Access Control Friendly Query Verification for Outsourced Data Publishing

Hong Chen[1], Xiaonan Ma[2] *, Windsor Hsu[2] **, Ninghui Li[1], and Qihua Wang[1]

[1] CERIAS & Department of Computer Science, Purdue University
{chen131, ninghui, wangq}@cs.purdue.edu
[2] IBM Almaden Research Center
xiaonan.ma@gmail.com, windsorh@cs.berkeley.edu

Abstract. Outsourced data publishing is a promising approach to achieve higher distribution efficiency, greater data survivability, and lower management cost. In outsourced data publishing (sometimes referred to as third-party publishing), a data owner gives the content of databases to multiple publishers which answer queries sent by clients. In many cases, the trustworthiness of the publishers cannot be guaranteed; therefore, it is important for a client to be able to verify the correctness of the query results. Meanwhile, due to privacy concerns, it is also required that such verification does not expose information that is outside a client's access control area. Current approaches for verifying the correctness of query results in third-party publishing either do not consider the privacy preserving requirement, or are limited to one dimensional queries. In this paper, we introduce a new scheme for verifying the correctness of query results while preserving data privacy. Our approach handles multi-dimensional range queries. We present both theoretical analysis and experimental results to demonstrate that our approach is time and space efficient.

1 Introduction

The amount of data stored in databases is rapidly increasing in today's world. A lot of such data is published over the Internet or large-scale intranets. Given the large sizes of databases and the high frequency of queries, it is often desirable for data owners to outsource data publishing to internal or external publishers. In such a model, the data center of an organization gives datasets to internal publishers (for publishing within the organization's intranet), or external third-party publishers (for publishing over the Internet) [1–4].

Offloading the task of data publishing from data owners to dedicated data publishers offers the following advantages: (1) the publishers may have higher bandwidths; (2) the publishers may be geographically closer to the clients and have lower latencies; (3) having multiple publishers helps to avoid the data owner

* Currently with Schooner Information Technology, Inc.
** Currently with Data Domain, Inc.

being a single point of failure; (4) overall data management cost can be significantly reduced, by leveraging hardware and software solutions from dedicated data publishing service providers [5].

In many settings the trustworthiness of the data publishers cannot be guaranteed – the security of the publishers' servers is not under the control of the data owners. Historical computer security incidents have shown that securing large online systems is a difficult task. The threat of insider attacks from within a data publishing service provider cannot be overlooked either. Therefore it is critical for a client to be able to verify the correctness of query results.

There are three aspects of correctness: *authenticity*, *completeness* and *freshness* [1, 6–8]. A query result is authentic if all the records in the result are from the dataset provided by the data owner. A query result is complete if the publisher returns all data records that satisfy the query criteria. A query result is fresh if the query result reflects the current state of the owner's database.

Suppose a dataset contains numbers $12, 20, 5, 10, 18, 30, 16, 31$. The range query $[15, 25)$ asks for numbers between 15 (inclusive) and 25 (non-inclusive). The correct result is $\{16, 18, 20\}$. A result $\{16, 17, 18, 20\}$ is not authentic. And a result $\{18, 20\}$ is not complete.

Several approaches have been proposed for verifying the query results in third-party publishing, e.g., [1, 6, 4]. Most of these solutions use techniques from public-key cryptography. The data owner has a pair of public/private keys. Verification metadata is generated over the dataset using the private key by the owner, and the metadata is provided to the publishers with the dataset. When a client queries from a data publisher, the publisher returns the query result together with a proof called a *Verification Object (VO)* [1], which is constructed based on such metadata. The client can then verify the correctness of the query result using the corresponding *VO*, with the data owner's public key.

Due to increasing privacy[3] concerns for today's information management, preserving data privacy has become a critical requirement. The data owner and the publishers need to enforce access control policies so that each client can only access the information within her own accessible area. On the other hand, clients should be able to verify the correctness (namely authenticity, completeness and freshness) of the query results, even if the publishers could be malicious or be compromised. The data privacy should be preserved at least when the publishers are benign. When the publishers are compromised, the bottom line is that the publishers cannot cheat the clients by giving them bogus query results. Though in that case, data privacy might be violated. As the security model discussed in [7], the publishers are not fully trusted. Although it seems contradictory that the publishers are not fully trusted but yet are expected to protect data privacy, such a combination of requirements is reasonable given the following observations: (1) even highly secure systems cannot promise that they would never be compromised, and cautious clients may require correctness verification nevertheless. Leaking private information as a side effect of offering correctness

[3] By privacy we mean the data should not be accessed by any unauthorized party; from another perspective, it means the confidentiality of users' data.

verification (even when the publishers are benign) is problematic; (2) The semi-trusted publisher model also fits well when the correctness requirement and the privacy requirement are not equally stringent. For example, a data owner utilizing such semi-trusted servers may want to ensure that clients can verify data correctness. The data owner may be less concerned with data privacy (therefore could tolerate semi-trusted publishers) but still want decent protection such that information is not leaked voluntarily and access control is not trivially bypassed.

