7-2022

Lightweight privacy-preserving spatial keyword query over encrypted cloud data

Yutao YANG

Yinbin MIAO

Kim-Kwang Raymond CHOO

Robert H. DENG
Singapore Management University, robertdeng@smu.edu.sg

2022 IEEE 42nd International Conference on Distributed Computing Systems (ICDCS) | 978-1-6654-7177-0/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICDCS54860.2022.00045

Lightweight Privacy-Preserving Spatial Keyword Query over Encrypted Cloud Data

Yutao Yang*, Yinbin Miao*(Corresponding author), Kim-Kwang Raymond Choo†, and Robert H. Deng‡
*School of Cyber Engineering, Xidian University, Xi'an, China
†Department of Information Systems and Cyber Security, University of Texas at San Antonio, San Antonio, USA
‡School of Information Systems, Singapore Management University, Singapore
Email: yangyutao_13@163.com, ybmiao@xidian.edu.cn, raymond.choo@fulbrightmail.org, robertdeng@smu.edu.sg

***Abstract*—With the rapid development of geographic location technology and the explosive growth of data, a large amount of spatio-textual data is outsourced to the cloud server to reduce the local high storage and computing burdens, but at the same time causes security issues such as data privacy leakage. Thus, extensive privacy-preserving spatial keyword query schemes have been proposed. Most of the existing schemes use Asymmetric Scalar-Product-Preserving Encryption (ASPE) for encryption, but ASPE has proven to be insecure. And the existing spatial range query schemes require users to provide more information about the query range and generate a large amount of ciphertext, which causes high storage and computational burdens. To solve these issues, in this paper we introduce some random numbers and a random permutation to enhance the security of ASPE scheme, and then propose a novel privacy-preserving Spatial Keyword Query (SKQ) scheme based on the enhanced ASPE and Geohash algorithm. In addition, we design a more Lightweight Spatial Keyword Query (LSKQ) scheme by using a unified index for spatial range and multiple keywords, which not only greatly decreases SKQ's storage and computational costs but also requires users to provide little information about query region. Finally, formal security analysis proves that our schemes have Indistinguishability under Chosen Plaintext Attack (IND-CPA), and extensive experiments demonstrate that our enhanced scheme is efficient and practical.**

***Index Terms*—spatio-textual data, outsourcing data, cloud server, privacy-preserving, spatial keyword query**

I. Introduction

With the increasing popularity of GPS devices in the mobile Internet, spatio-textual data is widely used in various applications such as location-based services [1], [2], [3], mobile crowdsourcing [4], [5], task recommendation [6] and personalized query services. For example, Facebook can help users find like-minded friends, and Google Maps can help users search for places of interest [7]. Due to the rapid growth of data volume, more and more service providers (i.e., Google, Amazon, etc.) tend to outsource their spatio-textual data to cloud servers to reduce local storage and computing burden, but this will lead to security issues such as privacy leakages [8]. Thus, the privacy-preserving spatial keyword query has been extensively studied. However, there are still several issues to be solved.

The first issue is that most of the existing spatial range query schemes require users to provide more query information and need to generate a large number of ciphertexts for each spatial object. For example, for the search rectangle range [26], users need to provide at least two 2-dimensional coordinates of the lower left corner and the upper right corner of the rectangle, which generates two ciphertexts for two coordinates respectively. And for arbitrary geometric range query implemented by polynomial fitting technology [29], users need to provide an accurate closed curve formed by the query area, which generates two ciphertexts for the upper half and the lower half of the closed curve respectively. Thus, the above solutions require more information about the query region from users, and also produce a large number of ciphertexts, which incurs high storage and computational costs and more inconvenience to users in the real world.

The second issue is that many existing schemes use Asymmetric Scalar-Product-Preserving Encryption (ASPE) [9] to encrypt spatio-textual data. ASPE is a very efficient searchable encryption scheme to achieve scalar product based on vector search, which can be extended and applied to various search schemes. However, ASPE has been proven to be insecure, i.e., it cannot resist the known-plaintext attack as shown in [10]. Therefore, ASPE-based schemes may lead to the disclosure of a large number of private information, resulting in irreparable harm.

To solve the above issues, we devise a novel privacy-preserving spatial keyword query scheme. We use Geohash algorithm [11], [12], [13], [14] to achieve approximate spatial range query, which only requires users to provide less query information. Then, we build a unified index for spatial range and multiple keywords to realize a lightweight spatial keyword query scheme. And we enhance the security of ASPE by adding some random numbers and a random permutation. The main contributions of our work are shown as follows:

1) We use Geohash algorithm to realize approximate spatial range query, which only requires users to provide a spatial coordinate and an approximate range value. Then, we design a unified index for spatial range and multiple keywords to realize spatial keyword query.
2) We enhance the security of ASPE by adding some random numbers and a random permutation to make it Indistinguishability under Chosen Plaintext Attack (IND-CPA).
3) We first propose a naive scheme by simply combining

2575-8411/22/$31.00 ©2022 IEEE
DOI 10.1109/ICDCS54860.2022.00045

Geohash algorithm with enhanced ASPE, and then improve it to greatly reduce its storage and computational costs in data encryption, trapdoor generation and query phases.

4) We give the formal security analysis and extensive experiments to show the security and feasibility of our schemes.

II. Related Work

Privacy-preserving spatial query. Due to the rapid development of geographic location technology, spatial query has attracted extensive attention. To find the k-Nearest-Neighbor (kNN) objects, Choi *et al.* proposed a secure kNN scheme based on mutable order preserving encoding [15]. However, this scheme only supports searching a single spatial point. To search for rectangular areas, Wang *et al.* proposed a hierarchical encrypted index ($\hat{R}$-tree) by using ASPE, which enables secure and efficient execution of range queries over encrypted database [16]. To search for circular areas, Zhu *et al.* proposed an encrypted spatial range query algorithm using an improved homomorphic encryption over composite order group [17]. To search for areas of arbitrary shape, Xu *et al.* used secure kNN calculation, polynomial fitting technology and order-preserving encryption technology to propose an efficient geometric range query scheme, which support search and access control over encrypted spatial data [18].

Privacy-preserving keyword query. Keyword queries aim to find objects based on textual relevance ranking over encrypted cloud data. For example, Wang *et al.* used searchable symmetric encryption and order-preserving symmetric encryption to achieve secure ranked keyword search [19]. To further improve the efficiency of [19], Wang *et al.* leveraged the edit distance as a similarity metric and suppressing technique to construct a storage-efficient similarity keyword set [20]. But these schemes just support the single keyword search. To this end, Sun *et al.* proposed an effective privacy-preserving multi-keyword query by building the search index based on term frequency and the vector space model with cosine similarity measure [21]. To search for multiple keywords efficiently, Zheng *et al.* proposed an efficient keyword set containment search scheme, which builds a radix tree to effectively prune the keyword set and uses ASPE to design a set containment and set intersection encryption to achieve secure search [22].

Privacy-preserving spatial keyword query. To search for spatial and keyword at the same time, top-k spatial keyword query and Boolean spatial keyword query have been extensively studied. Su *et al.* proposed the anchor-based position determination technique and position-distinguished trapdoor generation technique for secure spatio-textual similarity computation to achieve secure top-k spatial keyword query [23]. Meng *et al.* proposed a new practical protocol to support top-k queries by building an Encrypted Hash List, which allows the servers to homomorphically evaluate the equality relation between two objects [24]. Negi *et al.* proposed a new privacy-preserving top-k spatio-textual keyword query scheme, which combines Boneh-Goh-Nissim homomorphic encryption and hash bucket techniques to protect spatial location and uses one-way hash function to protect text information [25]. For boolean spatial keyword query. Cui *et al.* devised a novel privacy-preserving spatial-textual Bloom Filter encoding structure and an encrypted R-tree index to maintain both spatial and text information together in a secure way [26]. Wang *et al.* designed a new encoding scheme, which uses Gray code and bitmap to uniformly encode spatial text data, and then used Symmetric-key Hiden Vector Encryption [28] to encrypt the encoded vector to solve the Boolean spatial keyword query problem [27]. Song *et al.* used polynomial fitting technology and ASPE to achieve secure arbitrary geometric range query, and then utilized polynomial function and random matrix multiplication to achieve secure multi-dimensional keyword similarity search based on access control [29].

