# Efficient Privacy-Preserving Spatial Range Query Over Outsourced Encrypted Data

Yinbin Miao, Yutao Yang, Xinghua Li, Zhiquan Liu, *Member, IEEE*, Hongwei Li, Kim-Kwang Raymond Choo, *Senior Member, IEEE*, and Robert H. Deng, *Fellow, IEEE*

***Abstract*— With the rapid development of Location-Based Services (LBS), a large number of LBS providers outsource spatial data to cloud servers to reduce their high computational and storage burdens, but meanwhile incur some security issues such as location privacy leakage. Thus, extensive privacy-preserving LBS schemes have been proposed. However, the existing solutions using Bloom filter do not take into account the redundant bits that do not map information in Bloom filter, resulting in high computational overheads, and reveal the inclusion relationship in Bloom filter. To solve these issues, we propose an efficient Privacy-preserving Spatial Range Query (PSRQ) scheme by skillfully combining Geohash algorithm with Circular Shift and Coalesce Bloom Filter (CSC-BF) framework and Symmetric-key Hidden Vector Encryption (SHVE), which not only greatly reduces the computational cost of generating token but also speeds up the query efficiency on large-scale datasets. In addition, we design a Confused Bloom Filter (CBF) to confuse the inclusion relationship by confusing the values of 0 and 1 in the Bloom filter. Base on this, we further propose a more secure and practical enhanced scheme PSRQ$^+$ by using CBF and Geohash algorithm, which can support more query ranges and achieve adaptive security. Finally, formal security analysis proves that our schemes are secure against Indistinguishability under Chosen-Plaintext Attacks (IND-CPA) and PSRQ$^+$ achieves adaptive IND-CPA, and extensive experimental tests demonstrate that our schemes using million-level dataset improve the query efficiency by 100$\times$ compared with previous state-of-the-art solutions.**

***Index Terms*— Location-based services, location privacy leakage, privacy-preserving, spatial range query.**

Manuscript received 19 January 2023; revised 26 May 2023; accepted 16 June 2023. Date of publication 21 June 2023; date of current version 3 July 2023. This work was supported in part by the National Natural Science Foundation of China under Grant 62072361, Grant 62125205, and Grant 62272195; in part by the Key Research and Development Program of Shaanxi under Grant 2022GY-019; in part by the Shaanxi Fundamental Science Research Project for Mathematics and Physics under Grant 22JSY019; in part by the Fundamental Research Funds for the Central Universities under Grant QTZX23091; in part by the 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. The work of Kim-Kwang Raymond Choo was supported by the Cloud Technology Endowed Professorship. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Mark Manulis. *(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).

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

Hongwei Li is with the School of Computer Science and Engineering, University of Electronic Science and Technology of China, Chengdu 610051, China (e-mail: hongweili@uestc.edu.cn).

Kim-Kwang Raymond Choo is with the Department of Information Systems and Cyber Security, The University of Texas at San Antonio, San Antonio, TX 78249 USA (e-mail: raymond.choo@fulbrightmail.org).

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/TIFS.2023.3288453

## I. Introduction

With the explosive growth of location-aware mobile devices, Location-Based Services (LBS) are becoming more and more popular in social and business fields [1]. LBS has been widely used in various practical applications such as navigation systems, leisure travel, social network, taxi hailing and personalized recommendation. With a geographical location, LBS can provide the most relevant Points of Interest (PoIs) to users [2]. For example, LBS can help users find nearby scenic spots, hospitals, restaurants, etc. Nowadays, the computational and storage overheads of LBS providers are exploding. It is more preferable for LBS providers (i.e., Google Map, Facebook, etc.) to outsource the massive amounts of spatial data to cloud servers. However, outsourcing data to cloud servers seriously threatens users' privacy as cloud servers can infer the user's daily itinerary and preference by using location information [3]. Thus, the secure spatial range query has been extensively studied. However, there are still two issues to be solved.

The first issue is that the traditional Bloom filter will reveal the inclusion relationship between 0 and 1 in Bloom filter. As shown in Fig. 1, "1" in Bloom filter represents inclusion and "0" represents exclusion. Then, the attacker can speculate whether a set contains an element according to the position of "1" in Bloom filter. Although Twin Bloom Filter (TBF) [4], [5] can hide inclusion relationship by confusing the values of 0 and 1 in Bloom filter, TBF needs to construct two rows of Bloom filters for a set, resulting in huge computational and storage overheads. It is worth noticing that the security of the existing schemes [23], [24] using Bloom filter depends on the security of the encryption algorithm. Although the Bloom filter can provide space efficiency to some extent, the unencrypted Bloom filter still enables attackers to deduce some plaintext data mapped into it, which cannot resist common attacks such as the chosen-plaintext attacks. Thus, it is necessary to encrypt the Bloom filter to further protect its privacy.

The second issue is that the existing privacy-preserving spatial range query schemes using Bloom filter do not take

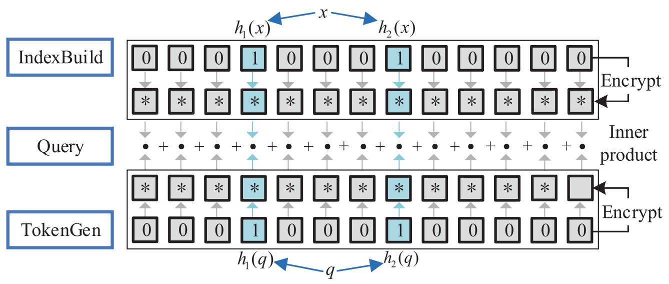


Fig. 1. Existing issues in previous solutions.

into account the redundant bits that do not map information in the Bloom filter (i.e., gray block in Fig. 1), which leads to huge computational and storage overheads. In token generation phase, the above schemes require users to map query information to a fixed-length Bloom filter. Although some privacy-preserving techniques such as Asymmetric Scalar-Product-Preserving Encryption (ASPE) [6] or Randomizable Matrix Multiplication (RMM) technique [29] can be used to encrypt Bloom filter, they still make the computation complexity of token generation linearly increase with the size of Bloom filter as all bits of Bloom filter need to be encrypted [23], [24]. Furthermore, these encryption mechanisms require the cloud server to compute the inner product between encrypted index and token bit by bit, which leads to the linear complexity of ciphertext retrieval [7], [11], [20], [23]. Especially, to reduce the false positive on the large-scale dataset, the length of Bloom filter will increase with the amount of data mapped into it, which in turn results in increased redundant bits. Therefore, the time of token generation and query will be too large under large-scale datasets, which makes existing schemes unsuitable for real-time scenarios. There are other range query schemes that usually use Gray code [8] or prefix code [9] to generate indexes and tokens, but lead to longer indexes and tokens, even incur huge computational overhead as all bits of indexes and tokens need to be calculated in token generation and query.

To solve the above issues, we first propose an efficient Privacy-Preserving Spatial Range Query (PSRQ) scheme by skillfully combining Geohash algorithm with Circular Shift and Coalesce Bloom Filter (CSC-BF) framework [33] and Symmetric-key Hidden Vector Encryption (SHVE) [34]. Specifically, PSRQ uses SHVE to encrypt the valid bits of Geohash code mapped to CSC-BF to generate a shorter token for users, and matches each bit of shorter token via CSC-BF during the query, which makes the computational costs of token generation and ciphertext retrieval independent of the length of Bloom filter. Then, we first design a Confused Bloom Filter (CBF) to confuse the inclusion relationship by confusing the values of 0 and 1 in the Bloom filter. Based on this, we further design a more secure and practical enhanced scheme PSRQ$^+$ by using CBF and converting the spatial location as a set containing 8 Geohash codes, which supports more query ranges and achieves adaptive security. The main contributions of our work are shown as follows:

1) We first propose an efficient Privacy-Preserving Spatial Range Query (PSRQ) scheme by skillfully combining the Geohash algorithm with CSC-BF and SHVE, which greatly reduces the computational cost of generating token by reducing the length of token by using Geohash algorithm and SHVE, and speeds up the query efficiency on large-scale datasets via CSC-BF.
2) We design a novel Confused Bloom Filter (CBF) structure, which can confuse the inclusion relationship by confusing the values of 0 and 1 in the traditional Bloom filter without incurring high storage overheads.
3) We propose a more secure and practical enhanced scheme PSRQ$^+$ by using CBF and converting the spatial location as a set containing 8 Geohash codes, which can support more query ranges and achieve adaptive security.
4) We give formal security analysis to prove that our schemes are secure against Indistinguishability under Chosen-Plaintext Attacks (IND-CPA) and PSRQ$^+$ achieves adaptive IND-CPA, and conduct extensive experiments to demonstrate that our schemes using million-level dataset improve the query efficiency by 100x compared with previous state-of-the-art solutions.

The rest of this paper is organized as follows. In Section II, we introduce some state-of-the-art related work. In Section III, we introduce some main preliminaries for our schemes. In Section IV, we define the system model, threat model, problem definition and design goals. In Section V, we demonstrate the detail of our schemes. In Section VI, we prove the security of our schemes. In Section VII, we conduct extensive experimental tests to analyze and evaluate the performance of our schemes. In Section VIII, we summarize our work.

## II. Related Work

Spatial range query can be divided into rectangular range query, circular range query and geometric range query according to the shape of query area.

### A. Privacy-Preserving Rectangular Range Query

Boneh and Waters [10] designed a Hidden Vector Encryption (HVE) and proposed a public-key scheme to determine whether a point was within a hyper-rectangle under ciphertext environment. To improve the query efficiency of [10], Wang et al. [11] proposed a hierarchical encrypted index $\hat{R}$-tree by combing ASPE, which can perform range query securely and efficiently over encrypted dataset. In addition, Wang et al. [12] used Order-Revealing Encryption (ORE) [13] to determine the size relationship between the coordinate of spatial point and those of the lower left and upper right corners of query rectangle, which implements secure range query. However, the above schemes only achieve single-dimensional spatial range query. To this end, Shi et al. [14] proposed a multi-dimensional spatial range query scheme, which supports secure rectangular range query based on the identity encryption. In addition, Wang et al. [15] designed a tree-based multi-dimensional range query algorithm by combing $R$-trees and HVE to support efficient rectangular range queries.

### B. Privacy-Preserving Circular Range Query

Wang et al. [16] leveraged Shen-Shi-Waters (SSW) [17] to build a symmetric-key circle predicate encryption scheme, which verifies whether the point was within the circle range by using multiple concentric circles. But the spatial data in [16] can only be integers. To this end, Zhu et al. [18] proposed a secure circle range query scheme based on improved homomorphic encryption over composite order group, which uses Boneh-Goh-Nissim (BGN) homomorphic encryption [19] to calculate the distance and evaluates the resulting range via a hash table. To reduce the computational overhead and improve the query efficiency of [18], Zheng et al. [20] designed a privacy-preserving circular range query scheme by using ASPE, Order-Preserving Encryption (OPE) [21] and $R$-tree. In addition, Li et al. [22] proposed an Inner Product Range Encryption (IPRE) to judge whether a spatial data was within the given circular region without revealing vectors.