Achieving completeness of query results while preserving privacy is challenging. To achieve completeness, one needs to show that there exists no other record in the query region other than the ones that are returned in the query result. Most existing approaches leak information that is outside the query region and the client's accessible area, violating the privacy preserving requirement.

For instance, back to the above example. Recall that the user's query is $[15, 25)$ and the correct result is $\{16, 18, 20\}$. Assume that the access control area for the user is $[10, 29)$, meaning that she can only access numbers in $[10, 29)$. To prove the completeness of the query result, the publisher should show that the following numbers are not in the dataset: $15, 17, 19, 21 - 24$. Most existing solutions rely on proving the adjacency of data points. The data owner signs the data pair $(20, 30)$ to indicate that there are no data points between 20 and 30 in the dataset. The publisher can return the signed data pair as part of the proof of completeness. However, in doing so, the data point 30, which is outside the client's access control area, is revealed to the client and thus data privacy is violated. Besides data privacy, we also considers *policy privacy*, where the client should not know the boundaries of other users' access control areas (details in Section 3.3). In [7] Pang et al. proposed a scheme which allows correctness verification while preserving the privacy of data records. However, the solution applies only to one dimensional range queries, which is a significant limitation given the multidimensional nature of the relational databases today.

In this paper, we present a novel scheme for proving the correctness of query results produced by data publishers. Our approach preserves data privacy and therefore can be used to perform access control. Our approach does not rely on proving the adjacency of neighboring data points for completeness verification, and can handle multi-dimensional queries.

The rest of the paper is organized as follows. Section 2 discusses the related work. Section 3 gives the security model and formalizes the problem. Section 4 presents our verification scheme. Section 5 analyzes the time and space efficiency of our scheme and discusses experimental results. Section 6 concludes the paper.

2 Related Work

Query-answering with third party publishing has been studied in the computer security and cryptography community under the name authenticated dictionary [1, 9, 10]. Schemes using Merkle Tree [11] and skip lists have been proposed; however these approaches assume that the data is public and do not consider the access control requirement. In [1], a scheme based on Merkle Tree is pro-

posed to guarantee the authenticity and completeness. In [12], this approach is generalized and applied to other authenticated data structures. Data structures based on space and data partitioning are introduced in [13] for verifying multi-dimensional query results. In these approaches, data privacy is not preserved. A scheme to verify the integrity of the query results in edge computing is proposed in [14]. The scheme does not check the completeness of query results. The security model in [15], where semi-trusted service providers answer user queries on behalf of the data owner, is similar to ours.

In order to preserve the data privacy, [7] proposed another scheme to solve the problem. This approach handles one-dimensional case well, but it cannot be applied to two or higher dimensional cases. In [6], the overheads and performances of different approaches to guarantee the authenticity and completeness are compared.

Several approaches are proposed for enforcing access control policies in outsourced XML documents [16–18]. The XML documents, which can be viewed as trees, present different structures from relational databases.

The data structure and algorithms derived in this paper uses the divide-and-conquer strategy developed in data structures and algorithms for range searching by the computational geometry community [19–21]. Our approach is unique in that our approach addresses the specific requirements of outsourced data publishing, especially the need to efficiently prove the search results. Also our solution incurs low communication and storage overhead when updating the data items and/or the access policies of clients.

3 Problem Overview

In this section we discuss the security model and requirements of the problem.

3.1 Security Model

The security model for our scheme is the same as that of [7]. We assume the data owners are secure and trusted. Each data owner maintains one public/private key pair for signing data and verifying data. We also assume that the publishers and clients can obtain the correct public keys for each data owner through some secure channel. The clients should be able to verify the authenticity, completeness and freshness of the query results using the public keys of the data owners. When the clients perform the query verification, they should not be able to gain any information that they do not have access to.

3.2 Formalization

The dataset contains k attributes $A_1, \ldots, A_k$. We assume that each attribute is of an integer. We assume the ranges of all attributes are $[0, N)$, and N is a power of two. Hence each record can be represented by a point in the k-space. Let T denote the set of all the points in the dataset, we have $T \subseteq [0, N)^k$.

Given any record $r \in T$, let $A_i(r)$ denote the value of the ith attribute of the record. The client may issue a range query $Q(L_1, R_1, \ldots, L_k, R_k)$, which defines a *query space* $q = [L_1, R_1) \times \cdots \times [L_k, R_k) \subseteq [0, N)^k$ A client submits Q to get the result $T' = T \cap q$. Upon receiving the query, the publisher will provide the client with the query result T' with a verification object VO to guarantee the authenticity and completeness.

