TU Delft

Delft University of Technology

Document Version
Final published version

Citation (APA)
Li, M., Zhang, M., Gao, J., Lal, C., Conti, M., & Alazab, M. (2022). Repetitive, Oblivious, and Unlinkable SkNN Over Encrypted-and-Updated Data on Cloud. In C. Alcaraz, L. Chen, S. Li, & P. Samarati (Eds.), *Information and Communications Security - 24th International Conference, ICICS 2022, Proceedings* (pp. 261-280). (Lecture Notes in Computer Science; Vol. 13407 LNCS). Springer. https://doi.org/10.1007/978-3-031-15777-6_15

Important note
To cite this publication, please use the final published version (if applicable).
Please check the document version above.

Copyright
In case the licence states “Dutch Copyright Act (Article 25fa)”, this publication was made available Green Open Access via the TU Delft Institutional Repository pursuant to Dutch Copyright Act (Article 25fa, the Taverne amendment). This provision does not affect copyright ownership.
Unless copyright is transferred by contract or statute, it remains with the copyright holder.

Sharing and reuse
Other than for strictly personal use, it is not permitted to download, forward or distribute the text or part of it, without the consent of the author(s) and/or copyright holder(s), unless the work is under an open content license such as Creative Commons.

Takedown policy
Please contact us and provide details if you believe this document breaches copyrights.
We will remove access to the work immediately and investigate your claim.

*This work is downloaded from Delft University of Technology.*



# Repetitive, Oblivious, and Unlinkable S*k*NN Over Encrypted-and-Updated Data on Cloud

Meng Li[1(✉)], Mingwei Zhang[1], Jianbo Gao[1], Chhagan Lal[2], Mauro Conti[2,3], and Mamoun Alazab[4]

[1] Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education; School of Computer Science and Information Engineering, Hefei University of Technology; Anhui Province Key Laboratory of Industry Safety and Emergency Technology; and Intelligent Interconnected Systems Laboratory of Anhui Province (Hefei University of Technology), Hefei, China
mengli@hfut.edu.cn, {mwzhang,jianbogao}@mail.hfut.edu.cn

[2] Department of Intelligent Systems, CyberSecurity Group, Delft University of Technology, Delft, The Netherlands
c.lal@tudelft.nl

[3] Department of Mathematics and HIT Center, University of Padua, Padua, Italy
conti@math.unipd.it

[4] College of Engineering, IT and Environment, Charles Darwin University, Darwin City, Australia
alazab.m@ieee.org

**Abstract.** Location-Based Services (LBSs) depend on a Service Provider (SP) to store data owners' geospatial data and to process data users' queries. For example, a Yelp user queries the SP to retrieve the $k$ nearest Starbucks by submitting her/his current location. It is well-acknowledged that location privacy is vital to users and several prominent Secure $k$ Nearest Neighbor (S$k$NN) query processing schemes are proposed. We observe that no prior work addresses the requirement of *repetitive query* after index update and its privacy issue, i.e., how to match a data item from the cloud repetitively in an oblivious and unlinkable manner. Meanwhile, a malicious SP may skip some data items and recommend others due to unfair competition.

In this work, we formally define the repetitive query and its privacy objectives and present an Repetitive, Oblivious, and Unlinkable SkNN scheme ROU. Specifically, we design a multi-level structure to organize locations to further improve search efficiency. Second, we integrate data item identity into the framework of existing S$k$NN query processing. Data owners encrypt their data item identity and location information into a secure index, and data users encrypt a customized identity range of a previously retrieved data item and location information into a token. Next, the SP uses the token to query the secure index to find the specific data item via privacy-preserving range querying. We formally prove the privacy of ROU in the random oracle model. We build a prototype based on a server to evaluate the performance with a real-world dataset.

© Springer Nature Switzerland AG 2022
C. Alcaraz et al. (Eds.): ICICS 2022, LNCS 13407, pp. 261–280, 2022.
https://doi.org/10.1007/978-3-031-15777-6_15

Experimental results show that ROU is efficient and practical in terms of computational cost, communication overhead, and result verification.

**Keywords:** Cloud computing · S$k$NN · Repetitive query · Privacy

# 1 Introduction

## 1.1 Background

Smartphones are now equipped with a Global Positioning System (GPS) module and various applications that support location-based service (LBS) [1–4]. It works by sending a data user's current location query to a Service Provider (SP). The SP matches the query with data items from data owners and retrieves corresponding results to the data user. For instance, Google Maps enable data users to find Starbucks, bars, and restaurants near their current location.

