c, r)$). As such, we shall not go into the details of this case.

In the above discussion, we have assumed only one layer of partitioning. We can easily extend the scheme to work with the VR-tree. All that is needed is to verify that no internal nodes are tampered with and dropped unnecessarily. This can be done as described above since the internal nodes are also signature chained.

4.5 kNN Authentication in Metric Space: iDistance Based Scheme

In Section 4.4, we have looked at how to authenticate kNN queries in the native data space. In this section, we shall look at the problem when points are stored in the metric space. Many data structures have been designed for processing kNN queries in metric space. We shall discuss the method that is based on the iDistance [41] scheme here.

iDistance is an efficient technique for kNN search that can be adapted to different data distributions. In iDistance, the data space is partitioned according to a set of reference points. By indexing the distance of each data point to the reference point of its partition,

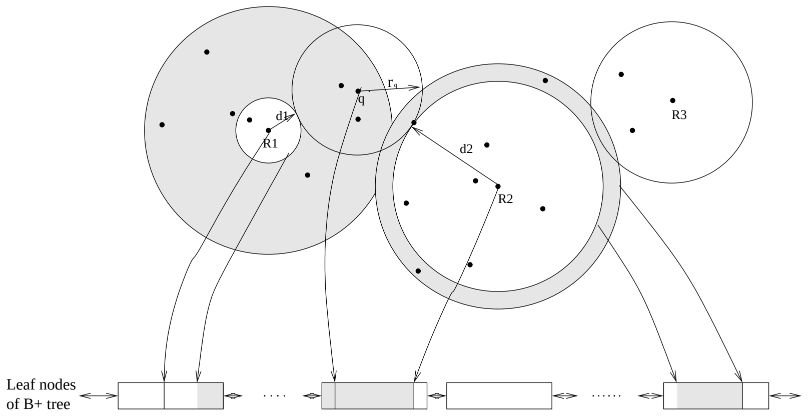


Figure 4.6: iDistance based scheme

high-dimensional points are transformed into points in a single dimensional space and indexed by a classical B+-tree. In particular, points in a partition are mapped into a range of values in the single dimensional space such that no two partitions have overlapping ranges. Thus, all points in partition P_i is located to the left side of points in partition P_{i+1} in the B+-tree.[2]

Within the same partition, data points are ordered by their distance from the data point to its reference point. Referring to Figure 4.6, we have 3 partitions formed by 3 reference points R1, R2 and R3 respectively. A range query with center at q and radius r will need to access data points in the shaded region shown in the figure.

In iDistance data structure, data partitioning is independent of the spatial location of the data points but only related to the selection of reference points. Moreover, the shape of partitions in iDistance structure is a hyper-sphere that is centered at its reference point O_j with radius $r_{P_j} = max(dist(r_i, O_j))$. Let a hyper-sphere query be centered at Q with radius r_q. Partition P_j does not overlap with the query and can be pruned from further consideration if the following holds:

$$dist(Q, O_j) \geq r_{P_j} + r_q \tag{4.1}$$

[2]We note that the original iDistance scheme did not discuss how partitions are ordered. Here, we adopt a simple strategy that orders the partition based on the values of the first dimension of the reference point.

On the other hand, if $dist(Q, O_j) < r_{P_j} + r_q$, we have to return the detailed information to show that all the query results contained in this partition are returned correctly. Now, as reported in [41], the set of points that need to be examined are bounded by the following inequality

$$dist(Q, O_j) - r_q \leq dist(O_j, r_i) \leq dis(Q, Q_j) + r_q \tag{4.2}$$

In the authentication model, we build up the signature chain directly on top of the B+-tree. Let $O_j = (O_{j1}, O_{j2}, \ldots, O_{jd})$ be the reference point for partition P_j. The signature of each data point r_i is

$$sig(r_i) = s(g(r_{i-1})|g(r_i)|g(r_{i+1})) \tag{4.3}$$

where $g(r_i) = h(h(r_i)|h(dist(r_i, O_j))$. Moreover, for each partition P_j,

$$sig(P_j) = s(h(O_j)|h(max(dist(r_i, O_j)))|h(k)) \tag{4.4}$$

where $h(O_j) = h(h(O_{j1})|h(O_{j2})| \ldots |h(O_{jd}))$ and k is the number of data points contained in partition P_j.

