Distributed kNN Query Authentication

Ce Zhang, Cheng Xu, Jianliang Xu, Byron Choi
Department of Computer Science, Hong Kong Baptist University, Hong Kong
{cezhang, chengxu, xujl, bchoi}@comp.hkbu.edu.hk

***Abstract*—With the prevalence of location-based services and geo-functioned devices, the trend of spatial data outsourcing is rising. In the data outsourcing scenario, result integrity must be ensured by means of a query authentication scheme. However, most of the existing studies are confined to a centralized environment. In this paper, we investigate the query authentication problem in distributed environments and focus on the k nearest neighbor (kNN) query, which is widely used in spatial data analytics. We design a new distributed spatial authenticated data structure (ADS), distributed MR-tree, to facilitate efficient kNN processing. Furthermore, we propose a basic algorithm to process authenticated kNN queries based on the new ADS. Apart from the results, some verification objects are generated to guarantee the results' integrity. We also design two optimized algorithms to reduce the size of verification objects as well as the verification cost. Our experiments validate the good performance of the proposed techniques in terms of query cost, communication overhead, and verification time.**

***Index Terms*—kNN, Query Authentication, Distributed Systems**

I. Introduction

The amount of spatial data has been increasing at an exponential pace owing to the prevalence of geo-functioned devices such as smartphones and IoT sensors. Querying the huge spatial datasets for big data analytics is often out of the computation capacity of a traditional data owner (DO). Meanwhile, it is normally unaffordable for a small- or medium-business DO to maintain large datasets. As a remedy, database outsourcing has been widely adopted to alleviate the DO's computation overhead and the maintenance cost [1]. More specifically, a third-party service provider (SP) is employed and the owner transfers the data and index to the SP. The client sends queries to the SP, which processes the queries and returns the results to the client on behalf of the DO.

However, the SP is out of the control of the DO and the query results can be tampered with or incomplete. The SP might want to reduce the computation resources and, as a result, return only a subset of the results. The SP can also be sponsored by the DO's rivals and return fictitious results so as to favor those sponsors. As such, the client should verify the results' soundness and completeness. The soundness implies that all the results are originated from the DO. The completeness means that none of the true results are missing. To achieve this, there have been many works on authenticated query processing [1]–[12]. The basic idea is that the DO sends the authenticated data structure (ADS), along with the outsourced data, to the SP. After receiving a query from the client, the SP returns the query results as well as a *verification object* (VO) for the client to verify the soundness and completeness of results.

To date, most of the authenticated query processing studies are confined to a centralized environment. Due to the large amount of data, the SP can opt to store and process data in a distributed framework. For example, systems such as Spatial Hadoop [13] and Geospark [14] provide spatial queries based on a cluster framework. Therefore, distributed spatial query authentication should be developed to satisfy the SP's guarantee of soundness and completeness. However, the challenge is that the ADS should be designed to well fit the distributed setting and facilitate distributed query processing. Moreover, the VO size should be small enough so that the communication overhead between the SP and the client is light and the client's verification time is short.

In this paper, we consider the distributed query authentication problem for kNN (k nearest neighbor) queries. We propose a new distributed ADS, i.e., distributed MR-tree. It has local and global layers and perfectly suits the distributed master-slave system. After receiving a kNN query from the client, the master node emits a message to some slaves, which will process the query and generate the partial results and VOs. The reducer then consolidates all the partial results as well as the partial VOs and sends them to the client for result verification. We also propose two optimized algorithms to reduce the VO size. The two optimized algorithms spend more query time in return for smaller VOs. The experiments show that the optimized algorithms outperform the basic algorithm in terms of the VO size and only sacrifice a little query cost. Furthermore, it is demonstrated that the system scales well with the node number in terms of the system throughput.

The paper's contributions are summarized as follows:

- For the first time in the literature, we study the problem of authenticated kNN query processing in a distributed environment.
- We propose distributed MR-tree as an ADS to suit the distributed query processing environment.
- We develop a basic algorithm and two optimization techniques to efficiently process authenticated kNN queries in a distributed fashion.
- We conduct extensive experiments to validate the performance of the proposed techniques in terms of query cost, communication overhead, and verification time.

The rest of this paper is organized as follows. In Section II, we introduce the cryptographic primitives, spatial ADS, and local authenticated kNN processing as the preliminaries. In Section III, we show the design of the distributed MR-tree, the basic algorithm to process the authenticated kNN queries, and the client verification algorithm. Two optimized algorithms are introduced in Section IV. Section V presents the experiment results. The related work is discussed in Section VI. Finally, Section VII concludes the paper.

II. Preliminaries

A. *Cryptographic Primitives*

Hash Function: A one-way hash function $H(\cdot)$ transforms an arbitrary message m to a fixed-length digest $H(m)$. One-way indicates that given a message m, the computation of $H(m)$ is easy. However, to get m from $H(m)$ is computationally infeasible. In this paper, we use SHA-1 as the hash function, which maps a given message to a 160-bit digest.

Digital Signature: The asymmetric signature is used to verify the integrity of the data. The owner of data keeps a secret key and publishes the corresponding public key to the verifier. The owner signs the data using the secret key and the verifier can decrypt the signature using the public key and verify the integrity of the data. RSA is one of the popular asymmetric signature algorithms.

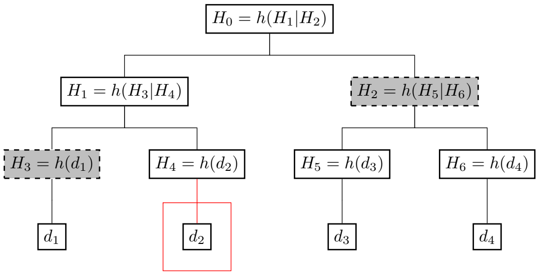


Fig. 1. Merkle Hash Tree

Merkle Hash Tree: A Merkle Hash Tree (MHT) [15] is used to authenticate a series of data points. It is a binary tree and is built in a bottom-up manner. Each leaf node of an MHT stores the hash value of the corresponding data point. The hash value of an internal node is the hash function of the concatenation of its two child nodes' hash values. The root hash value is signed by the data owner. Given a data point and its sibling hash values on the path from the root to the object, the verifier can reconstruct the root hash value and check whether it is the same as the decrypted signature. If they match, the authenticity of the data point is assured. Figure 1 shows a Merkle Hash Tree. We assume the query is d_2. The hash values of the grey nodes in Figure 1 are added into the VO, which contains (H_3,d_2,H_2). The client can compute the root hash value by $h(h(H_3|h(d_2))|H_2)$ and check its consistency with the decrypted signature. Here $h(\cdot)$ denotes the hash function.