While LBSs provide practical benefits, the privacy concerns rooted in location revelation and untrusted SP [5,6] are a major hindrance towards the broad adoption of LBSs. First, the submitted locations may include data users' sensitive locations. Second, locations are tightly correlated to human activities, such as visiting a cancer hospital and meeting a friend in a hotel. Besides, there are many reports on the data leakage incidents caused by cyber attacks, hardware malfunction, or misoperation for the past decade [7,8]. Therefore, it is highly important to protect data stored on SPs. To protect data stored on SPs, Secure $k$ Nearest Neighbor (S$k$NN) query processing has been proposed [9–12].

## 1.2 Motivations

**Motivation I (Blurry Memory):** Our motivation arises from real-world applications. For example, say a data user Bob submitted a S4NN query (*location X*, "*pizza shop*", 4) to the SP and obtained 4 results, namely Papa Johns, Mr. Pizza, Domino's Pizza, and Big Pizza. Among the results he went to, he was quite satisfied with the Domino's Pizza. Days later, when Bob is near the same query location and wants to dine at the previous Domino's Pizza. Unfortunately, Bob's memory of this shop somehow blurs and he submits a *repetitive query* to the SP to find the preferred pizza shop. **Motivation II (Index Update):** Following the example above, even if Bob remembers the name of the shop, the SP may happen to update the index tree as depicted in Fig. 1 such that the preferred location will not be included in the query results. In the previous query, the SP found the Domino's Pizza at $di_5$, i.e., data item 5, among the first 4 matched data items $\{di_2, di_4, di_5, di_6\}$. After the index update, $di_5$ will not be returned to Bob who submits the same S4NN query because the data items are reordered and $di_5$ is not among the top 4 matched data items. **Motivation III (Malicious SP):** The SP could be malicious in the sense that it has a secret agreement with some data owners to deliberately order data items, which is similar to unfair ranking where some search engines treat websites unfairly [17]. In this case, the user-preferred data item, which is not "favored" by the SP,

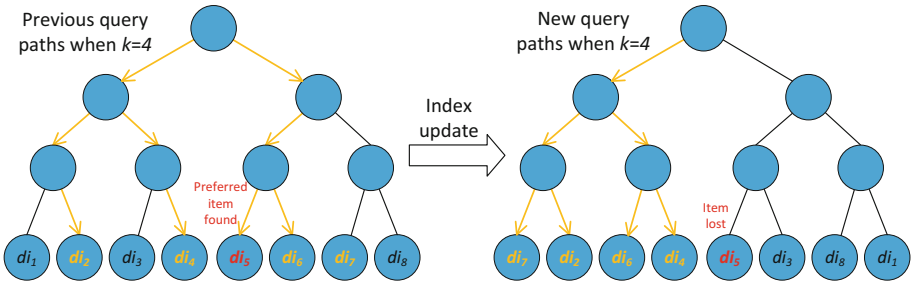



**Fig. 1.** Query paths before/after index update. (Data items marked in yellow are matched data items. The data item marked in red is matched and user-preferred data item. After index update, the preferred data item $di_5$ is moved back on the leaf level such that the search paths change. The user will not receive such an item by using the same S4NN query because the SP finds 4 matched data items before $di_5$.) (Color figure online)

may be put way back in the data item queue. **Motivation IV (Improving Efficiency):** There are many methods of processing locations in S$k$NN, such as Voronoi diagram [9], Paillier cryptosystem [10,18], and projection functions [11]. Improving search efficiency is always an ongoing goal.

Based on the first three motivations, no previous studies [9–16] have considered the requirement of repetitive query and its potential privacy problems, which lead to the following new requirements for S$k$NN.

- **Repetitive query after index update**: A data user queries the same data item that is returned in the previous query especially after the SP has updated the index.
- **Obliviousness**: We need to prevent the SP from knowing the data user's requirement to retrieve a previously matched data item.
- **Unlinkability**: Prevent the service provider from knowing that the specific data item has been previously matched to the data user.
- **Exclusiveness**: Prevent the SP from abandoning the preferred data item in case the SP gives priority to other data items.

