DOI: 10.1109/TKDE.2023.3283020

Efficient Privacy-Preserving Spatial Data Query in Cloud Computing

Yinbin Miao, *Member, IEEE*, Yutao Yang, Xinghua Li, Linfeng Wei, Zhiquan Liu, *Member, IEEE*, and Robert H. Deng, *Fellow, IEEE*

***Abstract*—With the rapid development of geographic location technology and the explosive growth of data, a large amount of spatial data is outsourced to the cloud server for reducing the local high storage and computing burdens, but at the same time causes security issues. Thus, extensive privacy-preserving spatial data query schemes have been proposed. Most of the existing schemes use Asymmetric Scalar-Product-Preserving Encryption (ASPE) to encrypt data, but ASPE has proven to be insecure against known plaintext attack. And the existing schemes require users to provide more information about query range and thus generate a large amount of ciphertexts, which causes high storage and computational burdens. To solve these issues, based on enhanced ASPE designed in our conference version, we first propose a basic Privacy-preserving Spatial Data Query (PSDQ) scheme by using a new unified index structure, which only requires users to provide less information about query range. Then, we propose an enhanced PSDQ scheme (PSDQ$^+$) by using Geohash-based R-tree structure (called GR-tree) and efficient pruning strategy, which greatly reduces the query time. Formal security analysis proves that our schemes achieve Indistinguishability under Chosen Plaintext Attack (IND-CPA), and extensive experiments demonstrate that our schemes are efficient in practice.**

***Index Terms*—Cloud server, privacy-preserving, query range, security issues, spatial data.**

Manuscript received 21 June 2022; revised 2 May 2023; accepted 1 June 2023. Date of publication 5 June 2023; date of current version 27 November 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62072361, in part by the Key Research and Development Program of Shaanxi under Grant 2022GY-019, in part by Shaanxi Fundamental Science Research Project for Mathematics and Physics under Grant 22JSY019, in pat by the National Natural Science Foundation of China under Grant 62125205, in part by Fundamental Research Funds for the Central Universities under Grant QTZX23091, in part by Fellowship of China Postdoctoral Science Foundation under Grant 2022T150507, and in part by the Opening Project of Intelligent Policing Key Laboratory of Sichuan Province under Grant ZNJW2023KFMS002. Recommended for acceptance by X. Xiao. *(Corresponding author: Zhiquan Liu.)*

Yinbin Miao and Yutao Yang are with the School of Cyber Engineering, Xidian University, Xi'an 710071, China (e-mail: ybmiao@xidian.edu.cn; yangyutao_13@163.com).

Xinghua Li is with the State Key Laboratory of Integrated Service Networks, School of Cyber Engineering, Xidian University, Xi'an 710071, China, and also with the Engineering Research Center of Big data Security, Ministry of Education, Xi'an 710071, China (e-mail: xhli1@mail.xidian.edu.cn).

Linfeng Wei is with the School of Cyber Security, Jinan University, Guangzhou 510632, China (e-mail: twei@jnu.edu.cn).

Zhiquan Liu is with the College of Cyber Security, Jinan University, Guangzhou 510632, China (e-mail: zqliu@vip.qq.com).

Robert H. Deng is with the School of Information Systems, Singapore Management University, Singapore 178902 (e-mail: robertdeng@smu.edu.sg).

Digital Object Identifier 10.1109/TKDE.2023.3283020

I. Introduction

With the increasing popularity of positioning devices in the mobile Internet, Location Based Services (LBS) [1], [2], [3] are becoming more and more popular. The goal of LBS is to find Points of Interest (PoIs) that satisfy the user's query requirements. Generally, PoIs contain spatial location information and keywords, which are also called as spatial data objects. With the continuous enrichment of query services, spatial data query is also widely used in various fields such as spatial crowdsourcing [4], task recommendation [5], ride service [6], smart city [7], and epidemic prevention and control [8]. Due to the explosive growth of data, more and more data owners tend to outsource their spatial data to cloud servers for reducing local storage and computing burdens, but this operation is subject to privacy leakages. Thus, the privacy-preserving spatial data query has been extensively studied. However, there are still two issues to be solved.

The first issue is that most of the existing privacy-preserving spatial data query schemes require users to provide more information about the query range, which incurs huge computational and storage overheads on query users. For example, for rectangle range search [9], [10], [30], [35], users need to provide at least two 2-dimensional coordinates of the lower left corner and the upper right corner of the rectangle, which results in two ciphertexts for two coordinates respectively. And for geometric range query implemented by polynomial fitting technology [4], [23], [34], it requires users to provide an accurate closed curve formed by the query area, which also incurs two ciphertexts for the upper half and the lower half of the closed curve respectively.

The second issue is that many previous privacy-preserving spatial data query solutions [5], [13], [30], [35] based on Asymmetric Scalar-Product-Preserving Encryption (ASPE) [11] do not achieve higher security as ASPE is insecure against known-plaintext attack [12]. Although there are already some schemes to enhance the security of ASPE, there are still some problems. For example, [13] proposed an enhanced version of ASPE by adding Laplacian noise after the index vector, namely ASPE with Noises (ASPEN), which is secure against chosen-plaintext attack and known-plaintext attack. However, this scheme will lead to inaccurate query results as the scalar product of ASPEN is disturbed by noises.

To solve the above issues, our conference version [14] proposed an Enhanced ASPE (EASPE) scheme by adding some

random numbers and a random permutation, which can achieve Indistinguishability under Chosen Plaintext Attack (IND-CPA) and guarantee that query accuracy is not affected. Based on EASPE, [14] first proposed a naive Spatial Keyword Query (SKQ) scheme by using Geohash algorithm, which only requires users to provide less information about query range. However, SKQ incurs huge computational and storage overheads. Then, [14] proposed an Lightweight SKQ (LSKQ) scheme by designing an efficient unified index structure for spatial range and multiple keywords, which greatly reduces the computational and storage costs of SKQ in all phases. However, LSKQ leads to false positives. To further improve solutions in our conference version, in this paper, we first design a new unified index construction method to improve the accuracy of [14], which dose not cause false positive. Then, we propose a Geohash-based R-tree structure (called GR-tree) and an efficient pruning strategy to further reduce query time. Compared with the solutions in our conference version [14], this paper has new contributions shown as follows:

1) We use Geohash algorithm to achieve spatial range query, which only requires users to provide less information about query range. And we first propose a linear Privacy-preserving Spatial Data Query (PSDQ) scheme by using a new unified index structure for spatial range and multi-keyword, which dose not cause any loss of accuracy.
2) Then, we propose a more efficient PSDQ scheme (PSDQ$^+$) by using Geohash-based R-tree structure (called GR-tree) and pruning strategy based on both spatial range and keywords, which achieves the sublinear search complexity.
3) We give the formal security analysis to prove that our schemes are secure against IND-CPA, and conduct extensive experiments to demonstrate that our schemes are efficient in practice, especially the query efficiency of PSDQ$^+$ is about 10x faster than that of our conference version.

II. Related Work

To achieve privacy-preserving spatial data query, we will mainly focus on the following topics such as secure spatial range query, secure keyword query, and secure spatial keyword query.

Privacy-Preserving Spatial Range Query: Spatial range query aims to find all points within the query range. To search for rectangular areas, Wang et al. [15] proposed a hierarchical encrypted index ($\hat{R}$-tree) by using ASPE, which enables secure and efficient range queries over encrypted database. But this scheme only supports single dimension range query. So Wang et al. [16] designed a tree-based public key multi-dimensional range search encryption scheme by combining R-tree and hidden vector encryption [17] to support multi-dimensional rectangular range query. To search for circular areas, Wang et al. [18] leveraged Shen-Shi-Waters [19] to build a symmetric-key circle predicate encryption scheme, which can verify whether a point is inside a circle by using a number of concentric circles. To reduce the computational and storage overheads of [18], Zheng et al. [20] devised a secure circular range search system by using ASPE, Order-Preserving Encryption (OPE) [21], [22] and R-tree structure. With continuous research, it has been realized to query arbitrary geometric regions, including but not limited to rectangles and circles. For example, Zhang et al. [23] employed the polynomial fitting technique [24] to fit query range to achieve accurate arbitrary geometric range query, and used Randomizable Matrix Multiplication (RMM) to protect data privacy. They propose a personalized privacy protection index for location-based services, which can ensure quantitative privacy protection while realizing information sharing.

Privacy-Preserving Keyword Query: Keyword query aims to find objects related to query keywords. Boneh et al. [25] proposed a Public Key Encryption (PKE) scheme with keyword search over encrypted data, which enables the server to test whether a specific keyword is included in the encrypted keyword set without revealing any other information. But this scheme just supports single keyword query. To search multiple keywords, Liu et al. [26] proposed a privacy-preserving multi-keyword searchable encryption scheme based on a subset decision mechanism, which can determine whether an input set is a subset of another input set. In addition, Tang et al. [27] first proposed a Personalized Privacy Protection Index (PPI) to ensure quantitative privacy protection, then proposed a PPI abstract method [28], which can quantitatively control the privacy leakage for multi-keyword document query. And Tang et al. [29] proposed a novel authentication framework, which achieves the lightweight verification of data version freshness on massive data update streams.

Privacy-Preserving Spatial Keyword Query: Spatial keyword query aims to find objects that satisfy both spatial location and keywords of the query. Cui et al. [30] proposed a novel spatial textual Bloom filter encoding method by mapping spatial and textual information into bloom filters, and encrypted them by using ASPE. After that, they used the inner-product based matching operation to answer query without decryption. However, the scheme can only search the rectangular range. To support searching for more shapes, Wang et al. [31] used Gray code [32] and bitmap to encode spatio-textual objects into uniform index vectors, and used Symmetrickey Hiden Vector Encryption (SHVE) [33] to encrypt them for secure vector matching. Then, they designed a Bitmap Quad-tree (BQ-tree) to further improve query performance over large scale datasets. However, the search shapes that [31] can support are still limited. To further support arbitrary geometric range query, Yang et al. [34] used the polynomial fitting technique and vector space model to encode the spatial location and keyword into a uniform index, and then used Randomizable Matrix Multiplication (RMM) technique to encrypt index matrix. In the query process, the cloud server finds the object that satisfies the query requirement by calculating the trace of matrix product. To sort the results by keyword similarity, Tong et al. [35] designd a comparable product encoding strategy to encode spatio-textual objects as a unified index, and used ASPE to encrypt them, which allows the cloud server to check whether the object falls in the query range and to deduce its textual similarity simultaneously.

However, the above solutions cannot solve our problems simultaneously. Therefore, we design a new unified index

TABLE I
COMPARISON BETWEEN PREVIOUS SCHEMES AND OUR SCHEMES

Scheme	Range query	Keyword query	Index structure	Encryption method	Resist attack
[15]	✓	✗	$\hat{R}$-tree	ASPE	KPA
[20]	✓	✗	R-tree	ASPE/OPE	CPA
[23]	✓	✗	Sublinear	RMM	COA/CPA
[25]	✗	Single	Linear	PKE	CKA
[26]	✗	Multiple	Linear	PKE	KGA
[30]	✓	Multiple	Sublinear	ASPE	KBA
[31]	✓	Multiple	BQ-tree	SHVE	CPA
[34]	✓	Multiple	Linear	RMM	COA/CPA
[35]	✓	Ranked	R-tree	ASPE	SCPA
PSDQ	✓	Multiple	Linear	EASPE	COA/CPA
PSDQ$^+$	✓	Multiple	GR-tree	EASPE	COA/CPA

Notes: KPA: Known-Plaintext Attack; CKA: Chosen-Keyword Attack; KGA: Keyword-Guessing Attack; KBA: Known-Background Attack; SCPA: Selective Chosen-Plaintext Attack.

structure to propose a Privacy-Preserving Spatial Fata Query (PSDQ) scheme by using Geohash algorithm, which only requires users to provide less information about the query range. The challenge issue of PSDQ is how to convert Geohash algorithm and multi-keyword query into a unified index and achieve spatial data query by calculating inner products. Then, we design a GR-tree structure based on Geohash algorithm and R-tree, which avoids the range intersection problem between non-leaf nodes in R-tree. We encode each node of GR-tree as a unified index, and design a pruning strategy based on both spatial information and keywords, which achieves spatial keyword query with sub-linear search complexity. The comparison between our schemes and the previous scheme is shown in Table I.