Similar to the R-tree based scheme, authentication of kNN queries for the iDistance based scheme contains the following two steps:(a) Verify that no overlapped partitions is missing; (b) Verify that no result points inside the overlapped partition is tampered or dropped.

To verify that all overlapped partitions are returned, the publisher need to return the following information to the client:

- For each partition P_j, return O_j, r_{P_j}, k and $sig(P_j)$. With these information, the client can verify that the partition information has not been tampered with. Moreover, the client can safely prune away partitions that satisfy Equation 4.1 from further verification.

Here, we assume that the client knows the number of partitions; otherwise, additional information has to be provided (e.g., the signature for the total number of partitions, and the number of partitions). We note that this phase can be optimized by chaining the partitions to minimize the amount of information to be sent to the client. This is similar to the process of verifying partitions in the R-tree based scheme.

Now, for each partition P_j that overlaps the query hyper-sphere, we need to verify that no points has been tampered or dropped. The publisher returns the following information to facilitate verification:

- The continuous sequence of signature chain within P_j that satisfy Equation 4.2. Since the signatures are ordered by the distance to the reference point, those points matching the inequality would form a continuous signature chain and should be returned to the user as verification objects. Since not all points with the same distance are answer points, this chain of points contain both answer points $\mathcal{A}$ and false positives $\mathcal{F}$. For each point $p_i \in \mathcal{A}$, the publisher returns p_i and $sig(p_i)$. For each point $p_j \in \mathcal{F}$, the publisher returns a reference point $S = (S_1, S_2, \ldots, S_d)$ on the hyper-sphere (in the native space) as well as the corresponding (partial) digest. As in the R-tree based scheme, different false positive points could share a same reference point S as long as the following condition holds: $if\ r_{iz} < Q_z, S_z \in (r_{iz}, O_z); else\ S_z \in (O_z, r_{iz}), 1 \leq z \leq d.$

- The publisher also returns the (partial) digests of the two points bounding the continuous sequence of signature chain above. Essentially, these two points allow the client to verify that no other points within the partition has been dropped. Each of these points is also associated with a reference point.

We note that the verification process is done in the native space. Once the client receives all the verification objects, it operates in the native space in the same manner as that described in the R-tree based scheme. In other words, with the k answer points, it can determine the hyper-sphere query and hyper-cube query. For each of the non-answer points, the client uses its associated reference point to verify that it lies outside the hyper-sphere.

4.6 Performance Study

We have implemented the proposed solution for verifying kNN queries and conducted a series of experiments to study their performance. For our VR-tree, we implemented the R*-tree data structure [8]. In [12], we also presented a metric-based scheme using the B^+-tree based iDistance structure [41]. The codes for both mechanisms are implemented in C++. The performance metrics used in our study is the authentication overhead introduced and the I/O access cost. The authentication overhead is computed as $the\ number\ of\ overhead\ points/k$, where the number of overhead points refer to the number of non-answer points returned.

Unless stated otherwise, we use the following default parameter settings. The number of dimensions is 4. The data distribution is Gaussian, the number of data points is 100K, the domain of each dimension is [0, 1M]. The node capacity is 30 (i.e., each node holds up to 30 data points). Queries are generated by randomly picking a point from the database, and the value of k for the kNN query is 10. For each experiment, we vary one of the above parameters, run 200 queries, and take the average score.

4.6.1 Effect of Number of Dimensions

We first vary the number of dimensions from 2 to 32. Figure 4.7 summarizes the result. As expected, a higher dimensionality introduces more overhead for both mechanisms adopted, as more non-answer points are required to verify the completeness of the query.