The publishers enforce access control policies on the clients to protect privacy. Suppose we have a payroll database, and each record contains salary, rank and other information of an employee. Access control policies enforce that a particular accountant can only access the records with the salary below $20,000$ and the rank below 10. We call this *accessible space* of a user. The access policy enforced on a user is represented as $AC(L_1, R_1, \ldots, L_k, R_k)$. The accessible space ac of a user is a sub-space in the k-space:

$$ac = [L_1, R_1) \times \cdots \times [L_k, R_k) \subseteq [0, N)^k \tag{1}$$

3.3 Requirements

In order to prove the correctness of the query results, authenticity and completeness need to be satisfied. At the same time, privacy needs to be preserved: (1) the client should not get any information about any data that is outside the client's access control area; (2) the client should not learn any information about the access control policies except her own access control policy.

Suppose the result of a query is T'', and the database is T, we say that the result is **authentic** when $T'' \subseteq T$. Suppose the range query space is q, if $\forall r \in T \quad r \in q \Rightarrow r \in T''$, we say the result is **complete**.

Suppose the accessible space for the user is ac. The records within the user's accessible space are $v = T \cap ac$, and the points that are outside the accessible space are $v' = T - v$. Suppose $v_0 \subseteq [0, N)^k$ is a set of data points, we use $P_1(v' = v_0)$ to denote the probability that the data points outside ac are a particular set v_0 when the user observes all the data points in ac. After observing the verification object, the user has another probability function $P_2(v' = v_0)$ which denotes the probability that the data points outside ac are a particular set v_0. The **privacy is preserved** if

$$\forall v_0 \subseteq [0, N)^k \; P_1(v' = v_0) = P_2(v' = v_0) \tag{2}$$

In our approach, we use a more practical (and stronger) definition for privacy preserving: data privacy is preserved if VO only depends on the access control policy of the user and the records within the user's accessible space:

$$VO = VO(v, ac) \tag{3}$$

4 Query Result Verification

As described in some related works, there are several ways to guarantee the authenticity of the query result. The main challenge is to prove the completeness

while preserving the privacy. Thus our main focus is on proving completeness. For simplicity and easier description, we assume that the data owner signs each data point in the database for authenticity verification. Previous works on more efficient data integrity verification and data freshness verification can be combined with our scheme [1, 6, 8].

Our solution to the problem is based on the following insight: when we are limited to using only information in a region, proving a positive statement saying that there exists a record in a region is easy, but proving a negative statement saying that there does not exist a record in a region in a privacy-preserving way is difficult. Our approach aims to turn a negative proof for completeness into a positive proof.

4.1 Solution Overview

Suppose the query space is q and the access control space is ac. There are n_q records in the query space. The publisher returns those n_q records as the query results. To prove those are the only records in the query space, the publisher proves:

1. There are totally n_{ac} records in ac.
2. There are at least $n_{ac} - n_q$ records in $ac - q$ (the area outside q and inside ac).

An example is shown in Figure 1. Data points A, B, C, D, E are inside ac, where A, B are inside q. F, G, H, I are outside ac. The publisher first returns A, B as the query result. To prove there are only two records in the query space, the publisher will prove that: (1) there are totally 5 records inside ac; (2) there are at least 3 records in $ac - q$.

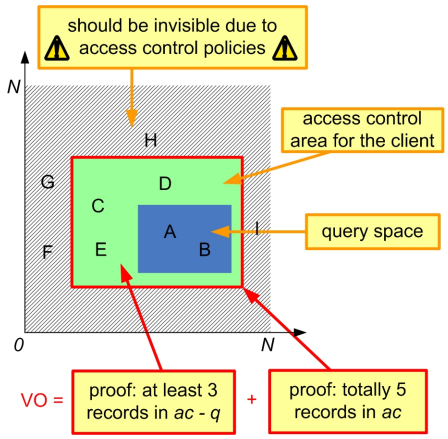


Fig. 1. Solution Overview

To achieve this, the data owner will provide a signature indicating that there are n_{ac} data points in ac. Also the data owner needs to provide efficient proof that there are at least $n_{ac} - n_q$ data points in $ac - q$. We propose a data structure to provide the efficient proof.

Intuitively, the reason we use the data points in $ac - q$ is that it is easier to prove "existence" than "non-existence". As in Figure 1, if we only construct the *VO* using data points in q, we need to prove that all the space in q is empty except for A and B. This is not an easy task since the query space can be any subspace inside ac and is not predictable. In contrast, to prove the existence of data points only requires using data points in $ac - q$. In this case, the publishers only need to provide the number of data points in $ac - q$, which can be produced efficiently utilizing aggregated counting data structures, instead of the actual values of these data points.

To prove the existence of a number of records in $ac - q$, we need an efficient proof. A trivial solution is to return all the records in $ac - q$, which is too expensive therefore not practical. To solve this problem, we design a data structure called *Canonical Range Tree*, the details of which are discussed in section 4.2.

Thus, there are three components in the *VO*: the authentication data structure which proves the authenticity of the data records in the query result; the number of records in the accessible space of the client, which is signed by the data owner; and the number of records in $ac - q$ which is also authenticated by the data owner. Note that although $ac - q$ is different for different queries, our approach does not require the data publisher to contact the data owner for each query. The authentication data structure we developed allows the data publisher to efficiently prove to the client the number of data records in a particular $ac - q$.