III. PRELIMINARIES

In this section, we mainly review some related background knowledge, including Geohash algorithm [36], [37], [38], [39], Enhanced Asymmetric Scalar-Product-Preserving Encryption (EASPE) [14] and R-tree [40], [41].

A. Geohash Algorithm

Geohash algorithm is a geocoding method that recursively divides the geographic space into smaller grids, and converts the two-dimensional latitude and longitude of each grid into Geohash code according to Base32. Each Geohash code represents a specific rectangular region on the earth, as shown in Fig. 1.

The Geohash code provides arbitrary precision. The length of the Geohash code gets shorter as the range increases. Given a longer Geohash code, we can get a more precise geographic location. For example, in Fig. 1, the red dot (8, 13) can be denoted as "$s33jxb67$" with ± 19 m rectangular range, "$s33jxb6$" with ± 76 m, or "$s33jxb$" with ± 610 m. The precision of each Geohash code is shown in Table II.

Therefore, we can determine the distance between two points by matching the Geohash codes. For example, in Fig. 1, the Geohash codes of the red dot and the green dot have the same prefix "$s33jxb6$", then we can know that the green dot is within ± 76 m of the red dot. Therefore, we can judge whether the spatial point is within the query range of the query point by

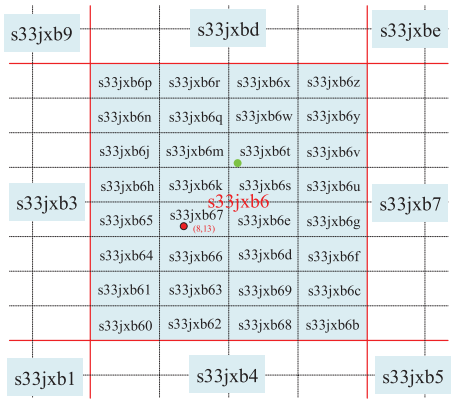

Fig. 1. An example of Geohash coding.

TABLE II
ACCURACY OF GEOHASH CODING

Geohash Length	1	2	3	4
Km Error (km)	± 2500	± 630	± 78	± 20
Geohash Length	5	6	7	8
Km Error (km)	± 2.4	± 0.61	± 0.076	± 0.019

judging the number of the same prefix of the two Geohash codes. Moreover, when the length of the Geohash code is 8, the accuracy is about 19 meters, then an 8-character Geohash code can accurately correspond to a spatial point in the real world.

B. Enhanced Asymmetric Scalar-Product-Preserving Encryption (EASPE)

ASPE [11] is an important technology for similarity search over encrypted data, which can calculate the inner product of two vectors without revealing privacy. However, ASPE is insecure against known-plaintext attack as shown in [12]. Therefore, we enhanced the security of ASPE in our conference version [14].

Assuming that p and q are two d-dimensional vectors, the specific structure of EASPE is as follows:

- $\mathsf{EASPE.KeyGen}(1^\zeta) \rightarrow \mathcal{SK}$: Given a security parameter ζ, this algorithm generates a secret key $\mathcal{SK}$

$$\mathcal{SK} = \{\boldsymbol{s}, \boldsymbol{M}_1, \boldsymbol{M}_2, \pi, r_1, r_2, r_3, r_4, r_5, r_6\},$$

  where $\boldsymbol{s}$ is a random $(d+3)$-dimensional bit vector, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(d+3) \times (d+3)$ invertible matrices, $\pi : \mathbb{R}^{(d+3)} \rightarrow \mathbb{R}^{(d+3)}$ is a random permutation, r_1, r_2, r_3, r_4, r_5 and r_6 are random numbers and satisfy $r_1 r_4 + r_2 r_5 + r_3 r_6 = 0$.
- $\mathsf{EASPE.Enc}(\boldsymbol{p}, \mathcal{SK}) \rightarrow C$: Given the secret key $\mathcal{SK}$ and d-dimensional vector $\boldsymbol{p}$, this algorithm first extends $\boldsymbol{p}$ as a $(d+3)$-dimensional vector $\boldsymbol{p_v} = (\boldsymbol{p}, r_1, r_2, r_3)$ and then permutes it as $\widehat{\boldsymbol{p_v}} = \pi(\boldsymbol{p_v})$ by π. Finally, this algorithm encrypts $\widehat{\boldsymbol{p_v}}$ as C by (1)

$$C = (\boldsymbol{M}_1^T \widehat{\boldsymbol{p_v}}', \boldsymbol{M}_2^T \widehat{\boldsymbol{p_v}}''), \tag{1}$$

where $\widehat{\boldsymbol{p}_v}$ is split into two vectors $\widehat{\boldsymbol{p}_v}'$, $\widehat{\boldsymbol{p}_v}''$ by using the bit vector $\boldsymbol{s}$, which is shown by (2)

$$\begin{cases} \widehat{\boldsymbol{p}_v}'[\kappa] = \widehat{\boldsymbol{p}_v}''[\kappa] = \widehat{\boldsymbol{p}_v}[\kappa], & if\, \boldsymbol{s}[\kappa] = 0; \\ \widehat{\boldsymbol{p}_v}'[\kappa] + \widehat{\boldsymbol{p}_v}''[\kappa] = \widehat{\boldsymbol{p}_v}[\kappa], & if\, \boldsymbol{s}[\kappa] = 1. \end{cases} \tag{2}$$

- EASPE.TrapGen$(\boldsymbol{q}, \mathcal{SK}) \rightarrow T_Q$: Given the secret key $\mathcal{SK}$ and d-dimensional vector $\boldsymbol{q}$, this algorithm first extends $\boldsymbol{q}$ as a $(d+3)$-dimensional vector $\boldsymbol{q}_v = (\boldsymbol{q}, r_4, r_5, r_6)$ and then permutes it as $\widehat{\boldsymbol{q}_v} = \pi(\boldsymbol{q}_v)$ by π. Finally, this algorithm encrypts $\widehat{\boldsymbol{q}_v}$ as as T_Q by (3)

$$T_Q = (\boldsymbol{M}_1^{-1}\widehat{\boldsymbol{q}_v}', \boldsymbol{M}_2^{-1}\widehat{\boldsymbol{q}_v}''), \tag{3}$$

where $\widehat{\boldsymbol{q}_v}$ is split into two vectors $\widehat{\boldsymbol{q}_v}'$, $\widehat{\boldsymbol{q}_v}''$ by using the bit vector $\boldsymbol{s}$, which is shown by (4)

$$\begin{cases} \widehat{\boldsymbol{q}_v}'[\kappa] + \widehat{\boldsymbol{q}_v}''[\kappa] = \widehat{\boldsymbol{q}_v}[\kappa], & if\, \boldsymbol{s}[\kappa] = 0; \\ \widehat{\boldsymbol{q}_v}'[\kappa] = \widehat{\boldsymbol{q}_v}''[\kappa] = \widehat{\boldsymbol{q}_v}[\kappa], & if\, \boldsymbol{s}[\kappa] = 1. \end{cases} \tag{4}$$

- EASPE.Query$(C, T_Q) \rightarrow \boldsymbol{p}^T \cdot \boldsymbol{q}$: This algorithm calculates the inner product of C and T_Q by (5)

$$\begin{aligned} C^T \cdot T_Q &= ((\widehat{\boldsymbol{p}_v}')^T \boldsymbol{M}_1) \cdot (\boldsymbol{M}_1^{-1}\widehat{\boldsymbol{q}_v}') \\ &\quad + ((\widehat{\boldsymbol{p}_v}'')^T \boldsymbol{M}_2) \cdot (\boldsymbol{M}_2^{-1}\widehat{\boldsymbol{q}_v}'') \\ &= \boldsymbol{p}_v^T \cdot \boldsymbol{q}_v = \boldsymbol{p}^T \cdot \boldsymbol{q}. \end{aligned} \tag{5}$$

Compared with the ASPE scheme, EASPE scheme introduces three random numbers and a random permutation π before encryption. It is worth noting that this process requires the data owner and the data user to preprocess the vectors in practice, and take the confused vectors $\widehat{\boldsymbol{p}_v}$ and $\widehat{\boldsymbol{q}_v}$ as the original plaintext vectors.

C. R-Tree

The function of R-tree is to recursively classify spatial data based on the distance between spatial points. In R-tree, there are two types of nodes, namely leaf nodes and internal nodes. Each leaf node represents a spatial data, and each internal node represents a range composed of a Minimum Bounding Rectangle (MBR), note that the child nodes of internal nodes must be within the range. The search of R-tree can be divided into the following two phases.

1) *Filtration Phase:* This phase starts from the root node and traverses the entire tree, and its goal is to find all deepest internal nodes intersected with the given searching rectangle. If such internal nodes exist, this phase adds all leaf nodes of the internal nodes to the candidate set and moves to the next step; otherwise, it returns null.
2) *Verification Phase:* This phase checks whether each node in the candidate set meets the query requirements in turn. If such nodes exist, it returns them to the user; otherwise, it returns null.

Fig. 2 shows an example of R-tree. All spatial data points are first divided into MBRs sequentially and then formed an R-tree. Assume that o_1 satisfies the query requirements, the filtration phase first traverses from top to bottom and finds the deepest intersecting internal node S_4, and then takes the child nodes of S_4 as candidate nodes. The verification phase verifies whether

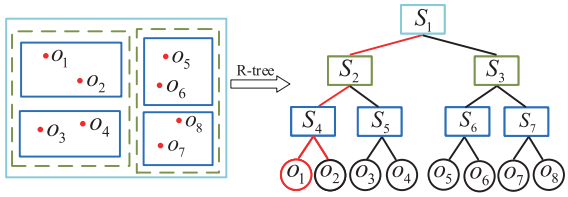

Fig. 2. A example of R-tree.

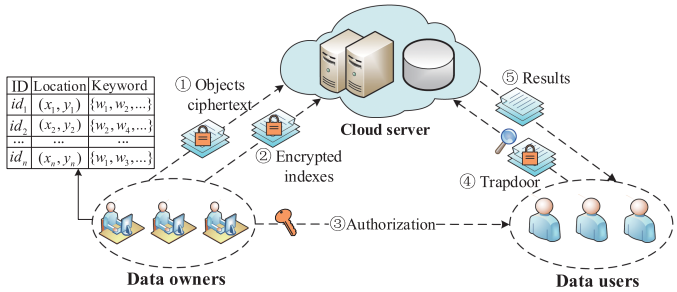

Fig. 3. System model of our schemes.

the candidate nodes o_1 and o_2 meet the query requirements. Finally, o_1 is returned to the user.

IV. Problem Formulation

In this section, we formalize our system model, problem definition, threat model and design goals.

A. System Model

In this article, we consider a cloud data outsourcing scenario. As depicted in Fig. 3, the system model of our schemes consists of three entities, namely data owner, data user and cloud server. The role of each entity is shown as follows:

- *Data owner:* The data owner (individual or organization) owns the spatial dataset and is responsible for building the searchable index for each object in the dataset, and encrypting the spatial data objects and indexes before outsourcing them to the cloud server.
- *Data user:* The authorized data user has access to the outsourced spatial data. He first encrypts the query content as trapdoor, and then submits the trapdoor to the cloud server.
- *Cloud server:* The cloud server which has unlimited storage and computing capabilities, stores encrypted objects and indexes outsourced by the data owner, and provides query services for data users.

The data owner first builds an index for each spatial data object in dataset D, and then outsources the encrypted spatial data objects and indexes to the cloud server (step ①, ②). The data user is authorized after registering himself with the data owner (step ③). When the data user wants to make a search request Q, he/she encrypts the query content as a trapdoor, and submits the trapdoor to the cloud server (step ④). After receiving the trapdoor, the cloud server finds the spatial data objects that meet

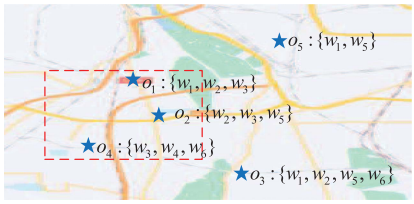

Fig. 4. A example of spatial data query.

the data user's query requirements and returns them to the data user (step ⑤).

B. Problem Definition

The problem of spatial data query is to find objects that are within the query range and contain all query keywords. As shown in the example in Fig. 4, the red area is the query range and the query keywords are $\{w_2, w_5\}$, so o_2 is the result of query.