### C. Privacy-Preserving Geometric Range Query

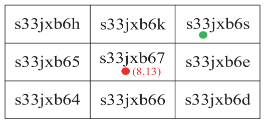
Geometric range query includes but is not limited to rectangles and circles. Wang et al. [23] enumerated all integer points within query range and mapped them into a Bloom filter, and then determined if the point was within query range by computing the inner product of encrypted Bloom filters. To reduce the number of integers inserted into Bloom filter, Cui et al. [24] provided a linear transformation strategy to transform the geographic coordinates as some small integers, which deals with the issues of dimension expansion and space reduction. To improve query efficiency of [24], Luo et al. [25] used circumscribed polygons to replace various types of geometries, transformed all shapes of the query into a unified form, and used ASPE to protect the privacy of data. Afterwards, Li et al. [26] pointed out that [25] could not achieve declared security, so they improved [25] to make it secure against known-background attacks. In addition, Wang et al. [27] used Quadtree to construct index for spatial dataset, and mapped spatial location and query range as Gray code, then used SHVE to encrypt and obtain range query results under ciphertext. To reduce the index length of [27], Yang et al. [28] mapped spatial data and query range as Geohash code, and converted them into indexes based on Base32. Then, they used Enhanced ASPE algorithm (EASPE) for encryption and determined whether spatial data was within query range by calculating the inner product of encrypted index. However, the search shapes that the above schemes can support are still limited. To further support arbitrary geometric range query, Zhang et al. [29] proposed a geometric range query scheme based on a two-server model, which employs polynomial fitting technique to achieve accurate arbitrary range queries and uses RMM to protect data privacy.

However, the above solutions do not consider the redundant bits in index, resulting in huge computational overhead and low query efficiency. Therefore, we first propose a PSRQ scheme by using the Geohash algorithm with CSC-BF and SHVE, which not only greatly reduces the computational cost of generating token but also speeds up the query efficiency on the large-scale datasets. Then, we propose a more practical scheme PSRQ$^+$, which can support more query ranges and achieve adaptive security. The detailed comparison between the previous schemes and our schemes is shown in TABLE I.

TABLE I
COMPARISON BETWEEN OUR SCHEMES AND PREVIOUS SCHEMES

| Scheme | Index structure | Encryption method | Resist attack type | TokenGen complexity | Query complexity |
|---|---|---|---|---|---|
| [11] | $\hat{R}$-tree | ASPE | KPA | $\mathcal{O}(d^2)$ | $\mathcal{O}(d\log n)$ |
| [16] | Linear | SSW | IND-CPA | $\mathcal{O}(d)$ | $\mathcal{O}(dn)$ |
| [20] | $R$-tree | ASPE/OPE | CPA | $\mathcal{O}(d^2)$ | $\mathcal{O}(d\log n)$ |
| [24] | Linear | ASPE | KBA | $\mathcal{O}(d^2)$ | $\mathcal{O}(dn)$ |
| [26] | $R$-tree | ASPE | KBA | $\mathcal{O}(d^2)$ | $\mathcal{O}(d\log n)$ |
| [27] | Quadtree | SHVE | IND-CPA | $\mathcal{O}(d)$ | $\mathcal{O}(d\log n)$ |
| [28] | Linear | EASPE | IND-CPA | $\mathcal{O}(d^2)$ | $\mathcal{O}(d\log n)$ |
| [29] | Linear | RMM | COA/CPA | $\mathcal{O}(d^3)$ | $\mathcal{O}(nd^3)$ |
| PSRQ | Sublinear | SHVE | IND-CPA | $\mathcal{O}(kb)$ | $\mathcal{O}(kbr)$ |
| PSRQ$^+$ | Sublinear | SHVE | Adaptive security | $\mathcal{O}(kb)$ | $\mathcal{O}(kbr)$ |

**Notes:** KPA: Known-Plaintext Attack; KBA: Known-Background Attack; COA: Ciphertext-Only Attack; CPA:Chosen-Plaintext Attack; $d$: Dimension of index and $d > kb$; $n$: Size of dataset and $n \gg kbr$; $k$: Number of hash functions; $b$: Number of partitions; $r$: Number of repetitions.



(a)

| 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 |

(b)

Fig. 2. (a) An example of Geohash coding; (b) Accuracy of Geohash coding.

## III. PRELIMINARIES

In this section, we introduce some related technologies for our schemes, mainly including the Geohash algorithm, Circular Shift and Coalesce Bloom Filter (CSC-BF) framework, and Symmetric-key Hidden Vector Encryption (SHVE).

### A. Geohash Algorithm

Geohash algorithm [3], [30], [31], [32] is a common geocoding method, which can recursively divide the entire geographical space as smaller grids and convert the longitude and latitude of each grid as a Geohash code according to Base32. As shown in the example in Fig. 2, each Geohash code is a unique specific rectangular range, and provides arbitrary precision. For example, the red point $(8, 13)$ can be denoted as "$s33jxb67$" with $\pm 19m$ rectangular range or "$s33jxb6$" with $\pm 76m$.

Thus, we judge the distance between two spatial points by matching their Geohash codes. As shown in the example in Fig. 2, the Geohash codes of red point and green point have the same prefix "$s33jxb6$," so we can know that the green point is within $\pm 76m$ of red point. Thus, we can determine whether a point is within the query range by judging the number of characters with the same prefix of two Geohash codes.

### B. Circular Shift and Coalesce Bloom Filter (CSC-BF)

The Bloom filter is a data structure used to check whether an element is included in a set. Given a set $S = \{s_1, \ldots, s_e\}$ and an element $q$, the Bloom filter executes the following processes to determine whether $q \in S$.

- *Init*: Generate an $m$-bit array $\boldsymbol{BF}$ with each bit is 0, and choose $k$ independent hash functions $\mathcal{H} = \{h_0, \ldots, h_{k-1}\}$.

- *Insertion*: For each an element $s \in S$, this process sets the values of $k$ locations $\boldsymbol{BF}[(\mathcal{H}(s)\%m]$ in array $\boldsymbol{BF}$ as 1, where $\mathcal{H}(s)\%m = \{h_0(s)\%m, \ldots, h_{k-1}(s)\%m\}$.
- *Query*: For a query $q$, this process checks $k$ locations $\boldsymbol{BF}[(\mathcal{H}(q)\%m] \overset{?}{=} 1$. If $k$ locations are all 1, then $q \in S$.

Thus, given $n$ sets $\mathcal{S} = \{S_1, \ldots, S_n\}$, $1 \le i \le n$, the time complexity of Bloom filter determining which set $S_i \in \mathcal{S}$ contains $q$ is $\mathcal{O}(kn)$. Circular Shift and Coalesce Bloom Filter (CSC-BF) [33] is an efficient Bloom filter framework, which does not need to build a $\boldsymbol{BF}$ for each set $S_i \in \mathcal{S}$, thereby greatly reducing query time. The time complexity of CSC-BF to determine which set $S_i \in \mathcal{S}$ contains $q$ is only $\mathcal{O}(kbr)$, where $br \ll n$. Fig. 3 shows the framework of CSC-BF, and the specific structure of CSC-BF is as follows.

CSC-BF is $r$ $m$-bit Bloom filter $\{\boldsymbol{CSC}_t | 1 \le t \le r\}$ with $k$ independent hash functions $\mathcal{H} = \{h_0, \ldots, h_{k-1}\}$ and all bits are initialized to 0. Given $n$ sets $\mathcal{S} = \{S_1, \ldots, S_n\}$, $1 \le i \le n$, CSC-BF defines $r$ repetitions. Note that each repetition $R_t$ has a unique partition function $g_t$ which uniformly maps $n$ sets into $b$ ($b \ll n$) random partitions $\{P_l | 1 \le l \le b\}$, where $g_t(i) = l$ and $1 \le t \le r$. To support set inclusion query, CSC-BF needs to perform the following two processes:

- *Insertion*: For each repetition $R_t$, this process only generates a unique Bloom filter $\boldsymbol{CSC}_t$ for all $n$ sets $\mathcal{S}$. For each set $S_i$, this process sets the values of $k$ locations $\boldsymbol{CSC}_t[(\mathcal{H}(s)\%m + g_t(i))\%m]$ in $\boldsymbol{CSC}_t$ as 1 to insert each element $s \in S_i$, where $\mathcal{H}(s)\%m = \{h_0(s)\%m, \ldots, h_{k-1}(s)\%m\}$ are $k$ anchor locations and $g_t(i)$ is offset.
- *Query*: For a query element $q$ in repetition $R_t$, this process calculates its $k$ anchor locations $\mathcal{H}(q)\%m$, and then checks the next $b$ offset locations $(\mathcal{H}(q)\%m+0)\%m$, $\ldots$, $(\mathcal{H}(q)\%m + b - 1)\%m$ of $\boldsymbol{CSC}_t$ to determine which partitions contain $q$. That is, $\boldsymbol{CSC}_t[(\mathcal{H}(q)\%m + l)\%m] = 1$ indicates that $q \in P_l$. For partitions that report the existence of element $q$, this process takes the union of sets mapped into these partitions to generate a candidate membership set $M_t$. Finally, this process takes the intersection of $r$ candidate membership sets $\{M_1, \ldots, M_r\}$ to find which sets contain $q$.

The element $s$ in different sets has the same anchor, and then is divided into $b$ different partitions by offset. In query phase, CSC-BF only needs to calculate the anchor of $q$ and checks the next $b$ locations to find which partitions contain $q$. Then, CSC-BF takes all sets within these partitions as a candidate member set $M$. To reduce false positives, CSC-BF needs to repeat the above process $r$ times, and finally takes the intersection of $r$ candidate member sets as the result.

### C. Symmetric-Key Hidden Vector Encryption (SHVE)

SHVE [34] is an encryption scheme that supports equality, conjunctive and comparison queries over encrypted data. For vectors $\boldsymbol{v} = \{v_0, \ldots, v_{m-1}\} \in \Sigma^m$ and $\boldsymbol{x} = \{x_0, \ldots, x_{m-1}\} \in \Sigma^m$, where $\Sigma \in \{0, 1\}$ is a finite set of attributes, we have:

$$P_{\boldsymbol{v}}^{SHVE}(\boldsymbol{x}) = \begin{cases} 1, & \forall 0 \le j \le m-1 (v_j = x_j), \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

That is, if $\boldsymbol{x}$ matches $\boldsymbol{v}$ in all dimensions, $P_{\boldsymbol{v}}^{SHVE}(\boldsymbol{x}) = 1$.

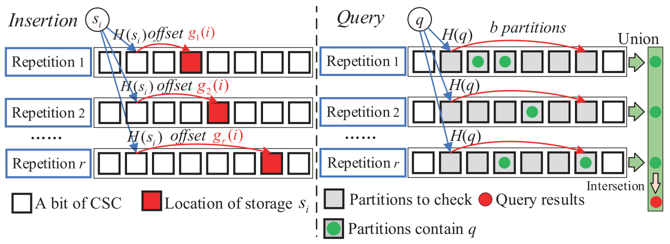


Fig. 3. The framework of CSC-BF.

<u>SHVE.Setup($1^\lambda$)</u>: Input $\lambda$, this algorithm uniformly samples a master secret key $msk \xleftarrow{\$} \{0,1\}^\lambda$, and it defines a message space $\mathcal{M}$, and then outputs $(msk, \mathcal{M})$.
<u>SHVE.KeyGen($msk, \boldsymbol{v} \in \Sigma^m$)</u>: Input the master secret key $msk$ and a vector $\boldsymbol{v} = \{v_0, \ldots, v_{m-1}\} \in \Sigma^m$, this algorithm first samples $K \xleftarrow{\$} \{0,1\}^{\lambda+\log\lambda}$, and then calculates the following: $d_0 = \oplus_{j \in m}(F(msk, v_j||j)) \oplus K, d_1 = \mathrm{Sym.Enc}(K, 0^{\lambda+\log\lambda})$. Finally, it outputs the decryption key: $\boldsymbol{s} = (d_0, d_1)$.
<u>SHVE.Enc($msk, \mu \in \mathcal{M}, \boldsymbol{x} \in \Sigma^m$)</u>: Input $msk$, a message $\mu$, and a vector $\boldsymbol{x} = \{x_0, \ldots, x_{m-1}\} \in \Sigma^m$, the algorithm calculates $c_j = F(msk, x_j||j)$. Finally, the algorithm outputs the ciphertext $\boldsymbol{c} = (\{c_j\}_{1 \le j \le m-1})$.
<u>SHVE.Query($\boldsymbol{s}, \boldsymbol{c}$)</u>: Input the decryption key $\boldsymbol{s}$ and the ciphertext $\boldsymbol{c}$, this algorithm first calculates $K' = (\oplus_{j \in m} c_j) \oplus d_0$, and then computes $\mu' = \mathrm{Sym.Dec}(K', d_1)$. If $\mu' = 0^{\lambda+\log\lambda}$ (i.e., $P_{\boldsymbol{v}}^{SHVE}(\boldsymbol{x}) = 1$), this algorithm outputs $\mu$; otherwise outputs $\perp$.