### 1.3 Possible Solutions and Technical Challenges

A simple way of finding a preferred data item in an oblivious and unlinkable manner is to query all the locations with the same type as the preferred data item. However, this brings too many computational costs in result searching and communication overhead. Assume that the data user needs the data item $di$ with a sequence number $*$. Intuitively, there are three approaches to finding $*$ in SP's index tree: (1) *Start from* $*$ and continue to find the other $k-1$ data items. (2) *End at* $*$ and return the obtained data items. (3) *Randomly choose* $r$ data items before $*$ and find $k-r-1$ data items after $*$. These three approaches require special treatment on locating $*$ which makes it difficult for the SP not to notice this difference. This first one may traverse the whole index and the

second one may return all the matched data items if $di_*$ is in the last leaf node. The last one is faced with an uncertain choice of $r$. Therefore, the **technical challenge I** is solving the contradiction between locating the preferred data item and treating all data items equally. To enable repetitive query, we can introduce an identity to each data item. Before uploading the index to the SP, the data owner has to append the identity to the location for each data item. When the data user queries the SP about a previous data item, the SP uses the identity as the extra query condition. While matching the previous data item precisely, this approach, however, excludes other location-matched data items which may expose the data user's requirement of repetitive matching. To match other data items, we cannot use the identity directly. Therefore, the **technical challenge II** lies in the contradiction between using a preferred identity and matching other data items.

To address the above challenges, we propose ROU: a Repetitive, Oblivious, and Unlinkable query processing scheme. Specifically, we first divide the location map into a $l$-leveled pyramid and each level consists of a number of grids. For each level, we use the similar space encoding technique to save computational cost for both data users and the SP. The data owner (data user) encodes the data items' locations (current location) and obtains a set of leveled location codes. At the $l$th level, we assign an identity to each data item. The data owner computes a prefix family of the data item identity and integrates it with location codes at the $l$-th level. Next, the data owner inserts the integrated codes into an Indistinguishable Bloom Filter (IBF) as a secure index. The data user who is about to submit a repetitive query generates a customized identity range to compute a minimum set of prefixes. Next, the data user also integrates the prefixes with the location codes similarly and computes a query token. Finally, the SP searches the secure index by querying the token on it and returns matched data items to the data user. Our contributions are summarized as follows.

- To the best of our knowledge, we are the first to focus on the repetitive query in S$k$NN and we propose a repetitive, oblivious, and unlinkable query processing scheme.
- We achieve the three above-mentioned new requirements via customized identity transformation and privacy-preserving range querying. We design a multi-level structure to encode locations to accelerate the search efficiency.
- We formally define privacy and then prove it in the random oracle model. We build a prototype of ROU based on a server and a real-world dataset. Experimental results demonstrate its efficiency and practicability.

### 1.4 Paper Organization

The remaining of this paper is organized as follows. We discuss related work in Sect. 2. We elaborate on the system model, threat model, and design objectives in Sect. 3. In Sect. 4, we introduce the proposed space encoding. In Sect. 5, we present the ROU scheme. We formally analyze the privacy of the ROU in Sect. 6. We implement the ROU scheme and analyze its performance in Sect. 7. Lastly, we draw conclusions in Sect. 8.

## 2 Related Work

### 2.1 S$k$NN

Yao et al. [9] proposed SNN methods by asking the SP, given only an encrypted query point $E(p)$ and an encrypted database $E(D)$, to return a corresponding (encrypted) partition $E(G)$ satisfying that $E(G)$ contains SNN query answer. They name their method the secure Voronoi diagram (SVD) method that is based on special partitions over $D$ and the Voronoi diagram of $D$. They partition the database $D$ into small groups and then store the encrypted groups on the SP. Instead of returning the whole encrypted database, the SP retrieves one encrypted group for any SNN query. The SVD method does not require any new encryption schemes, but only depends on any standard encryption scheme $E$ (e.g., RSA and AES) which means its security is the same as $E$.

Elmehdwi et al. [10] proposed an $k$-nearest neighbor search protocol based on two non-colluding semi-honest SPs that preserves both the data privacy and query privacy. They first design a basic protocol and show why it is not secure and present a fully secure $k$NN protocol. The basic protocol allows the data user to retrieve $k$ records that are closest to his query by using Paillier cryptosystem [18] and secure squared Euclidean distance. The advanced protocol, however, utilizes secure bit-decomposition, secure minimum out of $n$ numbers, secure bit-OR to avoid exposing the data access patterns in the basic protocol.

Lei et al. [11] proposed a secure and efficient query processing protocol SecEQP. They leveraged some primitive projection functions to convert the neighbor regions of a given location. Given the codes of two converted locations, the service provider computes the proximity of the two locations by judging whether the two codes are the same. This is an improvement over their previous work [14] since the two-dimensional location data is projected to high-dimensional data which expands the location space to make the converted location more secure. The data owner further embeds the codes into a similar IBFTree in order to build a secure index. The data user computes similar trapdoors by a keyed hash message authentication code. The final secure query processing is the same as [14].