Let $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ be a total keyword set, $D = \{o_1, o_2, \ldots, o_n\}$ be a spatial dataset owned by the data owner. Each object o_i in D has a unique identity id_i and can be represented as tuple $(o_i.l, o_i.k)$, where $o_i.l = (x_i, y_i)$ is spatial geographic location, $o_i.k = \{w_1, w_2, \ldots, w_g\} \subseteq \mathcal{W}$ is a set of keywords. We first give the definition of spatial data query as follows:

Definition 1: (Spatial Data Query) Given a query request $Q = \{R, Q.k\}$, where R represents the query range (i.e., rectangle) and $Q.k = \{w'_1, w'_2, \ldots, w'_k\} \subseteq \mathcal{W}$ represents a set of query keywords, SDQ is to retrieve a subset $Result = \{id_1, id_2, \ldots, id_h\}$ from D, such that $\forall id_e \in Result, 1 \leq e \leq h, o_e.l \in R$ and $Q.k \subseteq o_e.k$.

Our conference version [14] proposed a naive Spatial Keyword Query (SKQ) scheme and Lightweight SKQ (LSKQ) scheme to solve spatial data query problem by using Geohash algorithm, where the query range $R = (Q.l, r)$ consists of a coordinate $Q.l$ and a number r. However, SKQ needs to generate 8 index vectors for each object o_i or query request Q, which incurs high computation and storage overheads. Although LSKQ only needs to generate one index vector for o_i or Q to improve efficiency, it leads to false positives when mapping each Geohash code to a binary index vector based on Base32. To eliminate the false positive of LSKQ in our conference version, we attempt to design a new unified index construction method, which does not have any loss of accuracy.

C. Threat Model

Consistent with most of previous schemes, the cloud server in our schemes is considered to be honest-but-curious, which honestly performs the established protocol but may be curious to collect or analyze the meaningful information. Data owner and data user are assumed to be fully trusted. They execute specified operations properly and do not collude with the cloud server and unauthorized data users. In addition, the authorization, access and transmission channels between all entities are also assumed to be secure.

- *Ciphertext-Only-Attack (COA) Model:* In this model, the cloud server knows the ciphertexts of all the outsourced spatial data and can observe the encrypted trapdoors, but cannot obtain the corresponding plaintext.
- *Chosen-Plaintext-Attack (CPA) Model:* In this model, the cloud server can access object ciphertexts corresponding to plaintexts of its choice in addition to knowing the encrypted spatial data, index and trapdoor.

D. Design Goals

To achieve a secure spatial keyword query, our schemes should satisfy the following requirements. Note that, access pattern privacy means that the cloud server does not know the specific location in memory of the encrypted index being accessed during the search process, it is beyond the scope of our discussion and will be solved as an important issue in future work.

- *Data Privacy:* The plaintext dataset should not be leaked to the cloud server and unauthorized entities. And they should not obtain any information about these data.
- *Trapdoor Privacy:* Only authorized data users can generate valid trapdoors for search queries, but it requires that the cloud server cannot get the actual query content or infer the query over the trapdoors.
- *Trapdoor Unlinkability:* The cloud server should not be able to associate one trapdoor with another, which means that it cannot determine whether two different trapdoors are generated from the same query.
- *Result Privacy:* The query results returned by the cloud server should not be disclosed to others, while data users can obtain the real data that meets the query conditions.

V. Our Proposed Scheme

In this section, we first briefly review the problems of our conference version [14] and introduce the main idea to solve these problems. Then, we propose a linear Privacy-Preserving Spatial Data Query (PSDQ) scheme by designing a new unified index structure. And we design a Geohash-based R-tree structure, called GR-tree, and propose a more efficient Privacy-Preserving Spatial Data Query (PSDQ$^+$) scheme to improve query efficiency.

A. Main Idea

The existing spatial data query schemes require data users to provide more information about the query range and generate a large number of ciphertexts, resulting in huge storage and computational overheads. To solve this problem, our conference version [14] first proposed a naive scheme (i.e., SKQ) by simply combining Geohash algorithm with EASPE, which needs to generate 8 indexes for each object o_i or query request Q. Then, [14] improved the SKQ to construct a Lightweight scheme (i.e., LSKQ), which only needs to generate one index for each o_i or Q to greatly reduce the computational and storage overheads. Specifically, in the data encryption phase, the data owner first maps the geographic location $o_i.l = (x_i, y_i)$ of the object o_i

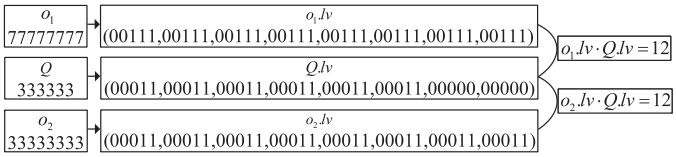

Fig. 5. An example of LSKQ leading false positives.

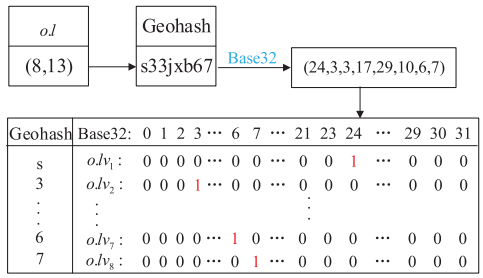

Fig. 6. An example of converting spatial location as vectors.

as 8-character Geohash code by using Geohash algorithm and encodes the Geohash code as an 8-dimensional order vector $\boldsymbol{o_i.l}$ according to Base32. Then, the data owner converts each dimension of $\boldsymbol{o_i.l}$ as 5-bit binary vector $\{\boldsymbol{o_i.lv_j}\}_{1\le j\le 8}$ to generate a (5×8)-dimensional spatial vector $\boldsymbol{o_i.lv} = \{\boldsymbol{o_i.lv_1}, \ldots, \boldsymbol{o_i.lv_8}\}$. For query request Q, the generation of query spatial vector $\boldsymbol{Q.lv}$ is same as that of $\boldsymbol{o_i.lv}$. In the query process, the cloud server determines whether the object o_i is within the query range by calculating the inner product of $\boldsymbol{o_i.lv}$ and $\boldsymbol{Q.lv}$. However, LSKQ leads to false positives. For example, as shown in Fig. 5, the Geohash code of o_1 is "7777777" will be encoded as $\boldsymbol{o_1.lv} = (00111, 00111, 00111, 00111, 00111, 00111, 00111, 00111)$, and the Geohash code of o_2 is "33333333" will be converted as $\boldsymbol{o_2.lv} = (00011, 00011, 00011, 00011, 00011, 00011, 00011, 00011)$. Suppose the Geohash code of query Q is "333333", it will be converted as $\boldsymbol{Q.lv} = (00011, 00011, 00011, 00011, 00011, 00011, 00000, 00000)$, since $\boldsymbol{o_1.lv} \cdot \boldsymbol{Q.lv} = \boldsymbol{o_2.lv} \cdot \boldsymbol{Q.lv}$, both o_1 and o_2 will be returned as the result, but obviously o_1 is not required by the data user.

To improve the accuracy of [14], we design a new index structure to propose a Privacy-Preserving Spatial Data Query (PSDQ) scheme, which will not cause false positives. Specifically, in the data encryption phase, the data owner first maps the geographic location $o_i.l = (x_i, y_i)$ of the object o_i as an 8-character Geohash code by using Geohash algorithm and encodes the Geohash code as an 8-dimensional order vector $\widetilde{\boldsymbol{o_i.l}}$ according to Base32. Then, the data owner converts each dimension of $\widetilde{\boldsymbol{o_i.l}}$ as a 32-dimensional spatial vector $\{\boldsymbol{o_i.lv_j}\}_{1\le j\le 8}$ to generate a (32×8)-dimensional spatial vector $\boldsymbol{o_i.lv} = \{\boldsymbol{o_i.lv_1}, \ldots, \boldsymbol{o_i.lv_8}\}$. And the data owner encodes the keyword set $o_i.k$ of object o_i as an m-dimensional textual vector $\boldsymbol{o_i.kv}$. Thus, the data owner generates a data vector $\boldsymbol{o_i.v} = (\boldsymbol{o_i.lv}, \boldsymbol{o_i.kv}, -1)$ for each o_i, and then uses EASPE to encrypt it as C_i. In the trapdoor generation phase, for query request $Q = \{R = (Q.l, r), Q.k\}$, the data user first maps $Q.l$ as t-character Geohash code according to r and converts it as an t-dimensional order vector $\widetilde{\boldsymbol{Q.l}}$. Then, the data user encodes each dimension of $\widetilde{\boldsymbol{Q.l}}$ as 32-dimensional query spatial vectors $\{\boldsymbol{Q.lv_j}\}_{1\le j\le t}$. If $t < 8$, each dimension of remaining $(8-t)$ 32-dimensional query spatial vectors is set as 0. Finally, the data user converts R as a (32×8)-dimensional vector $\boldsymbol{Q.lv} = (\boldsymbol{Q.lv_1}, \ldots, \boldsymbol{Q.lv_8})$. And the data user converts the query keyword set $Q.k$ as an m-dimensional vector $\boldsymbol{Q.kv}$. Thus, the data user generates a query vector $\boldsymbol{Q.v} = (\boldsymbol{Q.lv}, \boldsymbol{Q.kv}, t+k)$, and then uses EASPE to encrypt it as trapdoor T_Q. In the query phase, the cloud server can determine whether the object o_i meets the query conditions by calculating the inner product of each C_i and T_Q. However, PSDQ is linear and its query complexity is $\mathcal{O}(n)$, which leads to huge computational overhead.

To improve the query efficiency of PSDQ, we use the properties of Geohash code and R-tree to design a Geohash-based R-tree, called GR-tree. Then, we propose a more efficient PSDQ scheme (PSDQ$^+$) based on GR-tree, which achieves the sublinear search complexity. The details about GR-tree will be presented in Section V-C.

B. Privacy-Preserving Spatial Data Query (PSDQ)

We first design a new index structure by using Geohash algorithm, then propose a Privacy-Preserving Spatial Data Query (PSDQ) scheme.

Framework: PSDQ includes the following four phases: KeyGeneration, DataEncryption, TrapdoorGeneration and Query. In KeyGeneration phase, the data owner generates encryption key. In DataEncryption phase, the data owner encodes each spatial data object o_i as an index vector $\boldsymbol{o_i.v}$ and encrypts $\boldsymbol{o_i.v}$ as C_i, then outsources C_i to the cloud server. In TrapdoorGeneration, the data user encodes the query request Q as vector $\boldsymbol{Q.v}$ and encrypts it as a trapdoor T_Q, then submits it to the cloud server. In Query phase, the cloud server calculates the inner product of the encrypted index C_i and trapdoor T_Q, then returns the spatial data object whose inner product value is equal to 0 to the data user. The specific construction is as follows.

Let $\mathcal{W} = \{w_1, w_2, \ldots, w_m\}$ be a total keyword set and $D = \{o_1, o_2, \ldots, o_n\}$ be a spatial dataset. Each object o_i has a unique identity id_i and can be represented as tuple $(o_i.l, o_i.k)$, where $o_i.l = (x_i, y_i)$ represents spatial geographic location, $o_i.k = \{w_1, w_2, \ldots, w_g\} \subseteq \mathcal{W}$ represents a set of keywords owned by the object o_i.

KeyGeneration. The data owner generates a encryption key $\mathcal{SK}$ by calling EASPE.KeyGen, where $\boldsymbol{s}$ is a random $(260+m)$-dimensional binary vector, $\boldsymbol{M_1}$ and $\boldsymbol{M_2}$ are two random $(260+m)\times(260+m)$ invertible matrices.

DataEncryption. The details are shown in Algorithm 1. For each object o_i in D, the data owner first maps $o_i.l$ as an 8-character Geohash code by using Geohash algorithm and encodes the Geohash code as an 8-dimensional order vector $\widetilde{\boldsymbol{o_i.l}}$ according to Base32. Then, the data owner converts each dimension of $\widetilde{\boldsymbol{o_i.l}}$ as a 32-dimensional spatial vector $\{\boldsymbol{o_i.lv_j}\}_{1\le j\le 8}$. The specific process of spatial vectors is shown in Fig. 6. Given a point $o.l = (8, 13)$, the data owner first encodes it as

Algorithm 1: Data Encryption.