Fig. 4. Construction details of SHVE.

SHVE is achieved by using a Pseudo Random Function (PRF) $F$: $\{0,1\}^\lambda \times \{0,1\}^{\lambda+\log\lambda} \to \{0,1\}^{\lambda+\log\lambda}$ and a symmetric encryption (Sym.Enc, Sym.Dec) with IND-CPA security, where $\lambda$ is a security parameter. The details of the construction are shown in Fig. 4.

## IV. PROBLEM FORMULATION

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

### A. System Model & Threat Model

In this paper, we consider an encrypted cloud data outsourcing scenario. As shown in Fig. 5, the system model of our schemes includes three entities, namely LBS providers, mobile users and the cloud server. The role of each entity is shown as follows:

- **LBS Provider:** The LBS provider owns the spatial dataset and is responsible for building index for each spatial data, and then encrypts spatial dataset and indexes before outsourcing them to the cloud server.
- **Mobile User:** The authorized mobile user has access to the outsourced spatial dataset. He/She first encrypts the query information as token, and then submits it to the cloud server.
- **Cloud Server:** The cloud server which has unlimited computing and storage capabilities stores encrypted data and indexes outsourced by the LBS provider, and provides LBS for mobile users.

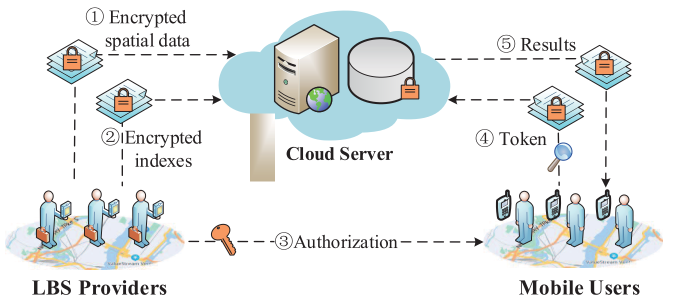


Fig. 5. System model of our schemes.

TABLE II
NOTATIONS AND DESCRIPTIONS

| Notation | Description |
|---|---|
| $n$ | Number of objects in dataset DB, $i \in [1, n]$ |
| $m$ | Size of a CSC-BF $\boldsymbol{CSC}$, $j \in [0, m-1]$ |
| $r$ | Number of repetitions $R_t$, $t \in [1, r]$ |
| $b$ | Number of partitions $P_l$, $l \in [1, b]$ |
| $s_i$ | Geohash code corresponding to the spatial location $o_i.l$ |
| $\mathcal{H}$ | $k$ independent hash functions $\mathcal{H} = \{h_0, \ldots, h_{k-1}\}$ |
| HF | A hash function, HF : $\{\rho\} \rightarrow \{0, 1\}$ |
| $\gamma, \rho$ | Random numbers |
| $g$ | $r$ independent hash functions $\{g_1, \ldots, g_r\}$ |

LBS provider first builds an index for each spatial data in dataset $\mathsf{DB}$, and then outsources the encrypted spatial dataset and indexes to the cloud server (steps ①, ②). Mobile users are authorized after registering themselves with LBS provider (step ③). When the mobile user wants to make query requests $\mathsf{Q}$, he/she encrypts the query information as token, and submits it to the cloud server (step ④). After receiving the token, the cloud server finds the spatial data satisfying query requests and then returns them to the mobile user (step ⑤).

*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 curiously collects or analyzes the meaningful information such as locations and mobile user's query information. The LBS provider and the mobile user are assumed to be completely trusted, which means that they execute specified operations properly and cannot collude with any unauthorized mobile users and the cloud server. In addition, the access, authorization and transmission channels between all entities are also assumed to be secure. And we consider the following threat model:

- *Chosen-Plaintext-Attack Model:* Apart from the ciphertexts of all spatial data, indexes, and query information, the cloud servers can also access the ciphertext of a spatial data corresponding to the plaintext it selects.

## B. Problem Definition

The existing solutions using Bloom filter do not take into account the invalid bits that do not map information in Bloom filter, resulting in high computational overheads. For example, as shown in the example in Fig. 1, given a query request $q$, the mobile user first initializes an $m$-bit Bloom filter index $\boldsymbol{BF} = (0, 0, \ldots, 0)$, then sets $k$ positions $\boldsymbol{BF}[h_1(q)\%m], \ldots, \boldsymbol{BF}[h_k(q)\%m]$ of $\boldsymbol{BF}$ as 1, and finally encrypts each bit $\boldsymbol{BF}[j]$ by using an encryption algorithm (e.g., ASPE [6]) for generating a token $\mathsf{T_Q}$, where $0 \le j \le m-1$. And the cloud server needs to calculate the inner product of encrypted index and token both described by $\boldsymbol{BF}$ bit by bit. Therefore, there are $(m-k)$ invalid 0-bits in each $\boldsymbol{BF}$, which need to be encrypted and involved in the inner product operation. Especially on the large-scale dataset, to reduce false positives, $m$ will increase with the amount of data mapped into Bloom filter, which will lead to an increase in the number of invalid 0-bits. And the computation complexities of token generation and query are proportional to the size of $m$, which makes the time of token generation and query too large under large datasets. In addition, directly encrypting the Bloom filter will reveal the inclusion relationship between 0 and 1. For example, the cloud server can judge whether the bit corresponding to the Bloom filter is the ciphertext of "1" if a set contains an element, thereby inferring the whole Bloom filter. Next, we give the formal definition of our schemes as follows:

*Definition 1 (Spatial Range Query, SRQ): Given a spatial database* $\mathsf{DB} = \{o_1, \ldots, o_n\}$ *and a query request Q, all spatial data* $\mathsf{DB}$ *are mapped as Geohash codes to generate a set* $\mathcal{S} = \{S_1, \ldots, S_n\}$*, and Q is mapped as a Geohash code q. SRQ runs the Insertion and Query algorithms to retrieve a subset* $Result = \{id_1, id_2, \ldots, id_h\}$ *from D, such that* $\forall id_e \in Result$, $1 \le e \le h$, $q \in S_e$.

## C. Design Goals

To achieve an efficient and secure spatial range query, our solutions should meet the following requirements:

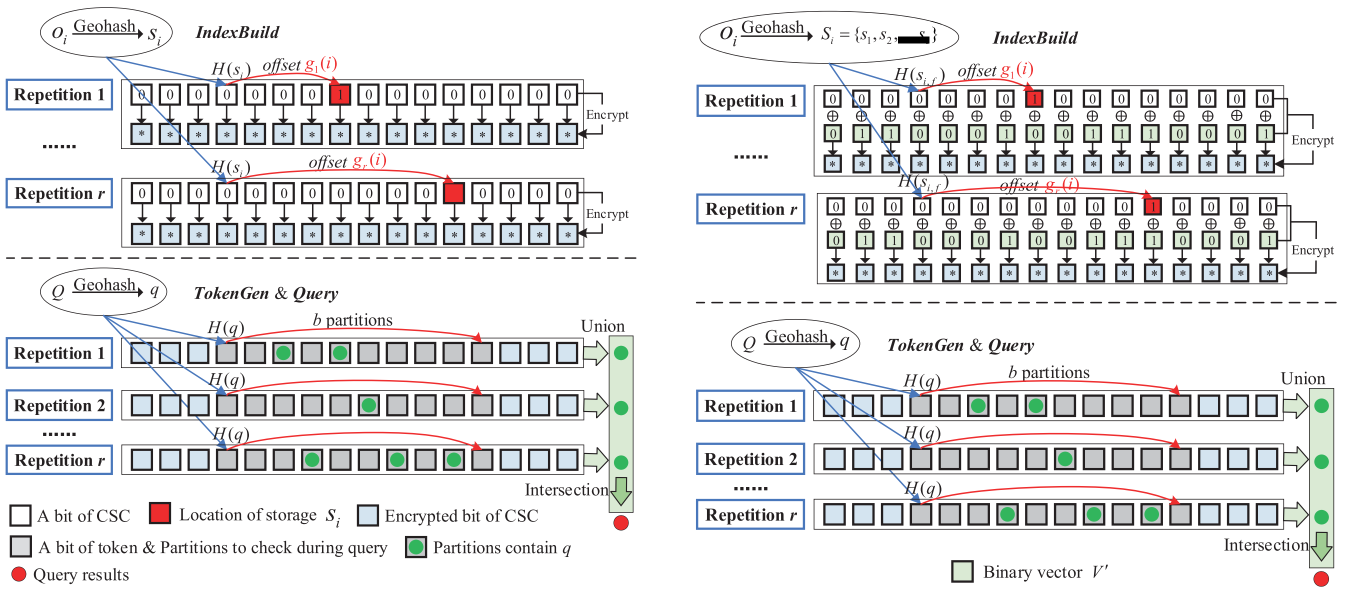
- *Data Privacy:* The plaintext dataset cannot be disclosed to any unauthorized entities and the cloud server. And they should not obtain any sensitive about these data.
- *Query Privacy:* Only authorized mobile users can generate tokens for queries and the cloud server cannot get or infer any actual query content over the token.
- *Efficiency:* Our scheme should achieve efficient token generation and query over large-scale datasets in real-time scenarios.

## V. OUR PROPOSED SCHEME

In this section, we first propose an efficient Privacy-preserving Spatial Range Query (PSRQ) scheme by combining Geohash algorithm with CSC-BF and SHVE. And we design a Confused Bloom Filter (CBF) by using hash function and XOR. Then, we propose a more practical enhanced scheme PSRQ$^+$ by using CBF to query more ranges and achieve adaptive security. Before introducing our schemes in detail, we first give the description of some important notations used in our proposed schemes in TABLE II.

## A. Basic Scheme: Efficient Privacy-Preserving Spatial Range Query (PSRQ)

The commonly used homomorphic encryption is difficult to implement fast queries under large-scale data sets due to



(a) Framework of PSRQ

(b) Framework of PSRQ$^+$

Fig. 6. Construction process of our schemes.

its high computational overhead, and ASPE has been proved to be insecure against known-plaintext attacks [35] despite its low overhead. Therefore, we choose SHVE algorithm by comprehensively considering the cost and security. In addition, there are some range query schemes that usually use Gray code or prefix code to generate encrypted indexes and tokens, but lead to longer encrypted indexes and tokens, even incur huge computational overhead as all bits of encrypted indexes and tokens need to be calculated in token generation and query phases. Therefore, we use Geohash algorithm to achieve range query, which greatly reduces the burden on mobile users by converting the query range as an 8-character Geohash code whose length is far less than those of Gray code and prefix code. However, Geohash algorithm trades efficiency for approximate query, i.e., it can only achieve approximate range query as shown in Fig. 2 (b).