### 2.2 Privacy-Preserving Range Querying

Li et al. [13] presented the first range query processing protocol which achieved index indistinguishability under the indistinguishability against chosen keyword attack (IND-CKA). A data owner converts each data item $dt_i$ by prefix encoding [19] and organizes each prefix family of encoded item $F(di_i)$ into a PBTree. Then the data owner makes the PBtree privacy-preserving by a keyed hash message authentication code $\mathsf{HMAC}$ and Bloom filters. For each prefix $pr_i$, the data owner computes several hashes $\mathsf{HMAC}(K_j, pr_i)$ and inserts a randomized version $\mathsf{HMAC}(r, \mathsf{HMAC}(K_j, pr_i))$ into a Bloom filter. Each $r$ corresponds to a node and each node relates to a prefix family, i.e., data item. Next, a data user converts a range into a minimum set of prefixes and computes several hashes

$\mathsf{HMAC}(K_j, pr_i)$ for each $pr_i$ as a trapdoor. The service provider searches in the PBtree to find a match by using the trapdoor.

Li et al. [14] concerned processing conjunctive queries including keyword conditions and range conditions in a privacy-preserving way and presented a privacy-preserving conjunctive query processing protocol supporting adaptive security, efficient query processing, and scalable index size at the same time. Specifically, they adopt prefix encoding as in their earlier work [13] and design an indistinguishable Bloom filter (IBF), i.e., twin Bloom filter to replace the previous structure. A pseudo-random hash function $H$ to determine a cell location $H(h_{k+1}(h_j(pr_i)) \oplus r)$, i.e., which twin cell stores '1'. Instead of building a PBTree, they construct an IBTree as the secure index.

Different from the previous works, ROU scheme can support the three new features in S$k$NN, namely repetitive, oblivious, and unlinkable. The novelty of ROU is in realizing the function of repetitive query by mixing customized identity range query with existing S$k$NN query without sacrificing privacy.

## 3 Problem Formulation

Before we dive into the details of ROU, we elaborate on its system model, threat model, and design objectives. Specifically, we formally define the repetitive query and its privacy objectives.

### 3.1 System Model

The system model, as drawn in Fig. 2, consists of a data owner $O$, a data user $U$, and SP. We define $\mathcal{DI} = \{di_1, di_2, \cdots, di_n\}$ as the set of $n$ data items. A location $loc$ as a pair of coordinates.

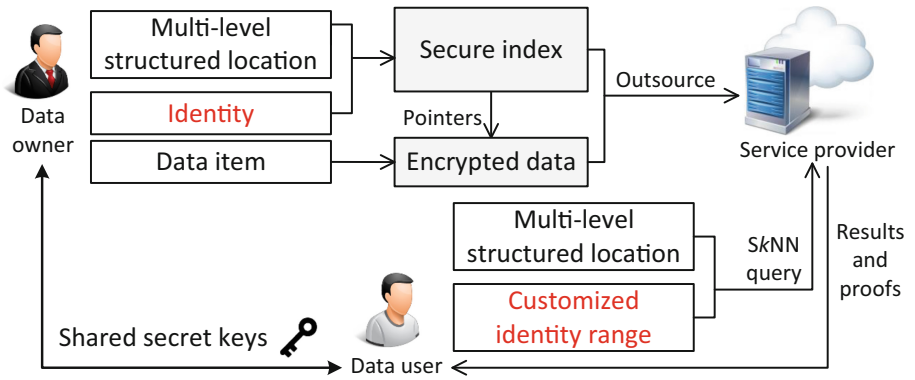



**Fig. 2.** ROU system model.

**Data Owner**: A data owner has some data items to be shared with data users. Each data item has type, location, and identity. The data owner extracts the

information of each data item and calculates a secure index by using secret keys. Next, he encrypts his data item by using another secret key and a standard encryption algorithm. Each secure index has a pointer to link to the ciphertext. Finally, the data owner uploads the index and the ciphertext to the SP. The secret keys are shared with data users. We assume that ROU has only one data owner for simplicity, but also supports the multi-owner setting.

**Data User**: A data user generates a query token by using the type, current location, and shared secret keys, and an identity range. If the data user does not have a specific preference on a data item, the identity range is set by default. Otherwise, the identity range is computed based on the identity of the preferred data item. Next, the data user submits the query token to the SP, which retrieves corresponding results and proofs to the data user. The data user decrypts and verifies the received results. We formally define the repetitive query as follows.