4.2 Canonical Range Tree

To better explain our idea, we define the following concepts in k-space.

Definition 1 (Cube). A sub-space of the k-space in the following form is called a *cube*: $[L_1, R_1) \times \cdots \times [L_k, R_k)$

Definition 2 (Shell). A sub-space of the k-space in the form $c_1 - c_2$ is called a *shell*. Here c_1 and c_2 are both k-dimensional cubes and $c_2 \subseteq c_1$.

From the problem definition, a query space is a cube, the accessible space of a client is also a cube. The space inside a user's accessible space but outside the query space is a shell.

Range tree [20, 22] is a data structure used in computational geometry to store points in k-space. In our solution, we use a modified version of range tree. It is called *Canonical Range Tree*, or CRT for short.

We use CRT to store the counting information for data points. And we will use a set of nodes of the tree as evidence of existence of records in the shell. We first discuss one dimensional CRT. In this case CRT is used to store a list of numbers $x_1, \ldots, x_n$. One dimensional CRT is a binary tree. Each node of the tree

corresponds to an interval. Suppose *node* is a CRT node, it stores the number of points in the interval $[node.l, node.r)$.

If the interval of a node is a unit interval ($node.l + 1 = node.r$) the node is a leaf node. The size of the interval of a node $node.r - node.l$ is always a power of 2. We call $[node.l, (node.r + node.l)/2)$ the left sub-interval, and $[(node.r + node.l)/2, node.r)$ the right sub-interval.

Suppose there are n' records in the left sub-interval, *node* will have a left child $node_1$ if $n' > 0$:

$$node_1.l = node.l \quad node_1.r = (node.r + node.l)/2 \quad node_1.cnt = n'$$

And *node* will have a right child $node_2$ if $n' < node.cnt$:

$$node_2.l = (node.l + node.r)/2 \quad node_2.r = node.r \quad node_2.cnt = node.cnt - n'$$

We refer to the left and right child of *node* as $node.c1$ and $node.c2$. Each of them can be *nil* if empty. The root node of the tree corresponds to the interval $[0, N)$. Figure 2 shows a CRT storing the list of 3 numbers: $5, 12, 15$.

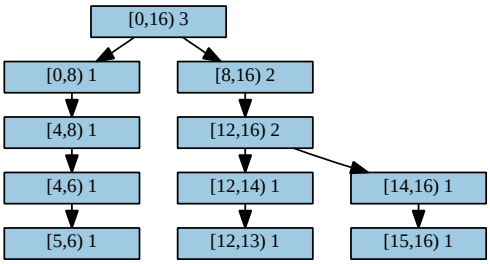


Fig. 2. 1-D CRT Example

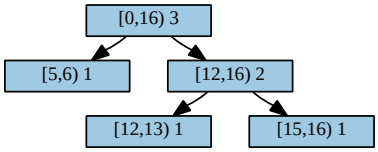


Fig. 3. Optimized 1-D CRT

Now we discuss how to build a CRT for 2-D space. Suppose we have a list of points $(x_1, y_1), \ldots, (x_n, y_n)$. We first build a 1-D CRT for the list of numbers $x_1, \ldots, x_n$. This tree is called the primary structure. Then for every node *node* of the primary structure, suppose there are n' points of which the first coordinate is in the interval $[node.l, node.r)$. By definition $node.cnt = n'$. Let $(x'_1, y'_1), \ldots, (x'_{n'}, y'_{n'})$ be those points. We then build a one dimensional CRT for this node, to store information for the numbers $y'_1, \ldots, y'_{n'}$. In this way, we build a primary structure on the first attribute of the data points, and for every node of the primary structure, we build a secondary structure. For each node *node* of the primary structure, *node.sec* stores the root of the corresponding secondary structure. This completes the CRT construction for 2-D space. Figure 4 gives an example of a 2-D CRT to store a list of 3 points: $(5, 10), (12, 4), (15, 19)$.

For two dimensional CRT, we call a node of the primary structure a 1st order node, and a node of the secondary structure a 2nd order node. A 1st order node *node* stores the number of points in the area $[node.l, node.r) \times [0, N)$. Suppose

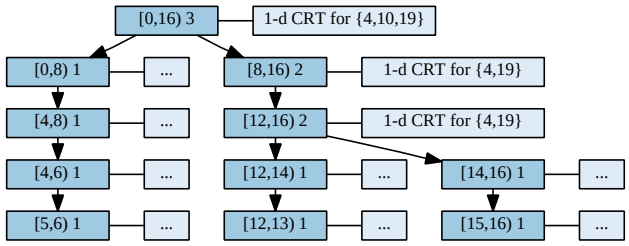


Fig. 4. Two-dimensional CRT Example

$node'$ is a node belongs to the secondary structure attached to $node$, then $node'$ stores the number of points in the area $[node.l, node.r) \times [node'.l, node'.r)$.