*Technical Overview of PSRQ:* The most existing solutions usually use ASPE algorithm to encrypt all bits of Bloom filter to generate token and then compute the inner product of encrypted Bloom filter bit by bit during query, resulting in excessive computational overheads of token generation and query. Therefore, we first propose an efficient PSRQ scheme by skillfully combining Geohash algorithm with CSC-BF and SHVE, which uses SHVE to encrypt the valid bits of Geohash code mapped to CSC-BF to generate a shorter token for mobile users, and matches its each bit via CSC-BF during the query. Thus, our PSRQ can make the complexity of token generation independent of the length of Bloom filter, and greatly reduce the computational cost of generating token and speeds up the query efficiency, as shown in Fig. 6 (a). Specifically, in IndexBuild phase, LBS provider maps the spatial location $o_i.l = (x_i, y_i)$ of each spatial data $o_i$ as an 8-character Geohash code $s_i$. Then, LBS provider generates $r$ $m$-bit Bloom filters $\{\boldsymbol{CSC}_1, \ldots, \boldsymbol{CSC}_r\}$ for $n$ Geohash codes $\{s_1, \ldots, s_n\}$ by calling *Insertion* algorithm of CSC-BF. Finally, LBS provider encrypts each bit $v_j$ of each $\boldsymbol{CSC}_t$ for generating $r$ encrypted indexes $\{\widehat{\boldsymbol{CSC}_1}, \ldots, \widehat{\boldsymbol{CSC}_r}\}$, where $1 \le t \le r$, $0 \le j \le m-1$. In TokenGen phase, the mobile user first encodes query location $Q.l = (x_Q, y_Q)$ as an 8-character Geohash code $q$, then calculates its $k$ anchor locations $\mathcal{H}(q)\%m$ and gets corresponding $b$ offset locations $\{(\mathcal{H}(q)\%m + 0)\%m, \ldots, (\mathcal{H}(q)\%m + b - 1)\%m\}$ by shifting $b$ times, where $\mathcal{H}(q)\%m = \{h_0(q)\%m, \ldots, h_{k-1}(q)\%m\}$. Finally, the mobile user encrypts $kb$ offset locations as $\{c_0, \ldots, c_{b-1}\}$ for generating a token $\mathsf{T_Q} = \{\mathcal{H}(q)\%m, c_0, \ldots, c_{b-1}\}$. In Query phase, the cloud server checks the next $b$ locations of $\mathcal{H}(q)\%m$ in each $\widehat{\boldsymbol{CSC}_t}$ to determine which partitions contain $q$, then takes all spatial data object identities $\{id_i\}$ in these partitions as a candidate membership set $M_t$. Finally, the cloud server returns the intersection of candidate membership set $\{M_1, \ldots, M_r\}$ to the mobile user. The concrete construction of PSRQ is as follows.

Setup. LBS provider generates a master key $msk$ by calling SHVE.Setup, and generates a PRF $F$, $k$ independent hash functions $\mathcal{H} = \{h_0, \ldots, h_{k-1}\}$ and $r$ independent hash functions $\{g_1, \ldots, g_r\}$.

IndexBuild. The specific construction is shown in Algorithm 1. Let $\mathsf{DB} = \{o_1, o_2, \ldots, o_n\}$ be a spatial database owned by LBS provider. Each spatial data $o_i$ in DB has a unique identity $id_i$ and its spatial geographic location $o_i.l = (x_i, y_i)$. For each $o_i$, LBS provider maps $o_i.l$ as an 8-character Geohash code $s_i$. As shown in the example in Fig. 2, the spatial location (8, 13) will be converted as "$s33jxb67$." Then, LBS provider sets $r$ repetitions $\{R_1, \ldots, R_r\}$ and initializes an $m$-bit Bloom filter $\boldsymbol{CSC}_t$ for each $R_t$, where $1 \le t \le r$. For each $R_t$, LBS provider maps all Geohash codes $\{s_1, \ldots, s_n\}$ as $b$ ($b < n$) random partitions $\{P_l | 1 \le l \le b\}$ by calculating $g_t(i)\%m$, and then sets the values of $k$ locations $\boldsymbol{CSC}_t[(\mathcal{H}(s_i)\%m + g_t(i))\%m]$ in $\boldsymbol{CSC}_t$ as 1 to insert each $s_i$

into $\boldsymbol{CSC}_t$, where $\mathcal{H}(s_i)\%m = \{h_0(s_i)\%m, \ldots, h_{k-1}(s_i)\%m\}$. Thus, LBS provider generates $r$ indexes $\{\boldsymbol{CSC}_1, \ldots, \boldsymbol{CSC}_r\}$ for DB. Then, LBS provider randomly samples $K \xleftarrow{\$} \{0,1\}^{\lambda+\log\lambda}$ and encrypts each bit $v_j$ of each $\boldsymbol{CSC}_t$ as $d_{j,0}$ and $d_{j,1}$ by Eq. 2, where $1 \le j \le m-1$. Finally, LBS provider outsources $r$ encrypted indexes EDB $= \{\widehat{\boldsymbol{CSC}}_1, \ldots, \widehat{\boldsymbol{CSC}}_r\}$ to cloud server.

**Algorithm 1** IndexBuild

```
Input: DB = $\{o_1, o_2, \ldots, o_n\}$
Output: EDB
1  for $1 \le i \le n$ do
2      Convert $o_i.l$ as an 8-character Geohash code $s_i$;
3  for $1 \le t \le r$ do
4      Initialize an $m$-bit Bloom filter $\boldsymbol{CSC}_t$;
5      Map $\{s_1, \ldots, s_n\}$ uniformly as $b$ random partitions
         $\{P_l | 1 \le l \le b\}$ by calculating $g_t(i)\%m$;
6      // Generate an index $\mathbf{CSC}_t$;
7      for $1 \le i \le n$ do
8          Set $k$ locations $\boldsymbol{CSC}_t[(\mathcal{H}(s_i)\%m + g_t(i))\%m]$
             in $\boldsymbol{CSC}_t$ as 1;
9      // Generate an encrypted index $\widehat{\mathbf{CSC}}_t$;
10     for $0 \le j \le m-1$ do
11         Encrypt each bit $v_j \in \boldsymbol{CSC}_t$ as:
               $d_{j,0} = F(msk, v_j||j) \oplus K,$
               $d_{j,1} = \mathrm{Sym.Enc}(K, 0^{\lambda+\log\lambda}).$      (2)
12 return EDB = $\{\widehat{\boldsymbol{CSC}}_1, \ldots, \widehat{\boldsymbol{CSC}}_r\}$.
```

**Algorithm 2** TokenGen

```
Input: $Q.l$
Output: $\mathsf{T_Q}$
1  Convert $Q.l$ as an 8-character Geohash code $q$;
2  Calculate $k$ anchor locations $\mathcal{H}(q)\%m$;
3  for $0 \le l \le b-1$ do
4      Calculate $(\mathcal{H}(q)\%m + l)\%m$ and encrypt it as:
           $c_l = F(msk, 1||\mathcal{H}(q)\%m + l)\%m);$      (3)
5  return $\mathsf{T_Q} = \{\mathcal{H}(q)\%m, c_0, \ldots, c_{b-1}\}$.
```

TokenGen. The specific construction is shown in Algorithm 2. Given a query location $Q.l = (x_Q, y_Q)$, the mobile user first encodes $Q.l$ as an 8-character Geohash code $q$. Then, the mobile user calculates its $k$ anchor locations as $\mathcal{H}(q)\%m$ and gets corresponding $b$ offset positions $\{\mathcal{H}(q)\%m + l)\%m | 0 \le l \le b-1\}$ by shifting $b$ times, where $\mathcal{H}(q)\%m = \{h_0(q)\%m, \ldots, h_{k-1}(q)\%m\}$. Then, the mobile user encrypts $kb$ offset positions as $\{c_0, \ldots, c_{b-1}\}$ by Eq. 3. Finally, the mobile user submits the token $\mathsf{T_Q}$ to the cloud server.

Query. The specific construction is in Algorithm 3. After receiving the mobile user's token $\mathsf{T_Q}$, the cloud server checks the next $b$ locations $\{\mathcal{H}(q)\%m + l)\%m | 0 \le l \le b-1\}$ of anchor locations $\mathcal{H}(q)\%m$ in each $\widehat{\boldsymbol{CSC}}_t$ to determine which partitions contain $q$ by Eq. 4. If $\mu' = 0^{\lambda+\log\lambda}$, i.e., $\boldsymbol{CSC}_t[(\mathcal{H}(q)\%m + l)\%m] = 1$, then $P_l$ contains $q$. The cloud server takes all spatial data object identities $\{id_i\}$ in partition $P_l$ as a candidate membership set $M_t$. Finally, the cloud server returns the intersection of the candidate membership set $\{M_1, \ldots, M_r\}$ obtained by each repetition as Result to the mobile user.

**Algorithm 3** Query

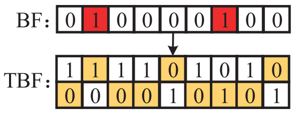
```
Input: EDB = $\{\widehat{\boldsymbol{CSC}}_t | 1 \le t \le r\}$ and
       $\mathsf{T_Q} = \{\mathcal{H}(q)\%m,\ c_0, c_1, \ldots, c_{b-1}\}$
1  Initialize a set Result = $\emptyset$;
2  for $1 \le t \le r$ do
3      Initialize a candidate membership set $M_t$;
4      for $0 \le l \le b-1$ do
5          $j = \mathcal{H}(q)\%m + l$;
6          Determine if partition $P_l$ contains $q$ as:
               $K' = c_l \oplus d_{j,0},$
               $\mu' = \mathrm{Sym.Dec}(K', d_{j,1});$      (4)
7          if $\mu' = 0^{\lambda+\log\lambda}$ then
8              $M_t.add(id_i),\ \forall o_i \in P_l$;
9  Result = $\cap_{t=1}^{r} M_t$;
10 return Result.
```

**Correctness**. For each offset location $(\mathcal{H}(q)\%m + l)\%m$ in each $\widehat{\boldsymbol{CSC}}_t$, $c_l \oplus d_{j,0}$ is calculated as Eq. 5,

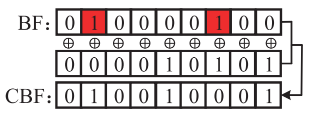
$$c_l \oplus d_{j,0} = F(msk, 1||\mathcal{H}(q)\%m + l)\%m) \oplus F(msk, v_j||j) \oplus K = K', \quad (5)$$

where $0 \le l \le b-1$, $j = (\mathcal{H}(q)\%m + l)\%m$. If $\mu' = \mathrm{Sym.Dec}(K', d_{j,1}) = 0^{\lambda+\log\lambda}$, we can know that $K = K'$ and $v_j = 1$. Therefore, the $(\mathcal{H}(q)\%m + l)\%m$-*th* bit of $\boldsymbol{CSC}_t$ is 1 and the $l$-*th* partition $P_l$ contains $q$, i.e., there is a Geohash code $s_i$ in $P_l$ which is the same as $q$.

*Remark 1:* As the length of each Geohash code is far less than those of other spatial codes such as Gray code, the Geohash code can be inserted into CSC-BF as a keyword to greatly improve the query efficiency. And PSRQ uses SHVE to encrypt the valid bits of Geohash code mapped to CSC-BF, and matches its each bit via CSC-BF during the query, which makes the time complexity $\mathcal{O}(kb)$ of TokenGen and the time complexity $\mathcal{O}(kbr)$ of Query independent of the length $m$ of Bloom filter. However, since the basic solution only encodes spatial data as an 8-character Geohash code, the mobile users can only search for PoIs within 19 meters, which is not practical for mobile users. In addition, PSRQ encrypts the generated Bloom filter directly, which may reveal the inclusion relationship between 0 and 1 in the Bloom filter. For example, the cloud server can judge whether the bit corresponding to Bloom filter is the ciphertext of "1" base on the partition added to candidate set, so that it can infer the entire Bloom filter.