However, the above solutions cannot solve our problems simultaneously. Therefore, we propose a lightweight privacy-preserving spatial keyword query scheme, which only requires users to provide less information about the query range to greatly reduce storage and computational costs and enhance the security of ASPE. The comparison between our schemes and the above scheme is shown in TABLE I.

TABLE I
COMPARISON BETWEEN PREVIOUS SCHEMES AND OUR SCHEMES.

Schemes	Spatial query	Keyword query	Range query	High efficiency	Resist attack types
[15]	✓	✗	✗	NO	IND-OCPA
[18]	✓	✗	✓	YES	KCA/KBA
[21]	✗	✓	✗	NO	KCA/KBA
[22]	✗	✓	✗	YES	COA/KPA
[23]	✓	✓	✗	YES	CPA/KPA
[26]	✓	✓	✓	NO	KBA
[29]	✓	✓	✓	NO	COA/CPA
SKQ	✓	✓	✓	NO	COA/CPA
LSKQ	✓	✓	✓	YES	COA/CPA

Notes: KCA:Known-Ciphertext Attack; CPA:Chosen-Plaintext Attack; KPA: Known-Plaintext Attack; COA: Ciphertext-Only Attack; KBA: Known-Background Attack.

III. Preliminaries

In this section, we mainly review some related background knowledge, including Geohash algorithm [11], [12], [13], [14], Asymmetric Scalar-Product-Preserving Encryption (ASPE) [9], which is treated as the basis of our schemes.

A. Geohash Algorithm

Geohash is a geocoding method, which 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 longer the Geohash code is, the smaller the divided grid and the more accurate the position of the spatial points are represented. For example, in Fig. 1, the red dot (8, 13) can be denoted as

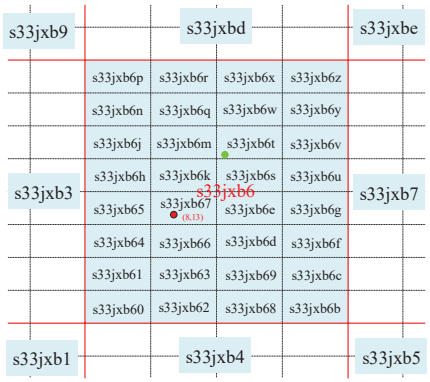

Fig. 1. An example of Geohash coding.

"$s33jxb67$" with $\pm 19m$ rectangular range, "$s33jxb6$" with $\pm 76m$, or "$s33jxb$" with $\pm 610m$. The precision of each Geohash code is shown in TABLE II.

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

Therefore, we can determine the distance between two points by matching the Geohash code. For example, in Fig. 1, the prefix of the Geohash code of the red dot and the green dot is "$s33jxb6$", then we can know that the green dot is within $\pm 76m$ of the red dot. Therefore, we can judge whether the spatial point is within the query range of the query point by 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. Asymmetric Scalar-Product-Preserving Encryption (ASPE)

ASPE is an important technology for similarity search over encrypted data, which can calculate the inner product of two vectors without revealing privacy. Assuming that $\boldsymbol{p}$ and $\boldsymbol{q}$ are two d-dimensional vectors, ASPE consists of the following four algorithms:

- ASPE.KeyGen(1^{ζ}) $\rightarrow$ sk: Given a security parameter ζ, this algorithm generates a secret key $sk = \{\boldsymbol{s}, \boldsymbol{M}_1, \boldsymbol{M}_2\}$, where $\boldsymbol{s}$ is a random d-dimensional bit vector, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $d \times d$ dimensional invertible matrices.
- ASPE.Enc($\boldsymbol{p}$, sk) $\rightarrow$ C: Given the secret key sk and a vector $\boldsymbol{p}$, this algorithm encrypts $\boldsymbol{p}$ as $C = (\boldsymbol{M}_1^T\boldsymbol{p}', \boldsymbol{M}_2^T\boldsymbol{p}'')$, where $\boldsymbol{p}$ is split into two vectors $\boldsymbol{p}'$, $\boldsymbol{p}''$ by using the bit vector $\boldsymbol{s}$ by Eq. 1.

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

- ASPE.TrapGen($\boldsymbol{q}$, sk) $\rightarrow$ T_Q: Given the secret key sk and a vector $\boldsymbol{q}$, this algorithm encrypts $\boldsymbol{q}$ as $T_Q = (\boldsymbol{M}_1^{-1}\boldsymbol{q}', \boldsymbol{M}_2^{-1}\boldsymbol{q}'')$, where $\boldsymbol{q}$ is split into two vectors $\boldsymbol{q}'$, $\boldsymbol{q}''$ by using the bit vector $\boldsymbol{s}$ by Eq. 2.

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

- ASPE.Query(C, T_Q) $\rightarrow$ $\boldsymbol{p}^T \cdot \boldsymbol{q}$: This algorithm calculates the inner product of C and T_Q as $\boldsymbol{p}^T \cdot \boldsymbol{q}$ by Eq. 3.

$$\begin{aligned} C^T \cdot T_Q = &((\boldsymbol{p}')^T\boldsymbol{M}_1) \cdot (\boldsymbol{M}_1^{-1}\boldsymbol{q}') \\ &+ ((\boldsymbol{p}'')^T\boldsymbol{M}_2) \cdot (\boldsymbol{M}_2^{-1}\boldsymbol{q}'') = \boldsymbol{p}^T \cdot \boldsymbol{q} \end{aligned} \tag{3}$$

IV. Problem Formulation

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

A. System Model

In this paper, we consider a cloud data outsourcing scenario. As depicted in Fig. 2, 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:

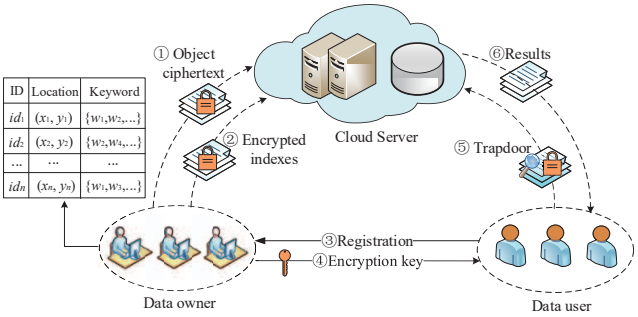

Fig. 2. System model.

- **Data Owner:** The data owner first generates a corresponding index for each object in the dataset D, and then encrypts the spatial data objects and indexes and outsources them to the cloud server.
- **Data user:** The user encrypts the query content as a trapdoor, and then submits the trapdoor to the cloud server.
- **Cloud Server:** The cloud server has unlimited storage and computing capabilities. It can store encrypted objects and indexes outsourced by the data owner, and provide query services for users.

The data owner outsources the encrypted objects and indexes to the cloud server (step ①, ②). The user is authorized after registering himself with the data owner (step ③), then obtains the encryption key from the data owner (step ④). When the user requests a query, he encrypts the query content as a trapdoor, and submits the trapdoor to the cloud server (step ⑤). After receiving the trapdoor, the cloud server finds the objects that meet the user's query requirements and returns the id of the object to the data user (step ⑥).

B. Problem Definition

Consider the example shown in Fig. 3, a user wants to find objects that meet all query keywords $\{w_2, w_5\}$ in the vicinity. As can be seen from Fig. 3, only o_3 contains all query keywords, which is most likely to be returned as a result.

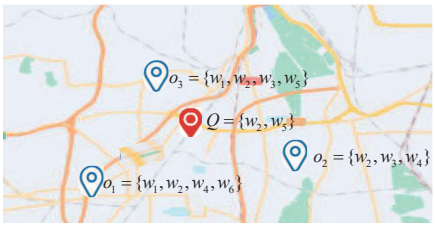

Fig. 3. A example of spatial keyword query.