We can construct k dimensional CRTs similarly. There are 1st, 2nd, ..., kth order nodes in a k-D CRT. Every tth ($t < k$) order node has a $t+1$-ary structure. The insertion and deletion algorithms of CRT are shown in Figure 5 and Figure 6.

```
function CRT_Insert(r, node, t)
    node.cnt ← node.cnt + 1
    if t < k then
        if node.sec = nil then
            create node.sec with [0, N)
        CRT_Insert(r, node.sec, t + 1)

    if node.r − node.l > 1 then
        mid = (node.l + node.r)/2
        if A_t(r) < mid then
            if node.c1 = nil then
                create node.c1 with [node.l, mid)
            CRT_Insert(r, node.c1, t)
        else
            if node.c2 = nil then
                create node.c1 with [mid, node.r)
            CRT_Insert(r, node.c2, t)
```

Fig. 5. CRT Insertion

```
function CRT_Delete(r, node, t)
    node.cnt ← node.cnt − 1

    if t < k then
        CRT_Delete(r, node.sec, t + 1)
        if node.sec.cnt = 0 then
            remove node.sec

    if node.r − node.l > 1 then
        mid = (node.l + node.r)/2
        if A_t(r) < mid then
            CRT_Delete(r, node.c1, t)
            if node.c1.cnt = 0 then
                remove node.c1
        else
            CRT_Delete(r, node.c2, t)
            if node.c2.cnt = 0 then
                remove node.c2
```

Fig. 6. CRT Deletion

4.3 Constructing Evidence in a Cube/Shell

Given a cube/shell, CRT nodes could be used as proof of existence of all the records in the cube/shell. In both cases, the evidence is a list of verified non-overlapping kth order CRT nodes. The counter of a such node gives the number of records in the corresponding sub-space. Summing up the counters we get a proof of existence of the records.

Suppose a cube is $c = [L_1, R_1) \times \cdots \times [L_k, R_k)$, recursive function Cnt_Cube($node, t, c$) in Figure 7 returns a list of CRT nodes as evidence in c.

Function Cnt_Shell($node, t, c, c'$) in Figure 8 returns a list of non-overlapping kth order nodes as evidence for shell $c - c'$, where $c' \subseteq c$.

```
function CNT_CUBE(node, t, c)
    if node.r ≤ L_t ∨ node.l ≥ R_t then
        return φ

    if node.l ≥ L_t ∧ node.r ≤ R_t then
        if t = k then
            return {node}
        else
            return Cnt_Cube(node.sec, t+1, c)
    else
        ret ← φ
        if node.c1 ≠ nil then
            ret ← ret ∪ Cnt_Cube(node.c1, t, c)
        if node.c2 ≠ nil then
            ret ← ret ∪ Cnt_Cube(node.c2, t, c)
        return ret
```

Fig. 7. Constructing Evidence in a Cube

```
function CNT_SHELL(node, t, c, c')
    if node = nil then
        return φ

    if c.L_t ≤ node.l ∧ node.r ≤ c'.L_t then
        x ← c
        x.R_t ← c'.L_t
        return Cnt_Cube(node, t, x)
    if c'.R_t ≤ node.l ∧ node.r ≤ c.R_t then
        x ← c
        x.L_t ← c'.R_t
        return Cnt_Cube(node, t, x)
    if c'.L_t ≤ node.l ∧ node.r ≤ c'.R_t then
        if t = k then
            return φ
        return Cnt_Shell(node.sec, t + 1, c, c')

    ret ← φ
    if [c.L_t, c.R_t)∪[node.c1.l, node.c1.r) ≠ φ then
        ret ← ret∪ Cnt_Shell(node.c1, t, c, c')
    if [c.L_t, c.R_t)∪[node.c2.l, node.c2.r) ≠ φ then
        ret ← ret∪ Cnt_Shell(node.c2, t, c, c')
    return ret
```

Fig. 8. Constructing Evidence in a Shell

Note that in the solution, we need to provide evidence outside the query space and inside the accessible space, which is a shell. The above algorithm is used to achieve this.

4.4 *VO* Construction

The data owner will maintain a k-dimensional CRT for all the records. Suppose there are n records in the database, the owner will first build an empty CRT and then insert the n records. The owner should also guarantee the integrity of the CRT nodes, so the user can use the CRT nodes to verify the number of records in a shell. Verifying whether a CRT node is generated by the owner is a membership testing problem. This problem is not the focus of this work, any solution for membership testing that incurs low storage and communication overhead can be used in our scheme, e.g., authenticated dictionary [9, 10].

Also, the data owner will maintain a counter for each access control space. Suppose there are m access control spaces $ac_1, \ldots, ac_m$, the data owner maintains and signs the pairs $(ac_1, cnt_1), \ldots, (ac_m, cnt_m)$. cnt_i is the number of records in access control space ac_i. The data owner can call the function Cnt_Cube for ac_m to get the list of nodes and add up the counters of the nodes to get cnt_i.