(a) An example of TBF



(b) Our method

Fig. 7. Technical overview of CBF.

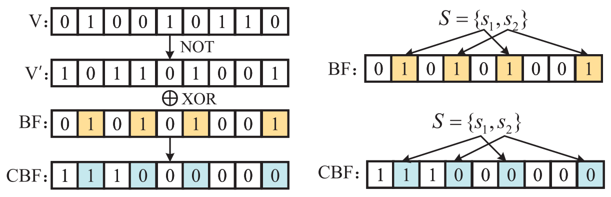


Fig. 8. An example of CBF.

## *B. Enhanced Scheme: Practical Privacy-Preserving Spatial Range Query (PSRQ$^+$)*

*1) Technical Overview of PSRQ$^+$:* To solve the flaws of basic scheme, we first encode the spatial location $o_i.l$ as a set containing 8 Geohash codes of different lengths according to the accuracy of Geohash algorithm, then allow the mobile user to encode the query request as a Geohash code of any length to achieve range query. To protect the values of 0 and 1 in Bloom filter, the most direct way is to use Twin Bloom Filter (TBF) [4], [5] to confuse the inclusion relationship of 0 and 1 to achieve adaptive security, but TBF needs to construct two rows of Bloom filters, resulting in huge computational and storage overheads, as shown in Fig. 7. Therefore, we first devise a Confused Bloom Filter (CBF) scheme by using hash function and XOR, which achieves the same adaptive security as TBF and generates only one row of Bloom filter. The specific construction is as follows.

*2) Construction of Confused Bloom Filter (CBF):* CBF confuses inclusion relationship of 0 and 1 in Bloom filter, which includes the following processes.

- *CBF.Setup*: Generate an $m$-bit Bloom Filter $\boldsymbol{BF}$ for a set $S$ through $k$ hash functions $\mathcal{H} = \{h_0, \ldots, h_{k-1}\}$. Choose a random number $\gamma$ and a hash function $\mathsf{HF} : \{\rho\} \rightarrow \{0, 1\}$, where $\rho$ is an arbitrary number.
- *CBF.Init*: Generate an $m$-bit binary vector $V$, where $V[j] = \mathsf{HF}(j + \gamma)$, $0 \leq j \leq m - 1$. Then, negate each bit of $V$ to generate an $m$-bit binary vector $V' = \neg V$, where $V'[j] = \neg\mathsf{HF}(j + \gamma)$ and "$\neg$" represents the sign of binary negation.
- *CBF.Confuse*: XOR each bit in $\boldsymbol{BF}$ with the bit in $V'$ to generate a $\boldsymbol{CBF}$.
- *CBF.Query*: For a query $q$, this process checks $k$ locations $\boldsymbol{CBF}[(\mathcal{H}(q)\%m] \overset{?}{=} \mathsf{HF}(\mathcal{H}(q)\%m + \gamma)$, where $\mathcal{H}(q)\%m = \{h_0(q)\%m, \ldots, h_{k-1}(q)\%m\} \in [0, m-1]$. If so, then $q \in S$.

**Correctness**. The Bloom filter sets the values of $k$ locations $\boldsymbol{BF}[j]$ as 1 to insert each element $s \in S$, where $j = \mathcal{H}(s)\%m$. Then, the Bloom filter determines whether $q \in S$ by checking whether $k$ locations $\boldsymbol{BF}[(\mathcal{H}(q)\%m] = 1$. Different from Bloom filter, in order to insert element $s$, the CBF sets the values of $k$ locations $\boldsymbol{CBF}[j]$ by Eq. 6.

$$\begin{aligned}\boldsymbol{CBF}[j] &= \neg\mathsf{HF}(j+\gamma) \oplus \boldsymbol{BF}[j] \\ &= \neg\mathsf{HF}(j+\gamma) \oplus 1 \\ &= \mathsf{HF}(j+\gamma) \oplus 1 \oplus 1 \\ &= \mathsf{HF}(j+\gamma). \end{aligned} \quad (6)$$

Then, CBF determines whether $q \in S$ by checking whether $\boldsymbol{CBF}[(\mathcal{H}(q)\%m] = \mathsf{HF}(\mathcal{H}(q)\%m + \gamma)$ holds. Since the values of $k$ locations $\mathsf{HF}(\mathcal{H}(s)\%m + \gamma)$ are 0 or 1, CBF can confuse the inclusion relationship of Bloom filter. Fig. 8 shows an example of CBF. To insert element $s_1$, CBF sets $\boldsymbol{CBF}[1] = 1$ and $\boldsymbol{CBF}[5] = 0$. Thus, even if an attacker knows that $s_1 \in S$, he cannot determine that the corresponding location is 1, i.e., 0 can also indicate that the location contains elements.

Then, we improve PSRQ to propose a more secure and practical enhanced scheme PSRQ$^+$, as shown in Fig. 6 (b). Specifically, LBS provider encodes each spatial location $o_i.l$ in dataset DB as a set containing 8 Geohash codes of different lengths and inserts them into CSC-BF, and then confuses CSC-BF with CBF and encrypts it with SHVE. The specific construction of PSRQ$^+$ is as follows.

Setup. LBS provider generates a master key $msk$ by calling SHVE.Setup, and generates a PRF $F$, $k$ hash functions $\mathcal{H} = \{h_0, \ldots, h_{k-1}\}$, $r$ independent hash functions $\{g_1, \ldots, g_r\}$, a random number $\gamma$ and a hash function $\mathsf{HF}$:$\{\rho\} \rightarrow \{0, 1\}$, where $\rho$ is an arbitrary number.

**Algorithm 4** IndexBuild

```
Input: DB = $\{o_1, o_2, \ldots, o_n\}$
Output: EDB
1  for $1 \leq i \leq n$ do
2      Convert $o_i.l$ as a set $S_i = \{s_1, s_2, \ldots, s_8\}$
         containing 8 Geohash codes of different lengths;
3  Generate an $m$-bit binary vector $V$;
4  Negate each bit of $V$ to generate $V'$;
5  for $1 \leq t \leq r$ do
6      Initialize a $m$-bit Bloom filter $\boldsymbol{CSC}_t$;
7      Map $\{S_1, \ldots, S_n\}$ uniformly as $b$ random partitions
         $\{P_l | 1 \leq l \leq b\}$ by calculating $g_t(i)\%m$;
8      for $1 \leq i \leq n$ do
9          for $s \in S_i$ do
10             Set $\boldsymbol{CSC}_t[(\mathcal{H}(s)\%m + g_t(i))\%m]$ as 1;
11     for each bit $v_j \in \mathbf{CSC}_t$ do
12         $v'_j = v_j \oplus V'[j]$ ;
13         Encrypt $v'_j$ as $d_{j,0}$ and $d_{j,1}$ by Eq. 2;
14 return EDB = $\{\widehat{\boldsymbol{CSC}'_1}, \ldots, \widehat{\boldsymbol{CSC}'_r}\}$.
```

IndexBuild. The specific construction is shown in Algorithm 4. Different from PSRQ, for each spatial data $o_i$, LBS provider maps the spatial location $o_i.l$ as 8 Geohash codes of different lengths for generating a set $S_i = \{s_1, s_2, \ldots, s_8\}$ according to the accuracy of Geohash algorithm. As shown in the example in Fig. 2, the spatial location (8, 13) will be converted as a set $S = \{s, s3, s33, \ldots, s33jxb67\}$ containing 8 Geohash codes.

Then, similar to PSRQ, LBS provider generates $r$ indexes $\{\boldsymbol{CSC}_1, \ldots, \boldsymbol{CSC}_r\}$ for $n$ sets $\{S_1, \ldots, S_n\}$.

To confuse the inclusion relationship of each bit in $\boldsymbol{CSC}_t$, LBS provider generates an $m$-bit binary vector $V$, where the value of $j$-$th$ bit is $\mathsf{HF}(j+\gamma)$, $0 \le j \le m-1$. And LBS provider negates each bit of $V$ to generate an $m$-bit binary vector $V'$. Then, LBS provider XORs each bit in each $\boldsymbol{CSC}_t$ with the bit in $V'$ to generate $r$ new indexes $\{\boldsymbol{CSC}'_1, \ldots, \boldsymbol{CSC}'_r\}$. Finally, LBS provider encrypts them as $\mathsf{EDB} = \{\widehat{\boldsymbol{CSC}'_1}, \ldots, \widehat{\boldsymbol{CSC}'_r}\}$ by Eq. 2, and outsources $\mathsf{EDB}$ to the cloud server.

$\mathsf{TokenGen}$. The specific construction is shown in Algorithm 5. Given a query request $\mathsf{Q} = \{Q.l, Q.r\}$, where $Q.l = (x_Q, y_Q)$ is geographic coordinate of query and $Q.r$ is search range, the mobile user can encode $Q.l$ as a Geohash code $q$ with a length of $t$-character according to $Q.r$. For example, $Q.l = (8, 13)$ is encoded as Geohash code $q =$ "$s33jxb$" according to $Q.r = 600m$, or $Q.l = (8, 13)$ is encoded as $q =$ "$s33jx$" according to $Q.r = 2km$. Then, similar to PSRQ, the mobile user gets corresponding $b$ offset positions $\{\mathcal{H}(q)\%m + l)\%m | 0 \le l \le b-1\}$. Then, the mobile user encrypts each offset position as $c_l$ by Eq. 7.

$$c_l = F(msk, \mathsf{HF}((\mathcal{H}(q)\%m + l)\%m + \gamma) || \mathcal{H}(q)\%m + l)\%m). \tag{7}$$

Finally, the mobile user submits the token $\mathsf{T_Q} = \{\mathcal{H}(q)\%m, c_0, c_1, \ldots, c_{b-1}\}$ to the cloud server.

**Algorithm 5** TokenGen
```
  Input: $\mathsf{Q} = \{Q.l, Q.r\}$
  Output: $\mathsf{T_Q}$
1 Convert $Q.l$ as an $t$-character Geohash code $q$
    according to $Q.r$;
2 Calculate $k$ anchor locations $\mathcal{H}(q)\%m$;
3 for $0 \le l \le b-1$ do
4     Calculate $(\mathcal{H}(q)\%m + l)\%m$ and encrypt it as $c_l$
        by Eq. 7.
5 return $\mathsf{T_Q} = \{\mathcal{H}(q)\%m, c_0, \ldots, c_{b-1}\}$.
```

$\mathsf{Query}$. This process is the same as that of PSRQ.

**Correctness**. Similar to PSRQ, for each offset location $(\mathcal{H}(q)\%m + l)\%m$ in each $\widehat{\boldsymbol{CSC}'_t}$, $c_l \oplus d_{j,0} = F(msk, \mathsf{HF}((\mathcal{H}(q)\%m + l)\%m + \gamma)||\mathcal{H}(q)\%m + l)\%m) \oplus F(msk, v'_j||j) \oplus K = K'$, where $0 \le l \le b-1, j = (\mathcal{H}(q)\%m + l)\%m$. If $\mu' = \mathrm{Sym.Dec}(K', d_{j,1}) = 0^{\lambda + \log\lambda}$, we can know that $K = K'$ and $v'_j = \mathsf{HF}((\mathcal{H}(q)\%m + l)\%m + \gamma)$, i.e., $\boldsymbol{CSC}'_t[(\mathcal{H}(q)\%m + l)\%m] = \mathsf{HF}((\mathcal{H}(q)\%m + l)\%m + \gamma)$. Therefore, according to the properties of CBF, the $l$-$th$ partition $P_l$ contains $q$, i.e., there is a set $S_i$ in $P_l$ that contains $q$.