B. *Spatial Authenticated Data Structure*

MR-tree [3], which is the combination of the R-tree and the Merkle Hash tree, is often used to process authenticated spatial queries. Each leaf node stores the pointers pointing to the data points and the hash value of the binary concatenation of the data points. The hash value of an internal node is computed by hashing the concatenation of each child node's MBR and hash value. We give an example based on Figure 2, which depicts the data points and their corresponding MR-tree. The leaf node N_3's hash value is $H(a|b|c)$, where '|' represents the binary concatenation. The non-leaf node's hash value $H(N_1) = H(N_3|H(N_3)|N_4|H(N_4))$. Here N_3, N_4 represent the MBR of node N_3 and N_4, respectively. We only show three hash values in Figure 2 and other hash values are computed similarly.

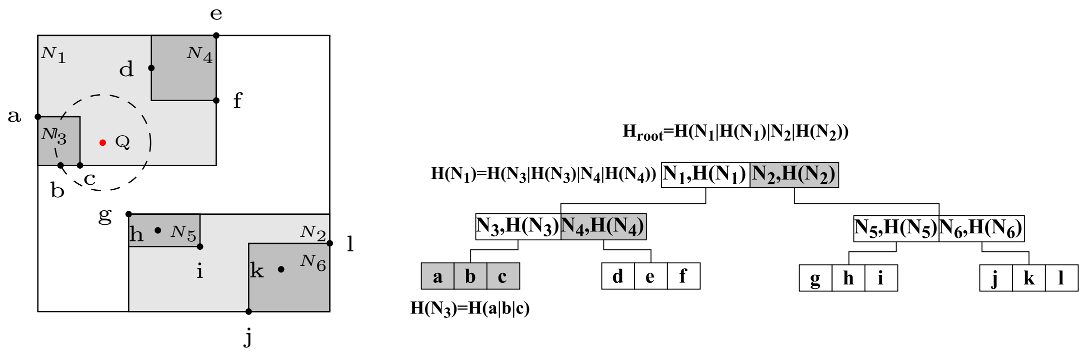
C. *Local Authenticated kNN Processing*

There are mainly two different methods of authenticated kNN processing based on the MR-tree. The algorithm proposed by Yang *et al.* [3] mainly focuses on the authenticated range query and transforms the kNN query to a range query. The method separates the generation of the results and the VO. It first retrieves kNN results by using any kNN search method. Based on the k results, we can draw a circle centered at the query point q with the radius of the distance between q and the k^{th} result. Then, the authenticated range query is executed to generate the VO set. Another method is to generate the results and the VO set together while traversing the whole MR-tree. Su *et al.* [5] proposed the authentication of top-k spatial keyword queries and we can transform this query to the authenticated kNN query easily. Algorithm 1 shows the procedure of generating the VO and results altogether.

Algorithm 1 Local Authenticated kNN Query

```
Input $k$, Point $q$, Root of MR-tree
Output $RS$, $VO$
$RS$ = $\emptyset$ ; $counter = 0$;
$VO$ = ('[', each root entry $R_i, R_i.hash$, ']')
$PQ$ Enqueues each Root entry $R_i$;
while $counter < k$ && $PQ \neq \emptyset$ do
    Dequeue priority queue $PQ$ and get $object$
    if $object$ is an MBR $R_i$ then
        Enqueue each $R_j$ or $O_j$ in $R_i$'s child node into $PQ$
        Replace $R_i, R_i.hash$ with '[', each $R_j$,$R_j$.hash (or $O_j$) in $R_i$'s child node, and ']' in $VO$
    else
        $RS$.add($O_i$)
        $counter$++
    end if
end while
return $(RS, VO)$
```

The algorithm traverses the MR-tree in a best first search manner [16]. We maintain a priority queue PQ. In the while loop, the current nearest object to the point q is dequeued from the priority queue PQ. If the object is a node rather than a data point, each of its child node R_j will be enqueued to the PQ. Otherwise, the object is added to the result set because it is a point. The while loop terminates until k results



(a)

(b)

Fig. 2. Local authenticated kNN processing

are collected or the PQ is empty. The VO set changes along with the expansion of the object. The sign marks '[' and ']' are used to decide the scope of entries in a node and they can help the client reconstruct the root hash value. The VO set is initialized with '[', each root entry R_i, R_i.hash, and ']'. When the object is dequeued from the PQ, we replace R_i, R_i.hash with '[', each R_j and R_j.hash (or O_j) in R_i's child node, and ']'.

We give an example of the local authenticated kNN processing using Figure 2. Assume that k=2 and the red point Q is the query point. At first, the PQ contains N_1, N_2 and the VO is [[N_1,$H(N_1)$], [N_2,$H(N_2)$]]. Next, N_1 is removed and N_3, N_4 are added into the PQ. The VO changes to [[[N_3,$H(N_3)$],[N_4,$H(N_4)$]],[N_2,$H(N_2)$]. Finally, points c and b are computed as the results and the VO updates to [[[a,b,c],[N_4,$H(N_4)$]],[N_2,$H(N_2)$]]. The grey nodes in Figure 2(b) are included in the VO. We omit the client verification part here as we will illustrate the verification for the distributed kNN authentication in Section III-D.

III. Distributed kNN authentication

A. Problem Formulation

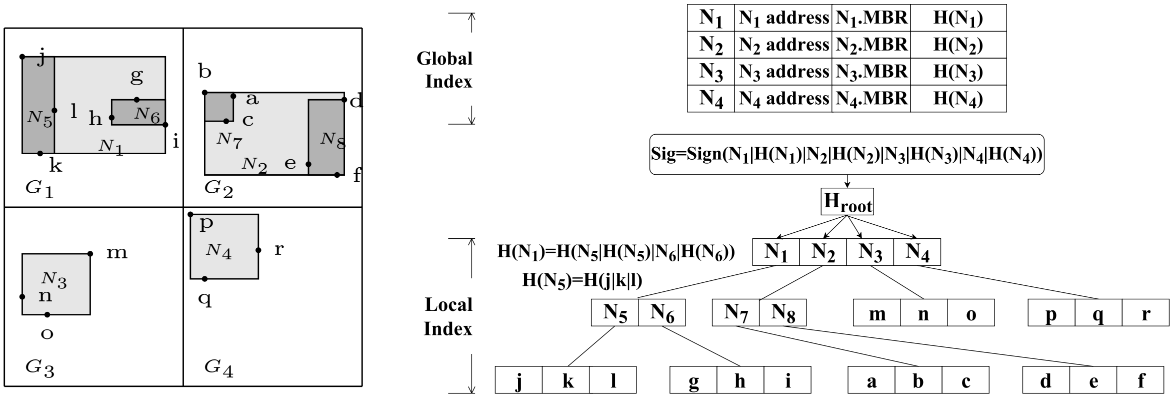
There are three parties in our system: the data owner (DO), the third-party distributed service provider (SP), and the client. The DO builds the ADS and signs the ADS to ensure the integrity of the data. The ADS and the signature are then sent to the SP. The distributed SP provides the service of storage and query processing. Since the SP is configured in a distributed environment, it consists of several types of nodes: $Master$, $Slave$, and $Reducer$. The $Master$ node in the SP is mainly responsible for dispatching jobs to the corresponding slaves. The $Slave$ nodes are the workers, which process the actual queries. The $Reducer$ consolidates all the partial results as well as the VOs computed by the $Slaves$ and sends them to the client. Finally, the client can verify the soundness and completeness of the results by using the VO, the DO's public key, and the root signature sent by the SP.