Let $W = \{w_1, w_2, \ldots, w_m\}$ be a total keyword set, $D = \{o_1, o_2, \ldots, o_n\}|_{1 \le i \le n}$ be a spatio-textual dataset owned by the data owner. Each object o_i in D has a unique identity id_i and a tuple $(o_i.l, o_i.k)$, where $o_i.l = (x_i, y_i)$ represents spatial geographic location, $o_i.k = \{w_1, w_2, \ldots, w_g\} \subseteq W$ represents a set of keywords. Let $Q = \{Q.l, Q.k, R\}$ be a search request, where $Q.l = (q_x, q_y)$ represents spatial geographic location of query and $Q.k = \{w'_1, w'_2, \ldots, w'_k\} \subseteq W$ represents a set of query keywords, R represents the query range. Now we give the definition of our scheme as follows:

Definition 1: *(Lightweight Spatial Keyword Query, LSKQ) Given a spatio-textual dataset $D = \{o_1, o_2, ..., o_n\}$, where each o_i is encoded as a vector $\boldsymbol{o_i.v}$ and then encrypted as C_i. Given a query $Q = \{Q.l, Q.k, R\}$, it is encoded as a vector $\boldsymbol{Q.v}$ and then encrypted to T_Q. A LSKQ is to retrieve a subset $Result = \{id_1, id_2, ..., id_h\}$ from D, such that $\forall id_r \in Result$, $1 \le r \le h$, $C_r^T \cdot T_Q = 0$.*

C. Threat Model

In this paper, the cloud server is considered to be honest-but-curious, which honestly performs the established protocol but may be curious to collect or analyze the meaningful information such as location, text information and user's queries, which incurs security concerns. We assume that the data owner and data users are trusted and do not collude with the cloud server and unauthorized data users. In addition, we assume that the authorization, access and transmission channels are secure. According to the available information known by the cloud server, we consider the following two threat models:

- **Ciphertext-Only-Attack Model:** In this model, it is assumed that the cloud server knows the ciphertext of all the outsourced data, and can observe the encrypted trapdoors, but cannot obtain the corresponding plaintext.
- **Chosen-Plaintext-Attack Model:** In this model, it is assumed that the cloud server can access object ciphertexts corresponding to the 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:

- **Data privacy:** The data privacy includes the privacy of dataset and query content. In other words, the plaintext dataset and query should not be leaked to the cloud server and unauthorized entities.
- **Unlinkability of trapdoor:** 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 search.

V. Our Proposed Scheme

In this section, we first enhance the security of ASPE, then propose a naive spatial keyword query scheme based on the enhanced ASPE and Geohash algorithm. Finally, we improve the naive scheme and propose a more lightweight scheme to greatly reduce its computational and storage overhead.

A. Technical Overview

ASPE is a very common encryption method in spatial keyword query schemes, but it has been proved that it cannot resist the known-plaintext attack as shown in [10]. Therefore, we first enhance the security of ASPE by adding some random numbers and a random permutation to make it Indistinguishability under Chosen Plaintext Attack (IND-CPA).

The existing spatial range query schemes require users to provide more information about the query area and generate a large number of ciphertexts, resulting in huge storage and computing overhead. Therefore, to solve these problems, we use the Geohash algorithm to implement a lightweight approximate range query. We first propose a naive spatial keyword query scheme by simply combining Geohash algorithm with enhanced ASPE. Specifically, in the data encryption phase, for each object o_i in the dataset D, which spatial location $o_i.l$ is first mapped as an 8-character Geohash code and then converted as an 8-dimensional order vector $\widetilde{\boldsymbol{o_i.l}}$ according to Base32, finally, each dimension of the order vector is encoded as a 32-dimensional spatial vector $\boldsymbol{o_i.lv_j}|_{1 \le j \le 8}$. The keyword set $o_i.k$ of o_i is converted as an m-dimensional text vector $\boldsymbol{o_i.kv}$. Thus, we generate 8 data vectors $\boldsymbol{o_i.v_j}$ for each object o_i by appending a text vector after each spatial vector, and then use enhanced ASPE to encrypt them as $C_{i,j}|_{1 \le i \le n, 1 \le j \le 8}$. In the trapdoor generation phase, for query $Q = \{Q.l, Q.k, R\}$, $Q.l$ is first mapped as a t-character Geohash code according to R and then converted as an 8-dimensional order vector $\widetilde{\boldsymbol{Q.l}}$, where the remaining $8 - t$ dimensions are set as 0, finally, each dimension of the order vector is encoded as a 32-dimensional spatial vector $\boldsymbol{Q.lv_j}|_{1 \le j \le 8}$. $Q.k$ is converted as an m-dimensional vector $\boldsymbol{Q.kv}$. The generation method of each query vector $\boldsymbol{Q.v_j}$ is similar to that of each data vector $\boldsymbol{o_i.v_j}$ and then is encrypted as 8 trapdoors $T_{Q_j}|_{1 \le j \le 8}$. 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 $C_{i,j}$ and T_{Q_j}. However, the naive scheme causes high storage and computational burdens.

Then we improved the naive scheme to construct a lightweight spatial keyword query scheme. Specifically, we convert each dimension of the order vector $\widehat{\boldsymbol{o_i.l}}$ of each object into 5-bit binary to generate a 40-dimensional spatial vector $\boldsymbol{o_i.lv}$, and form a data vector $\boldsymbol{o_i.v}$ by appending a text vector after spatial vector, so each object o_i contains only one ciphertext C_i. The user's query vector is similar to the data vector generation method, only one query vector $\boldsymbol{Q.v}$ needs to be generated and encrypted as a trapdoor T_Q. In the query phase, the cloud server determines whether the object o_i meets the query conditions only by calculating the inner product of each C_i and T_Q. Therefore, the lightweight scheme greatly improves the calculation and storage overhead in the KeyGeneration, TrapdoorGeneration and Query phases.

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

We enhance the security of ASPE by adding some random numbers and a random permutation. The specific structure is as follows:

- EASPE.KeyGen(1^ζ) $\rightarrow sk$: Given a security parameter ζ, this algorithm generates a secret key $sk = \{\boldsymbol{s}, \boldsymbol{M}_1, \boldsymbol{M}_2, \pi, r_1, r_2, r_3, r_4, r_5, r_6\}$, where $\boldsymbol{s}$ is a random $(d+3)$-dimensional bit vector, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(d+3)\times(d+3)$ dimensional invertible matrices, π is a random permutation, i.e., $\pi : \mathbb{R}^{(d+3)} \rightarrow \mathbb{R}^{(d+3)}$, r_1, r_2, r_3, r_4, r_5 and r_6 are random numbers and satisfy $r_1 r_4 + r_2 r_5 + r_3 r_6 = 0$.
- EASPE.Enc($\boldsymbol{p}$, sk) $\rightarrow$ C: Given the secret key sk and a d-dimensional vector $\boldsymbol{p}$, $\boldsymbol{p}$ is first extended to a $(d+3)$-dimensional vector $\boldsymbol{p_v} = (\boldsymbol{p}, r_1, r_2, r_3)$ and then permuted as $\widehat{\boldsymbol{p_v}} = \pi(\boldsymbol{p_v})$ by π. Finally, this algorithm encrypts $\widehat{\boldsymbol{p_v}}$ as $C = (\boldsymbol{M}_1^T \widehat{\boldsymbol{p_v}}', \boldsymbol{M}_2^T \widehat{\boldsymbol{p_v}}'')$, where $\widehat{\boldsymbol{p_v}}$ is split into two vectors $\widehat{\boldsymbol{p_v}}'$, $\widehat{\boldsymbol{p_v}}''$ by using the bit vector $\boldsymbol{s}$ by Eq. 4.

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

- EASPE.TrapGen($\boldsymbol{q}$, sk) $\rightarrow T_Q$: Given the secret key sk and a d-dimensional vector $\boldsymbol{q}$, $\boldsymbol{q}$ is first extended to a $(d+3)$-dimensional vector $\boldsymbol{q_v} = (\boldsymbol{q}, r_4, r_5, r_6)$ and then permuted as $\widehat{\boldsymbol{q_v}} = \pi(\boldsymbol{q_v})$ by π. Finally, this algorithm encrypts $\widehat{\boldsymbol{q_v}}$ as $T_Q = (\boldsymbol{M}_1^{-1} \widehat{\boldsymbol{q_v}}', \boldsymbol{M}_2^{-1} \widehat{\boldsymbol{q_v}}'')$, where $\widehat{\boldsymbol{q_v}}$ is split into two vectors $\widehat{\boldsymbol{q_v}}'$, $\widehat{\boldsymbol{q_v}}''$ by using the bit vector $\boldsymbol{s}$ by Eq. 5.

$$\begin{cases} \widehat{\boldsymbol{q_v}}'[\kappa] + \widehat{\boldsymbol{q_v}}''[\kappa] = \widehat{\boldsymbol{q_v}}[\kappa], & if \boldsymbol{s}[\kappa] = 0; \\ \widehat{\boldsymbol{q_v}}'[\kappa] = \widehat{\boldsymbol{q_v}}''[\kappa] = \widehat{\boldsymbol{q_v}}[\kappa], & if \boldsymbol{s}[\kappa] = 1. \end{cases} \tag{5}$$