*Remark 2:* In our enhanced scheme, the same query range can be generated as the same token, but does not leak the plaintext query information, which also achieves the acceptable security requirements as most of existing SSE solutions. Even though the cloud server can deduce that any two tokens are associated with the same Geohash code, these two result sets are still different as several adjacent spatial locations can be encoded as the same Geohash code. In addition, the cloud server may infer the approximate query range based on the same token, but cannot know the specific query range. However, PSRQ$^+$ also has some shortcomings, i.e., it only provides eight approximate query ranges due to the precision limitation of Geohash algorithm. Our current solutions focus on achieving efficient query on large-scale datasets, and arbitrary range query will be addressed as an important issue in future work.

## VI. Security Analysis

In this section, we give the formal security definition to prove that PSRQ is secure against IND-CPA and PSRQ$^+$ is secure against an adaptive IND-CPA adversary.

Before proving that PSRQ guarantees data privacy and query privacy against IND-CPA, we first simulate a security game played between an adversary $\mathcal{A}$ from the ideal security and a challenger $\mathcal{C}$.

*Definition 2 (IND-CPA of PSRQ): Let* $\Pi = (\mathsf{Setup}, \mathsf{IndexBuid}, \mathsf{TokenGen}, \mathsf{Query})$ *be a PSRQ scheme. The security game between an adversary* $\mathcal{A}$ *and a challenger* $\mathcal{C}$ *is defined as*:

- $\mathsf{Init}$*: Input a security parameter* $\lambda$, $\mathcal{A}$ *generates two databases* $\mathsf{DB}_0 = (d_{0,1}, d_{0,2}, \ldots, d_{0,n})$ *and* $\mathsf{DB}_1 = (d_{1,1}, d_{1,2}, \ldots, d_{1,n})$ *with the same number of spatial data to* $\mathcal{C}$*, where* $d_{0,i}$ *and* $d_{1,i}$ *are spatial data,* $1 \le i \le n$.
- $\mathsf{Setup}$*: The challenger* $\mathcal{C}$ *runs* $\mathsf{Setup}$ *to generate a master key msk and keeps msk private.*
- $\mathsf{Phase1}$*:* $\mathcal{A}$ *submits a series of requests, which are divided into the following two types*:
  1) *On the* $j$*-th ciphertext request, the adversary* $\mathcal{A}$ *outputs a spatial dataset* $\mathsf{DB}'_j = \{d'_{j,1}, \ldots, d'_{j,n}\}$*, where* $d'_{j,i}$ *is spatial data object,* $1 \le i \le n$*. Challenger* $\mathcal{C}$ *responses with an encrypted dataset* $\mathsf{EDB}_j$ *through* $\mathsf{IndexBuid}$.
  2) *On the* $j$*-th query request, the adversary* $\mathcal{A}$ *outputs a spatial range query* $\mathsf{Q}'_j = \{Q'_j.l, Q'_j.r\}$*. Challenger* $\mathcal{C}$ *responses with a token* $\mathsf{T}_{\mathsf{Q}_j}$ *through* $\mathsf{TokenGen}$.
- $\mathsf{Challenge}$*:* $\mathcal{C}$ *chooses a random bit* $b \in \{0, 1\}$*, and calculates the ciphertext* $\mathsf{EDB}_b$ *of* $\mathsf{DB}_b$ *through* $\mathsf{IndexBuid}$ *or token* $\mathsf{T}_{\mathsf{Q}_b}$ *of* $\mathsf{Q}_b$ *through* $\mathsf{TokenGen}$ *before sending it to* $\mathcal{A}$.
- $\mathsf{Phase2}$*:* $\mathcal{A}$ *can continue to adaptively submit a series of requests that are the same as those in phase 1.*
- $\mathsf{Guess}$*:* $\mathcal{A}$ *takes a guess* $b'$ *of* $b$.

We say that PSRQ is secure against IND-CPA if for any polynomial-time adversary in the above security game, it has at most a negligible advantage

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

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

*Theorem 1 (Data Privacy of PSRQ): PSRQ is IND-CPA data secure if there exists no adversary that can break the above IND-CPA game with non-negligible advantage.*

*Proof:* To prove the IND-CPA data privacy of PSRQ, we first simulate the above security game with an adversary $\mathcal{A}$. And we should prove that $\mathcal{A}$ cannot distinguish $\mathsf{EDB}_{0,i}$ and $\mathsf{EDB}_{1,i}$, even if the adversary $\mathcal{A}$ has oracle access to $\mathsf{IndexBuild}$. Assume that one of the messages in $\mathsf{DB}'_j$ is $d'_{j,i}$. Note that $d'_{j,i}$ is first converted into Geohash code $s_{j,i}$, then inserted into CSC-BF to generate an index $\boldsymbol{CSC}$ by calculating $(\mathcal{H}(s_{j,i})\%m + g_t(i))\%m$, where $\mathcal{H}$ and $g_t$ are independent hash functions. Finally, each bit $v_f$ of $\widehat{\boldsymbol{CSC}}$ is encrypted as $d_0 = F(msk, v_f||f) \oplus K$ and $d_1 = \text{Sym.Enc}(K, 0^{\lambda+\log\lambda})$, where $msk$ and $K$ are randomly sampled from $\{0,1\}^\lambda$ and $\{0,1\}^{\lambda+\log\lambda}$ respectively. Since $\mathcal{A}$ has no idea about $\mathcal{H}$, $g_t$, $msk$ and $K$, he cannot recover the ciphertext $\mathsf{EDB}_{j,i}$.

In Phase 1 and Phase 2 of security game, the adversary $\mathcal{A}$ chooses different $d'_{j,i}$ each time and observes its ciphertext $\mathsf{EDB}_{j,i}$. However, since $\widehat{\boldsymbol{CSC}}$ is a random vector determined by $\mathcal{C}$, $\mathcal{H}$ and $g_t$ are random independent hash functions, $msk \xleftarrow{\$} \{0,1\}^\lambda$ $K \xleftarrow{\$} \{0,1\}^{\lambda+\log\lambda}$, the ciphertext $\mathsf{EDB}_{j,i}$ is random to adversary $\mathcal{A}$. That is, for any message and its corresponding ciphertext selected by adversary $\mathcal{A}$, $\mathcal{A}$ cannot distinguish which message is actually encrypted. Thus, even if the adversary $\mathcal{A}$ can access $\mathsf{IndexBuild}$, $b'$ can only be obtained by random guessing. So we have

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

Therefore, PSRQ is IND-CPA data secure base on the above specific analysis. ■

*Theorem 2 (Query Privacy of PSRQ): PSRQ is IND-CPA query secure if there is no adversary that can break the adaptive IND-CPA game with non-negligible advantage.*

*Proof:* The proof of query privacy of PSRQ is similar to the proof of data privacy mentioned above. We first simulate the above security game to prove the query privacy of PSRQ. And we should prove that the adversary $\mathcal{A}$ cannot distinguish $\mathsf{T}_{\mathsf{Q}_0}$ and $\mathsf{T}_{\mathsf{Q}_1}$, even if $\mathcal{A}$ has oracle access to $\mathsf{TokenGen}$. According to the process of $\mathsf{TokenGen}$, $\mathsf{Q}'_j$ is first converted into Geohash code $q_j$, then encrypted as $c = F(msk, 1||\mathcal{H}(q_j)\%m)\%m)$, where $\mathcal{H}$ and $g_t$ are independent hash functions, $msk$ and $K$ are randomly sampled from $\{0,1\}^\lambda$ and $\{0,1\}^{\lambda+\log\lambda}$ respectively. Since $\mathcal{A}$ has no idea about $\mathcal{H}$, $g_t$, $msk$ and $K$, the ciphertext $\mathsf{T}_{\mathsf{Q}_j}$ is random to $\mathcal{A}$ who cannot recover the ciphertext $\mathsf{T}_{\mathsf{Q}_j}$. for any message and its corresponding ciphertext selected by adversary $\mathcal{A}$, the adversary $\mathcal{A}$ cannot distinguish which query request is actually encrypted. Thus, even if $\mathcal{A}$ can access $\mathsf{TokenGen}$, $b'$ can only be obtained by random guessing. So we have

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

Therefore, PSRQ is IND-CPA query secure base on the above specific analysis. ■

PSRQ$^+$ improves the security of PSRQ by confusing the inclusion relationship between 0 and 1 in Bloom filter. We next demonstrate that PSRQ$^+$ is adaptive security.

*Theorem 3: PSRQ$^+$ is secure against an adaptive IND-CPA adversary on condition that the adversary can only break the above security game.*

*Proof:* "1" in traditional Bloom filter represents inclusion and "0" represents exclusion, i.e., 1-bit in Bloom filter indicates that there is an element inserted at this location. Our CBF scheme confuses the inclusion relationship of Bloom filter, i.e., "0" can also represent inclusion and "1" can represent exclusion. Therefore, the value of each bit of Bloom filter no longer has any meaning, and the adversary cannot determine whether an element has been inserted based on whether the location is 1.

Specifically, we use several pseudo-random hash functions $F$, $\mathcal{H}$, $g_t$, $\mathsf{HF}$, and a random number $\gamma$ to generate encrypted index and token. Note that, a hash function is a pseudo-random hash function if and only if the polynomial time adversary cannot distinguish between the output of this function and that of a truly random function. We consider a probabilistic polynomial-time adaptive adversary who can access the result of past encrypted indexes, tokens and results before choosing the next query in the simulator. We first construct a simulator that can build a simulated encrypted index and token. The simulator first uses $\mathcal{H}$ and $g_t$ to insert the spatial data $s$ into $\boldsymbol{CSC}$, then uses $\mathsf{HF}$ and $\gamma$ to confuse 0 and 1 in $\boldsymbol{CSC}$ to generate a new $\boldsymbol{CSC}'$, and finally uses $F$ to encrypt it as a simulated index. For query $Q$, the simulator uses $\mathsf{HF}$ and $\gamma$ to confuse $\mathcal{H}(Q)$, and then uses $F$ to encrypt it as a token. Now if a probabilistic polynomial time adversary wants to issue a search request, the simulator generates a token for this search as above. The adversary cannot distinguish between the token given by the simulator and the query result generated by the simulation index as $F$, $\mathcal{H}$, $g_t$ and $\mathsf{HF}$ are pseudo-random functions and $\gamma$ is a random number. Thus, PSRQ$^+$ is secure against an adaptive IND-CPA adversary. ■

## VII. Performance Analysis

In this section, we first conduct detailed accuracy analysis and test for our schemes, and then conduct specific performance analysis, and extensive experimental tests for $\mathsf{IndexBuid}$, $\mathsf{TokenGen}$ and $\mathsf{Query}$. The whole experiments are carried out by using C++ programming language on 64-bit Windows 10 system and completed on Intel(R) Core(TM) i7-10700 CPU @2.90GHz server.

### A. Accuracy Analysis

We use CSC-BF, an improved Bloom filter, to build index. Thus, the accuracy of our schemes is mainly affected by the false positives of CSC-BF.

We first introduce the false positives of traditional Bloom filter. For $q \notin S$, the Bloom filter may incorrectly report $q \in S$, which leads to false positives due to its corresponding $k$ bits $h_0(q), \ldots, h_{k-1}(q)$ may be set as 1 by other elements. If $m$ is very large, the false positive rate of Bloom filter incorrectly reporting $q \in S$ can be approximately calculated as

$$\epsilon_{bf}(m, k, |S|) \approx (1 - (1 - \frac{1}{m})^{k|S|})^k \approx (1 - e^{-k|S|/m})^k,$$

where $|S|$ represents the number of elements in set $S$.