Threat Model In this system, we consider the DO as the trusted party. The SP is untrusted and can return incorrect results. Given the whole dataset D, the client's query point q, and the parameter k, the SP returns several results. If the result number does not match the value k, the client can find that some results are missing. Suppose $R_k = \{r_1, r_2, ..., r_k\}$ are the k true kNN results. The SP can (1) return a point p and $p \notin D$; (2) return a point $p \in D$ but $dist(q, p) > dist(q, r_k)$,where $dist(\cdot, \cdot)$ denotes the Euclidean distance. The first and second cases violate the soundness and completeness conditions, respectively.

The system's performance can be measured in these metrics: (1) ADS construction time; (2) query processing cost; (3) client's verification time; (4) VO size. We assume that there is no data update in this system. Therefore, the ADS construction is a one-off operation by the DO and the cost can be amortized by the queries. The VO size can influence the communication overhead between the SP and the client as well as the time of client's verification. Therefore, the VO size should be minimized.

B. Distributed MR-Tree

To adapt the MR-tree index structure to the distributed environment, the index structure has two layers: the local index and the global index. The DO first employs the Grid partition or the Sort-Tile-Recursive [17] method to partition the entire dataset into several splits. These methods guarantee that there is no overlap between any two partitions' minimum bounding rectangles (MBRs). Thanks to this non-overlapping characteristic, only a few partitions are used to process the kNN query, which saves the computation resources. The local indexes are constructed using the data points in each partition. The global index is then constructed, and it contains each local MR-tree's root MBR and, root hash value, and the pointers towards each local index. After the construction of the index structure, the DO signs the root node of the global index using the private key. Then, the signature and the entire index will be sent to the SP. The $Master$ of the SP is responsible for dispatching the local indexes to the $Slaves$ according to the current workload of the distributed system. Since the global index is small, it will be stored in the main memory of all the


j
g
b
a
d
N_5
l
h
N_6
c
N_7
N_8
N_1
i
N_2
e
f
k
G_1
G_2
p
m
N_4
r
N_3
n
q
o
G_3
G_4
Global Index
N_1 | N_1 address | N_1.MBR | $H(N_1)$
N_2 | N_2 address | N_2.MBR | $H(N_2)$
N_3 | N_3 address | N_3.MBR | $H(N_3)$
N_4 | N_4 address | N_4.MBR | $H(N_4)$
Sig=Sign(N_1|H(N_1)|N_2|H(N_2)|N_3|H(N_3)|N_4|H(N_4))
H_{root}
H(N_1)=H(N_5|H(N_5)|N_6|H(N_6))
H(N_5)=H(j|k|l)
Local Index
N_1 N_2 N_3 N_4
N_5 N_6
N_7 N_8
m n o
p q r
j k l
g h i
a b c
d e f

(a)

(b)

Fig. 3. Data points and index

nodes, which can speed up the kNN query processing. Figure 3(a) shows the four partitions using the Grid partition method. Figure 3(b) depicts the entire structure of distributed MR-tree. In each partition, the data points are used to build a local MR-tree. We use N_1, N_2, N_3 and N_4 to represent the four local indexes. The global index is a directory table and can prune the search space.

C. Authenticating Distributed kNN Query Processing

For easy illustration, we first give some definitions, which will be used in the authenticating distributed kNN query processing.

Definition 1: Given a query point q and a set of local MR-trees T={T_1, T_2, ... , T_n}, T_i is a *home index* if $dist(q, T_i.MBR) < dist(q, T_j.MBR)$ where $i, j \in [1..n]$ and $j \neq i$. Here the $dist(\cdot,\cdot)$ denotes the minimum distance between a point and a rectangle.

Definition 2: Given a query point q and a set of current partial kNN results $\{r_1, r_2, ..., r_k\}$, a *rcircle* is a circle centered at q with the radius of $dist(q, r_k)$. Here the $dist(\cdot,\cdot)$ denotes the Euclidean distance.

Definition 3: Given a *rcircle* and a set of local MR-trees T={T_1, T_2, ... , T_n}, CT is a set of *candidate trees* if $CT \subseteq T$ and for each $CT_i \in CT$, $CT_i.MBR \cap$ *rcircle* $\neq \emptyset$ and $CT_i \neq$*home index*.

Definition 4: Given a set of slave nodes $Slaves = \{Slave_1$, ..., $Slave_n\}$, a *home slave* $HSlave \in Slaves$ is the slave which stores the *home index*. And a set of *candidate slaves* $CSlaves \subseteq Slaves$ are the set of slaves which stores *candidate trees CT*.

Definition 5: We define the VO computed by *HSlave* is hVO; the VO computed by *CSlaves* is cVO; the VO of non-processed slaves is nVO.

Definition 6: We define the results computed by *HSlave* are in $\{rs_h\}$; the results computed by *CSlaves* are in $\{rs_c\}$.

We assume that in the distributed system, queries are executed by the slaves who store the corresponding local indexes. For example, since the $HSlave$ stores the *home index*, the first local kNN query should be sent to the $HSlave$ by the $Master$ and the $HSlave$ will execute the query locally.

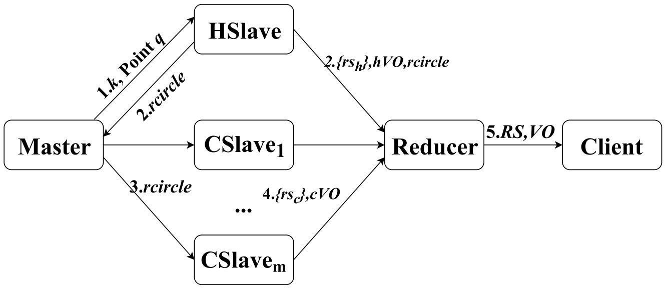
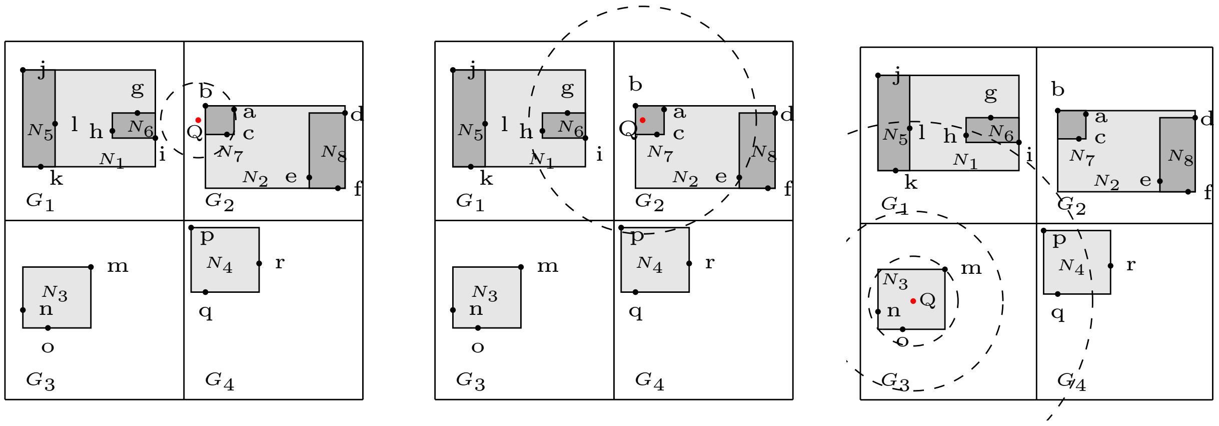