Input: $D = \{o_1, o_2, \ldots, o_n\}$
Output: $Enc(D)$
Initialize a set $Enc(D) = \emptyset$;
for $1 \leq i \leq n$ **do**
  Initialize a set $Enc(o_i) = \emptyset$;
  $o_i.l$ is encoded as an 8-character Geohash code and then converted as an 8-dimensional order vector $\widetilde{\boldsymbol{o_i.l}}$;
  for $1 \leq j \leq 8$ **do**
    Convert the j-*th* dimension of $\widetilde{\boldsymbol{o_i.l}}$ as $\boldsymbol{o_i.lv_j}$;
  $\boldsymbol{o_i.lv} = (\boldsymbol{o_i.lv_1}, \boldsymbol{o_i.lv_2}, \ldots, \boldsymbol{o_i.lv_8})$;
  $o_i.k$ is converted as $\boldsymbol{o_i.kv}$;
  $\boldsymbol{o_i.v} = (\boldsymbol{o_i.lv}, \boldsymbol{o_i.kv}, -1)$;
  $C_i = (\boldsymbol{M_1^T}\widehat{\boldsymbol{o}}_i\boldsymbol{.v'}, \boldsymbol{M_2^T}\widehat{\boldsymbol{o}}_i\boldsymbol{.v''})$;
  $Enc(D).add(C_i)$;
return $Enc(D)$.

a Geohash code "$s33jxb67$", then generates the order vector $(24, 3, 3, 17, 29, 10, 6, 7)$. For example, the order of character 's' in Base32 is 24, '3' is 3. Finally, each dimension of the order vector is converted as a spatial vector. For example, 24 is denoted as (0, ..., 1, ..., 0), where only the 24-*th* position is set as 1.

Finally, the data owner converts spatial location $o_i.l$ of object o_i as a (32×8)-dimensional spatial vector by (6)

$$\boldsymbol{o_i.lv} = (\boldsymbol{o_i.lv_1}, \boldsymbol{o_i.lv_2}, \ldots, \boldsymbol{o_i.lv_8}). \tag{6}$$

For the keyword set $o_i.k$ owned by the object o_i, the data owner converts $o_i.k$ as an m-dimensional vector $\boldsymbol{o_i.kv}$ by (7)

$$\boldsymbol{o_i.kv} = (\lambda_1, \lambda_2, \ldots, \lambda_m), \tag{7}$$

where m is the size of $\mathcal{W}$, and if $w_\zeta \in \mathcal{W}$ is in $o_i.k$, $\lambda_\zeta = 1$; otherwise, $\lambda_\zeta = 0$. For example, given a total keyword set $\mathcal{W} = \{w_1, w_2, w_3, w_4, w_5\}$ and a keyword set $o.k = \{w_2, w_3, w_5\}$ owned by object o, we can obtain $o.kv = (0, 1, 1, 0, 1)$.

Then, the data owner generates a $(32 * 8 + m + 1)$-dimensional data vector $\boldsymbol{o_i.v}$ for each object o_i by (8)

$$\boldsymbol{o_i.v} = (\boldsymbol{o_i.lv}, \boldsymbol{o_i.kv}, -1). \tag{8}$$

Finally, the data owner encrypts spatial dataset D as $Enc(D) = \{C_i | 1 \leq i \leq n\}$ by calling EASPE.Enc and outsources $Enc(D)$ to the cloud server.

TrapdoorGeneration. The details are shown in Algorithm 2. Given a query request $Q = \{R, Q.k\}$, where $R = (Q.l, r)$ represents the query range consisting of a coordinate $Q.l$ and a number r, and $Q.k = \{w'_1, w'_2, \ldots, w'_k\} \subseteq \mathcal{W}$ represents a set of query keywords, the data user first encodes $Q.l$ as a Geohash code with a length of t-character according to r, and converts it as an t-dimensional order vector $\widetilde{\boldsymbol{Q.l}}$ according to Base32. Then, the data user encodes each dimension of $\widetilde{\boldsymbol{Q.l}}$ as 32-dimensional query spatial vectors $\{\boldsymbol{Q.lv_j}\}_{1 \leq j \leq t}$. If $t < 8$, each dimension of remaining $(8 - t)$ 32-dimensional query spatial vectors is set as 0. The specific process of query spatial vectors is shown in Fig. 7. Given a point $Q.l = (8, 13)$ and $r = 600\ m$, the data user first encodes $Q.l$ as Geohash

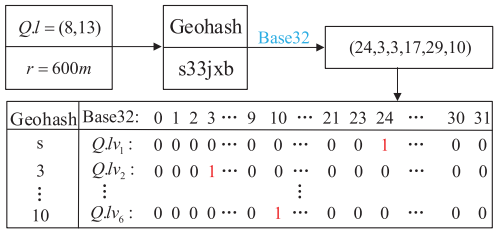

Fig. 7. An example of converting the query range as vectors.

Algorithm 2: Trapdoor Generation.

Input: $Q = \{Q.l, Q.k, R\}$
Output: T_Q
$Q.l$ is encoded an t-character Geohash code and then converted as an t-dimensional order vector $\widetilde{\boldsymbol{Q.l}}$;
for $1 \leq j \leq 8$ **do**
  if $j \leq t$ **then**
    Convert the j-*th* dimension of $\widetilde{\boldsymbol{Q.l}}$ as $\boldsymbol{Q.lv_j}$;
  else
    Each dimension of $\boldsymbol{Q.lv_j}$ is 0;
$\boldsymbol{Q.lv} = (\boldsymbol{Q.lv_1}, \boldsymbol{Q.lv_2}, \ldots, \boldsymbol{Q.lv_8})$;
$Q.k$ is converted as $\boldsymbol{Q.kv}$;
$\boldsymbol{Q.v} = (\boldsymbol{Q.lv}, \boldsymbol{Q.kv}, t + k)$;
$T_Q = (\boldsymbol{M_1^{-1}}\widehat{\boldsymbol{Q}}\boldsymbol{.v'}, \boldsymbol{M_2^{-1}}\widehat{\boldsymbol{Q}}\boldsymbol{.v''})$;
return T_Q.

code "$s33jxb$" and converts it as an 6-dimensional order vector $(24, 3, 3, 17, 29, 10)$ according to Base32. Then, the data user converts $Q.l$ as six 32-dimensional query spatial vectors $\{\boldsymbol{Q.lv_1}, \boldsymbol{Q.lv_2}, \ldots, \boldsymbol{Q.lv_6}\}$, and generates two other vectors $\boldsymbol{Q.lv_7} = (0, \ldots, 0)$ and $\boldsymbol{Q.lv_8} = (0, \ldots, 0)$.

Finally, the data user converts query range R as a (32×8)-dimensional query vector by (9)

$$\boldsymbol{Q.lv} = (\boldsymbol{Q.lv_1}, \boldsymbol{Q.lv_2}, \ldots, \boldsymbol{Q.lv_8}). \tag{9}$$

For query keyword set $Q.k$, the data user converts $Q.k$ as an m-dimensional vector $\boldsymbol{Q.kv}$ by (10)

$$\boldsymbol{Q.kv} = (\lambda'_1, \lambda'_2, \ldots, \lambda'_m), \tag{10}$$

where m is the size of $\mathcal{W}$, and if $w_\zeta \in \mathcal{W}$ is in $Q.k$, $\lambda'_\zeta = 1$; otherwise, $\lambda'_\zeta = 0$. For example, given a total keyword set $\mathcal{W} = \{w_1, w_2, w_3, w_4, w_5\}$ and a query keyword set $Q.k = \{w_2, w_5\}$, we can obtain $Q.kv = (0, 1, 0, 0, 1)$.

Then, the data user generates a $(32 * 8 + m + 1)$-dimensional query vector $\boldsymbol{Q.v}$ by (11)

$$\boldsymbol{Q.v} = (\boldsymbol{Q.lv}, \boldsymbol{Q.kv}, t + k), \tag{11}$$

where k is the number of query keywords.

Finally, the data user encrypts query Q as a trapdoor T_Q by calling EASPE.TrapGen and outsources T_Q to the cloud server.

Query. The cloud server calculates the inner product of C_i and T_Q, then returns the identity id of object that meets the query requirement to the data user. The details are shown in Algorithm 3.

Algorithm 3: Query.

Input: $Enc(D) = \{C_1, C_2, \ldots, C_n\}_{1 \le i \le n}$ and T_Q
Output: $Result$

Initialize a set $Result = \emptyset$;
for $1 \le i \le n$ **do**
  Calculate $C_i^T \cdot T_Q = \boldsymbol{o_i.v}^T \cdot \boldsymbol{Q.v}$;
  if $C_i^T \cdot T_Q = 0$ **then**
    $Result.add(id_i)$;
return $Result$.

Correctness: For each $o_i \in D$, the cloud server calculates $C_i^T \cdot T_Q = 0$ as (12) by calling EASPE.Query

$$C_i^T \cdot T_Q = (\boldsymbol{o_i.lv}, \boldsymbol{o_i.kv}, -1)^T \cdot (\boldsymbol{Q.lv}, \boldsymbol{Q.kv}, t + k). \quad (12)$$

If $(\boldsymbol{o_i.lv} \cdot \boldsymbol{Q.lv}) - t = 0$, it indicates that $o_i.l$ and $Q.l$ have t same Geohash prefixes, that is, o_i is within the query range. If $(\boldsymbol{o_i.kv} \cdot \boldsymbol{Q.kv}) - k = 0$, it indicates that $o_i.k$ contains all query keywords. $C_i^T \cdot T_Q = 0$ indicates that o_i is within the query range and contains all query keywords, that is, o_i meets query requirements.

Remark: Compared with our conference version [14], the way in which PSDQ builds indexes for spatial data objects and queries does not lead to false positives. In the query phase, the cloud server needs to calculate the inner product of each encrypted spatial data index and trapdoor, PSDQ has linear query complexity $\mathcal{O}(n)$. Therefore, we need to improve the scheme to greatly improve its query efficiency.

C. Efficient Privacy-Preserving Spatial Data Query (PSDQ$^+$)

PSDQ is a linear search scheme, which is difficult to meet the needs of large-scale databases. Therefore, we first design a Geohash-based R-tree structure, called GR-tree, and then devise a pruning strategy based on both spatial range and keywords. Based on these two techniques, we finally propose an efficient Privacy-preserving Spatial Data Query (PSDQ$^+$) scheme to greatly improve the query efficiency of PSDQ.

Construction of GR-tree: According to the Gehash algorithm in Section III, we can know that the Geoash codes of the points located in the same range have the same prefix, so we can build a GR-tree based on Geohash code and R-tree. Each Geohash code has 8 dimensions in total, note that each dimension has 32 possible characters, and each character represents a unique range. The first dimension of Geohash code represents the largest spatial range, and the range represented by each dimension decreases in turn. Therefore, the maximum depth of GR-tree is 9 layers, where the nodes of the first 8 layers are internal nodes, and the nodes of the 9-*th* layer are leaf nodes. In particular, each node of the GR-tree contains spatial information and keywords. For each internal node $S = (S.l, S.k)$, $S.l$ and $S.k$ contain the location and keywords of all spatial data objects within this range respectively. Each leaf node represents a unique spatial data object o_i. If the data owner builds GR-tree as a binary tree (resp. quad-tree), he/she randomly selects 16 (resp. 8) characters from 32 possible characters and assigns them to internal nodes to divide the range in turn. Randomly dividing the range can prevent the cloud server from guessing the data user's query range based on the traversal path of the tree. Therefore, each internal node S only needs to represent the 16 (resp. 8) spatial ranges corresponding to the 16 (resp. 8) characters contained in $S.l$.

Fig. 8 shows an example of GR-tree, we only give a binary tree with 4 layers due to space limitations. In the query phase, the cloud server just compares whether each dimension of the query Geohash code belongs to the Geohash code of the internal node in turn, and checks whether the internal node contains all the query keywords. Suppose the Geohash code of the query range is "s33jxb" and the query keyword is $Q.k = w_4$, the cloud server traverses the internal nodes of the tree from top to bottom. When the cloud server traverses the first layer, $S_2.l$ contains "s" and $Q.k \in S_2.k$, so it continues to traverse the child nodes of S_2. Specifically, when it traverses the second layer, $S_6.l$ contains "3" and $Q.k \in S_6.k$, so it continues to traverse the child points of S_6. Then, the cloud server traverses the remaining internal nodes in turn to find the internal node with the deepest intersection, and compares the query request with all leaf nodes of the internal node in turn, and finally returns the leaf nodes that meet the requirements.