Canonical range tree has a nice property: given any k dimensional rectangular space S, say there are a points from T that are inside S. CRT can use a small number of non-overlapping nodes that are completely within S, to prove that there are at least a points in S. This property is very useful in our *VO* construction.

The data owner will give the signed CRT and the signed list of access control counters to the publisher. When the client submits a query with query space q, the publisher will return the query result to the client with the following Verification Object(suppose the access control space of the client is ac):

- The metadata for the user to verify the integrity of the records in the query result.
- The verifiable number of records in the user's accessible space ac.
- A set of verifiable CRT nodes that can be used to verify that there are at least $n_{ac} - n_q$ records in $ac - q$. This evidence could be collected by calling the function Cnt_Shell.

4.5 Optimization

In Figure 2, we can see several "redundant" nodes, such as $\langle[0, 8), 1\rangle$, $\langle[4, 8), 1\rangle$, $\langle[4, 6), 1\rangle$. These nodes do not provide additional information. If we want to use, e.g., $\langle[4, 6), 1\rangle$ in the VO, we can always use $\langle[5, 6), 1\rangle$ instead. The size of VO will not be increased, and the privacy is preserved (if the user is allowed to gain information in $[4, 6)$, she is also allow to gain information in $[5, 6)$.

Generally, if a node has only one child, we can simply remove this node. After removing all redundant nodes, each non-leaf node will have exactly two child nodes. After optimization, the CRT in Figure 2 becomes the tree CRT in Figure 3.

Extending this optimization to higher dimensional cases is non-trivial. we have two versions of optimization for higher dimensional cases:

- Applying the optimization to the last dimensional nodes. This version is called *LastDimOpt scheme*. The CRT nodes of this version are a subset of *basic scheme*, and data updating for LastDimOpt scheme is more efficient than the basic scheme.
- Applying the optimization to all dimensional nodes. This version is called *AllDimOpt scheme*. The CRT nodes of this version are a subset of the LastDimOpt scheme. If there are only insertions, the amortized updating cost of this version is smaller than the basic scheme and the LastDimOpt scheme. But the cost of inserting or deleting a specific record could be large. Therefore LastDimOpt scheme can be used when the database is static, insertion-only, or does not require frequent deletions.

Note that the number of CRT nodes in VO is the same for all three schemes. There is a one to one mapping of the CRT nodes in the VOs of the three schemes.

5 Evaluation

5.1 Theoretical Analysis

This section discusses the cost of various CRT operations. The theorems can be proved through induction. The proofs are omitted here due to space constraints.

Following theorems give the upper-bound of the number of nodes needed to prove the existence of all nodes in a cube/shell, and the cost of inserting a record into CRT.

Theorem 1. *If there are k attributes for every record, the range of the values for each attribute is $[0, N)$, at most $2^k(\log N)^k$ CRT nodes are needed to cover all the data points in a k dimensional cube.*

Theorem 2. *At most $4 \cdot (2 \log N)^k$ nodes of the CRT are needed to cover all the data points in a k dimensional shell.*

Theorem 3. *To insert a record to a k dimensional CRT, the insertion algorithm will visit at most $\frac{\log N+1}{\log N} \cdot \left((\log N + 1)^k - 1\right)$ nodes. Creating a node is also considered as visiting a node.*

The theorem implies that the time complexity for inserting a record into CRT is $O((\log N)^k)$.

The cost of different operations are listed below. Suppose there are m different access control areas:

- The storage overhead to store the CRT is bounded by $n \cdot \frac{\log N+1}{\log N} \cdot \left((\log N + 1)^k - 1\right) \approx n \cdot (\log N)^k$ (when $\log N$ is sufficiently large).
- The size of the evidence for all records in a cube is bounded by $(2 \log N)^k$.
- The size of the evidence of a shell is bounded by $4 \cdot (2 \log N)^k$. The time cost to insert or delete a record in the database is $O((\log N)^k + m)$. This cost includes the cost incurred in the data owner site and the cost in the publisher site.
- The time cost to build the database is $O(n(\log N)^k)$.
- The time cost to setup an access control policy is $O((2 \log N)^k)$. The communication cost is $O(1)$.
- The time cost to construct the VO is $O((2 \log N)^k)$.