Fig. 4. Framework of distributed authenticated kNN query

Our query authentication framework consists of three main procedures: (1) the $HSlave$ processes the local authenticated kNN query to compute $\{rs_h\}$ and hVO; (2) $CSlaves$ process additional range queries to compute $\{rs_c\}$ and the cVO; (3) the $Reducer$ finally selects the k nearest results from the partial results and generates the overall VO. Figure 4 depicts the overview of the basic distributed kNN query processing.

The main objective is to use a small amount of partitions to compute the correct kNN results as well as the VO. The non-overlapping characteristic of the partition method prunes some local indexes, which saves the computation resources. The $HSlave$ first finds the local kNN results in the *home index* which contains the correct results in a high probability. However, if there are some other points which are closer than the ones in $\{rs_h\}$, $CSlaves$ need to process extra range queries to find the correct results. The $rcircle$ is used to check whether $CSlaves$ should process the extra queries. Algorithm 2 summarizes the procedure of the $Master$ node and the $Reducer$ node. The procedure of the $HSlave$ and the $CSlave$ is omitted because they simply compute the local queries and send the partial VO and results to the $Reducer$.

After receiving the kNN query from the client, the $Master$ locates the *home index* and sends the kNN query request to



(a) case 1

(b) case 2

(c) special case

Fig. 5. Three cases of kNN processing

Algorithm 2 Distributed Authenticated kNN Procedure

```
Master Procedure
Input $k$, Point $q$
Output kNN request, range query requests
Send $kNN$ request to $HSlave$
Receive $rcircle$, decide $CT$ list and $CSlaves$
if $CT \neq \emptyset$ then
    Send range query with $rcircle$ to the $CSlaves$
end if
Reducer Procedure
Input $rs_h$, $hVO$, $rcircle$, $rs_c$, $cVO$
Output $RS$, $VO$
Receive $rs_h$, $hVO$, $rcircle$ from $HSlave$
Compute $CT$ list using $rcircle$ and global index
if $CT$==$\emptyset$ then
    Add $hVO$ and $nVO$ to $VO$
    Send $rs_h$ and $VO$ to the $Client$
else
    Receive $\{rs_c\}$ and $cVO$ from $CSlaves$
    Add $k$ nearest results in $rs_c \cup rs_h$ to $RS$
    Add $hVO$, $cVO$, $nVO$ to $VO$
    Send $RS$ and $VO$ to the $Client$
end if
```

the *HSlave*. The hVO, $\{rs_h\}$ and $rcirlce$ will be computed by $HSlave$ using Algorithm 1 and are sent to the *Reducer*. Also, the $rcirlce$ is sent to the $Master$. Given the $rcircle$, the *candidate trees* and the corresponding $CSlaves$ are identified (Definition 3 and 4) by the $Master$. When the CT list is not empty, the $rcircle$ is sent to the $CSlaves$ by the $Master$. Each of the $CSlaves$ computes the local range query in parallel. The $Reducer$ consolidates rs_h, $\{rs_c\}$, hVO, cVO, and nVO. Then, it selects k nearest results from the union of $\{rs_h\}$ and $\{rs_c\}$. The final VO list consists of hVO, cVO, and nVO. The nVO can be derived using the global index and the $rcircle$. If a local index's MBR does not intersect with the $rcircle$, the MBR and hash value of the index will be added into nVO.

Figure 5 shows some cases of the distributed kNN processing. In case 1, we assume that k equals 3 and Q is the query point. According to Definition 1, the *home index* is N_2. Then the $Master$ sends the kNN request to the corresponding *HSlave*. The $\{rs_h\}$ includes $\{b,c,a\}$ and the hVO is $\{[[a,b,c], [N_8,H(N_8)]]\}$. Since the $rcircle$ has no intersection with other local tree's MBR, the CT list is empty. The non-processed VO set nVO is $\{[N_1,H(N_1)], [N_3,H(N_3)], [N_4,H(N_4)]\}$. Finally, the *Reducer* sends the union of hVO and nVO and the results to the client.

In case 2, we assume that k equals 4. The query point Q locates in N_2. The corresponding $HSlave$ computes the $rs_h = \{b, c, a, e\}$ and also the hVO=$\{[[a,b,c],[d,e,f]]\}$. The radius of $rcircle$ is the distance between Q and point e. Using the global index and $rcircle$, the $Master$ node finds the $CT = \{N_1, N_4\}$ and sends the range query requests to the $CSlaves$. The two local result sets $\{rs_c\}_{N_1} = \{g, h, i\}$ and $\{rs_c\}_{N_4} = \{p\}$ are computed by each $CSlave$. Meanwhile, cVO=$\{[[N_5,H(N_5)],[g,h,i]],[p,q,r]\}$ is computed. The *Reducer* selects the 4 nearest result points from the union of partial results and the final result set is $RS = \{b, c, a, i\}$. The local tree N_3 does not intersect with the $rcircle$, so that $[N_3,H(N_3)]$ will be added to the nVO. The final VO consists of hVO, cVO, and nVO.

There is a special case of the distributed kNN processing when the number of data points in a partition is less than k. A skewed distribution of data points may lead to this case. The previous algorithm cannot compute the correct kNN results. In this case, we double the radius of the $rcircle$ and process the range query iteratively. The $Reducer$ checks the result number and sends the message to the $Master$ if the result number is less than k. The $Master$ doubles the $rcircle'$s radius and the new $rcircle$ is sent to the updated $CSlaves$. This process runs iteratively until the result number exceeds k. Figure 5(c) illustrates the special case. Assume that k is 4 and Q is the query point. N_3 is the *home index*. During the third iteration, we find 8 points and the iteration terminates. Finally, the $Reducer$ selects 4 nearest points and returns the final RS and VO to the client. In this special case, the VO size is rather large since the radius of $rcircle$ grows exponentially. To avoid this case, in our experiments we use the Sort-Tile-Recursive (STR) [17] partition method to split the dataset. The STR partition splits the near data points in the same partition and each partition has nearly the same number of data points.