It is worth noting that the range of each node in the GR-tree is independent and GR-tree avoids the range intersection problem between non-leaf nodes in R-tree. Then, we encode the spatial information and keywords of each node in GR-tree into a unified index vector. The spatial data can be filtered according to the spatial information and keywords at the same time by calculating the inner product of the unified index, so as to achieve spatial keyword query with sub-linear search complexity.

Framework: With the above GR-tree, we construct PSDQ$^+$ which also consists of four phases: KeyGeneration, DataEncryption, TrapdoorGeneration and Query. Unlike PDSQ, in the DataEncryption phase, the data owner needs to encrypt the entire tree index. The specific construction is as follows.

KeyGeneration. The data owner generates a encryption key $\mathcal{SK}$ by calling EASPE.KeyGen, where $\boldsymbol{s}$ is a random $(260 + m)$-dimensional binary vector, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(260 + m) \times (260 + m)$ invertible matrices.

DataEncryption. The details are shown in Algorithm 4. For each internal node S in the first layer of GR-tree, the data owner first encodes its owning character $S.l$ as an 8-dimensional order vector $\widetilde{\boldsymbol{S.l}}$ according to Base32, and then encodes $\widetilde{\boldsymbol{S.l}}$ as a 32-dimensional vector $\boldsymbol{S.lv_1}$. Note that, the data owner sets the corresponding position of $\boldsymbol{S.lv_1}$ as 1 according to each dimension of $\widetilde{\boldsymbol{S.l}}$, and sets other positions as 0. For example, in Fig. 8, $\boldsymbol{S_1.lv_1} = (1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0)$. In particular, the data owner sets each dimension of remaining 32-dimensional vectors $\{\boldsymbol{S.lv_j} | j = 2, 3, \ldots, 8\}$ as 1. Finally, the data owner converts $S.l$ as a (32×8)-dimensional vector $\boldsymbol{S.lv} = (\boldsymbol{S.lv_1}, \ldots, \boldsymbol{S.lv_8})$. And the data owner encodes the keywords $S.k$ owned by S as m-dimensional vector $\boldsymbol{S.kv}$ by using (7). Finally, the data owner encodes each internal node as

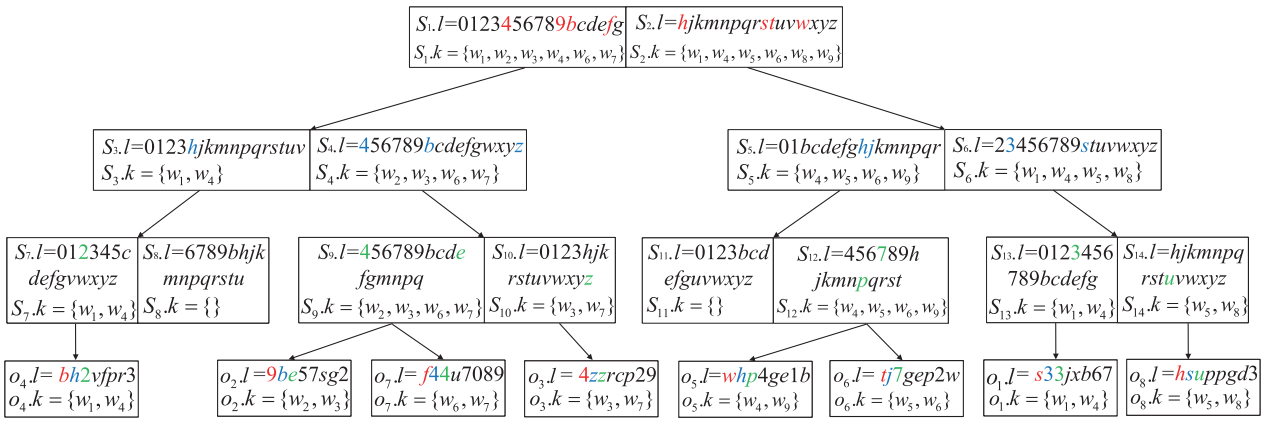

Fig. 8. An example of GR-tree.

Algorithm 4: Data Encryption.

Input: GR-tree
Output: Encrypted GR-tree

for *each internal node* S **do**
  if S *in the* ι *layer* **then**
    $S.l$ is encoded as a 32-dimensional vector $\boldsymbol{S.lv_\iota}$;
    Each dimension of the remaining 7 vectors $\{\boldsymbol{S.lv_j}|j \in [1,8] \ \& \ j \neq \iota\}$ is set as 1;
  $\boldsymbol{S.lv} = (\boldsymbol{S.lv_1}, \boldsymbol{S.lv_2}, \ldots, \boldsymbol{S.lv_8})$;
  $S.k$ is encoded as an m-dimensional vector $\boldsymbol{S.kv}$;
  $\boldsymbol{S.v} = (\boldsymbol{S.lv}, \boldsymbol{S.kv}, -1)$;
  $Enc(S) = (\boldsymbol{M}_1^T \widehat{\boldsymbol{S}}.\boldsymbol{v'}, \boldsymbol{M}_2^T \widehat{\boldsymbol{S}}.\boldsymbol{v''})$;
for *each leaf node* o_i **do**
  //(Same as Algorithm 1);
  $o_i.l$ is encoded as an 8-dimensional order vector $\widetilde{\boldsymbol{o_i.l}}$;
  Each dimension of $\widetilde{\boldsymbol{o_i.l}}$ is encoded as $\boldsymbol{o_i.lv_j}$;
  $\boldsymbol{o_i.lv} = (\boldsymbol{o_i.lv_1}, \boldsymbol{o_i.lv_2}, \ldots, \boldsymbol{o_i.lv_8})$;
  $o_i.k$ is converted as $\boldsymbol{o_i.kv}$;
  $\boldsymbol{o_i.v} = (\boldsymbol{o_i.lv}, \boldsymbol{o_i.kv}, -1)$;
  $C_i = (\boldsymbol{M}_1^T \widehat{\boldsymbol{o}}_i.\boldsymbol{v'}, \boldsymbol{M}_2^T \widehat{\boldsymbol{o}}_i.\boldsymbol{v''})$;
return Encrypted GR-tree.

$(32 * 8 + m + 1)$-dimensional vector $\boldsymbol{S.v}$ by (13)

$$\boldsymbol{S.v} = (\boldsymbol{S.lv}, \boldsymbol{S.kv}, -1). \tag{13}$$

The generation method of the index vectors $\boldsymbol{S.v}$ of internal nodes of other layers in GR-tree is the same as (13), except that the construction method of $\boldsymbol{S.lv}$ is different. For example, for each internal node in the second layer of GR-tree, its owning characters are encoded as a 32-dimensional vector $\boldsymbol{S.lv_2}$, the remaining $\{\boldsymbol{S.lv_j}|j = 1, 3, \ldots, 8\}$ are all 32-dimensional vectors in which all elements are 1. For each internal node in the third layer of GR-tree, its owning characters are encoded as a 32-dimensional vector $\boldsymbol{S.lv_3}$, the remaining $\{\boldsymbol{S.lv_j}|j = 1, 2, 4, \ldots, 8\}$ are all 32-dimensional vectors in which all elements are 1. And so on, the data owner encodes all internal nodes in GR-tree.

The index vectors of leaf nodes are generated in the same way as (8). Finally, the data owner encrypts the index vectors of the internal nodes and leaf nodes in generated GR-tree as $Enc(S)$ and C_i by calling EASPE.Enc, respectively.

TrapdoorGeneration. This process is the same as that of PSDQ. The details are shown in Algorithm 2.

Query. Unlike PSDQ, the cloud server needs to perform the following two phases to find objects that meet query requirements. The details are shown in Algorithm 5.

1) *Filtration phase:* This phase first calculates the inner product of the trapdoor and the internal node, i.e., $Enc(S)^T \cdot T_Q$. If $Enc(S)^T \cdot T_Q = 0$, it means that the internal node intersects the query range and contains all query keywords. Then, this phase continues to traverse the child nodes of the internal node until the deepest internal node (i.e. the internal node of layer 8) is found. If there exist deepest intersecting internal nodes, it adds all leaf nodes of the internal nodes to the candidate set and moves to the next step; otherwise, it returns null.
2) *Verification phase:* For all leaf nodes in the candidate set, this phase calculates the inner product of the trapdoor and leaf nodes, i.e., $C_i^T \cdot T_Q$. If $C_i^T \cdot T_Q = 0$, it means that the leaf node meets the query requirements. Then, this phase returns the leaf node to the data user; otherwise, it returns null.

It should be noted that, since the number of layers of GR-tree is determined by the dimension of the Geohash code and each dimension of the Geohash code represents a fixed range, the cloud server can infer the data user's query range based on the number of layers traversing GR-tree. Thus, if the dimension of Geohash code of the query is less than 8, the cloud server still needs to traverse all the layers (i.e., 8 layers) of the GR-tree.

Remark: PSDQ$^+$ is based on GR-tree, which can greatly improve the query efficiency and its query complexity is only $\mathcal{O}(\log n)$. However, PSDQ$^+$ also has some shortcomings, i.e., it cannot guarantee access pattern privacy, which may lead to data leakage. And the same query will traverse the same path of the index to find the same query result, the cloud server can link

Algorithm 5: Query.

Input: Encrypted GR-tree
Output: $Result$

Initialize a set $Result = \emptyset$;
//*Filtration phase*;
Initialize a candidate set $CS = \emptyset$;
for *internal node* S **do**
  if S *is on the 8th layer* **then**
    Calculate $Enc(S)^T \cdot T_Q$;
    if $Enc(S)^T \cdot T_Q = 0$ **then**
      $CS.add(o_i)$;
  else
    Calculate $Enc(S)^T \cdot T_Q$;
    if $Enc(S)^T \cdot T_Q = 0$ **then**
      Traversing child nodes of the internal node;
//*Verification phase*;
for *each leaf node* $o_i \in CS$ **do**
  Calculate $C_i^T \cdot T_Q$;
  if $C_i^T \cdot T_Q = 0$ **then**
    $Result.add(id_i)$;
return $Result$.

trapdoors with the same query results to the same query due to access pattern leakage. Fortunately, some existing hidden access pattern solutions can be better integrated into our solutions. For example, Zhang et al. [42] first proposed a privacy notion called access pattern unlinkability to better balance security and performance requirements, and then used Private Information Retrieval (PIR) and oblivious shuffling as building blocks of hidden access pattern to design a secure data retrieval structure that can provide access pattern unlinkability. Therefore, we will draw on the experience of [42] in future work to redesign a new index structure and algorithms to protect access pattern privacy.

VI. Security Analysis

In this section, we analyze the security of our schemes under the threat models defined in Section IV. The results show that our schemes make the cloud server unable to learn any critical information except encrypted objects and trapdoors. As described in the threat models, the adversary who conducts CPA has more information than the adversary launching COA, so the adversary launching CPA is more powerful. Therefore, we directly prove that our two schemes are secure against IND-CPA. And because the two schemes use the same encryption method, we take PSDQ as a representative for detailed proof.

ASPE has been proven unable to resist the Known-Plaintext Attack (KPA) as shown in [12]. Specifically, in [12], given d d-dimensional plaintext vectors $\{p_1, \ldots, p_d\}$ and corresponding ciphertexts $\{C_1, \ldots, C_d\}$, a d-dimensional query vector q and corresponding trapdoor T_Q, the adversary calculates $C_1^T \cdot T_Q = p_1 \cdot q, \ldots, C_d^T \cdot T_Q = p_d \cdot q$. Since this is a d linear equation system with d unknown variables in q, which has a unique solution for q, the adversary can infer the d-dimensional plaintext q. Once the adversary finds the d linearly independent trapdoors, he can infer the whole plaintext dataset. Therefore, we confuse index vectors by adding random numbers and random permutations before encryption. According to the above attack, the attacker can only obtain $\widehat{\boldsymbol{q_v}}$, he cannot speculate the q, thereby not recovering the spatial dataset. We will prove that our schemes are secure against IND-CPA. Since CPA is stronger than KPA, our solutions can also resist KPA.

A. Data Encryption

We will first simulate the security game played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ in chosen-plaintext attack.