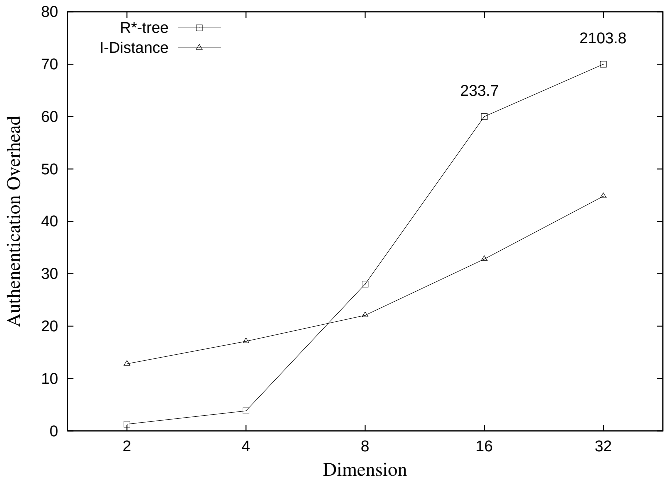


Figure 4.7: Authentication Overhead on Different Data Dimension

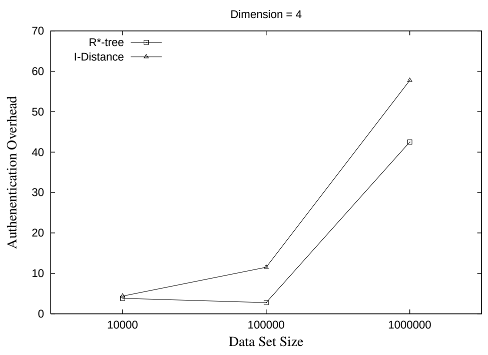
Moreover, as the number of dimensions increases, the data space "expands" correspondingly; with a fixed dataset size, the data points for higher dimensional dataset are spread more sparsely. Thus, given kNN queries with the same k value, the radius of the corresponding hyper-sphere in a higher dimensional dataset is much larger than its radius in a lower dimensional dataset.

Another observation is that for small number of dimensions, the R*-tree based mechanism yields lower authentication overhead. However, the iDistance based mechanism is superior when the number of dimensions is higher. This is reasonable as R*-tree has its own structural restriction when the dimensionality is high.

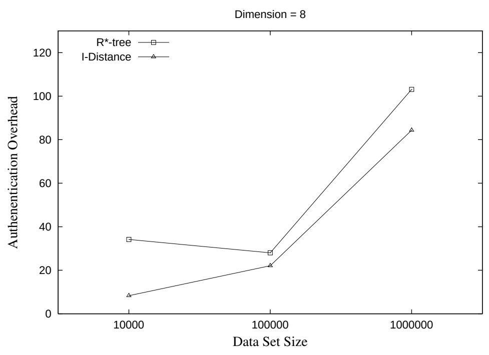
4.6.2 Effect of Different Dataset Size

In our second experiment, we study the effect of different dataset size for a fixed data space. Figure 4.8 shows the authentication overhead of the two schemes under different dataset size.

From the result, we observe that as the dataset size increases, the authentication overhead for iDistance based method increases as well. However, for the R*-tree based



(a) d=4



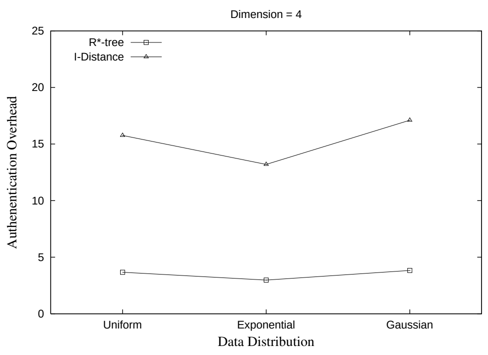
(b) d=8

Figure 4.8: Authentication Overhead on different Dataset Size

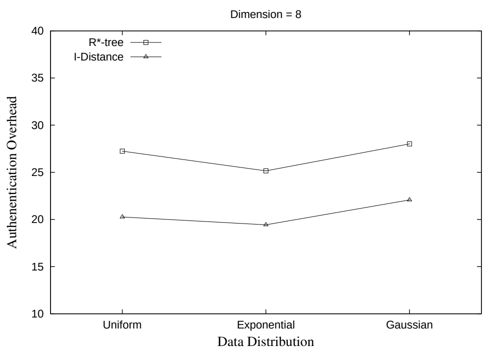
mechanism, the overhead decreases initially. Our investigation suggests the following reasons - the increasing dataset size reduces the size of the kNN query, which actually reduces the radius of its corresponding hyper-sphere. The R*-tree based method is more sensitive to this kind of reduction because of the overlaps in the MBR of its internal nodes in the structure. However, as the dataset size increases further, given the fixed data space, the space becomes too dense, resulting in larger overhead.