D. Client Verification

The client uses the VO to verify the results' soundness and completeness. The soundness is verified first by reconstructing the root hash value of the distributed MR-tree. The root hash value is the hash value of the concatenation of each local tree's MBR and hash value. If the reconstructed hash value matches the root hash decrypted by the public key of the DO, the soundness is satisfied. The completeness can be verified in the following method. All of the data points and MBRs are extracted from the VO set and the client selects the first k nearest data points and compares them with the RS. If they match, it means that no other points are closer than the RS data points. Also, all of the $MBRs$ must be farther than the k^{th} result point. Algorithm 3 summarizes the procedure of client's verification.

Algorithm 3 Client Verification

```
Input $k$, Point $q$, $RS$, $VO$
Output Boolean
$DOroot$ = decryptedByPublicKey($sig$)
$root$ = reconstruct($VO$)
if $DOroot \neq root$ then
    return False;
end if
Extract datapoints and MBRs from $VO$ and put them into $rs$, $mbr$ list
Sort $rs$ by distance to $q$
for $i = [1 : k]$ do
    if $RS[i] \neq rs[i]$ then
        return False
    end if
end for
for $obj$ in $mbr$ do
    if $dist(q, kobj[k]) > dist(q, obj)$ then
        return False
    end if
end for
```

Take Figure 5(b) as an example. The overall VO consists of hVO={[[a,b,c], [d,e,f]]}, cVO={[[N_5,$H(N_5)$], [g,h,i]], [p,q,r]} and $nVO = \{[N_3, H(N_3)]\}$. The client can recompute N_2 using hVO. N_7, N_8 as well as their corresponding hash values are derived by {a,b,c} and {d,e,f}, respectively. $H(N_7)$ is the hash value of the concatenation of points a, b, and c. Similarly, $H(N_8)$ can be computed. N_2 and $H(N_2)$ are generated by N_7, N_8, and their hash values. Points g, h, and i are used to compute N_6 and $H(N_6)$. N_1 with its hash value can be derived by N_5, N_6, and their hash values. N_4 and $H(N_4)$ are computed using points p, q, and r in cVO. The N_3 and $H(N_3)$ are given in nVO. Therefore, the root hash value is computed using four local MR-trees' root hash values and MBRs. After that, the data points and MBRs are extracted from the union of VO. The correct results are {b,c,a,i} and all of the non-result points will be checked whether they are closer than point i. Meanwhile, the MBR of N_5 and N_3 are checked whether they are farther from point i.

E. Robustness Analysis

Lemma 1: If there is no need to invoke the range queries ($CT = \emptyset$), the local kNN result rs_h will be the final correct results.

Proof: This can be proved by contradiction. If there is a point p which is one of the correct kNN results but not included in the local kNN result, it is either inside or outside the partition. If it is inside the partition, it should be outside the $rcircle$ since all of the points in the $rcirlce$ are added into rs_h. There are k points inside the $rcircle$, so p must be at least $(k+1)^{th}$ nearest point. This contradicts the assumption that it is one of the correct results. If it is outside the partition, as there is no intersection among the partitions, it must be farther than the current k^{th} result and this contradicts the correct result assumption. □

Lemma 2: All of the correct kNN results must reside in the $rcircle$.

Proof: This can be proved by contradiction. Suppose there is a correct kNN result p which resides outside the $rcircle$. The radius of $rcircle$ is computed by the distance between the k^{th} current nearest point and the query point. There are at least k data points inside the $rcircle$ because apart from the k local results, there are probably some points in other partitions that are closer than the current k^{th} point. Since p is outside the $rcircle$, it is at least the $(k+1)^{th}$ nearest point, which contradicts the assumption. □

Theorem 1: Algorithm 2 can produce the correct result set.

Proof: This can be proved using Lemmas 1 and 2. □

Algorithm 3 summarizes the procedure of client verification. The root hash reconstruction guarantees the $soundness$ of the results and the distance comparison protects the $completeness$. We give the following theorem of its correctness of verification.

Theorem 2: Algorithm 3 verifies the $soundness$ and $completeness$ of the results and can detect the threats mentioned in Section III-A.

Proof: If the adversary returns a fictitious point, which is not included in the dataset, the reconstructed root hash must be different from the decrypted hash value received from the SP because of the collision-resistance characteristic of the hash function. If one data point changes, its corresponding hash value will be different, resulting in the change of the root hash value of MR-tree. Without knowing the secret key, the SP cannot generate a valid signature of the wrong root hash value, which means that the signature is unforgeable. Therefore, the violation of soundness can be detected. If the reconstructed hash value matches the one decrypted, the incorrect result point must reside in the VO set but farther than the correct results. After the client sorts all the data points in the VO set by distance and gets the first k nearest points, the client can notice the difference between k points extracted from the VO and the points in RS. Then the results cannot pass the verification of the completeness. □

IV. Optimization of VO size

The VO size determines the communication cost and also the client verification time. In this section, we focus on the optimization of the VO size. Intuitively, the $rcircle$ decides the VO size. A larger $rcircle$ makes more nodes to be visited and hence and more data points to be added to the VO. Therefore, the objective is to reduce the radius of the $rcircle$. As we can see in Figure 5(b), the $rcircle$ is much bigger than the circle with the radius of the distance between Q and point i. This is because the distance between Q and e is larger than the distance between Q and i. In Algorithm 2, the $rcircle$ is decided by the local results in *home index* and will not change during the whole procedure. To compute the minimum $rcircle$, the optimized algorithm adds the sequential computation of the $rcircle$ before the parallel authenticated range query in Algorithm 2.

```
Algorithm 4 VO Optimization Algorithm
Master Procedure
Input $k$, Point $q$
Output kNN request, range query requests
Send $kNN$ request to $HSlave$
Receive $rcircle$
Compute $CT$ list and decide $CSlaves$
if $CT \neq \emptyset$ then
    Sort $CT$ list with minimum distance
    for $i = 1; i \leq |CT|; i++$ do
        if $!rcircle$.intersects $CT[i]$.MBR then
            break
        end if
        Send $rcircle$ to $CSlave[i]$
        Receive $rs_c$ and $rs_h += rs_c$
        Select $k^{th}$ nearest point $kobj$ in $rs_h$
        Update $rcircle$'s radius with $dist(kobj, q)$
    end for
    Recompute $CSlaves$ using final $rcircle_f$
    Send $rcircle_f$ to $HSlave$ and $CSlaves$
end if
```

Algorithm 4 summarizes the procedure of the $Master$ node. The procedure of slaves is omitted because they simply compute the partial results and VOs. First, the $Master$ sends the kNN request to the $HSlave$ and uses the received $rcircle$ to decide the candidate tree list CT and $CSlaves$. If the CT list is not empty, the $Master$ sequentially updates the $rcircle$. Since the closer local indexes have the kNN results in a higher probability, the CT list is sorted with the minimum distance. In the *for* loop in Algorithm 4, the $rcircle$ will be sequentially sent to each $CSlave$ and the rs_c will be added into rs_h to find the current k^{th} nearest point $kobj$, which is used to update the $rcircle$. Note that the *for* loop terminates immediately when the updated $rcircle$ does not intersect with the $CT[i].MBR$. This prunes some search space and improves the performance. The $CSlaves$ list is recomputed after the computation of the final $rcircle$. Since the $rcircle$ changes, the rs_h and hVO are recomputed. Also, the $Reducer$ should discard the previously received rs_h and hVO when it finds that CT list is not empty. The $Master$ sends the $rcircle_f$ to the $HSlave$ and the $CSlaves$, which compute the range query and VOs in parallel. The $Reducer$ receives all kinds of results and VOs from the $Slaves$. Then it selects k nearest results and adds them to the RS list. The nVO is also added into the VO. Finally, the RS and VO are sent to the $Client$.