- Given a security parameter λ, the adversary $\mathcal{A}$ generates two datasets $D_0 = (d_{0,1}, d_{0,2}, \ldots, d_{0,n})$ and $D_1 = (d_{1,1}, d_{1,2}, \ldots, d_{1,n})$ with the same dimension to $\mathcal{C}$, where $d_{i,j}$ is a spatio-textual data, $i \in \{0, 1\}$, $j \in \{1, n\}$.
- The challenger $\mathcal{C}$ runs KeyGeneration to generate the secret key.
- Phase 1: $\mathcal{A}$ submits $d_{i,j}$ to $\mathcal{C}$, $i \in \{0, 1\}$, $j \in \{1, n\}$. Then, $\mathcal{C}$ responses with a ciphertext $C_{i,j}$ through Data Encryption.
- With $D_0, D_1, \mathcal{C}$ chooses a uniform bit $b \in \{0, 1\}$ and calculates the ciphertext $C_{b,j}$ of $d_{b,j}$ through Data Encryption. After that, $\mathcal{C}$ returns $C_{b,j}$ to $\mathcal{A}$.
- Phase2: $\mathcal{A}$ selects a number of messages and submits them to $\mathcal{C}$.
- The adversary $\mathcal{A}$ takes a guess b' of b.

Definition 2: PSDQ achieves IND-CPA security in the data encryption phase if for any polynomial time adversary $\mathcal{A}$, it has at most a negligible advantage $negl(\lambda)$, such that

$$Adv_{PSDQ,\mathcal{A}}^{IND-CPA}(1^\lambda) = \left| Pr(b' = b) - \frac{1}{2} \right| \leq negl(\lambda),$$

where $negl(\lambda)$ represents a negligible function.

Theorem 1: PSDQ achieves IND-CPA security in the data encryption phase under the CPA security game.

Proof: According to the above analysis, we should prove that $\mathcal{A}$ cannot distinguish $C_{0,j}$ from $C_{1,j}$, even if the adversary $\mathcal{A}$ has oracle access to DataEncryption. Assume that one of the messages in D_i is $d_{i,j} = (d_{i,j}.l, d_{i,j}.k)$. According to the process of DataEncryption, $d_{i,j}$ is extended to a $(260 + m)$-dimensional vector $\boldsymbol{d_{i,j}.v} = (\boldsymbol{d_{i,j}.lv}, \boldsymbol{d_{i,j}.kv}, -1, r_1, r_2, r_3)$, where r_1, r_2, r_3 are random numbers. Then, $\boldsymbol{d_{i,j}.v}$ is permuted as $\widehat{\boldsymbol{d_{i,j}.v}} = \pi(\boldsymbol{d_{i,j}.v})$ by the random permutation π and encrypted to $C_{i,j} = (\boldsymbol{M}_1^T\widehat{\boldsymbol{d_{i,j}.v'}}, \boldsymbol{M}_2^T\widehat{\boldsymbol{d_{i,j}.v''}})$ by the random binary vector $\boldsymbol{s}$ and the matrices $\boldsymbol{M}_1, \boldsymbol{M}_2$. In Phase 1 and Phase 2 of the game, $\mathcal{A}$ can choose different $d_{i,j}$ each time and observe its corresponding ciphertext $C_{i,j}$. However, since $\widehat{\boldsymbol{d_{i,j}.v}}$ is a random vector determined by $\mathcal{C}$, r_1, r_2, r_3 are random numbers, π is a random permutation, $\boldsymbol{s}$ is a random binary vector, and $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random invertible matrices, the ciphertext $C_{i,j}$ is random to $\mathcal{A}$. In other words, for any message selected by $\mathcal{A}$ and its corresponding ciphertext, $\mathcal{A}$ cannot distinguish which messages are actually encrypted. Therefore, even if $\mathcal{A}$ has the ability to access DataEncryption, b' can only be obtained by

random guessing. So we have

$$Adv_{PSDQ,\mathcal{A}}^{IND-CPA}(1^{\lambda}) = \left| Pr(b' = b) - \frac{1}{2} \right| \leq negl(\lambda).$$

Thus, the Data Encryption phase of the PSDQ scheme is also secure against IND-CPA.

B. Trapdoor Generation

We first simulate the security game played between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$ in chosen-plaintext attack.

- Given a security parameter λ, the adversary $\mathcal{A}$ generates two different query requests $Q_0 = (q_0.l, q_0.k, R_0)$ and $Q_1 = (q_1.l, q_1.k, R_1)$ with the same dimension to $\mathcal{C}$, where $i \in \{0, 1\}$.
- The challenger $\mathcal{C}$ runs KeyGeneration to generate the secret key.
- Phase 1: $\mathcal{A}$ submits Q_i to $\mathcal{C}$, $i \in \{0, 1\}$. Then, $\mathcal{C}$ responses with a trapdoor T_{Q_i} through TrapdoorGeneration.
- With Q_0 and Q_1, $\mathcal{C}$ chooses a uniform bit $b \in \{0, 1\}$ and calculates the trapdoor T_{Q_i} of Q_i through TrapdoorGeneration. After that, $\mathcal{C}$ returns T_{Q_i} to $\mathcal{A}$.
- Phase2: $\mathcal{A}$ selects a number of messages and submits them to $\mathcal{C}$.
- The adversary $\mathcal{A}$ takes a guess b' of b.

Definition 3: PSDQ achieves IND-CPA security in the trapdoor generation phase if for any polynomial time adversary $\mathcal{A}$, it has at most a negligible advantage $negl(\lambda)$, such that

$$Adv_{PSDQ,\mathcal{A}}^{IND-CPA}(1^{\lambda}) = \left| Pr(b' = b) - \frac{1}{2} \right| \leq negl(\lambda),$$

where $negl(\lambda)$ represents a negligible function.

Theorem 2: PSDQ achieves IND-CPA security in the trapdoor generation phase under the CPA security game.

Proof: According to the above analysis, we should prove that $\mathcal{A}$ cannot distinguish Q_0 and Q_1, even if the adversary $\mathcal{A}$ has oracle access to TrapdoorGeneration. According to the process of TrapdoorGeneration, Q_0 is extended to a $(260 + m)$-dimensional vector $\boldsymbol{q_i.v} = (\boldsymbol{q_i.lv}, \boldsymbol{q_i.kv}, t + k, r_1, r_2, r_3)$, where r_1, r_2, r_3 are random numbers. Then, $\boldsymbol{q_i.v}$ is permuted as $\widehat{\boldsymbol{q}}_{\boldsymbol{i}}.\boldsymbol{v} = \pi(\boldsymbol{q_i.v})$ by the random permutation π and encrypted to $T_{Q_i} = (\boldsymbol{M}_1^T \widehat{\boldsymbol{q}}_{\boldsymbol{i}}.\boldsymbol{v'}, \boldsymbol{M}_2^T \widehat{\boldsymbol{q}}_{\boldsymbol{i}}.\boldsymbol{v''})$ by the random binary vector $\boldsymbol{s}$ and the matrices $\boldsymbol{M}_1$, $\boldsymbol{M}_2$. Since $\mathcal{A}$ has no idea about the random value r_1, r_2, r_3, the random permutation π, the split vector $\boldsymbol{s}$, and two invertible matrix $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$, he cannot recover ciphertexts T_{Q_i}.

In Phase 1 and Phase 2 of the game, $\mathcal{A}$ can observe Q_i corresponding trapdoor T_{Q_i}. However, since $\widehat{\boldsymbol{q}}_{\boldsymbol{i}}.\boldsymbol{v}$ is a random vector determined by $\mathcal{C}$, r_1, r_2, r_3, π, $\boldsymbol{s}$, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are all random, the ciphertext T_{Q_i} is random to $\mathcal{A}$. Therefore, even if $\mathcal{A}$ has the ability to access TrapdoorGeneration, b' can only be obtained by random guessing. So we have

$$Adv_{PSDQ,\mathcal{A}}^{IND-CPA}(1^{\lambda}) = Pr(b' = b) - \frac{1}{2} \leq negl(\lambda).$$

□

Thus, according to the above specific analysis, the TrapdoorGeneration phase of the PSDQ scheme is IND-CPA.

C. Query

The cloud server has all spatial data ciphertexts $C_i = (\boldsymbol{M}_1^T \widehat{\boldsymbol{o}}_{\boldsymbol{i}}.\boldsymbol{v'}, \boldsymbol{M}_2^T \widehat{\boldsymbol{o}}_{\boldsymbol{i}}.\boldsymbol{v''})$ and the trapdoor $T_Q = (\boldsymbol{M}_1^{-1} \widehat{\boldsymbol{Q}}.\boldsymbol{v'}, \boldsymbol{M}_2^{-1} \widehat{\boldsymbol{Q}}.\boldsymbol{v''})$. Since M_1, M_2 are random matrices, $\boldsymbol{s}$ is a random binary vector and π is a random permutation, the cloud server cannot obtain $\boldsymbol{o_i.v}$ and $\boldsymbol{Q.v}$. Thus, the cloud server cannot know the original dataset and the data user's query information. Then, the cloud server calculates $C_i^T \cdot T_Q = \boldsymbol{o_i.v}^T \cdot \boldsymbol{Q.v} = \widehat{\boldsymbol{o}}_{\boldsymbol{i}}.\boldsymbol{v}^T \cdot \widehat{\boldsymbol{Q}}.\boldsymbol{v} = (\boldsymbol{o_i.lv}, \boldsymbol{o_i.kv}, -1, r_1, r_2, r_3) \cdot (\boldsymbol{Q.lv}, \boldsymbol{Q.kv}, t + k, r_4, r_5, r_6) = 0$ and checks whether $C_i^T \cdot T_Q = 0$. Because $r_1, r_2, r_3, r_4, r_5, r_6$ are unknown random numbers, so even if the cloud server can judge whether $C_i^T \cdot T_Q = 0$, it does not know whether $(\boldsymbol{o_i.lv} \cdot \boldsymbol{Q.lv}) - t = 0$ and $(\boldsymbol{o_i.kv} \cdot \boldsymbol{Q.kv}) - k = 0$. So the cloud server cannot infer the original data according to the calculation results, thereby protecting the privacy and security of the data.

VII. Performance Analysis

In this section, we conduct a detailed theoretical analysis, and extensive performance tests on the processes of Data Encryption, TrapdoorGeneration and Query. The whole experiments were carried out by using Python 3.9 programming language on 64-bit Windows 10 system and completed on Intel(R) Core(TM) i5-8300 h CPU @2.30 GHz server.

A. Theoretical Analysis

We analyze the theoretical complexity of the PSDQ, PSDQ$^+$, SKQ [14], LSKQ [14] and TSKS [34] in terms of computational and storage overheads of DataEncryption, TrapdoorGeneration and Query. Table III gives the detailed comparison results of the five schemes.

PSDQ uses EASPE to encrypt data with encryption key is $\mathcal{SK} = \{\boldsymbol{M}_1, \boldsymbol{M}_2\}$, where $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(260 + m) \times (260 + m)$ invertible matrices. Without losing generality, we assume that the size of each ciphertext is $|X|$. In DataEncryption phase, the data owner needs to encrypt all objects in the dataset, the total time complexity of the process is $\mathcal{O}(2n(260 + m)^2)$, and corresponding storage overhead is $n|X|$. In TrapdoorGeneration phase, the total time complexity of the process is $\mathcal{O}(2(260 + m)^2)$, and corresponding storage cost is $|X|$. In Query phase, the total time complexity of the process is $\mathcal{O}(2n(260 + m))$.

PSDQ$^+$ constructs a tree index structure based on PSDQ. We take a binary tree with 9 layers as an example, in which additional 510 internal nodes are generated. Therefore, in DataEncryption phase, the total time complexity of the process is $\mathcal{O}(2(n + 510)(260 + m)^2)$, and corresponding storage overhead is $(n + 510)|X|$. In TrapdoorGeneration phase, the computational and storage overheads of this process is the same as that of PSDQ. In Query phase, the total time complexity of the process is $\mathcal{O}(2(260 + m) \log n)$.