- EASPE.Query(C, T_Q)$\rightarrow \boldsymbol{p}^T \cdot \boldsymbol{q}$: This algorithm calculates the inner product of C and T_Q by Eq. 6.

$$\begin{aligned} C^T \cdot T_Q &= ((\widehat{\boldsymbol{p_v}}')^T \boldsymbol{M}_1) \cdot (\boldsymbol{M}_1^{-1} \widehat{\boldsymbol{q_v}}') \\ &\quad + ((\widehat{\boldsymbol{p_v}}'')^T \boldsymbol{M}_2) \cdot (\boldsymbol{M}_2^{-1} \widehat{\boldsymbol{q_v}}'') \\ &= \boldsymbol{p_v}^T \cdot \boldsymbol{q_v} = \boldsymbol{p}^T \cdot \boldsymbol{q} \end{aligned} \tag{6}$$

Note that, compared with the ASPE scheme, EASPE scheme introduces three random numbers and a random permutation π for each ciphertext C_i and each trapdoor T_Q. Specifically, C_i and T_Q introduce random numbers $\{r_1, r_2, r_3\}$ and $\{r_4, r_5, r_6\}$ respectively. We will give a detailed security proof of the enhanced ASPE in Section VI, which is secure against IND-SCPA.

C. Naive Scheme: Spatial Keyword Query (SKQ)

We first propose a Spatial Keyword Query (SKQ) scheme by simply combining Geohash algorithm with the enhanced ASPE, which includes four phases of KeyGeneration, DataEncryption, TrapdoorGeneration and Query. The specific construction is as follows.

Let $W = \{w_1, w_2, \ldots, w_m\}$ be a total keyword set and $D = \{o_1, o_2, \ldots, o_n\}_{|1 \le i \le n}$ be a spatio-textual dataset. Each object o_i has a unique identity id_i and a tuple $(o_i.l, o_i.k)$, where $o_i.l = (x_i, y_i)$ represents spatial geographic location, $o_i.k = \{w_1, w_2, \ldots, w_g\} \subseteq W$ represents a set of keywords owned by the object o_i.

KeyGeneration. The data owner generates a encryption key $sk = \{\boldsymbol{s}, \boldsymbol{M}_1, \boldsymbol{M}_2, \pi, r_1, r_2, r_3, r_4, r_5, r_6\}$, where $\boldsymbol{s}$ is a random $(36+m)$-dimensional bit vector, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(36+m)\times(36+m)$ dimensional invertible matrices.

DataEncryption. The details are shown in Algorithm 1. For each object o_i in D, the data owner first encodes $o_i.l$ as an 8-character Geohash code, then converts the Geohash code into an order vector $\widehat{\boldsymbol{o_i.l}}$ according to the Base32, and finally generates eight 32-dimensional spatial vectors $\{\boldsymbol{o_i.lv_j}\}_{1 \le j \le 8}$. It is worth noticing that each spatial vector corresponds to a character of Geohash code, the corresponding position of this character on the base32 sequence is set as 1, and other positions are set as 0. The specific process of spatial vectors are shown in Fig. 4. Given a point $o.l$ = (8, 13), the data owner first encodes it as a Geohash code "$s33jxb67$", for each character, i.e., the order of 's' in Base32 is 24, '3' is 3. An order vector is generated as $(24, 3, 3, 17, 29, 10, 6, 7)$ [11]. Then, each element of the order vector is converted as a spatial vector, e.g., 24 is denoted as (0, ..., 0, 1, 0, ..., 0), where only the 24-th position is set as 1. Finally, $o_i.l$ is converted as 8 spatial vectors $\{\boldsymbol{o_i.lv_1}, \boldsymbol{o_i.lv_2}, \ldots, \boldsymbol{o_i.lv_8}\}$.

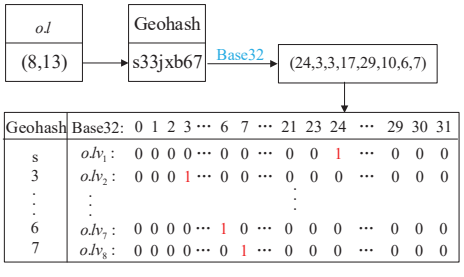

Fig. 4. An example of converting spatial location into vectors.

For each object o_i, $o_i.k$ is converted as an m-dimensional vector $\boldsymbol{o_i.kv}$ by Eq. 7,

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

where m is the size of the W, and if $w_r \in W$ is in $o_i.k$, $\lambda_r = 1$, otherwise, $\lambda_r = 0$. For example, let $W = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7\}$ and $o.k = \{w_1, w_2, w_4, w_5\}$, we obtain $o.kv = (1, 1, 0, 1, 1, 0, 0)$.

Then, the data owner generates eight $(33+m)$-dimensional data vectors $\boldsymbol{o_i.v_j}$ for each object o_i by Eq. 8,

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

Finally, D is encrypted as $Enc(D) = \{C_{i,j} | 1 \le i \le n, 1 \le j \le 8\}$ by EASPE.Enc and $Enc(D)$ is outsourced to the cloud server.

Algorithm 1: Data Encryption

Input: $D = \{o_1, o_2, \ldots, o_n\}$
Output: $Enc(D)$

Initialize a set $Enc(D) = \emptyset$;
for $1 \le i \le n$ **do**
  Initialize a set $Enc(o_i) = \emptyset$;
  $o_i.l$ is encoded an 8-character Geohash code and then converted as an 8-dimensional order vector $\widetilde{\boldsymbol{o_i.l}}$;
  $o_i.k$ is converted as $\boldsymbol{o_i.kv}$;
  for $1 \le j \le 8$ **do**
    Convert the j-th dimension of $\widetilde{\boldsymbol{o_i.l}}$ as $\boldsymbol{o_i.lv_j}$;
    $\boldsymbol{o_i.v_j} = (\boldsymbol{o_i.lv_j}, \boldsymbol{o_i.kv}, -1)$;
    $C_{i,j} = (\boldsymbol{M_1^T \widehat{o}_i.v'_j}, \boldsymbol{M_2^T \widehat{o}_i.v''_j})$;
    $Enc(o_i).add(C_{i,j})$;
  $Enc(D).add(Enc(o_i))$;
return $Enc(D)$.

TrapdoorGeneration. The details are shown in Algorithm 2. Given an query $Q = (Q.l, Q, k, R)$, where $Q.l = (q_x, q_y)$ represents spatial geographic location of query and $Q.k = \{w'_1, w'_2, \ldots, w'_k\} \subseteq W$ represents a set of query keywords, R represents the query range. The user encodes $Q.l$ as a Geohash code with a length of t characters according to the query range R, and then converts the Geohash code as an order vector $\widetilde{\boldsymbol{Q.l}}$ through the Base32 sequence. Since the number of characters in the Geohsah code can represent the query range, to protect the privacy of the query range, when $t < 8$, the user needs to set the remaining $8 - t$ dimensions to 0, and finally convert the 8-dimensional order vector into eight 32-dimensional query spatial vectors $\{\boldsymbol{Q.lv_j}\}_{1 \le j \le 8}$. The specific process of query spatial vectors are shown in Fig. 5. Given a point $Q.l = (8, 13)$ and $R = 600m$, $Q.l$ is first encoded as Geohash code "$s33jxb$" and then converted as an order vector $(24, 3, 3, 17, 29, 10, 0, 0)$ by the Base32 sequence. Finally, each element of the order vector is converted as a query spatial vector and $Q.l$ is converted as 8 query spatial vectors $\{\boldsymbol{Q.lv_1}, \boldsymbol{Q.lv_2}, ..., \boldsymbol{Q.lv_8}\}$.

Given $Q.k$, the user generates an m-dimensional vector $\boldsymbol{Q.kv}$ by Eq. 9,

$$\boldsymbol{Q.kv} = (\lambda'_1, \lambda'_2, \ldots, \lambda'_m), \tag{9}$$

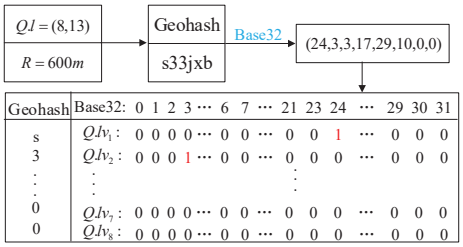

Fig. 5. An example of converting the query location into vectors.

where m is the size of the W, and if $w_r \in W$ is in $Q.k$, $\lambda'_r = 1$, otherwise, $\lambda'_r = 0$. For example, let $W = \{w_1, w_2, w_3, w_4, w_5, w_6, w_7\}$ and $Q.k = \{w_2, w_5\}$, we obtain $Q.kv = (0, 1, 0, 0, 1, 0, 0)$.