For a query element $q$ existing in $v$ sets out of $n$ sets $\{S_1, \ldots, S_n\}$, i.e., $v = |M_q|$. For a set $S_i, i \in \{1, \ldots, n\} \setminus M_q$,

TABLE III

LENGTH OF A CSC-BF INDEX WHEN FALSE POSITIVE IS CLOSE TO 0

| | $n =$ | $1 \times 10^6$ | $1.5 \times 10^6$ | $2 \times 10^6$ | $2.5 \times 10^6$ | $3 \times 10^6$ |
|---|---|---|---|---|---|---|
| $r = 3$ | $b = 70$ | $3.9 \times 10^7$ | $5.2 \times 10^7$ | $6.68 \times 10^7$ | $8.7 \times 10^7$ | $1.42 \times 10^8$ |
| | $b = 80$ | $3.75 \times 10^7$ | $5.13 \times 10^7$ | $6.6 \times 10^7$ | $8.6 \times 10^7$ | $1.3 \times 10^8$ |
| | $b = 90$ | $3.7 \times 10^7$ | $5.1 \times 10^7$ | $6.55 \times 10^7$ | $8.56 \times 10^7$ | $1.24 \times 10^8$ |
| $r = 4$ | $b = 50$ | $3.56 \times 10^7$ | $4.72 \times 10^7$ | $6.37 \times 10^7$ | $8.28 \times 10^7$ | $9.95 \times 10^7$ |
| | $b = 60$ | $3.4 \times 10^7$ | $4.6 \times 10^7$ | $6.2 \times 10^7$ | $8.1 \times 10^7$ | $9.8 \times 10^7$ |
| | $b = 70$ | $3.32 \times 10^7$ | $4.45 \times 10^7$ | $6.1 \times 10^7$ | $8.1 \times 10^7$ | $9.67 \times 10^7$ |
| $r = 5$ | $b = 50$ | $3 \times 10^7$ | $4.1 \times 10^7$ | $5.7 \times 10^7$ | $7.5 \times 10^7$ | $9.23 \times 10^7$ |
| | $b = 60$ | $2.92 \times 10^7$ | $4.1 \times 10^7$ | $5.62 \times 10^7$ | $7.43 \times 10^7$ | $9.2 \times 10^7$ |
| | $b = 70$ | $2.9 \times 10^7$ | $4 \times 10^7$ | $5.5 \times 10^7$ | $7.4 \times 10^7$ | $9.15 \times 10^7$ |

the false positive rate $\epsilon$ of CSC-BF incorrectly reporting $q \in S_i$ is bounded as

$$\epsilon \lessapprox (1 - (1 - \epsilon_{bf}(m, k, \sum_{i=1}^{n} |S_i|))(1 - \frac{1}{b})^v)^r, \tag{8}$$

where $r \geq 2$ and $\epsilon_{bf}(m, k, \sum_{i=1}^{n} |S_i|)$ is the false positive rate of a Bloom filter with $m$ bits, $k$ hash functions and $\sum_{i=1}^{n} |S_i|$ elements inserted [33].

For a more detailed analysis of CSC-BF false positives, please refer to [33]. We can know that the main factors causing false positives are the variable $r$, the length $m$ of CSC-BF, and the number $b$ of partitions. Note that $r$ and $m$ together determine the cost of IndexBuid, $r$ and $b$ together determine the costs of TokenGen and Query. To provide the mobile users with better LBS, we will use real dataset (USA road network[1]) to test the conditions (i.e., $r, b, m$) when false positives are close to 0, on condition that the overheads of TokenGen and Query are as low as possible by varying the size of dataset (i.e., $n$). Therefore, we take PSRQ$^+$ as the representative and show its experimental results in TABLE III. Since the information that PSRQ needs to map to CSC-BF is much less than that of PSRQ$^+$, which also applies to PSRQ.

## *B. Theoretical Analysis*

We analyze the theoretical complexities of PSRQ, PSRQ$^+$, PPBSKQ [24] and PPTR [29] in terms of computational and storage overheads of IndexBuid, TokenGen and Query. TABLE IV gives a specific theoretical complexity comparison results of four schemes.

Both PSRQ and PSRQ$^+$ use SHVE to encrypt each bit of index. The difference is that PSRQ and PSRQ$^+$ need to map 1 and 8 Geohash codes corresponding to each object to Bloom filter respectively. Without losing generality, we assume that the size of each ciphertext bit is $|X|$. In IndexBuid phase, the time complexity of PSRQ is $\mathcal{O}(n(k + r) + rm))$ and its corresponding storage overhead is $rm|X|$, thus the time complexity of PSRQ$^+$ is $\mathcal{O}(8n(k + r) + rm)$ and its corresponding storage overhead is $rm|X|$. In TokenGen phase, the time complexities of PSRQ and PSRQ$^+$ are both $\mathcal{O}(kb)$, and their corresponding storage overheads are both $kb|X|$. In Query phase, the total cost of set intersection and

[1]http://www.diag.uniroma1.it//challenge9/download.shtml

TABLE IV

COMPARISON OF PERFORMANCE ANALYSIS

| Scheme | Phase | Computational overhead | Storage overhead |
|---|---|---|---|
| PRSQ | IndexBuid | $\mathcal{O}(n(k + r) + rm)$ | $rm\|X\|$ |
| | TokenGen | $\mathcal{O}(kb)$ | $kb\|X\|$ |
| | Query | $\mathcal{O}(kbr + ((n - v)\epsilon + v)r)$ | $--$ |
| PRSQ$^+$ | IndexBuid | $\mathcal{O}(8n(k + r) + rm)$ | $rm\|X\|$ |
| | TokenGen | $\mathcal{O}(kb)$ | $kb\|X\|$ |
| | Query | $\mathcal{O}(kbr + ((n - v)\epsilon + v)r)$ | $--$ |
| PPBSKQ [24] | IndexBuid | $\mathcal{O}(nk + 2nw^2)$ | $2nw\|Y\|$ |
| | TokenGen | $\mathcal{O}(k + 2w^2)$ | $2w\|Y\|$ |
| | Query | $\mathcal{O}(2nw)$ | $--$ |
| PPTR [29] | IndexBuid | $\mathcal{O}(n(\eta + 8)^3)$ | $n\|Z\|$ |
| | TokenGen | $\mathcal{O}((\eta + 8)^3)$ | $2\|Z\|$ |
| | Query | $\mathcal{O}(2n(\eta + 8)^3)$ | $--$ |

TABLE V

COMPUTATIONAL OVERHEAD OF Query IN DIFFERENT DATASETS

| Scheme | $n =$ | USA Road Network[1] | Road Network of North America[1] | Gowalla dataset[2] |
|---|---|---|---|---|
| PSRQ | $1 \times 10^6$ | 0.0346s | 0.0368s | 0.0356s |
| | $2 \times 10^6$ | 0.0541s | 0.0549s | 0.0553s |
| | $3 \times 10^6$ | 0.0795s | 0.0803s | 0.0798s |
| PSRQ$^+$ | $1 \times 10^6$ | 0.0385s | 0.0397s | 0.0391s |
| | $2 \times 10^6$ | 0.0594s | 0.0597s | 0.0595s |
| | $3 \times 10^6$ | 0.0813s | 0.0821s | 0.0824s |
| PPBSKQ [24] | $1 \times 10^6$ | 2.304s | 2.315s | 2.309s |
| | $2 \times 10^6$ | 5.137s | 5.149s | 5.141s |
| | $3 \times 10^6$ | 7.784s | 7.789s | 7.792s |
| PPTR [29] | $1 \times 10^6$ | 16.575s | 16.472s | 16.311s |
| | $2 \times 10^6$ | 32.492s | 33.887s | 33.455s |
| | $3 \times 10^6$ | 50.215s | 49.534s | 50.744s |

union is $\mathcal{O}(((n - v)\epsilon + v)r)$, so the time complexities of PSRQ and PSRQ$^+$ are both $\mathcal{O}(rb + ((n - v)\epsilon + v)r)$.

PPBSKQ first inserts spatial data into Bloom filters to generate indexes, then encrypts the indexes by using ASPE, and finally calculates the inner product of encrypted indexes to obtain query results. We assume that the length of a Bloom filter is $w$, and the size of each ciphertext bit is $|Y|$. In IndexBuid phase, the time complexity is $\mathcal{O}(nk+2nw^2)$, and its corresponding storage overhead is $2nw|Y|$. In TokenGen phase, the time complexity is $\mathcal{O}(k + 2w^2)$, and its corresponding storage overhead is $2w|Y|$. In Query phase, the total time complexity is $\mathcal{O}(2nw)$.

PPTR first employs polynomial fitting technique to build indexes, and then uses randomizable matrix multiplication technology to encrypt them, in which the encryption keys are $(\eta + 8) \times (\eta + 8)$ random invertible matrices. We assume that the degree of polynomial fit is $\eta$ and the size of each ciphertext index is $|Z|$. In IndexBuid phase, the time complexity is $\mathcal{O}(n(\eta + 8)^3)$, and its corresponding storage overhead is $n|Z|$. In TokenGen phase, the time complexity is $\mathcal{O}(2(\eta + 8)^3)$, and its corresponding storage overhead is $2|Z|$. In Query phase, the time complexity is $\mathcal{O}(2n(\eta + 8)^3)$.

## *C. Experimental Evaluation*

We use the real spatial dataset (USA road network[1]) to conduct sufficient experiments to perform performance testing

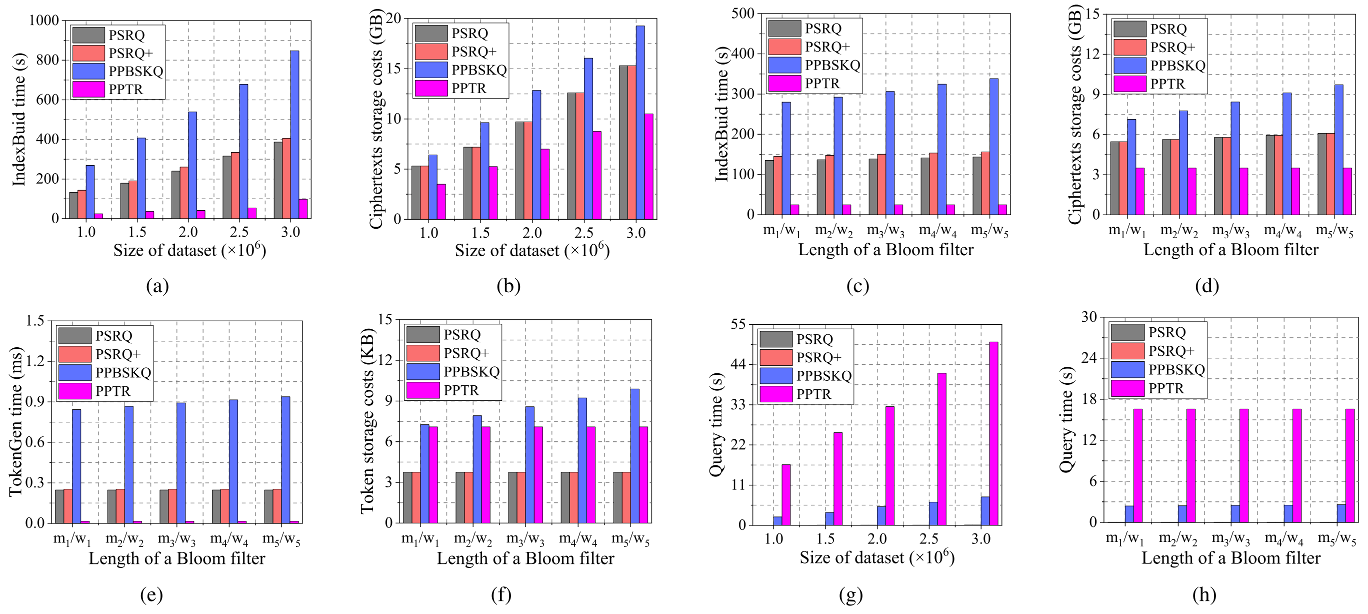


Fig. 9. (1) Overhead of IndexBuid: (a) (b) (c) (d); (2) Overhead of TokenGen: (e) (f); (3) Overhead of Query: (g) (h).