SKQ uses EASPE to encrypt data with encryption key is $\mathcal{SK} = \{\boldsymbol{M}_1, \boldsymbol{M}_2\}$, where $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(36 + m) \times (36 + m)$ invertible matrices. Without losing generality, we assume that the size of each ciphertext is $|U|$. In

TABLE III
COMPARISON OF THEORETICAL ANALYSIS

Scheme	Computation overhead			Storage overhead	
	DataEncryption	TrapdoorGeneration	Query	DataEncryption	TrapdoorGeneration
PSDQ	$\mathcal{O}(2n(260+m)^2)$	$\mathcal{O}(2(260+m)^2)$	$\mathcal{O}(2n(260+m))$	$n\|X\|$	$\|X\|$
PSDQ$^+$	$\mathcal{O}(2(n+510)(260+m)^2)$	$\mathcal{O}(2(260+m)^2)$	$\mathcal{O}(2(260+m)\log n)$	$(n+510)\|X\|$	$\|X\|$
SKQ [14]	$\mathcal{O}(16n(36+m)^2)$	$\mathcal{O}(16(36+m)^2)$	$\mathcal{O}(16n(36+m))$	$8n\|U\|$	$8\|U\|$
LSKQ [14]	$\mathcal{O}(2n(44+m)^2)$	$\mathcal{O}(2(44+m)^2)$	$\mathcal{O}(2n(44+m))$	$n\|V\|$	$\|V\|$
TSKS [34]	$\mathcal{O}(4n(\eta+m+3)^3)$	$\mathcal{O}(4n(\eta+m+3)^3)$	$\mathcal{O}(2n(\eta+m+3)^3)$	$n\|Z\|$	$2\|Z\|$

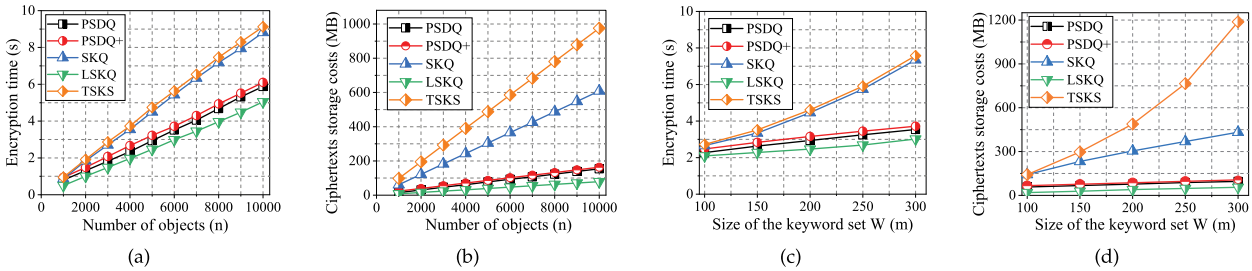

Fig. 9. Overhead of DataEncryption. (a) Computational overhead varying with n; (b) Storage overhead varying with n; (c) Computational overhead varying with m; (d) Storage overhead varying with m.

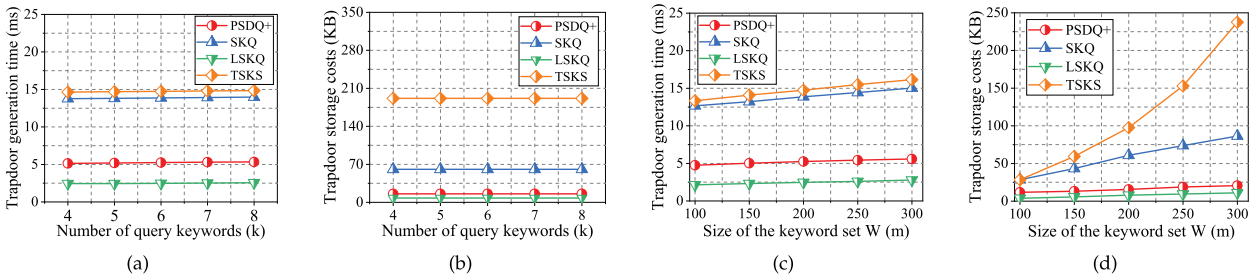

Fig. 10. Overhead of TrapdoorGeneration. (a) Computational overhead varying with k; (b) Storage overhead varying with k; (c) Computational overhead varying with m; (d) Storage overhead varying with m.

DataEncryption phase, the data owner needs to encrypt all objects in the dataset, and each encrypted object contains 8 ciphertexts. Therefore, the total time complexity of the process is $\mathcal{O}(16n(36+m)^2)$, and corresponding storage overhead is $8n|U|$. In TrapdoorGeneration phase, the data user needs to generate 8 ciphertexts for query request, so the total time complexity of the process is $\mathcal{O}(16(36+m)^2)$, and corresponding storage cost is $8|U|$. In Query phase, the cloud server needs to calculate the product of each encrypted object and the trapdoor, where they all contain 8 ciphertexts, so the total time complexity of the process is $\mathcal{O}(16n(36+m))$.

LSKQ is similar to SKQ, but the LSKQ only needs to generate one ciphertext for each object or query by using encryption key is $\mathcal{SK}=\{\boldsymbol{M}_1, \boldsymbol{M}_2\}$, where $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(44+m)\times(44+m)$ invertible matrices. Without losing generality, we assume that the size of each ciphertext is $|V|$. In DataEncryption phase, the total time complexity of the process is $\mathcal{O}(2n(44+m)^2)$, and corresponding storage overhead is $n|V|$. In TrapdoorGeneration phase, the total time complexity of the process is $\mathcal{O}(2(44+m)^2))$, and corresponding storage cost is $|V|$. In Query phase, the total time complexity of the process is $\mathcal{O}(2n(44+m))$.

TSKS uses polynomial fitting technology to fit the query range and uses randomizable matrix multiplication technology to encrypt data, in which the encryption key includes six $(\eta+m+3)\times(\eta+m+3)$ random invertible matrices. Without losing generality, we assume that the degree of polynomial fit is η, the ciphertext size of each ciphertext is $|Z|$. In the DataEncryption phase, the total time complexity of the process is $\mathcal{O}(4n(\eta+m+3)^3)$, and corresponding storage overhead is $n|Z|$. In the TrapdoorGeneration phase, the total time complexity of the process is $\mathcal{O}(8(\eta+m+3)^3)$, and corresponding storage cost is $2|Z|$. In Query phase, the total time complexity of the process is $\mathcal{O}(2n(\eta+m+3)^3)$.

B. Performance Evaluation

We use the real dataset (Yelp dataset [1]) to conduct sufficient experiments to perform performance testing and specific analysis on the Data Encryption, TrapdoorGeneration and Query processes of the PSDQ, PSDQ$^+$, SKQ [14], LSKQ [14] and TSKS [34].

[1]https://www.yelp.com/dataset

TABLE IV
COMPARISON OF COMPUTATION OVERHEAD

Phase	Variable		PSDQ	PSDQ$^+$	SKQ [14]	LSKQ [14]	TSKS [34]
DataEncryption	$m = 200, n =$	1000	0.796s	0.943s	0.863s	0.499s	0.952s
		5000	2.938s	3.225s	4.501s	2.469s	4.744s
		10000	5.879s	6.101s	8.821s	5.052s	9.124s
	$n = 5000, m =$	100	2.275s	2.471s	2.658s	2.094s	2.719s
		200	2.938s	3.155s	4.457s	2.468s	4.669s
		300	3.534s	3.720s	7.351s	3.016s	7.576s
TrapdoorGeneration	$m = 200, k =$	5	5.198ms	5.198ms	13.811ms	2.443ms	14.694ms
		6	5.248ms	5.248ms	13.871ms	2.480ms	14.744ms
		7	5.298ms	5.298ms	13.926ms	2.522ms	14.794ms
	$k = 5, m =$	100	4.748ms	4.748ms	12.662ms	2.146ms	13.346ms
		200	5.248ms	5.248ms	13.871ms	2.480ms	14.744ms
		300	5.585ms	5.585ms	15.021ms	2.781ms	16.144ms
Query	$m = 200, n =$	1000	0.0175s	0.0042s	0.0192s	0.0169s	0.138s
		5000	0.195s	0.0162s	0.335s	0.177s	0.714s
		10000	0.397s	0.0301s	0.734s	0.376s	1.788s
	$n = 5000, m =$	100	0.1045s	0.0135s	0.175s	0.081s	0.375s
		200	0.195s	0.0162s	0.335s	0.177s	0.714s
		300	0.296s	0.0193s	0.494s	0.275s	1.178s

DataEncryption. For PSDQ and PSDQ$^+$, the factors that affect the computational costs are the dimension of the vector and the size of the dataset n (i.e. the number of objects in the dataset), and the dimension of the vector is only determined by the size of the keyword set $\mathcal{W}$ (i.e. m). And we take a 9 layers binary tree as an example for PSDQ$^+$. Therefore, (1) let $m = 200$, we test the computational and storage overheads of the five schemes by varying n from 1000 to 10000. The test results are shown in Fig. 9(a), (b); (2) let $n = 5000$, we test the computational and storage overheads of the five schemes by varying m from 100 to 300. The test results are shown in Fig. 9(c), (d).

TrapdoorGeneration. Since the TrapdoorGeneration phases of PSDQ and PSDQ$^+$ are same, only PSDQ$^+$ is used as a representative. This process is similar to the data encryption process, which is one-time encryption of the data user's query content. Therefore, (1) let $m = 200$, we test the computational and storage overheads of the five schemes by changing the number of query keywords (i.e. k). The test results are shown in Fig. 10(a), (b); (2) let $k = 5$, we test the computational and storage overheads of the five schemes by varying m from 100 to 300. The test results are shown in Fig. 10(c), (d).

Query. At this phase, the cloud server needs to calculate the product of each encrypted index and trapdoor. Therefore, the factors that affect the computational costs are the size of the dataset n and the dimension of the index vector. Therefore, (1) Let $m = 200$, we test the computational overhead of the five schemes by varying n from 1000 to 10000. The test results are shown in Fig. 11(a). (2) Let $n = 5000$, we test the computational overheads of the five schemes by varying m from 100 to 300. The test results are shown in Fig. 11(b).

Table IV shows the comprehensive comparison of the computational overhead of PSDQ, PSDQ$^+$, SKQ [14], LSKQ [14] and TSKS [34] under different variables in the Data Encryption, TrapdoorGeneration and Query phases. It can be seen from the extensive specific experimental results that the PSDQ$^+$ scheme is very efficient in practice.

Because the size of the keyword set has a great impact on the costs of our schemes, we let $n = 5000$ and test PSDQ, PSDQ$^+$, SKQ [14], LSKQ [14] and TSKS [34] by increasing the size of m. Consist with the experimental settings in most papers [31], [35], $m = 2000$ is a large enough value to represent the total number of different keywords in the corpus. The experimental results are shown in Table V. The results show that PSDQ$^+$ is still very efficient for users and the costs of the TrapdoorGeneration and Query are still at the millisecond level. For example, when $m = 2000$, the trapdoor generation time and the query time are only 9.937 ms and 43.5 ms, respectively.

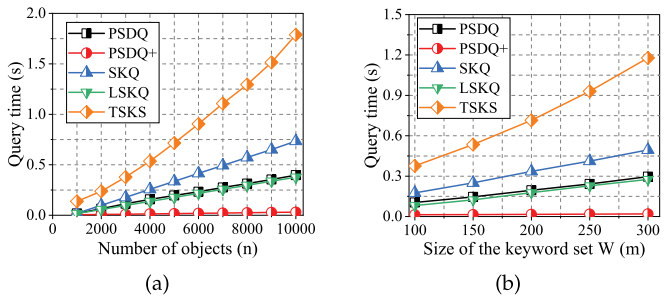

Fig. 11. Overhead of Query. (a) Computational overhead varying with n; (b) Computational overhead varying with m.

TABLE V
COMPARISON OF COMPUTATION OVERHEAD BY VARYING m

Scheme	TrapdoorGeneration			Query		
	m=1000	m=1500	m=2000	m=1000	m=1500	m=2000
PSDQ	7.583ms	8.672ms	9.937ms	0.559s	0.669s	0.768s
PSDQ$^+$	7.583ms	8.672ms	9.937ms	0.0281s	0.0363s	0.0435s
SKQ [14]	24.928ms	30.121ms	38.726ms	0.824s	1.258s	1.607s
LSKQ [14]	4.589ms	5.843ms	7.143ms	0.518s	0.605s	0.696s
TSKS [34]	0.639s	0.812s	1.053s	23.467s	30.873s	39.791s

VIII. CONCLUSION

In this article, we improve our conference version [14]. Specifically, to avoid false positives of [14] and improve query efficiency, we first propose a linear Privacy-preserving Spatial Data Query (PSDQ) scheme by designing a new unified index

structure, which does not cause any loss of accuracy. Then, we propose a more efficient PSDQ scheme (PSDQ$^+$) by designing a Geohash-based R-tree structure (called GR-tree) and a pruning strategy based on both spatial range and keywords, such that the search complexity is sublinear. Finally, We give the formal security analysis to prove that our schemes are secure against IND-CPA, and conduct extensive experiments to demonstrate that our schemes are efficient in practice. As part of our future work, we will attempt to protect access pattern to further reduce privacy disclosure and improve the security of the scheme.