Then, the user generates eight $(33 + m)$-dimensional data vectors $\boldsymbol{Q.v_j}$ by Eq. 10,

$$\boldsymbol{Q.v_j} = \begin{cases} (\boldsymbol{Q.lv_j}, \boldsymbol{Q.kv}, 1 + k), & 1 \le j \le t \\ (\boldsymbol{Q.lv_j}, \boldsymbol{Q.kv}, k), & t < j \le 8, \end{cases} \tag{10}$$

where k is the number of query keywords.

Finally, Q is generated as trapdoors $\{T_{Q_j} | 1 \le j \le 8\}$ by EASPE.TrapGen and T_{Q_j} are outsourced to the cloud server.

Algorithm 2: Trapdoor Generation

Input: $Q = \{Q.l, Q.k, R\}$
Output: T_Q

Initialize a set $T_Q = \emptyset$;
$Q.l$ is encoded an t-character Geohash code and then converted as an 8-dimensional order vector $\widetilde{\boldsymbol{Q.l}}$;
$Q.k$ is converted as $\boldsymbol{Q.kv}$;
for $1 \le j \le 8$ **do**
  Convert the j-th dimension of $\widetilde{\boldsymbol{Q.l}}$ as $\boldsymbol{Q.lv_j}$;
  if $j \le t$ **then**
    $\boldsymbol{Q.v_j} = (\boldsymbol{Q.lv_j}, \boldsymbol{Q.kv}, 1 + k)$;
  else
    $\boldsymbol{Q.v_j} = (\boldsymbol{Q.lv_j}, \boldsymbol{Q.kv}, k)$;
  $T_{Q_j} = (\boldsymbol{M_1^{-1} \widehat{Q}.v'_j}, \boldsymbol{M_2^{-1} \widehat{Q}.v''_j})$;
  $T_Q.add(T_{Q_j})$;
return T_Q.

Query. After receiving the user's trapdoor, the cloud server returns the id of the object that meets the query requirements to the user. The details are shown in Algorithm 3.

Correctness. For each $o_i \in D$, $C_{i,j}^T \cdot T_{Q_j}$ is calculated as $C_{i,j}^T \cdot T_{Q_j} = \boldsymbol{o_i.v_j}^T \cdot \boldsymbol{Q.v_j}$ by EASPE.Query. For example, suppose $\boldsymbol{o_i.v_1}^T \cdot \boldsymbol{Q.v_1} = (\boldsymbol{o_i.lv_1}, \boldsymbol{o_i.kv}, -1) \cdot (\boldsymbol{Q.lv_1}, \boldsymbol{Q.kv}, 1 + k) = 0$, then $(\boldsymbol{o_i.lv_1} \cdot \boldsymbol{Q.lv_1}) - 1 = 0$ indicates that the first character of the Geohash code of the object o_i and query Q is the same, and $(\boldsymbol{o_i.kv} \cdot \boldsymbol{Q.kv}) - k = 0$ indicates that o_i contains all query keywords. Thus, $\{C_{i,j}^T \cdot T_{Q_j} = 0\}_{1 \le j \le 8}$ indicate that the object o_i meets the users query conditions, its identifier id_i is returned to the user.

Algorithm 3: Query

Input: $Enc(D) = \{C_{1,j}, C_{2,j}, \ldots, C_{n,j}\}_{1 \le i \le n, 1 \le j \le 8}$ and $\{T_{Q_j}\}_{1 \le j \le 8}$
Output: $Result$

Initialize a set $Result = \emptyset$;
for $1 \le i \le n$ **do**
  $\tau = 0$;
  for $1 \le j \le 8$ **do**
    Calculate $C_{i,j}^T \cdot T_{Q_j}$;
    if $C_{i,j}^T \cdot T_{Q_j} = 0$ **then**
      $\tau + 1$;
    else
      break;
  if $\tau = 8$ **then**
    $Result.add(id_i)$;
  else
    break;
return $Result$.

Remark 1: Since the naive solution needs to generate 8 encrypted indexes and 8 trapdoors for each object and user respectively, which incurs high storage and computational burdens. In the query phase, the cloud server needs to calculate the inner product of each encrypted index and each trapdoor, which causes the query time to be too long. Therefore, we need to improve the scheme to greatly improve its efficiency.

D. Enhanced Scheme: Lightweight Spatial Keyword Query (LSKQ)

The enhanced scheme also includes four phases of Key Generation, DataEncryption, TrapdoorGeneration and Query. Unlike SKQ, we convert an 8-dimensional order vector into a 40-dimensional spatial vector instead of eight 32-dimensional spatial vectors. The details are as follows.

KeyGeneration. The data owner generates a encryption key $sk = \{\boldsymbol{s}, \boldsymbol{M}_1, \boldsymbol{M}_2, \pi, r_1, r_2, r_3, r_4, r_5, r_6\}$, where $\boldsymbol{s}$ is a random $(44+m)$-dimensional bit vector, $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(44+m) \times (44+m)$ dimensional invertible matrices.

DataEncryption. The details are shown in Algorithm 4. For each object o_i in D, $o_i.l$ is converted as an order vector, and then each dimension of the order vector is converted as a 5-bit binary vector to generate a 40-dimensional spatial vector. The specific process of spatial vector is shown in Fig. 6. Given a point $o.l$ = (8, 13), it is first encoded into Geohash code "$s33jxb67$", then converted as an order vector $(24, 3, 3, 17, 29, 10, 6, 7)$ through Base32. Finally, the order vector is converted as a spatial vector $\boldsymbol{o_i.lv}$ = $(11000, 00011, 00011, 10001, 11101, 01010, 00110, 00111)$.

Then, the data owner generates a $(41+m)$-dimensional data vectors $\boldsymbol{o_i.v}$ for each object o_i by Eq. 11,

$$\boldsymbol{o_i.v} = (\boldsymbol{o_i.lv}, \boldsymbol{o_i.kv}, -1). \tag{11}$$

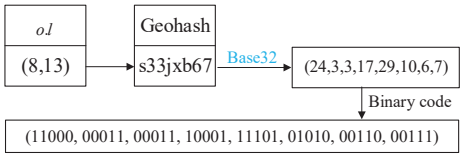

Fig. 6. An example of converting spatial location into a vector.

Finally, D is encrypted as $Enc(D) = \{C_i | 1 \le i \le n\}$ by EASPE.Enc and $Enc(D)$ is outsourced to the cloud server.

Algorithm 4: Data Encryption

Input: $D = \{o_1, o_2, \ldots, o_n\}$
Output: $Enc(D)$

Initialize a set $Enc(D) = \emptyset$;
for $1 \le i \le n$ **do**
  $o_i.l$ is encoded an 8-character Geohash code and then converted as an 8-dimensional order vector $\widetilde{\boldsymbol{o_i.l}}$;
  $\widetilde{\boldsymbol{o_i.l}}$ is converted as a 40-dimensional binary vector $\boldsymbol{o_i.lv}$;
  $o_i.k$ is converted as $\boldsymbol{o_i.kv}$;
  $\boldsymbol{o_i.v} = (\boldsymbol{o_i.lv}, \boldsymbol{o_i.kv}, -1)$;
  $\boldsymbol{o_i.v}$ is encrypted as $C_i = (\boldsymbol{M}_1^T \widehat{\boldsymbol{o}}_i.\boldsymbol{v'}, \boldsymbol{M}_2^T \widehat{\boldsymbol{o}}_i.\boldsymbol{v''})$;
  $Enc(D).add(C_i)$;
return $Enc(D)$.

TrapdoorGeneration. The details are shown in Algorithm 5. Given an query $Q = (Q.l, Q.k, R)$, $Q.l$ is converted as an 8-dimensional order vector and then each dimension of the order vector is converted as a 5-bit binary vector to generate a 40-dimensional query spatial vectors $\boldsymbol{Q.lv}$. The specific process of query spatial vectors are shown in Fig. 7. Given a point $Q.l = (8, 13)$ and $R = 600m$, $Q.l$ is first encoded as an order vector $(24, 3, 3, 17, 29, 10, 0, 0)$ by the Base32 sequence and then converted as a query spatial vector $\boldsymbol{Q.lv}$ = $(11000, 00011, 00011, 10001, 11101, 01010, 00000, 00000)$.