**Definition 1** (Repetitive Query)**.** *A repetitive query is a single location-time predicate or a combination of location-time predicates linked by the Boolean operators [20]. Let* $Q = (pid, \mathcal{T}(t, loc, id, R))$ *be a SkNN query submitted by a data user and it is a pair of pseudo-identity pid and a query token* $\mathcal{T}$*.* $\mathcal{T}$ *is composed of type* $t$*, location* $loc$*, identity of a previously matched data item* $id$*, and an identity range* $R$*. Let* $Q_i = (pid_i, \mathcal{T}_i(t_i, loc_i, id_i, R_i))$ *be the* $i$*th query of data owner* $O$*. A repetitive query event, denoted by* ReQuery*, is expressed as* $(pid_i \neq pid_j) \wedge (\mathcal{T}_i \neq \mathcal{T}_j) \wedge (t_i = t_j) \wedge (loc_i = loc_j) \wedge (id_i = id_j) \wedge (R_i \neq R_j) \wedge (time.j > time.i)$ *for two queries* $Q_i$ *and* $Q_j$*.*

**SP**: The SP helps the data owner to authorize the query service to a set of data users. The SP stores the secure indexes and ciphertexts uploaded from the data owners. It responds to data users' query tokens by searching over the secure indexes and returning corresponding results and proofs to data users.

### 3.2 Threat Model

The threats mainly arise from the behaviors of the internal entities, including the semi-honest (honest-but-curious) data owner and data users. This assumption is proposed by [21] and has been well acknowledged by existing work [11,13,14,22–24]. The SP is malicious [12,15]. Although it acts as a bridge between the data owner and data users to offer query services, it may also behave maliciously, i.e., it ignores some data items when searching the index in its database.

### 3.3 Design Objectives

There are four design objectives in this work: functionality, privacy, security, and efficiency.

**Functionality**, i.e., repetitive query after index update. ROU allows data users to query a previously matched data item even if the SP has updated the index.

**Privacy**. (1) Data/Index/Token Privacy. From the encrypted data item, index, and token, the adversary cannot learn any useful information about the data, data item's location, query location, and type [25–29]. (2) Obliviousness. ROU prevents the SP from knowing that the data user submitted a repetitive query. (3) Unlinkability. ROU prevents the SP from knowing that the data item referred to in the repetitive query was a retrieved data item of the data user. We define two experiments $\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi}$ and $\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi}$, based on a Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$ and the ROU scheme $\Pi = (\mathsf{Setup}, \mathsf{Index}, \mathsf{Token}, \mathsf{Query})$, and a function $\mathsf{S}$ computing the minimum set of prefixes. The formal definitions are as follows.

**The Adversarial Obliviousness Experiment** $\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi}$**:**

1. $\mathcal{A}$ is given the size $m$ of IBF and number of pseudo-random hash functions $p$, and outputs a pair of quintuples $q_0 = (t_0, loc_0, id'_0, R_0)$, $q_1 = (t_1, loc_1, id'_1, R_1)$ satisfying $t_0 = t_1$, $loc_0 = loc_1$, $id'_0 = 0$, $id'_1 \in \{1, n\}$, and $\mid \mathsf{S}(R_0)\mid = \mid\mathsf{S}(R_1)\mid$.
2. Secret keys are generated by using $\mathsf{Setup}$, and a uniform bit $b \in \{0, 1\}$ is chosen. A query token $\mathcal{T}_b \leftarrow \mathsf{Token}(q_b)$ is computed and given to $\mathcal{A}$. We refer to $\mathcal{T}_b$ as the challenge token.
3. $\mathcal{A}$ outputs a bit $b'$.
4. The output of the experiment is defined to be 1, i.e., $\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi} = 1$ and $\mathcal{A}$ succeeds, if $b' = b$, and 0 otherwise.

**The Adversarial Unlinkability Experiment** $\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi}$**:**

1. $\mathcal{A}$ is given the size $m$ of IBF and number of pseudo-random hash functions $p$, and outputs a pair of quintuples $q_0 = (t_0, loc_0, id'_0, R_0)$, $q_1 = (t_1, loc_1, id'_1, R_1)$ satisfying $t_0 = t_1$, $loc_0 = loc_1$, $di_{id'_1} \in \mathsf{Query}(q_0, \mathcal{I})$, and $\mid \mathsf{S}(R_0) \mid = \mid \mathsf{S}(R_1) \mid$, where $\mathcal{I}$ is the index tree.
2. Secret keys are generated by using $\mathsf{Setup}$, and a uniform bit $b \in \{0, 1\}$ is chosen. A query token $\mathcal{T}_b \leftarrow \mathsf{Token}(q_b)$ is computed and given to $\mathcal{A}$. We refer to $\mathcal{T}_b$ as the challenge token.
3. $\mathcal{A}$ outputs a bit $b'$.
4. The output of the experiment is defined to be 1, i.e., $\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi} = 1$ and $\mathcal{A}$ succeeds, if $b' = b$, and 0 otherwise.