REFERENCES

[1] R. Li, A. X. Liu, A. L. Wang, and B. Bruhadeshwar, "Fast and scalable range query processing with strong privacy protection for cloud computing," *IEEE Trans. Netw.*, vol. 24, no. 4, pp. 2305–2318, Aug. 2016.

[2] G. Xiao, F. Wu, X. Zhou, and K. Li, "Probabilistic top-k range query processing for uncertain datasets," *J. Intell. Fuzzy Syst.*, vol. 31, no. 2, pp. 1109–1120, 2016.

[3] K. Xue, S. Li, J. Hong, Y. Xue, N. Yu, and P. Hong, "Two-cloud secure database for numeric-related SQL range queries with privacy preserving," *IEEE Trans. Inf. Forensics Secur.*, vol. 12, no. 7, pp. 1596–1608, Jul. 2017.

[4] C. Zhang, L. Zhu, C. Xu, J. Ni, C. Huang, and X. Shen, "Location privacy-preserving task recommendation with geometric range query in mobile crowdsensing," *IEEE Trans. Mobile Comput.*, vol. 21, no. 12, pp. 4410–4425, Dec. 2022, doi: 10.1109/TMC.2021.3080714.

[5] F. Song, Z. Qin, D. Liu, J. Zhang, X. Lin, and X. Shen, " Privacy-preserving task matching with threshold similarity search via vehicular crowdsourcing," *IEEE Trans. Veh. Technol.*, vol. 70, no. 7, pp. 7161–7175, Jul. 2021, doi: 10.1109/TVT.2021.3088869.

[6] H. Xie, Y. Guo, and X. Jia, "A privacy-preserving online ride-hailing system without involving a third trusted server," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 3068–3081, 2021, doi: 10.1109/TIFS.2021.3065832.

[7] D. Negi, S. Ray, and R. Lu, "Pystin: Enabling secure LBS in smart cities with privacy-preserving top-k SpatialCTextual query," *IEEE Internet Things J.*, vol. 6, no. 5, pp. 7788–7799, Oct. 2019.

[8] Z. Peng, C. Xu, H. Wang, J. Huang, J. Xu, and X. Chu, "P ^{2} B-Trace: Privacy-preserving blockchain-based contact tracing to combat pandemics," in *Proc. Int. Conf. Manage. Data*, 2021, pp. 2389–2393.

[9] X. Wang, J. Ma, and X. Liu, "Enabling efficient and expressive spatial keyword queries on encrypted data," in *Proc. IEEE Int. Conf. Acoust., Speech Signal Process.*, 2021, pp. 2670–2674.

[10] X. Wang, J. Ma, Y. Miao, X. Liu, D. Zhu, and R. H. Deng, "Fast and secure location-based services in smart cities on outsourced data," *IEEE Internet Things J.*, vol. 8, no. 24, pp. 17639–17654, Dec. 2021.

[11] W. K. Wong, D. W. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc. Int. Conf. Manage. Data*, 2009, pp. 139–152.

[12] W. Lin, K. Wang, Z. Zhang, and H. Chen, "Revisiting security risks of asymmetric scalar product preserving encryption and its variants," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2017, pp. 1116–1125.

[13] S. Su, Y. Teng, X. Cheng, K. Xiao, G. Li, and J. Chen, "Privacy-preserving top-k spatial keyword queries in untrusted cloud environments," *IEEE Trans. Serv. Comput.*, vol. 11, no. 5, pp. 796–809, Sep./Oct. 2018.

[14] Y. Yang, Y. Miao, K.-K. R. Choo, and R. H. Deng, "Lightweight privacy-preserving spatial keyword query over encrypted cloud data," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2022, pp. 392–402.

[15] P. Wang and C. V. Ravishankar, "Secure and efficient range queries on outsourced databases using Rp-trees," in *Proc. Int. Conf. Data Eng.*, 2013, pp. 314–325.

[16] B. Wang, Y. Hou, M. Li, H. Wang, and H. Li, "Maple: Scalable multi-dimensional range search over encrypted cloud data with tree-based index," in *Proc. ACM Symp. Inf., Comput. Commun. Secur.*, 2014, pp. 111–122.

[17] D. Boneh and B. Waters, "Conjunctive, subset, and range queries on encrypted data," in *Proc. Theory Cryptogr. Conf.*, 2007, pp. 535–554.

[18] B. Wang, M. Li, H. Wang, and H. Li, "Circular range search on encrypted spatial data," in *Proc. IEEE Conf. Commun. Netw. Secur.*, 2015, pp. 182–190.

[19] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proc. Theory Cryptogr. Conf.*, 2009, pp. 457–473.

[20] Z. Zheng, Z. Cao, and J. Shen, "Practical and secure circular range search on private spatial data," in *Proc. IEEE Int. Conf. Trust, Secur. Privacy Comput. Commun.*, 2020, pp. 639–645.

[21] A. Boldyreva, N. Chenette, Y. Lee, and A. O'Neil, "Order-preserving symmetric encryption," in *Proc. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2009, pp. 224–241.

[22] F. Kerschbaum and A. Schropfer, "Optimal average-complexity IdealSecurity order-preserving encryption," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2014, pp. 275–286.

[23] C. Zhang, L. Zhu, C. Xu, J. Ni, C. Huang, and X. S. Shen, "Location privacy-preserving task recommendation with geometric range query in mobile crowdsensing," *IEEE Trans. Mobile Comput.*, 2021, doi: 10.1109/TMC.2021.3080714.

[24] P. Strobach, "Solving cubics by polynomial fitting," *J. Comput. Appl. Math.*, vol. 235, no. 9, pp. 3033–3052, 2011.

[25] D. Boneh, G. D. Crescenzo, R. Ostrovsky, and G. Persiano, "Public key encryption with keyword search," in *Proc. Int. Conf. Theory Appl. Cryptogr. Techn.*, 2004, pp. 506–522.

[26] X. Liu, G. Yang, W. Susilo, J. Tonien, X. Liu, and J. Shen, "Privacy-preserving multi-keyword searchable encryption for distributed systems," *IEEE Trans. Parallel Distrib. Syst.*, vol. 32, no. 3, pp. 561–574, Mar. 2021.

[27] Y. Tang, L. Liu, A. Iyengar, K. Lee, and Q. Zhang, "e-PPI:Locator service in information networks with personalized privacy preservation," in *Proc. IEEE Int. Conf. Distrib. Comput. Syst.*, 2014, pp. 186–197.

[28] Y. Tang and L. Liu, "Privacy-preserving multi-keyword search in information networks," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 9, pp. 2424–2437, Sep. 2015.

[29] Y. Tang, T. Wang, L. Liu, X. Hu, and J. Jang, "Lightweight authentication of freshness in outsourced key-value stores," in *Proc. Annu. Comput. Secur. Appl. Conf.*, 2014, pp. 176–185.

[30] N. Cui, J. Li, X. Yang, B. Wang, M. Reynolds, and Y. Xiang, "When geo-text meets security: Privacy-preserving boolean spatial keyword queries," in *Proc. IEEE Int. Conf. Data Eng.*, 2019, pp. 1046–1057.

[31] X. Wang et al., "Search me in the dark: Privacy-preserving boolean range query over encrypted spatial data," in *Proc. IEEE Int. Conf. Comput. Commun.*, 2020, pp. 2253–2262.

[32] J. R. Bitner, G. Ehrlich, and E. M. Reingold, "Efficient generation of the binary reflected gray code and its applications," *Commun. ACM*, vol. 19, no. 9, pp. 517–521, 1976.

[33] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, and D. Mukhopadhyay, "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. ACM Conf. Comput. Commun. Secur.*, 2018, pp. 745–762.

[34] Y. Yang, Y. Miao, Z. Ying, J. Ning, X. Meng, and K. -K. R. Choo, "Privacy-preserving threshold spatial keyword search in cloud-assisted IIoT," *IEEE Internet Things J.*, vol. 9, no. 18, pp. 16990–17001, Sep. 2022, doi: 10.1109/JIOT.2021.3138136.

[35] Q. Tong, Y. Miao, H. Li, X. Liu, and R. Deng, "Privacy-preserving ranked spatial keyword query in mobile cloud-assisted fog computing," *IEEE Trans. Mobile Comput.*, vol. 22, no. 6, pp. 3604–3618, Jun. 2023, doi: 10.1109/TMC.2021.3134711.

[36] R. Guo, Y. Wu, R. Liu, and H. Chen, "LuxGeo: Efficient and secure enhanced geometric range queries," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 2, pp. 1775–1790, Feb. 2023, doi: 10.1109/TKDE.2021.3093909.

[37] Q. Huang, J. Du, G. Yan, Y. Yang, and Q. Wei, "Privacy-preserving spatio-temporal keyword search for outsourced location-based services," *IEEE Trans. Serv. Comput.*, vol. 15, no. 6, pp. 3443–3456, Nov./Dec. 2021, doi: 10.1109/TSC.2021.3088131.

[38] N. Davis, G. Raina, and K. Jagannathan, "Taxi demand forecasting: A. HEDGE-Based tessellation strategy for improved accuracy," *IEEE Trans. Intell. Transp. Syst.*, vol. 19, no. 11, pp. 3686–3697, Nov. 2018.

[39] R. Gelda, K. Jagannathan, and G. Raina, "Forecasting supply in Voronoi regions for app-based taxi hailing services," in *Proc. IEEE Int. Conf. Adv. Logistics Transport*, 2017, pp. 1–6.

[40] A. Guttman, "R-trees: A dynamic index structure for spatial searching," in *Proc. Int. Conf. Manage. Data*, 1984, pp. 47–57.

[41] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin, "Enabling efficient and geometric range query with access control over encrypted spatial data," *IEEE Trans. Inf. Forensics Secur.*, vol. 14, no. 4, pp. 870–885, Apr. 2019.

[42] Z. Zhang et al., "Practical access pattern privacy by combining PIR and oblivious shuffle," in *Proc. ACM Int. Conf. Inf. Knowl. Manage.*, 2019, pp. 1331–1340.

Yinbin Miao (Member, IEEE) received the BE degree from the Department of Telecommunication Engineering from Jilin University, Changchun, China, in 2011, and the PhD degree from the Department of Telecommunication Engineering, Xidian University, Xi'an, China, in 2016. He is also a postdoctoral researcher with Nanyang Technological University, Singapore from September 2018 to September 2019, and a postdoctoral researcher with the City University of Hong Kong, Hong Kong from December 2019 to December 2021. He is currently an associate professor with the Department of Cyber Engineering, Xidian University. His research interests include information security and applied cryptography.

Yutao Yang received the BE degree from the Department of Electronical Information Science and Technology, Henan Agricultural University, Zhengzhou, China, in 2021. He is currently working toward the ME degree with the Department of Cyber Engineering, Xidian University, Xi'an, China. His research interests include information security and applied cryptography.

Xinghua Li received the ME and PhD degrees in computer science from Xidian University, Xi'an, China, in 2004 and 2007, respectively. He is currently a professor with the School of Cyber Engineering, Xidian University, China. His research interests include wireless networks security, privacy protection, cloud computing, and security protocol formal methodology.

Linfeng Wei received the BS degree from the School of Information Science and Technology, Jinan University, Guangzhou, China, in 2010, and the PhD degree from the School of Cyber Security, Jinan University, Guangzhou, China, in 2020. He is currently an associate professor with the School of Cyber Security, and assistant director of the National and Local Joint Engineering Research Center for Cyber Security Detection and Protection, Jinan University, Guangzhou, China, and his current research focuses on cyber threat intelligence, data security and privacy preserving, AI security, and etc.

Zhiquan Liu (Member, IEEE) received the BS degree from the School of Science, Xidian University, Xi'an, China, in 2012, and the PhD degree from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2017. He is currently an associate professor with the College of Cyber Security, Jinan University, Guangzhou, China, and his current research focuses on trust management and privacy preservation in vehicular networks and UAV networks.

Robert H. Deng (Fellow, IEEE) is AXA chair professor of cybersecurity and professor of Information Systems with the School of Information Systems, Singapore Management University since 2004. His research interests include data security and privacy, multimedia security, network and system security. He served/is serving on the editorial boards of many international journals, including *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*. He has received the Distinguished Paper Award (NDSS 2012), Best Paper Award (CMS 2012), Best Journal Paper Award (IEEE Communications Society 2017).