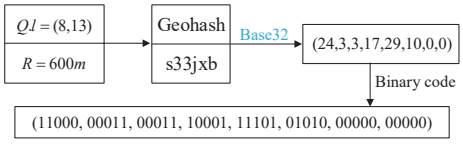

Fig. 7. An example of converting the query location into a vector.

Then, the user generates eight $(41+m)$-dimensional data vectors $\boldsymbol{Q.v}$ by Eq. 12,

$$\boldsymbol{Q.v} = (\boldsymbol{Q.lv}, \boldsymbol{Q.kv}, \theta + k), \tag{12}$$

where θ is the number of "1" in $\boldsymbol{Q.lv}$ and k is the number of query keywords.

Finally, the query Q is encrypted as a trapdoor T_Q by EASPE.TrapGen and T_Q is outsourced to the cloud server.

Algorithm 5: 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 8-dimensional order vector $\widetilde{\boldsymbol{Q.l}}$;
$\widetilde{\boldsymbol{Q.l}}$ is converted as a 40-dimensional binary vector $\boldsymbol{Q.lv}$;
$Q.k$ is converted as $\boldsymbol{Q.kv}$;
$\boldsymbol{Q.v} = (\boldsymbol{Q.lv_j}, \boldsymbol{Q.kv}, \theta + k)$;
$\boldsymbol{Q.v}$ is encrypted as $T_Q = (\boldsymbol{M_1}^{-1}\widehat{\boldsymbol{Q}}\boldsymbol{.v'}, \boldsymbol{M_2}^{-1}\widehat{\boldsymbol{Q}}\boldsymbol{.v''})$;
return T_Q.

Query. After receiving the user's trapdoor, the cloud server returns the id of the object that meets the query requirements to the user. The details are shown in Algorithm 6.

Algorithm 6: 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 $C_i^T \cdot T_Q$;
    if $C_i^T \cdot T_Q = 0$ **then**
        $Result.add(id_i)$;
return $Result$.

Correctness. For each $o_i \in D$, $C_i^T \cdot T_Q$ is calculated as $C_i^T \cdot T_Q = \boldsymbol{o_i.v}^T \cdot \boldsymbol{Q.v}$ by EASPE.Query. For example, if $\boldsymbol{o_i.v}^T \cdot \boldsymbol{Q.v} = (\boldsymbol{o_i.lv}, \boldsymbol{o_i.kv}, -1) \cdot (\boldsymbol{Q.lv}, \boldsymbol{Q.kv}, \theta + k) = 0$, because $(\boldsymbol{o_i.lv} \cdot \boldsymbol{Q.lv}) - \theta = 0$ indicates that o_i and Q have t same Geohash prefixes, i.e., o_i is within the query range, $(\boldsymbol{o_i.kv} \cdot \boldsymbol{Q.kv}) - k = 0$ indicates that o_i contains all query keywords, o_i meets query requirements and its identity identifier id_i is returned to the user.

Remark 2: By converting the order vector as a binary vector, each object only needs to generate an encrypted index and the user only needs to generate a trapdoor, which greatly reduces the calculation and storage overhead of the SKQ.

VI. Security Analysis

In this section, we analyze the security of our two schemes under the threat models defined in Section IV. There are three main operations: DataEncryption, TrapdoorGeneration and Query. The results show that our schemes make the cloud server unable to learn any critical information except encrypted objects and trapdoors.

A. Data Encryption

As described in the threat model defined in Section IV, the adversary who conducts chosen-plaintext attack has more information than the adversary launching ciphertext-only attack, so the adversary launching chosen-plaintext attack is more powerful. In other words, if our two schemes can safely resist chosen-plaintext attack, they can also safely resist ciphertext-only attack. 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 LSKQ as a representative for detailed proof.

Under the chosen-plaintext attack, the cloud server can obtain not only the ciphertext but also the corresponding plaintext. For example, this can happen when some adversaries submit their data and collude with the cloud server. 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, $i \in \{0, 1\}$, $j = 1, 2, \ldots, n$.
- The challenger $\mathcal{C}$ runs KeyGeneration to generate the secret key.
- Phase 1: $\mathcal{A}$ submits $d_{i,j}$ to $\mathcal{C}$, $i \in \{0, 1\}$, $j = 1, 2, \ldots, n$. 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 $C_{b,j}$ is returned 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: *LSKQ guarantees the IND-CPA data privacy if for any polynomial time adversary $\mathcal{A}$, it has at most a negligible advantage $negl(\lambda)$, such that*

$$Adv^{IND-CPA}_{PESKQ,\mathcal{A}}(1^\lambda) = Pr(b' = b) - \frac{1}{2}| \le negl(\lambda).$$

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

Theorem 1: *LSKQ guarantees the IND-CPA data privacy.*

Proof: According to the above analysis, we should prove that $\mathcal{A}$ cannot distinguish $C_{0,j}$ and $C_{1,j}$, even if the adversary $\mathcal{A}$ has oracle access to DataEncryption. Assume that one of the messages in D_0 is $d_{i,j} = (d.l, d.k)$. According to the process of DataEncryption, $d_{i,j}$ is extended to a $(44 + m)$-dimensional vector $\boldsymbol{d.v} = (\boldsymbol{d.lv}, \boldsymbol{d.kv}, -1, r_1, r_2, r_3)$, where r_1, r_2, r_3 are random numbers. Then, $\boldsymbol{d.v}$ is permuted as $\widehat{\boldsymbol{d}}\boldsymbol{.v} = \pi(\boldsymbol{d.v})$ by the random permutation π and encrypted to $C_0 = (\boldsymbol{M}_1^T\widehat{\boldsymbol{d}}\boldsymbol{.v'}, \boldsymbol{M}_2^T\widehat{\boldsymbol{d}}\boldsymbol{.v''})$ 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, and the split vector $\boldsymbol{s}$, he cannot recover the secret key $\boldsymbol{M}_1$, $\boldsymbol{M}_2$ with the ciphertexts C_0.

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 $\widehat{\boldsymbol{d}}\boldsymbol{.v}$ is a one-time random vector determined by $\mathcal{C}$, r_1, r_2, r_3 are random numbers and π is a random permutation, 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

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

■

Thus, according to the above analysis, the LSKQ scheme is secure under ciphertext-only attack and chosen-plaintext attack.

B. Trapdoor Generation

Theorem 2: *LSKQ scheme guarantees the IND-CPA query content privacy.*

Proof: The operations involved in DataEncryption and TrapdoorGeneration are almost the same, so the security definition and analysis of DataEncryption is applicable to TrapdoorGeneration. We skip further details due to space limitations.

■

C. Query

The cloud server has all ciphertexts $C_i = (\boldsymbol{M}_1^T\widehat{\boldsymbol{o_i}}.\boldsymbol{v'}, \boldsymbol{M}_2^T\widehat{\boldsymbol{o_i}}.\boldsymbol{v''})$ 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 $\boldsymbol{o_i}.\boldsymbol{v}$ and $\boldsymbol{Q}.\boldsymbol{v}$, thus, it cannot know the original dataset and the user's query information. Then, the cloud server calculates $C_i^T \cdot T_Q = \boldsymbol{o_i}.\boldsymbol{v}^T \cdot \boldsymbol{Q}.\boldsymbol{v} = \widehat{\boldsymbol{o_i}}.\boldsymbol{v}^T \cdot \widehat{\boldsymbol{Q}}.\boldsymbol{v} = (\boldsymbol{o_i}.\boldsymbol{lv}, \boldsymbol{o_i}.\boldsymbol{kv}, -1, r_1, r_2, r_3) \cdot (\boldsymbol{Q}.\boldsymbol{lv}, \boldsymbol{Q}.\boldsymbol{kv}, \theta + k, r_4, r_5, r_6) = 0$ can get whether $C_{i,j}^T \cdot T_{Q_j} = 0$. Because $r_1, r_2, r_3, r_4, r_5, r_6$ are unknown and they are all random numbers, so even if the cloud server can judge whether $C_{i,j}^T \cdot T_{Q_j} = 0$, it does not know whether $(\boldsymbol{o_i}.\boldsymbol{lv} \cdot \boldsymbol{Q}.\boldsymbol{lv}) - \theta = 0$ and $(\boldsymbol{o_i}.\boldsymbol{kv} \cdot \boldsymbol{Q}.\boldsymbol{kv}) - k = 0$. 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 Python3.9 programming language on 64-bit Windows 10 system and completed on Intel(R) Core(TM) i5-8300h CPU @2.30GHz server.