**Definition 2** (Obliviousness)**.** *Given a repetitive query event* ReQuery*, the SkNN scheme* $\Pi$ *is oblivious if for every* $\mathcal{A}$*, it holds that* $\Pr[\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi} = 1] = \frac{1}{2}$*. In other words, it is trivial for* $\mathcal{A}$ *to succeed with probability* $1/2$ *by outputting a random guess. Obliviousness requires that it is impossible for any* $\mathcal{A}$ *to do better.*

**Definition 3** (Unlinkability)**.** *Given a repetitive query event* ReQuery*, the SkNN scheme* $\Pi$ *is unlikable if for every* $\mathcal{A}$*, it holds that* $\Pr[\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi} = 1] = \frac{1}{2}$.

**Security**, i.e., exclusiveness. ROU prevents the SP from abandoning the preferred data item in case the SP gives priority to some locations. In other words, the data users can verify the query results that should include the preferred data item.

**Efficiency**. ROU should satisfy two types of efficiency requirements. (1) Low computational cost: the data owner/data user/SP spends a reasonable amount of time on computing index, token, and searching. (2) Low query latency. The data user can get the result within a reasonable amount of time. (3) Low communication overhead. It requires an acceptable amount of transmitted messages between the data owner, data users, and the SP.

## 4 The Proposed Space Encoding

The proposed space encoding technique is constructed on a multi-level structure to process data items. As shown in Fig. 3, there are four levels in the pyramid-like structure, i.e., $l_1$, $l_2$, $l_3$, and $l_4$. All the levels refer to the whole service area, but they are divided based on different granularity. From the second level $L_2$, the area is divided into more than one grid. There are 4, 16, and 64 grids in $L_2$, $L_3$, $L_4$, respectively. Each level encodes its grids from the number 1 prefixed with the level number such that each grid has a unique number on each level.

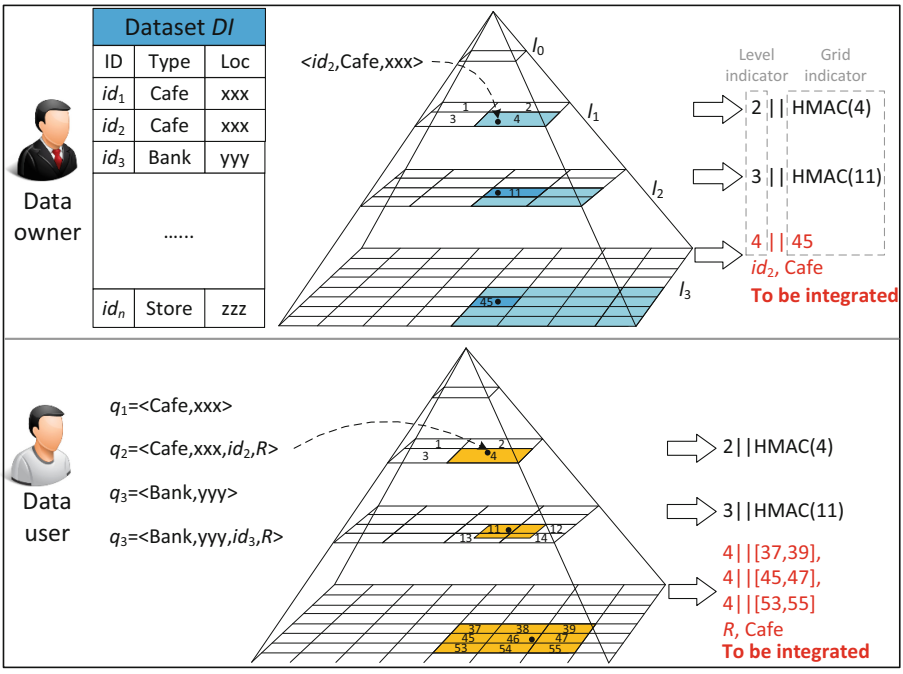



**Fig. 3.** Space encoding.

For the data item $< id_2$, Cafe, xxx>, the data owner encodes xxx from $L_2$ to $L_4$ to obtain three strings: "2||HMAC(4)", "3||HMAC(11)", and "4||HMAC(45)". Here, the real numbers before || stand for different levels and HMAC is a keyed

hash message authentication code. At the last level $L_4$, the data owner integrates its identity $id_2$ with "4||$\mathsf{HMAC}(45)$" as a foundation for repetitive query, which we will provide details in Sect. 5.2.