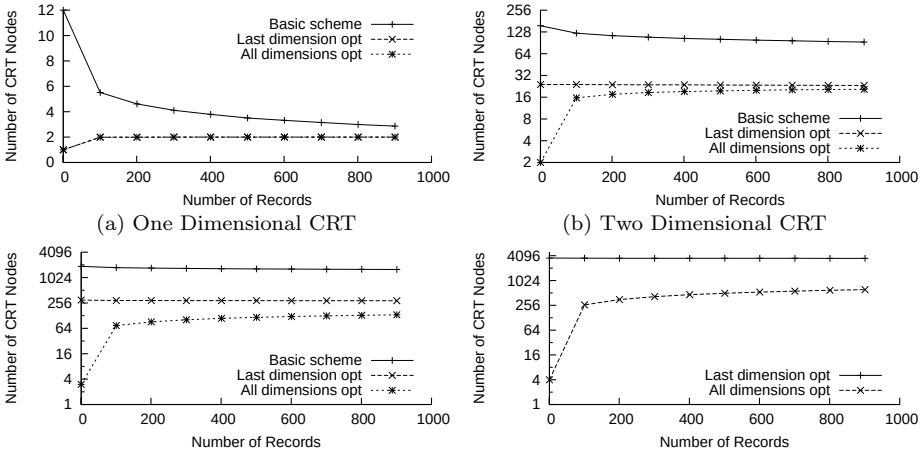
One advantage of our scheme is that the size of the proof for completeness is independent of the size of the query result. Also it is independent of the number of records in the database.

After last dimension optimization, the storage cost of k dimensional CRT can be reduced from $O(n(\log N)^k)$ to $O(n(\log N)^{k-1})$.

5.2 Experimental Results

In this section, we discuss the efficiency of our approach based on empirical data. We implemented three versions of the CRT data structure: the basic scheme, the LastDimOpt scheme, and the AllDimOpt scheme.

Storage Overhead The storage size needed to store the CRT is linear in the number of nodes. Figure 9 shows the average number of CRT nodes needed for each data record, against the size of the database. Each figure shows the results for three versions of CRT. The test data is generated randomly. For a k dimensional database, each record is generated independently. And each attribute



(a) One Dimensional CRT (b) Two Dimensional CRT

(c) Three Dimensional CRT (d) Four Dimensional CRT

X-axis: number of records in the database. Y-axis: average number of CRT nodes needed for each record. $N = 2048$ for all test cases. Each data point is the average of 10 independent runs using randomly generated data. In (d) we omit the data for basic scheme due to large overhead.

Fig. 9. Storage Overhead

value of a record is generated independently from a uniform distribution over $[0, N)$.

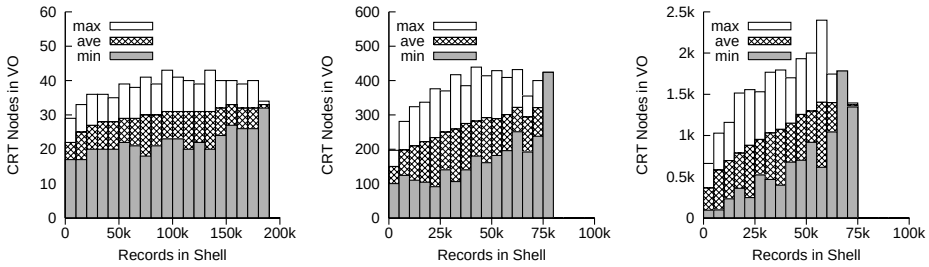
The basic scheme incurs the highest overhead. In basic scheme, the number of CRT nodes needed per record decreases when the number of records grows. The intuition is that more records "share" tree nodes when database grows.

The LastDimOpt scheme makes a big improvement over the basic scheme. Also the number of CRT nodes needed per records almost doesn't change with the growth of the database.

The AllDimOpt scheme improves over LastDimOpt significantly, especially in high dimensional cases. The number of CRT nodes needed per record grows slowly when database grows. If the application does not involve frequent updates, we can use optimization on all dimensions to save a lot of storage space.

Communication Overhead The communication overhead of verification depends on the *VO* size. Since a *VO* mostly consists of a set of CRT nodes, we use the number of CRT nodes in the *VO* to represent the *VO* size.

Figure 10 shows the number of CRT nodes in the *VO* against the number of points in the shell (which is inside the user's accessible space but outside the query space). In a k dimensional database, a query is generated as following. For each dimension i, four integers are generated independently by the uniform distribution over $[0, N]$. The four integers are sorted in non-decreasing order, we have $R_{i,1} \leq R_{i,2} \leq R_{i,3} \leq R_{i,4}$. In this query, the access control space of the user is $ac = [R_{1,1}, R_{1,4}) \times \cdots \times [R_{k,1}, R_{k,4})$ and the query space is $q = [R_{1,2}, R_{1,3}) \times \cdots \times [R_{k,2}, R_{k,3})$. Each query is generated independently.



$N = 2^{30}$, $200k$ records (a) One Dimensional
$N = 4096$, $100k$ records (b) Two Dimensional
$N = 256$, $100k$ records (c) Three Dimensional

There are 1000 random queries for each of 1-d, 2-d and 3-d cases. The x-axis is the number of records in $ac - q$. The y-axis is the number of CRT nodes in the VO. The result for each figure is grouped into 20 intervals of even sizes. For example, the left most bar in (a) shows the maximum, average and minimum number of CRT nodes in VO for all the queries that have 0 to 10000 records in $ac - q$.