A. Theoretical Analysis

We analyzed the theoretical complexity of the SKQ, LSKQ and Privacy-Preserving Keyword Similarity Search scheme (GRQ+MSSAC) [29] in terms of computational overheads, storage overheads and communication overheads of Data Encryption, TrapdoorGeneration and Query. TABLE IV gives the detailed comparison results of the there schemes.

SKQ uses ASPE to encrypt data, which encryption key is $sk = \{\boldsymbol{M}_1, \boldsymbol{M}_2\}$, where $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(36+m) \times (36+m)$ dimensional invertible matrices. Without losing generality, we assume that the size of each ciphertext is $|U|$. In DataEncryption phase, the data owner needs to encrypt all objects in the dataset, where each object contains 8 ciphertexts. Therefore, the total time complexity of the process is $O(16n(36+m)^2)$, and the storage overhead is $8n|U|$. In TrapdoorGeneration phase, the main overhead is taken by encrypting the query content once, so the total time complexity of the process is $O(16(36+m)^2)$, and the storage cost is $8|U|$. In Query phase, the cloud server needs to calculate the product of each encrypted object and the trapdoor, where each object contains only 8 ciphertexts, so the total time complexity of the process is $O(16n(36+m))$.

LSKQ is similar to the SKQ, but the LSKQ only needs to generate a ciphertext for each object and query user, which encryption key is $sk = \{\boldsymbol{M}_1, \boldsymbol{M}_2\}$, where $\boldsymbol{M}_1$ and $\boldsymbol{M}_2$ are two random $(44+m) \times (44+m)$ dimensional 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 $O(2n(44+m)^2)$, and the storage overhead is $n|V|$. In TrapdoorGeneration phase, the total time complexity of the process is $O(2(44+m)^2))$, and the storage cost is $|V|$. In Query phase, the total time complexity of the process is $O(2n(44+m))$.

GRQ+MSSAC first proposes a Geometric Range Query (GRQ) scheme, which uses ASPE to encrypt the spatial location to find objects located in the query region. Then, based on GRQ, a Multi-dimensional Spatial keyword Similarity search scheme with role Access Control (MSSAC) is proposed by integrating the polynomial function and randomizable matrix multiplication, which finds the object with the highest keyword similarity. This scheme does not support search for spatial location and keywords at the same time. Without loss of generality, we assume that the degree of polynomial fit is η, the number of roles is s, the size of each ciphertext is $|X|$ for GRQ, and the size of each ciphertext is $|Y|$ for MSSAC. In DataEncryption phase, the data owner needs to encrypt the spatial geographic coordinates of the object for GRQ, the keywords and roles of the object for MSSAC. Thus, the total time complexity of the process is $O(n(2(\eta+2)^2 + 3(m+3+2s)^3))$, and the storage cost is $t(|X| + |Y|)$. In TrapdoorGeneration phase, the query content needs to be encrypted, so the total time complexity of the process is $O(4(\eta+2)^2 + 3(m+3+2s)^3)$, and the storage overhead is $2|X|+|Y|$. In Query phase, the total time complexity of the process is $O(t(4(\eta+2)+(m+3+2s)^3))$.

B. Performance Evaluation

We use the real dataset Yelp academic dataset business [1] to perform performance testing and specific analysis on the Data Encryption, TrapdoorGeneration and Query processes of the SKQ, LSKQ and GRQ+MSSAC [29].

DataEncryption. For SKQ and LSKQ, the factors that affect the computational costs are the size 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 W (i.e. m). Therefore, (1) let m = 200 and s

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

TABLE III
COMPARISON OF THEORETICAL ANALYSIS.

Overhead	Phase	SKQ	LSKQ	GRQ+MSSAC [29]
Computation overhead	DataEncryption	$O(16n(36+m)^2)$	$O(2n(44+m)^2)$	$O(n(2(\eta+2)^2+3(m+3+2s)^3))$
	TrapdoorGeneration	$O(16(36+m)^2)$	$O(2(44+m)^2)$	$O(4(\eta+2)^2+3(m+3+2s)^3)$
	Query	$O(16n(36+m))$	$O(2n(44+m))$	$O(n(4(\eta+2)+(m+3+2s)^3))$
Storage overhead	DataEncryption	$8n\|U\|$	$n\|V\|$	$n(\|X\|+\|Y\|)$
	TrapdoorGeneration	$8\|U\|$	$\|V\|$	$2\|X\|+\|Y\|$
	Query	$--$	$--$	$--$

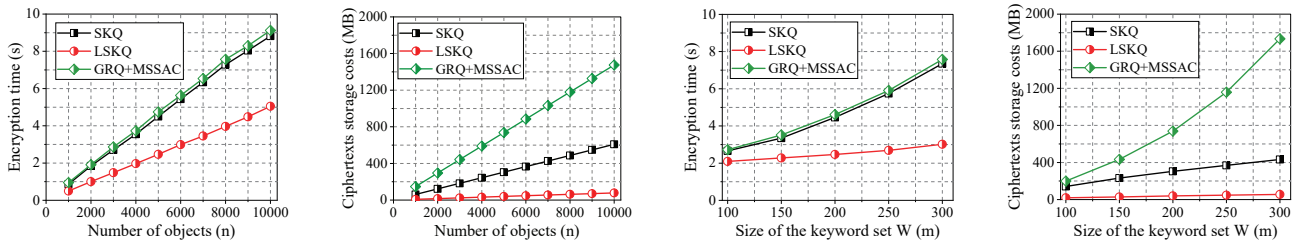

Fig. 8. Overhead of DataEncryption. (a) Computational overhead varying with n; (b) Storage overhead varying with n; (c) Computational overhead varying with m; (d) Storage overhead varying with m.

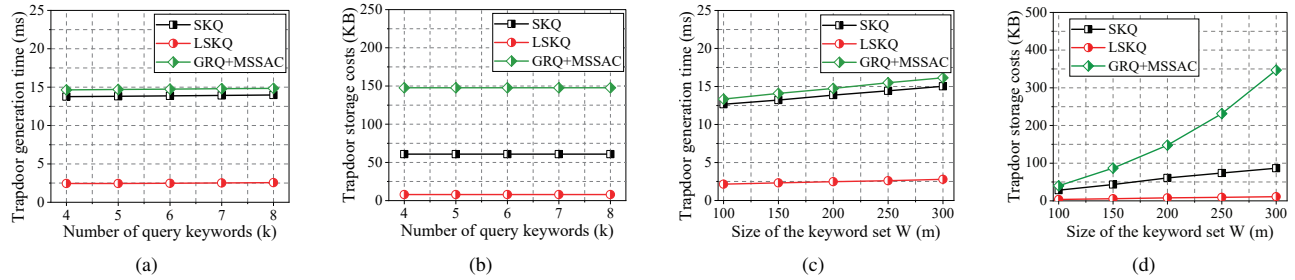

Fig. 9. Overhead of TrapdoorGeneration. (a) Computational overhead varying with k; (b) Storage overhead varying with k; (c) Computational overhead varying with m; (d) Storage overhead varying with m.

= 15, we test the computational and storage costs of the three schemes by varying n from 1000 to 10000. The test results are shown in Fig. 8 (a) (b); (2) let n = 5000 and s = 15, we test the computational costs and storage costs of the three schemes by varying m from 100 to 300. The test results are shown in Fig. 8 (c) (d).

TrapdoorGeneration. This process is similar to the data encryption process, which is one-time encryption of the user's query content. Therefore, (1) let m = 200 and s = 15, we test the computational and storage costs of the three schemes by changing the number of query keywords (i.e. k). The test results are shown in Fig. 9 (a) (b). (2) let k = 5 and s = 15, we test the computational costs and storage costs of the three schemes by varying m from 100 to 300. The test results are shown in Fig. 9 (c) (d).