4.6.3 Effect of Different Data Distributions

In this experiment, we study the effect of different data distributions. As shown in figure 4.9, the results are measured under three different distribution: Exponential, Uniform and Gaussian. We note that both methods incur lesser overheads with the exponential dataset. This is because the data generated under the exponential distribution are clustered toward one corner (the origin) of the data space, whereas they are more spread out under the other two distributions. Moreover, the relative performance of the two methods remains the same for different data distributions. This result is also consistent with the findings in [11] for multi-dimensional window queries.



(a) d=4



(b) d=8

Figure 4.9: Authentication Overhead on different Data Distribution

4.6.4 I/O Access Cost

Figure 4.10 shows the I/O access cost for the two mechanisms at the server. We see that the R*-tree based method outperforms the iDistance based method when the number of dimensions is small, while it incurs more I/O cost when the number of dimensions is large. This is consistent with previous works since the R*-tree method degenerates in performance as the number of dimensions increases.

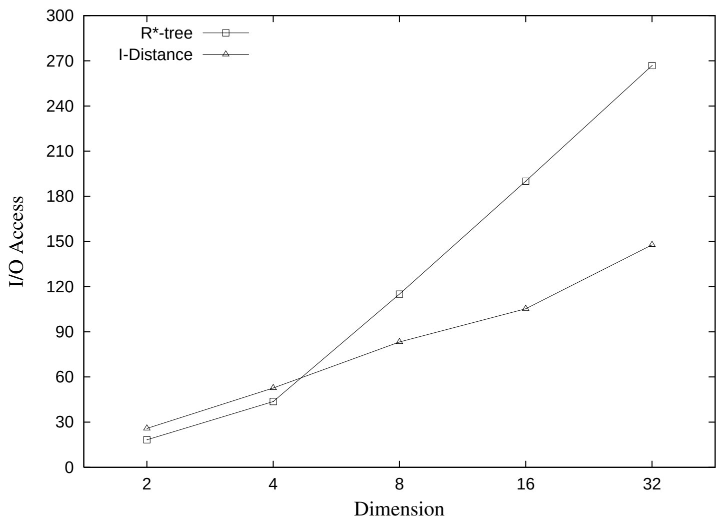


Figure 4.10: I/O Access Cost

4.7 Summary

In this chapter, we have introduced a solution for users to verify their answers when they query a multi-dimensional dataset. In particular, our scheme supports a wide range of query types, namely window, range, kNN and RNN queries. Our solution extends the signature chain scheme for multi-dimensional dataset. In this way, we can achieve authenticity and completeness. Moreover, our scheme introduces a positional reference point P for each non-answer point examined. This enables the scheme to achieve the minimality property. We have implemented the scheme for kNN queries. Our experimental study showed that the proposed method is effective and incurs low overhead.

Chapter 5

Conclusion and Future Work

5.1 Conclusion

In data outsourcing model, data owners engage third-party data servers (called publishers) to manage their data and process queries on their behalf. As these publishers may be untrusted or susceptible to attacks, it could produce incorrect query results to users.

In this thesis, we examined the issues of Multi-Dimensional Query results Authentication in Data Publishing. We first introduced a mechanism for users to verify that their query answers on a *multi-dimensional dataset* are correct, in the sense of being complete (i.e., no qualifying data points are omitted) and authentic (i.e., all the result values originated from the owner). Our approach is to add authentication information into a spatial data structure, by constructing certified chains on the points within each partition, as well as on all the partitions in the data space. Given a query, we generated proof that every data point within those intervals of the certified chains that overlap the query window either is returned as a result value, or fails to meet some query condition. We studied two instantiations of the approach: Verifiable KD-tree (VKDtree) that is based on space partitioning, and Verifiable R-tree (VRtree) that is based on data partitioning.