Figure 6 shows an example of the VO optimization. We assume that $k = 4$ and the query point Q lies in N_2. The *home index* is N_2 and the ordered CT list is $\{N_1, N_4\}$. The outer dashed circle is the initial $rcircle$ computed using the $\{rs_h\}$. The hVO is $\{[[a,b,c],[d,e,f]]\}$. Then the range query will be processed on N_1 first. The result of the range query rs_c is $\{g, h, i\}$. The $Master$ next finds the point i, which is the 4^{th} result in the union of $\{rs_c\}$ and $\{rs_h\}$. The $rcircle$ will be updated to the smaller dashed circle in Figure 6. Since the updated $rcircle$ does not intersect with the next local index N_4 in the CT list, the *for* loop terminates. The CT list and the $CSlaves$ are recomputed. The $rcircle_f$ will be sent to the $HSlave$ and $CSlave$ storing N_2 and N_1, respectively. The $Reducer$ receives rs_h=$\{a,b,c\}$, hVO=$[[a,b,c][N_8,H(N_8)]]$, rs_c=$\{i\}$, and cVO=$[[N_5,H(N_5)][g,h,i]]$. The final results contain $\{b,c,a,i\}$. The nVO contains $[N_3,H(N_3)]$ and $[N_4,H(N_4)]$. The updated $rcircle$ avoids the visit of local index N_4 and also the subtree N_8. Therefore the VO size is reduced by this algorithm.

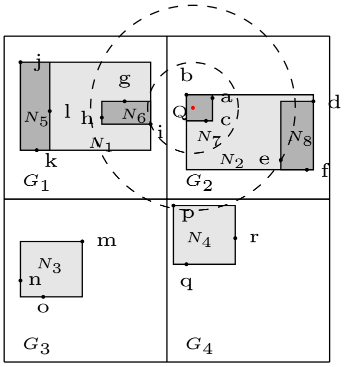


Fig. 6. Example of optimization of VO size

In Algorithm 4, the $rcircle$ is updated iteratively in the *for* loop. The definition of the $rcircle$ remains to be the same. As such, the correctness of results and VO generation can still be proved using Lemmas 1 and 2.

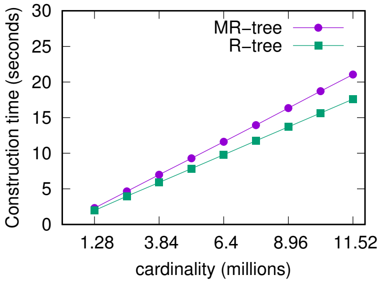
Obviously, Algorithm 4 spends more time than Algorithm

2 in query processing because of the sequential update of $rcircle$. If the $|CT|$ is large, the sequential update consumes too much time. Therefore, we design a hybrid algorithm, in which the $rcircle$ is updated only for l times, where $1 \leq l \leq |CT|$. Here the value l denotes the level of sequential processing. After l sequential queries, we resort to Algorithm 2 for parallel processing of distributed authentication. The hybrid algorithm uses less range queries to find the optimized $rcircle$ and it balances the trade-off between the VO size and the query time. Note that if the $|CT|$ equals one, the hybrid algorithm and the optimized algorithm are the same because there is only one update of the $rcircle$.

V. Performance evaluation

This section evaluates the proposed distributed query authentication algorithm and optimization techniques. We perform the query evaluation on a computer with Dual 6-core Intel Xeon E5-2620 2.4GHz CPU with 128G RAM. The SP runs on Apache Spark platform locally and we set the slaves (workers) to be 8 in the system. The DO side is set up on a desktop computer with Intel Core i7 CPU and 4GB RAM, running Linux. The dataset of all of the experiments is a New York map file from OpenStreetMap containing 11.56 million data points. We assume that there is no update of the dataset since the bulk loading of ADS construction is quite efficient. Therefore, we employ bulk-loaded STR-tree as our R-tree, which is implemented in JAVA. Moreover, we select the STR as the partition method. The fan-out of the MR-tree is set to 100. The hash function is 160bit SHA-1 and we use 1024-bit RSA algorithm to sign the root hash value. Both SHA-1 and RSA are imported from the java.security package. The index construction, query processing time, result verification time, and VO size are measured.

A. Cost of Index Construction



(a)

Fig. 7. DO's construction time

Figure 7 shows the DO's construction time of the MR-tree as well as the R-tree for comparison. The construction time grows linearly with the increasing cardinality. It takes less than half a minute to construct 11.52 million records, which is acceptable. The construction of ADS is a one-time process and further queries can amortize its cost. The construction time of MR-tree is around 20% longer than that of R-tree. This is because building the MR-tree involves the computation of hash values and binary concatenation. Table I shows the index size with different dataset cardinalities. When the cardinality is 11.52 million, the size of MR-tree is only 4 MB larger than the R-tree. The reason is that the SHA-1 hash value is only 160 bit, which is rather small. As a result, it takes the DO and the SP only a few MB to store the extra authenticated data structure.

TABLE I
The size of index with different cardinalities

Cardinality (Million)	1.28	3.84	6.4	8.96	11.52
MR-tree (MB)	69.8	209.4	349	488.6	628.2
R-tree (MB)	69.4	208.2	347	485.8	624.6