Fig. 10. VO Size

The parameters used in the testing are shown in Figure 10. From the figures we can see that the size of VO grows slowly with the number of records in the shell. For 1-D, 2-D and 3-D cases respectively, the VO sizes of queries are less than 45, 450 and $2.5k$, while the number of records in the shell is up to $200k$, $80k$ and $75k$. The VO size is quite small compared to the number of records in the shell.

From the figures we observe that with greater number of dimensions, the VO size grows faster with respect to the number of records in the shell. This trend is also shown by our theoretical analysis. In addition, our scheme favors denser data, because under the same k and same N, when the size of database grows, the size of VO does not grow significantly. Making the database denser will make the ratio of VO size against number of records in the shell smaller.

6 Conclusions

We formalize the query verification problem in third-party data publishing, in which we need to guarantee the authenticity and completeness of the query results, while preserving the data privacy. We provide a novel solution for this problem, in which we convert proving of non-existence to proving of existence of data records. Our solution can be used for multiple dimensional range queries and scales well to a reasonable number of dimensions. Based on the theoretical analysis and empirical data, we show that our solution is efficient in terms of storage and communication overhead.

References

1. Devanbu, P., Gertz, M., Martel, C., Stubblebine, S.: Authentic data publication over the internet. Journal of Computer Security **11** (2003) 291–314

2. Luo, Q., Krishnamurthy, S., Mohan, C., Pirahesh, H., Woo, H., Lindsay, B.G., Naughton, J.F.: Middle-tier database caching for e-business. In: SIGMOD. (2002)
3. Saroiu, S., Gummadi, K.P., Dunn, R.J., Gribble, S.D., Levy, H.M.: An analysis of internet content delivery systems. SIGOPS Oper. Syst. Rev. **36**(SI) (2002)
4. Cheng, W., Pang, H., Tan, K.L.: 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, Sophia Antipolis, France, Springer Berlin / Heidelberg (2006) 60–73
5. Hacigumus, H., Iyer, B.R., Mehrotra, S.: Providing database as a service. In: ICDE. (2002)
6. Li, F., Hadjieleftheriou, M., Kollios, G., Reyzin, L.: Dynamic authenticated index structures for outsourced databases. In: SIGMOD. (2006)
7. Pang, H., Jain, A., Ramamritham, K., Tan, K.: Verifying completeness of relational query results in data publishing. In: SIGMOD. (2005)
8. Mykletun, E., Narasimha, M., Tsudik, G.: Authentication and integrity in outsourced databases. In: NDSS. (2004)
9. Goodrich, M., Tamassia, R., Schwerin, A.: Implementation of an authenticated dictionary with skip lists and commutative hashing. In: DISCEX II. (2001)
10. Naor, M., Nissim, K.: Certificate revocation and certificate update. In: Proceedings of the 7th USENIX Security Symposium. (Jan 1998)
11. Merkle, R.C.: Secrecy, Authentication, and Public Key Systems. PhD thesis, Stanford University (1979)
12. Martel, C., Nuckolls, G., Devanbu, P., Gertz, M., Kwong, A., Stubblebine, S.: A general model for authenticated data structures (2001)
13. Cheng, W., Pang, H., Tan, K.L.: Authenticating multi-dimensional query results in data publishing. In: DBSec. (2006)
14. Pang, H., Tan, K.: Authenticating query results in edge computing. In: ICDE. (2004) 560–571
15. Haber, S., Horne, W., Sander, T., Yao, D.: Privacy-preserving verification of aggregate queries on outsourced databases. Technical Report HPL-2006-128, HP Labs (2006)
16. Miklau, G., Suciu, D.: Controlling access to published data using cryptography. In: VLDB. (2003) 898–909
17. Bertino, E., Carminati, B., Ferrari, E., Thuraisingham, B., Gupta, A.: Selective and authentic third-party distribution of xml documents. IEEE Transactions on Knowledge and Data Engineering **16**(10) (2004) 1263–1278
18. Carminati, B., Ferrari, E., Bertino, E.: Securing xml data in third-party distribution systems. In: CIKM '05: Proceedings of the 14th ACM international conference on Information and knowledge management, New York, NY, USA, ACM (2005) 99–106
19. Finkel, R.A., Bentley, J.L.: Quad trees: a data structure for retrieval on composite keys. Acta Informatica **4**(1) (March March, 1974) 1–9
20. Bentley, J.L., Shamos, M.I.: Divide-and-conquer in multidimensional space. In: STOC '76: Proceedings of the eighth annual ACM symposium on Theory of computing, New York, NY, USA, ACM (1976) 220–230
21. Agarwal, P., Erickson, J.: Geometric range searching and its relatives. Advances in Discrete and Computational GeometryContemporary Mathematics **223** (1999) 1–56
22. Chiang, Y., Tamassia, R.: Dynamic algorithms in computational geometry. Proceedings of IEEE, Special Issue on Computational Geometry **80**(9) (1992) 362–381