Query. At this phase, the cloud server needs to calculate the product of each encrypted data object and the trapdoor. Therefore, the factors that affect the computational costs are the size of the dataset n and the size of the vector. (1) Let m = 200 and s = 15, we test the computational costs of the three schemes by varying n from 1000 to 10000. The test results are shown in Fig. 10 (a). (2) Let n = 5000 and s = 15, we

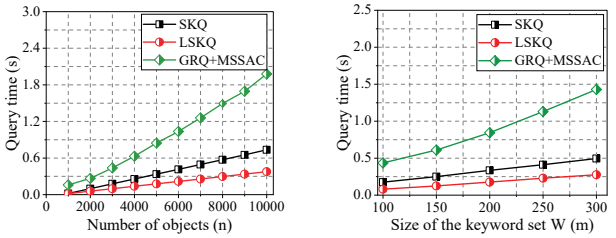

Fig. 10. Overhead of Query. (a) Computational overhead varying with n; (b) Computational overhead varying with m.

test the computational cost of the three schemes by varying m from 100 to 300. The test results are shown in Fig. 10 (b).

It can be seen from the above theoretical analysis and extensive specific experimental results that the LSKQ scheme is very efficient.

VIII. CONCLUSION

In this paper, we enhance the security of ASPE by adding some random numbers and a random permutation, and use

enhanced ASPE and Geohash algorithm to propose a naive privacy-preserving spatial keyword query scheme to achieve approximate spatial range query, which only requires users to provide less information about the query range, and then build a unified index for spatial range and multiple keywords to propose a lightweight spatial keyword query scheme, which greatly decreases storage and computational costs and is more in line with the actual query habits. As part of our future work, we will attempt to design a tree index to further improve search efficiency.

ACKNOWLEDGMENTS

This work was supported by the National Natural Science Foundation of China (No. 62072361, No.62125205), Key Research and Development Program of Shaanxi (No. 2022GY-019), the Fundamental Research Funds for the Central Universities (No. JB211505), Henan Key Laboratory of Network Cryptography Technology & State Key Laboratory of Mathematical Engineering and Advanced Computing (No. LNCT2020-A06), National Natural Science Foundation of China (No.U21A20464, No. 61872283, No.61802243), the Natural Science Basic Research Program of Shaanxi(No.2021JC-22) and the China 111Project (No.B16037). The work of Kim-Kwang Raymond Choo was supported only by the Cloud Technology Endowed Professorship.

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 ACM Transactions on Networking*, vol. 24, no. 4, pp. 2305−2318, 2016.

[2] G. Xiao, F. Wu, X. Zhou and K. Li, "Probabilistic top-k range query processing for uncertain datasets," *Journal of Intelligent and Fuzzy Systems*, 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 Transactions on Information Forensics and Security*, vol. 12, no. 7, pp. 1596−1608, 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 Transactions on Mobile Computing*, 2021, 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 Transactions on Vehicular Technology*, 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 Transactions on Information Forensics and Security*, 2021, doi: 10.1109/TIFS.2021.3065832.

[7] Y. Zheng, R. Lu, Y. Guan, J. Shao and H. Zhu, "Achieving Efficient and Privacy-Preserving Exact Set Similarity Search over Encrypted Data," *IEEE Transactions on Dependable and Secure Computing*, 2020, doi: 10.1109/TDSC.2020.3004442.

[8] Q. Tong, Y. Miao, H. Li, X. Liu and R. H. Deng, "Privacy-Preserving Ranked Spatial Keyword Query in Mobile Cloud-Assisted Fog Computing," *IEEE Transactions on Mobile Computing*, 2021, doi: 10.1109/TMC.2021.3134711.

[9] W. K. Wong, D. W. Cheung, B. Kao and N. Mamoulis, "Secure kNN computation on encrypted databases," *in Proc. International Conference on Management of data (COMAD'09)*, pp. 139−152, 2009.

[10] W. Lin, K. Wang, Z. Zhang and H. Chen, "Revisiting security risks of asymmetric scalar product preserving encryption and its variants," *in Proc. IEEE International Conference on Distributed Computing Systems (ICDCS'17)*, pp. 1116−1125, 2017.

[11] R. Guo, Y. Wu, R. Liu and H. Chen, "LuxGeo: Efficient and Secure Enhanced Geometric Range Queries," *IEEE Transactions on Knowledge and Data Engineering*, 2021, doi: 10.1109/TKDE.2021.3093909.

[12] Q. Huang, J. Du, G. Yan, Y. Yang and Q. Wei, " Privacy-Preserving Spatio-Temporal Keyword Search for Outsourced Location-Based Services," *IEEE Transactions on Services Computing*, 2021, doi: 10.1109/TSC.2021.3088131.

[13] N. Davis, G. Raina and K. Jagannathan, "Taxi Demand Forecasting: A HEDGE-Based Tessellation Strategy for Improved Accuracy," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 11, pp. 3686−3697, 2018.

[14] R. Gelda, K. Jagannathan, and G. Raina, "Forecasting supply in Voronoi regions for app-based taxi hailing services," *in Proc. International Conference on Advanced Logistics and Transport (ICALT'17)*, pp. 1−6, 2017.

[15] S. Choi, G. Ghinita, H. Lim and E. Bertino, "Secure kNN query processing in untrusted cloud environments," *IEEE Transactions on Knowledge and Data Engineering*, vol. 26, no. 11, pp. 2818-2831, 2014.

[16] P. Wang and C. V. Ravishankar, "Secure and Efficient Range Queries on Outsourced Databases Using $\hat{R}$-trees," *in Proc. IEEE International Conference on Data Engineering (ICDE'13)*, pp. 314−325, 2013.

[17] H. Zhu, R. Lu, C. Huang, L. Chen and H. Li, "An efficient privacy-preserving location-based services query scheme in outsourced cloud," *IEEE Transactions on Vehicular Technology*, vol. 65, no. 9, pp. 7729-7739, 2016.

[18] 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 Transactions on Information Forensics and Security*, vol. 14, no. 4, pp. 870−885, 2018.

[19] C. Wang, N. Cao, J. Li and K. Ren, "Secure ranked keyword search over encrypted cloud data," *in Proc. IEEE International Conference on Distributed Computing Systems (ICDCS'10)*, pp. 253−262, 2010.

[20] C. Wang, K. Ren, S. Yu and K. M. R. Urs, "Achieving usable and privacy-assured similarity search over outsourced cloud data," *in Proc. IEEE International Conference on Computer Communications (INFOCOM'12)*, pp. 451−459, 2012.

[21] W. Sun, B. Wang, N. Cao, M. Li, W. Lou, Y. T. Hou and H. liu, "Privacy-preserving multi-keyword text search in the cloud supporting similarity-based ranking," *in Proc. ACM Symposium on Information, Computer and Communications Security (ASIACCS'13)*, pp. 71−82, 2013.

[22] Y. Zheng, R. Lu, Y. Guan, J. Shao and H. Zhu, "Achieving Efficient and Privacy-Preserving Set Containment Search over Encrypted Data," *IEEE Transactions on Services Computing*, 2021, doi: 10.1109/TSC.2021.3065240.

[23] 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 Transactions on Services Computing*, vol. 11, no. 5, pp. 796−809, 2018.

[24] X. Meng, H. Zhu and G. Kollios, "Top-k Query Processing on Encrypted Databases with Strong Security Guarantees," *in Proc. IEEE International Conference on Data Engineering (ICDE'18)*, pp. 353−364, 2018.

[25] D. Negi, S. Ray and R. Lu, "Pystin: Enabling Secure LBS in Smart Cities With Privacy-Preserving Top-k SpatialCTextual Query," *IEEE Internet of Things Journal*, vol. 6, no. 5, pp. 1−1, 2019.

[26] 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. International Conference on Data Engineering (ICDE'19)*, pp. 1046−1057, 2019.

[27] X. Wang, J. Ma, X. Liu, R. H. Deng, Y. Miao, D. Zhu and Z. Ma, "Search me in the dark: Privacy-preserving boolean range query over encrypted spatial data," *in Proc. IEEE International Conference on Computer Communications (INFOCOM'20)*, pp. 2253−2262, 2020.

[28] S. Lai, S. Patranabis, A. Sakzad, J. K. Liu, and D. Mukhopadhyay, Result pattern hiding searchable encryption for conjunctive queries, *in Proc. of ACM Conference on Computer and Communications Security (CCS18)*, pp. 745−762, 2018.

[29] F. Song, Z. Qin, L. Xue, J. Zhang, X. Lin and X. Shen, "Privacy-Preserving Keyword Similarity Search over Encrypted Spatial Data in Cloud Computing," *IEEE Internet of Things Journal*, 2021, doi: 10.1109/JIOT.2021.3110300.