B. Distributed Authenticated kNN Query Cost

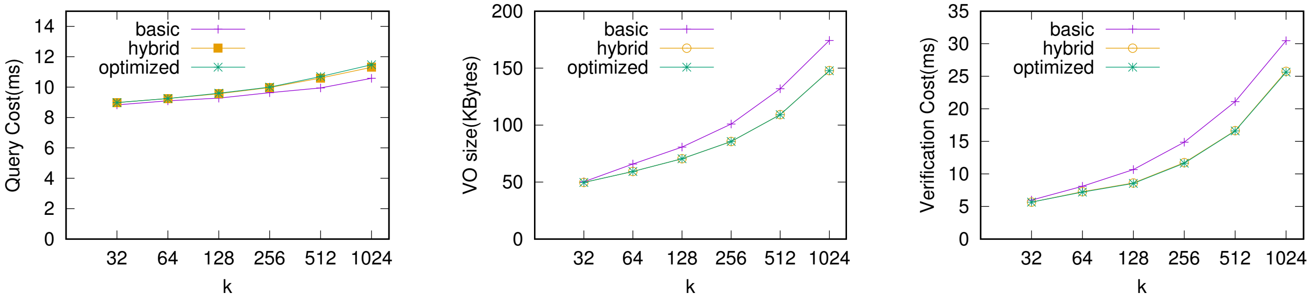
In this section, we evaluate the query cost of distributed authenticated kNN queries. The selection of the query point influences the query cost in our distributed setting. We call a kNN query as an ERQ if the $CT \neq \emptyset$ during its processing. If the query point is near the center of the partition's MBR and also the k is rather smaller, the query can be an ERQ in a small probability. However, if the query point is near the edge of the partition's MBR, even the k is small, it can be an ERQ in a higher probability. The range queries in an ERQ take more time. To test the average performance of the query cost with different k settings, we randomly select 1,000 data points located in the root MBR of the entire dataset and average the query time. We denote Algorithm 2 and Algorithm 4 as the basic and optimized algorithm, respectively. We compare the query cost performance of the basic, optimized, and hybrid algorithm, where l is set to one for simplicity.

Figure 8(a) shows that the query cost increases with the k's growth. Intuitively, searching more points in a local MR-tree takes longer time. Also, the number of $ERQs$ increases with the larger k and each ERQ takes more time. Table II shows the number of $ERQs$ in the 1,000 query points with increasing k. Furthermore, the basic algorithm runs faster than the other two optimized algorithms. The optimized algorithm and hybrid algorithm run more range queries to reduce the $rcircle$ and this leads to more local query time. The hybrid algorithm and optimized algorithm are the same when $|CT| = 1$, since both of them run one range query to reduce the $rcircle$. The efficiency of the hybrid algorithm is not obvious on average because the number of cases of $|CT| > 1$ is much less than the case of $|CT| = 1$ (see Table II).

TABLE II
# $ERQs$ in the 1000 query points

k	32	64	128	256	512	1024
$\|CT\| = 1$	38	44	51	59	77	88
$\|CT\| = 2$	0	0	1	1	2	3
$\|CT\| = 3$	0	0	0	0	1	2
total # $ERQs$	38	44	52	60	80	93

Figure 8(b) shows the average VO size with the three algorithms. Both of the optimized and hybrid algorithm reduce



(a) Query cost

(b) VO size

(c) Verification cost

Fig. 8. Query authentication performance

the VO size. In most cases, the query is not an ERQ (see Table II, there are only 93 ERQ queries out of the 1,000 queries when $k = 1,024$), which leads to the same VO size using the basic and two optimized algorithms. When k reaches 1,024, the basic algorithm outputs 174.32 KB VO on average and the two optimized algorithms output 147.82 KB and 147.72 KB on average, respectively. The curve of the hybrid and optimized algorithm overlap heavily because of the limited number of cases that $|CT| > 1$.

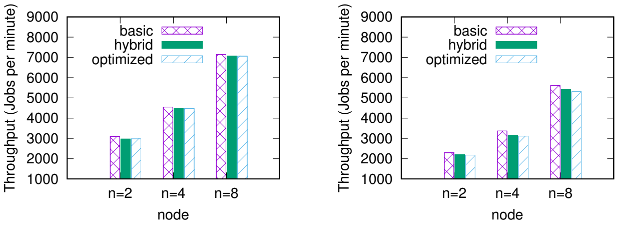
In order to compare the hybrid and optimized algorithm, we set $k = 1,024$ and choose three types of query points with different $|CT|$ from the previous selected 1000 queries and evaluate the VO size and query cost. The r_{hyb} is defined as $\frac{VO_{basic} - VO_{hybrid}}{VO_{basic}}$ and r_{opt} is similar. The t_{bas}, t_{hyb}, and t_{opt} represent the query cost of the three algorithms, respectively. In Table III, both the optimized and hybrid algorithm output smaller VO ($r_{hyb} > 0$, $r_{opt} > 0$) and take more query time ($t_{hyb} > t_{bas}$, $t_{opt} > t_{bas}$), compared with the basic algorithm. Moreover, the hybrid algorithm takes less query time ($t_{hyb} \leq t_{opt}$) and outputs larger VO ($r_{hyb} \leq r_{opt}$), compared to the optimized algorithm.

TABLE III
COMPARISON OF ERQ WITH DIFFERENT $|CT|$

type	r_{hyb}	r_{opt}	t_{bas}(ms)	t_{hyb}(ms)	t_{opt}(ms)
$\|CT\| = 1$	14.1%	14.1%	14.19	18.78	18.8
$\|CT\| = 2$	30%	40.2%	16.62	19.38	23
$\|CT\| = 3$	32.5%	43.3%	16.7	21.08	25.9

Figure 8(c) shows the average client verification time. Since the hybrid algorithm and the optimized algorithm output nearly the same size of VO on average, the two curves overlap. Obviously, the verification time is proportional to the VO size. The reduction of verification time becomes more obvious when k is larger and this corresponds to the reduction of VO size. When k is larger, the number of points in the VO increases. Also, each extra range query leads to visit of the corresponding index and contributes to a larger VO. Therefore, the client takes more time to reconstruct the root hash value of each local index. To verify the completeness of the results, the client should sort the results and also the points inside the VO. Therefore, the total verification time increases with larger k.

Finally, we evaluate how the three algorithms scale out with the slave node size varying from 2 to 8. The node size influences the performance of parallel computing and also the throughput of the system. If the number of slave nodes is small, there will be more partitions (local indexes) distributed into the same node, which leads to slower processing of the range query due to the limited cores. We test the throughput when $k = 32$ and $k = 1,024$. The throughput is calculated as the number of jobs completed per minute.



(a) $k = 32$

(b) $k = 1024$

Fig. 9. Throughput with different nodes

Figure 9 shows the throughput of the three algorithms under different node numbers. We can infer that the three algorithms scale well with the node number. The throughput of the hybrid and optimized algorithm is less than the throughput of the basic algorithm since the two optimized algorithms need more range queries to reduce the $rcircle$. The throughput of the three algorithms has similar performance when $k = 32$, because the ratio of ERQ is small. However, the throughput of the basic algorithm is obviously higher than the throughput of two optimized algorithms when $k = 1,024$ since the ratio of ERQ is higher and there are more cases of $|CT| > 1$.

VI. RELATED WORK

Authenticated query processing has been extensively investigated to verify the authenticity of the results computed by the service provider in the data outsourcing scenario. The Merkle Hash Tree (MHT), which is one of the important ADSs, has been proposed in [15] to verify a series of data points. By adding the signature chains as the authentication information to the spatial data structure, Cheng *et al.* [6] proposed VKDtree and VRtree to verify the results in multi-dimensional databases. Yang *et al.* [3] combined the MHT with the traditional spatial data structure R-tree and proposed the MR-tree with its variant to efficiently verify the spatial data. Cheng and Tan [2] extended the signature chain method and proposed a kNN authentication mechanism on multi-dimentional databases. A set of data pairs are returned to the client apart from the k results to guarantee the authenticity and the completeness.