For the query $q_2$=<Cafe, xxx$id_2, R$ >, the data user encodes her current location from $L_1$ to $L_3$ similarly. When reaching $L_3$, the data user computes a bigger grid that covers the current grid and obtain three grid number ranges, i.e., "[37,39]", "[45,47]", and "[53,55]". The size of the bigger grid is flexible and it is determined according to the data users. Next, the data user will also integrate the identity $id_2$ of the previously matched data item < $id_2$, Cafe, xxx> with "4|| [37, 39]", "4|| [45, 47]", and "4|| [53, 55]" similarly. The $R$ is an identity range that covers $id_2$ that we will provide details in Sect. 5.3.

## 5 The Proposed Scheme ROU

### 5.1 Overview

As depicted in Fig. 4, we use the level-based space encoding to obtain location codes. We adopt the privacy-preserving range querying to generate identity prefixes for data items. Further, we integrate the repetitive query problem with the location querying problem by mixing the location codes and identity prefixes. Lastly, we leverage IBFs to build secure indexes and achieve S$k$NN querying via membership checking. The data users decrypt and verify the received results.

For each data item, $O$ converts its location into a set of leveled location codes $\mathcal{LC}$ from level 2 to level $l$ and computes a concatenated prefix family $\mathcal{PF}$ based on $l$, grid number, identity, and type on level $l$. Next, $O$ inserts $\mathcal{LC}$ and $\mathcal{PF}$ into an IBF as a leaf node and encrypts the data item using symmetric encryption. When processing all data items, $O$ build an index tree from the bottom to up, and submits the index tree and corresponding ciphertexts to SP. A data user $U$ computes $\mathcal{LC}$ similarly and computes a concatenated minimum set of prefixes $\mathcal{MP}$ based on $l$, grid range, identity range, and type on level $l$. Next, $U$ computes a query token $qt$ based on $\mathcal{LC}$ and $\mathcal{MF}$ and submits it to SP. The SP searches the index tree by using the token and returns matched results and proofs to the $U$. Finally, $U$ decrypts and verifies the results.

### 5.2 Index Building

A data owner $O$ is holding a set of data items $\mathcal{DI}$. $di_i =< id_i, t_i, loc_i >$. We use $di_i$ as an example to show how to build an IBF in a leaf node. For each $di_i$, $O$ chooses a secret key $K_0$, converts $di_i$'s location into a set of grid numbers $\{g_{i2}, \cdots, g_{il}\}$ and encodes them into a set of leveled location codes:

$$\begin{aligned}\mathcal{LC}_i = \{lc_{i2}, lc_{i3}, \cdots, lc_{il}\} = \{2 \,||\, \mathsf{HMAC}_{K_0}(g_{i2}), \cdots, \\ l-1 \,||\, \mathsf{HMAC}_{K_0}(g_{il-1}), l \,||\, g_{il}\}.\end{aligned} \tag{1}$$

For the first $l-2$ levels, $O$ processes $di_i$'s location codes $\{lc_{i2}, lc_{i3}, \cdots, lc_{il-1}\}$ as follows. Given $p+1$ secret keys $K_1, K_2, \cdots, K_p, K_{p+1}$, $p$ pseudo-random hash

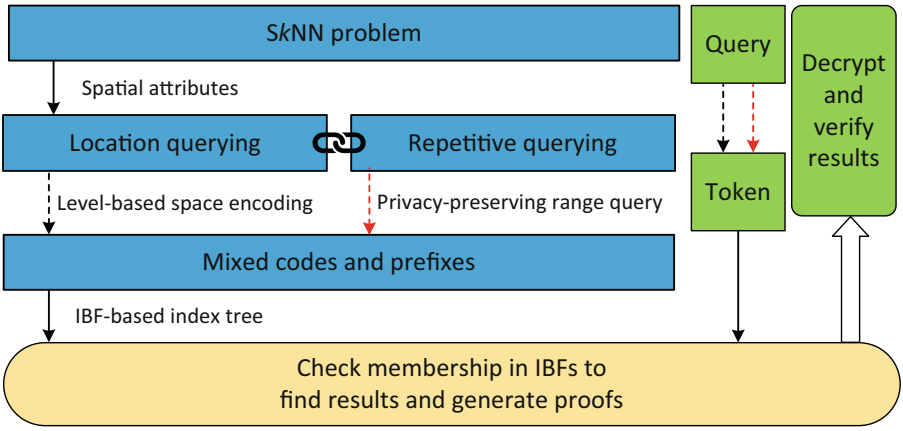



**Fig. 4.** ROU scheme overview.

functions $h_1, h_2, \cdots, h_p$ where $h_i = \mathsf{HMAC}_{K_i}(\cdot)$, and another hash function $H(.) = \mathsf{SHA256}(.)\%2$, $O$ creates an indistinguishable Bloom filter $IBF_O$ and embeds each location code $lc_{iu}$ and a randomly chosen number $r_i$ into $IBF_i$ by setting for all $u \in [2, l-1]$ and $v \in [1, p]$:

$$IBF_i[H(h_{K_{p+1}}(h_v(lc_{iu})) \oplus r_i)][h_v(lc_{iu})] = 1, \tag{2}$$

$$IBF_i[1 - H(h_{K_{p+1}}(h_v(lc_{iu})) \oplus r_i)][h_v(lc_{iu})] = 0. \tag{3}$$