The schemes are evaluated on window queries, and results show that VRtree is highly precise, meaning that few data points outside of a query result are disclosed in the course of proving its correctness.

As an extension, we examined the authentication of kNN query results in Multi-dimensional database, we introduce an authentication scheme for outsourced multi-dimensional databases. With the proposed scheme, users can verify that their query answers from a publisher are complete (i.e., no qualifying tuples are omitted) and authentic (i.e., all the result values are legitimate). In addition, our scheme guaranteed minimality (i.e. no non-answer points are returned in the plain). This scheme supports window, range, kNN and RNN queries on multi-dimensional databases. We have implemented the proposed scheme, and our experimental results on kNN queries show that our approach is a practical scheme with low overhead.

5.2 Future Work

5.2.1 Trust-Preserving Set Operations

Trust-Preserving Set Operation Problem is proposed by Ruggero et.al.in paper [24]. In this problem, the party performing the computation does not need to be trusted, but the result is a set which is trusted to the same extent as the original input. The techniques have a range of potential applications such as addressing the problem of securely reusing content-based search results in peer-to-peer (P2P) networks.

Given an example model with two trusted source nodes, s_1, s_2, each store an index in the form of S_1, S_2; an untrusted directory d; and a client c, standard set operation (such as union, difference, and intersection) are performed with problem raised on how to construct a scheme that allows c to verify that d didn't falsify the result of the query.

Current solution of this problem is accomplished by requiring trusted nodes to sign appropriates, defined digest of generated sets, and each such digest consists of an RSA accumulator and a Bloom filter. Two kinds of attacks might be performed: insertion attack and deletion attack. Current solution based on counting bloom filters compares the bloom filter, which is obtained as the element-by-element minimum of $Bl(S_1)$, $Bl(S_2)$, with the bloom filter $Bl(I')$ of the returned intersection to detect the insertion attacks. And the scheme also requires the directory to justify each gap (an index j is called a gap if $Bl(I)_j$ is strictly less than Bl_j) to make sure there is no deletion attack.

However, this solution with a simple compressed counting bloom filter would suffer from several limitations: The attacks such as insert an outside element into the intersection, although it can be solved at the cost of Bloom filters with a prohibitively large number of counters. Moreover, this simple scheme also suffers from the heavy load of the Bloom filter.

How to derive a simple and efficient scheme with lower overhead for the this set-operation scheme is an interesting and meaningful problem for us to investigate.

5.2.2 Authenticating Aggregation Queries in Outsourced Database Systems

Current wok on query authentication has focused on studying the general selection and projection queries. Another important aspect of query authentication in outsourced database system that has not been considered yet is handling aggregation queries.

When processing an aggregation query, although intermediate data might be involved during the computation, only result answers need to be returned. However, in a Third-party Publisher System, it would be infeasible for the user to authenticate the returned answer from publisher without the knowledge of the detailed data. In this case, we address the scenario where a user has the rights to know (at least some of) the detailed data underlying the aggregation it is given.

The most straight forward solution is, along with the aggregation result returned, the publisher sends all the answer-related detailed data to user. The user could first verify the returned data with authentication techniques such as Merkle Hash Tree [15] or Signature Chain [29] Methods, and then compute the result and verify authenticity its own. However, with this method, a ”sum” query might require the publisher returns all the values to the user, in this case this trivial solution is very inefficient. There are several drawbacks:

- Communication Cost: the communication between the publisher and the user might be expensive.
- Network Traffic: network traffic might be caused during data transmission especially when such large amount of data transferred.
- Access Control: Sometimes, the user might not be encouraged to know the detailed data of an aggregation query.
- Computation Workload: The user's workload might be too heavy when complicate calculations required.