More recently, moving kNN has been investigated by Yiu *et al.* [4]. Hu *et al.* [18] embedded the Voronoi neighbor information in the signature of the spatial datasets and proposed the VN-Auth to verify kNN, range queries and other advanced spatial queries. Hu *et al.* [10] addressed the preservation of the location privacy while processing authenticated range queries. Xu *et al.* [12] recently integrated the relational query authentication with fine-grained access control. There are also some other types of authenticated query processing. Chen *et al.* [7] developed a novel top-k query authentication based on the cryptography building blocks. Xu *et al.* [11] considered both authenticity and confidentiality for aggregate queries over set-valued data. Lin *et al.* [8] proposed a new ADS called MR-Sky-tree to facilitate the location-based authenticated skyline queries. Chen *et al.* [9] proposed the authenticated online data integration, which can support multi-source query authentication.

Meanwhile, there are a large body of research on distributed spatial queries. Aji *et al.* designed the Hadoop-GIS system [19], which is a spatial data warehousing system and integrates Hive. SpatialHadoop [13] adds the traditional spatial indexes to the native Hadoop framework and supports a variety of spatial queries like range query, kNN, spatial join etc. For in-memory computation over cluster machines, GeoSpark [14] has been proposed to support spatial queries like range, join, and kNN queries. More recently, an more efficient in memory spatial analytics system, called Simba [20], has been proposed. It supports rich spatial queries and has better throughput than SpatialHadoop and GeoSpark. Nevertheless, none of the system supports the authenticated query processing to guarantee the results' integrity.

VII. Conclusion

In this paper, we investigate the authenticated kNN processing in distributed environments. The distributed MR-tree has been designed to speed up the processing of kNN queries. We propose a basic algorithm to authenticate kNN queries in a distributed fashion, as well as two optimization techniques to reduce the VO size. Our experiments demonstrate that the query costs of the three algorithms are all efficient. The two optimization techniques reduce the VO size and the client's verification time. The hybrid algorithm balances the trade-off between the query cost and the VO size. Furthermore, the throughput of all the algorithms is investigated and the result reveals that the system scales well with increasing the number of nodes. In the future, we plan to design a distributed spatial query authentication system to support more spatial queries and efficient data updates.

Acknowledgements

This work was supported by Research Grants Council of Hong Kong under GRF Projects 12244916, 12202414, 12232716, and CRF Project C1008-16G.

References

[1] P. Devanbu, M. Gertz, C. Martel, and S. G. Stubblebine, "Authentic data publication over the internet1," *Journal of Computer Security*, vol. 11, no. 3, pp. 291–314, 2003.

[2] W. Cheng and K. L. Tan, "Authenticating kNN query results in data publishing," in *VLDB Conference on Secure Data Management*, 2007, pp. 47–63.

[3] Y. Yang, S. Papadopoulos, D. Papadias, and G. Kollios, "Authenticated indexing for outsourced spatial databases," *Vldb Journal*, vol. 18, no. 3, pp. 631–648, 2009.

[4] L. Y. Man, E. Lo, and D. Yung, "Authentication of moving kNN queries," in *IEEE International Conference on Data Engineering*, 2011, pp. 565–576.

[5] S. Su, H. Yan, X. Cheng, P. Tang, P. Xu, and J. Xu, "Authentication of top-k spatial keyword queries in outsourced databases," in *DASFAA*, 2015, pp. 567–588.

[6] W. Cheng, H. Pang, and K.-L. Tan, "Authenticating multi-dimensional query results in data publishing," in *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 2006, pp. 60–73.

[7] Q. Chen, H. Hu, and J. Xu, "Authenticating top-k queries in location-based services with confidentiality," *Proceedings of the Vldb Endowment*, vol. 7, no. 1, pp. 49–60, 2014.

[8] X. Lin, J. Xu, and H. Hu, "Authentication of location-based skyline queries," in *ACM International Conference on Information and Knowledge Management*, 2011, pp. 1583–1588.

[9] Q. Chen, H. Hu, and J. Xu, "Authenticated online data integration services," in *Proceedings of the 2015 ACM SIGMOD International Conference on Management of Data*. ACM, 2015, pp. 167–181.

[10] H. Hu, J. Xu, Q. Chen, and Z. Yang, "Authenticating location-based services without compromising location privacy," in *Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data*. ACM, 2012, pp. 301–312.

[11] C. Xu, Q. Chen, H. Hu, J. Xu, and X. Hei, "Authenticating aggregate queries over set-valued data with confidentiality," *IEEE Transactions on Knowledge and Data Engineering*, 2017.

[12] C. Xu, J. Xu, H. Hu, and M. H. Au, "When query authentication meets fine-grained access control: A zero-knowledge approach," in *Proceedings of the 2018 ACM SIGMOD International Conference on Management of Data*. ACM, 2018.

[13] A. Eldawy and M. F. Mokbel, "Spatialhadoop: A MapReduce framework for spatial data," in *IEEE International Conference on Data Engineering*, 2016, pp. 1352–1363.

[14] J. Yu, J. Wu, and M. Sarwat, "Geospark: a cluster computing framework for processing large-scale spatial data," in *Sigspatial International Conference on Advances in Geographic Information Systems*, 2015, p. 70.

[15] R. C. Merkle, "A certified digital signature," in *Advances in Cryptology - CRYPTO '89, International Cryptology Conference, Santa Barbara, California, Usa, August 20-24, 1989, Proceedings*, 1990, pp. 218–238.

[16] G. R. Hjaltason and H. Samet, "Distance browsing in spatial databases," *ACM Transactions on Database Systems (TODS)*, vol. 24, no. 2, pp. 265–318, 1999.

[17] S. T. Leutenegger, J. M. Edgington, and M. A. Lopez, "STR: A simple and efficient algorithm for R-Tree packing," in *International Conference on Data Engineering, 1997. Proceedings*, 1997, pp. 497–506.

[18] L. Hu, W. S. Ku, S. Bakiras, and C. Shahabi, "Spatial query integrity with voronoi neighbors," *IEEE Transactions on Knowledge and Data Engineering*, vol. 25, no. 4, pp. 863–876, 2013.

[19] A. Aji, F. Wang, H. Vo, R. Lee, Q. Liu, X. Zhang, and J. Saltz, "Hadoop-gis: A high performance spatial data warehousing system over mapreduce," 2013, pp. 1009–1020.

[20] D. Xie, F. Li, B. Yao, G. Li, L. Zhou, and M. Guo, "Simba: Efficient in-memory spatial analytics," in *International Conference on Management of Data*, 2016, pp. 1071–1085.