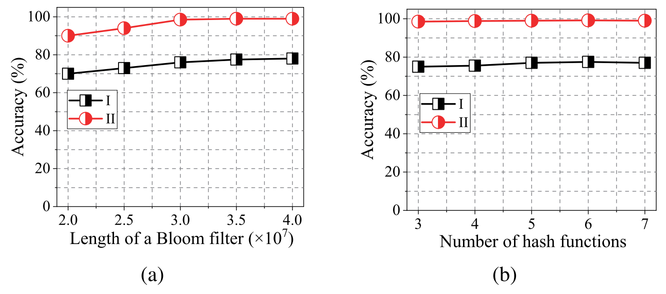


Fig. 10. Accuracy test by varying $m$ and $k$.

and detailed analysis on the IndexBuid, TokenGen and Query phases of the PSRQ, PSRQ$^+$, PPBSKQ [24], and PPTR [29].

Note that, since four schemes have false positives, we will test four schemes under the condition that the false positive is close to 0, i.e., PSRQ and PSRQ$^+$ is $r = 4, b = 60$ in TABLE III, PPBSKQ is $w = 200$ and PPTR is $\eta = 10$.

IndexBuid. We first test the computational and storage overheads of four schemes by varying the size of dataset $n$. The test results are shown in Fig. 9 (a) (b). Then, let $n = 1 \times 10^6$, we test the computational and storage overheads by varying the length of Bloom filter, i.e., for PSRQ and PSRQ$^+$, $m_1 = 3.5 \times 10^7, m_2 = 3.6 \times 10^7, m_3 = 3.7 \times 10^7, m_4 = 3.8 \times 10^7, m_5 = 3.9 \times 10^7$, and for PPBSKQ, $w_1 = 220, w_2 = 240, w_3 = 260, w_4 = 280, w_5 = 300$. Note that, PPTR is independent of the length of Bloom filter. The test results are shown in Fig. 9 (c) (d).

TokenGen. We test the computational and storage overheads by varying the length of Bloom filter, i.e., for PSRQ and PSRQ$^+$, $m_1 = 3.5 \times 10^7, m_2 = 3.6 \times 10^7, m_3 = 3.7 \times 10^7, m_4 = 3.8 \times 10^7, m_5 = 3.9 \times 10^7$, and for PPBSKQ, $w_1 = 220, w_2 = 240, w_3 = 260, w_4 = 280, w_5 = 300$. Note that, PPTR is independent of the length of Bloom filter. The test results are shown in Fig. 9 (e) (f).

Query. We first test the computational overheads of four schemes by varying the size of dataset $n$. The test results are shown in Fig. 9 (g). Then, let $n = 1 \times 10^6$, we test the computational overhead by varying the length of Bloom filter, i.e., the range of variation is the same as those in IndexBuid and TokenGen. The test results are shown in Fig. 9 (h).

PSRQ$^+$ is very efficient in the query process, which can provide millisecond level query service and its query efficiency is 100x higher than those of state-of-the-art solutions. To further prove the efficiency and practicality of PSRQ$^+$ in practice, we also conduct a large number of experimental tests for Query phase over different real spatial datasets, such as USA Road Network,[1] Road Network of North America[1] and Gowalla dataset.[2] Then, We set $r = 4, b = 60, w = 200$ and $\eta = 10$ to test the computational overhead of PSRQ, PSRQ$^+$, PPBSKQ [24], and PPTR [29] under different real spatial datasets. We can see from the results in TABLE V that the performance of PSRQ$^+$ is roughly the same under three different real spatial datasets, i.e., the performance of PSRQ will not be significantly affected by various datasets.

To further prove the accuracy of our schemes, let $n = 1 \times 10^6$, we change the length of the Bloom filter, where $k = 5$, and the number of hash functions, where $m = 3 \times 10^7$, to test the following schemes: (I) SHVE encrypts the traditional Bloom filter (i.e., $r = 1, b = 0$); (II) SHVE encrypts CSC-BF (i.e., our schemes). The results are shown in Fig. 10.

## VIII. Conclusion

In this paper, we first propose an efficient PrivacyPreserving Spatial Range Query (PSRQ) scheme by skillfully combining Geohash algorithm with CSC-BF and SHVE, which not only greatly reduces the computational cost of generating

[2]http://snap.stanford.edu/data/loc-Gowalla.html

token but also speeds up the query efficiency on large-scale dataset. Then, we design a CBF structure, which can confuse the inclusion relationship of traditional Bloom filter. Finally, we propose a more practical enhanced scheme PSRQ$^+$ by using CBF and converting the spatial location as a set containing 8 Geohash codes, which supports more query ranges and achieves adaptive security. In addition, we give formal security analysis to prove that PSRQ is secure against IND-CPA and PSRQ$^+$ achieves adaptive security, and conduct extensive experimental tests to demonstrate that our schemes using million-level dataset improve the query efficiency by 100x compared with previous schemes. As part of our future work, we will try to increase the query range diversity, not just limited to the range of a few precisions provided by Geohash algorithm.

## References

[1] J. Chen, K. He, Q. Yuan, M. Chen, R. Du, and Y. Xiang, "Blind filtering at third parties: An efficient privacy-preserving framework for location-based services," *IEEE Trans. Mobile Comput.*, vol. 17, no. 11, pp. 2524–2535, Nov. 2018.

[2] C. Yang, Z. Jia, and S. Li, "Privacy-preserving proximity detection framework for location-based services," in *Proc. Int. Conf. Netw. Netw. Appl. (NaNA)*, Oct. 2021, pp. 99–106.

[3] Q. Huang, J. Du, G. Yan, Y. Yang, and Q. Wei, "Privacy-preserving spatio-temporal keyword search for outsourced location-based services," *IEEE Trans. Services Comput.*, vol. 15, no. 6, pp. 3443–3456, Nov. 2022, doi: 10.1109/TSC.2021.3088131.

[4] R. Li and A. X. Liu, "Adaptively secure conjunctive query processing over encrypted data for cloud computing," in *Proc. IEEE 33rd Int. Conf. Data Eng. (ICDE)*, Apr. 2017, pp. 697–708.

[5] Q. Tong, Y. Miao, J. Weng, X. Liu, K. R. Choo, and R. H. Deng, "Verifiable fuzzy multi-keyword search over encrypted data with adaptive security," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 5, pp. 5386–5399, May 2023, doi: 10.1109/TKDE.2022.3152033.

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

[7] 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 Things J.*, vol. 9, no. 8, pp. 6184–6198, Apr. 2022, doi: 10.1109/JIOT.2021.3110300.

[8] 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, Sep. 1976.

[9] A. X. Liu and F. Chen, "Privacy preserving collaborative enforcement of firewall policies in virtual private networks," *IEEE Trans. Parallel Distrib. Syst.*, vol. 22, no. 5, pp. 887–895, May 2011.

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

[11] P. Wang and C. V. Ravishankar, "Secure and efficient range queries on outsourced databases using $\hat{R}$-trees," in *Proc. IEEE 29th Int. Conf. Data Eng. (ICDE)*, Apr. 2013, pp. 314–325.

[12] 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. (ICASSP)*, Jun. 2021, pp. 2670–2674.

[13] K. Lewi and D. J. Wu, "Order-revealing encryption: New constructions, applications, and lower bounds," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Oct. 2016, pp. 1167–1178.

[14] E. Shi, J. Bethencourt, T.-H.-H. Chan, D. Song, and A. Perrig, "Multi-dimensional range query over encrypted data," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2007, pp. 350–364.

[15] 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. 9th ACM Symp. Inf., Comput. Commun. Secur. (ASIA CCS)*, Jun. 2014, pp. 111–122.

[16] B. Wang, M. Li, H. Wang, and H. Li, "Circular range search on encrypted spatial data," in *Proc. IEEE 35th Int. Conf. Distrib. Comput. Syst. (CNS)*, Jun. 2015, pp. 794–795.

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

[18] H. Zhu, R. Lu, C. Huang, L. Chen, and H. Li, "An efficient privacy-preserving location-based services query scheme in outsourced cloud," *IEEE Trans. Veh. Technol.*, vol. 65, no. 9, pp. 7729–7739, Sep. 2016.

[19] D. Boneh, E.-J. Goh, and K. Nissim, "Evaluating 2-DNF formulas on ciphertexts," in *Proc. Theory Cryptogr. Conf. (TCC)*, 2005, pp. 325–341.

[20] Z. Zheng, Z. Cao, and J. Shen, "Practical and secure circular range search on private spatial data," in *Proc. IEEE 19th Int. Conf. Trust, Secur. Privacy Comput. Commun. (TrustCom)*, Dec. 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. Cryptograph. Techn. (EUROCRYPT)*, 2009, pp. 224–241.

[22] L. Li, R. Lu, and C. Huang, "EPLQ: Efficient privacy-preserving location-based query over outsourced encrypted data," *IEEE Internet Things J.*, vol. 3, no. 2, pp. 206–218, Apr. 2016.

[23] B. Wang, M. Li, and H. Wang, "Geometric range search on encrypted spatial data," *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 4, pp. 704–719, Apr. 2016.

[24] 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 35th Int. Conf. Data Eng. (ICDE)*, Apr. 2019, pp. 1046–1057.

[25] Y. Luo, S. Fu, D. Wang, M. Xu, and X. Jia, "Efficient and generalized geometric range search on encrypted spatial data in the cloud," in *Proc. IEEE/ACM 25th Int. Symp. Quality Service (IWQoS)*, Jun. 2017, pp. 1–10.

[26] X. Li, Y. Zhu, J. Wang, and J. Zhang, "Efficient and secure multi-dimensional geometric range query over encrypted data in cloud," *J. Parallel Distrib. Comput.*, vol. 131, pp. 44–54, Sep. 2019.

[27] X. Wang et al., "Search me in the dark: Privacy-preserving Boolean range query over encrypted spatial data," in *Proc. IEEE Conf. Comput. Commun. (INFOCOM)*, Jul. 2020, pp. 2253–2262.

[28] Y. Yang, Y. Miao, K. R. Choo, and R. H. Deng, "Lightweight privacy-preserving spatial keyword query over encrypted cloud data," in *Proc. IEEE 42nd Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jul. 2022, pp. 392–402.

[29] 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.

[30] R. Guo, B. Qin, Y. Wu, R. Liu, H. Chen, and C. Li, "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.

[31] 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.

[32] R. Gelda, K. Jagannathan, and G. Raina, "Forecasting supply in Voronoi regions for app-based taxi hailing services," in *Proc. 6th IEEE Int. Conf. Adv. Logistics Transp. (ICALT)*, Jul. 2017, pp. 47–52.

[33] R. Li et al., "Building fast and compact sketches for approximately multi-set multi-membership querying," in *Proc. Int. Conf. Manage. Data (SIGMOD)*, Jun. 2021, pp. 1077–1089.

[34] S. Lai et al., "Result pattern hiding searchable encryption for conjunctive queries," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Oct. 2018, pp. 745–762.

[35] W. Lin, K. Wang, Z. Zhang, and H. Chen, "Revisiting security risks of asymmetric scalar product preserving encryption and its variants," in *Proc. IEEE 37th Int. Conf. Distrib. Comput. Syst. (ICDCS)*, Jun. 2017, pp. 1116–1125.