As stated previously, communication just the result of a query is in many cases very efficient, but it does not give the guarantee of correctness.(example of random sampling) Thus it is a tradeoff between the query processing efficiency and result accuracy. In term of result authentication, we cannot do better than send all the detailed data related to the aggregation query to the user, which might be very inefficient in practice. We may set our goal of this problem is to reduce the communication cost between the user and publisher as well as achieve high accuracy of aggregation result.

Bibliography

[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] N. Beckmann, H. Kriegel, R. Schneider, and B. Seeger. The r*-tree: An efficient and robust access method for points and rectangles. In *SIGMOD Conference*, pages 322–331, 1990.

[9] J. Bentley. Multidimensional Binary Search Trees Used For Associative Searching. *Communications of the ACM*, 18(9):509–517, September 1975.

[10] 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.

[11] W. Cheng, H. Pang, and K. Tan. Authenticating multi-dimensional query results in data publishing. In *Proceedings of the 20th Annual IFIP WG 11.3 Working Conference on Data and Applications Security (DBSec'2006)*, pages 60–73, 2006.

[12] W. Cheng and K. Tan. Authenticating knn query results in data publishing. In *Proceedings of the 4th International Workshop on Secure Data Management (SDM'07)*, pages 47–63, 2007.

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

[14] P. Devanbu, M. Gertz, A. Kwong, C. Martel, G. Nuckolls, and S. Stubblebine. Flexible authentication of xml documents. In *Proceeding of the 8th ACM Conference on Computer and Commnunication Security(CCS-8)*, pages 136–145, 2001.

[15] 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.

[16] P. Devanbu, M. Gertz, C. Martel, and S. Stubblebine. Authentic Data Publication over the Internet. *Journal of Computer Security*, 11:291C314, 2003.

[17] H. Ferhatosmanoglu, I. Stanoi, D. Agrawal, and A. Abbadi. Constrained Nearest Neighbor Queries. In *Symposium on Spatial and Temporal Databases*, pages 257–278, 2001.

[18] 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.

[19] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic Authenticated Index Structures for Outsourced Databases. In *Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data*, page 121C132, 2006.

[20] 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.

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

[22] C. Martel, G. Nuckolls, P.Devanbu, M. Gertz, A. Kwong, and S.G.Stubblebine. A General Model for Authenticated Data Structures. *Algorithmica*, 39(1):21–41, 2004.

[23] 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.

[24] R. Morselli, S. Bhattacharjee, J. Katz, and P. J. Keleher. Trust-preserving set operations. In *INFOCOM*, 2004.

[25] 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.

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

[27] J. Nievergelt, H. Hinterberger, and K. Sevcik. The Grid File: An Adaptable, Symmetric Multikey File Structure. *ACM Transactions on Database Systems*, 9(1):38–71, March 1984.

[28] J. A. Orenstein and T. H. Merrett. A class of data structures for associative searching. In *Proceedings of the 3rd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*, pages 181–190, 1984.

[29] H. Pang, A. Jain, K. Ramamritham, and K. Tan. Verifying Completeness of Relational Query Results in Data Publishing. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, 2005.

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

[31] H. Pang and K. Tan. Verifying Completeness of Relational Query Answers from Online Servers. *ACM Transactions on Information and System Security (TISSEC), accepted for publication*, 2007.

[32] 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.

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

[34] 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.

[35] M. Roos, A. Buldas, and J. Willemson. Undeniable Replies for Database Queries. In *Proceedings of the Baltic Conference, BalticDB&IS*, pages 215–226, 2002.

[36] R.Tamassia and N. Triandopoulos. Efficient content authentication over distributed hash tables. Technical report, Brown University, 2005.

[37] H. Sagan. *Space-Filling Curves*. Springer-Verlag, New York, 1994.

[38] H. Samet. The Quadtree and Related Hierarchical Data Structures. *ACM Computing Surveys*, 16(2):187–260, June 1984.

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

[40] 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.

[41] C. Yu, B. Ooi, K. Tan, and H. Jagadish. Indexing the distance: An efficient method to knn processing. In *Proceedings of the 27th International Conference on Very Large Databases*, pages 421–430, 2001.