For the $l$th level, $O$ computes a prefix family $\mathcal{PF}_{i1}$ of $g_{il}$ by using prefix encoding [13] and a prefix family $\mathcal{PF}_{i2}$ of $id_i$'s identity $id_i$. Then, $O$ mixes $\mathcal{PF}_{i1}$ with $\mathcal{PF}_{i2}$ by concatenating their prefixes to obtain a mixed code set $\mathcal{MC}_i$. Further, $O$ prefixes each mixed code with the level number and the type (converted into a real number). In this way, we lay a foundation for the data user to meet the requirement of repetitive query. Next, $O$ inserts each code $mc_u$ in $\mathcal{MC}_i$ into $IBF_i$ by setting for all $u \in [1, \mid \mathcal{MC}_i \mid]$ and $v \in [1, p]$:

$$IBF_i[H(h_{K_{p+1}}(h_v(mc_u)) \oplus r_i)][h_v(mc_u)] = 1, \tag{4}$$

$$IBF_i[1 - H(h_{K_{p+1}}(h_v(mc_u)) \oplus r_i)][h_v(mc_u)] = 0. \tag{5}$$

When processing all data items, $O$ obtains $n$ IBFs and builds an index tree from the bottom to up. $O$ sorts the $n$ IBFs in a random order and organize them into a binary tree structure to achieve sublinear search time [11]. An index tree $\mathcal{I}$ is built as follows. Assume that $IBF_1$ is the father IBF of two children IBFs: $IBF_2$ (left child) and $IBF_3$ (right child), then for each $i \in [1, m]$, the value of $IBF_1$'s $i$th twin is the logical OR of $IBF_2$'s $i$th twin and $IBF_3$'s $i$th twin.

$$\begin{aligned} &IBF_1[H(h_{K_{p+1}}(i) \oplus r_1)][i] = \\ &IBF_2[H(h_{K_{p+1}}(i) \oplus r_2)][i] \vee IBF_3[H(h_{K_{p+1}}(i) \oplus r_3)][i]. \end{aligned} \tag{6}$$

$O$ encrypts the $n$ data items by using AES encryption and a symmetric key $sk$ to obtain ciphertexts $\mathcal{CT} = \{ct_1, ct_2, \cdots, ct_n\}$ and computes a root hash value $RT$ of IBFs from the hash value $HV$ of all the tree nodes based on the Merkle tree method [30]. Finally, $O$ submits to the SP index tree $\mathcal{I}$, a set of random numbers, $\mathcal{CT}$, and $RT$.

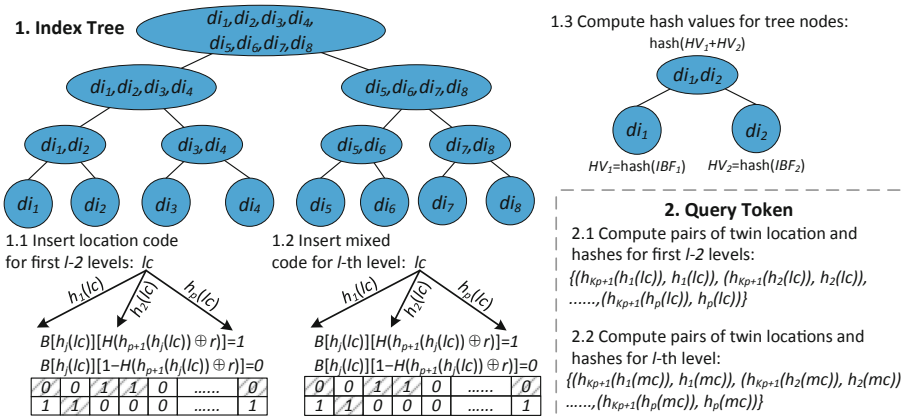



**Fig. 5.** Index tree and query token.

### 5.3 Token Generation

A data user $U$ is standing at location $loc$ and expecting to find the data item $id_i$. $U$ converts $loc$ into a set of leveled location codes:

$$\begin{aligned} \mathcal{LC} = \{lc_2, \cdots, lc_l\} = \{2 \mid\mid \mathsf{HMAC}_{K_0}(g_2), \cdots, \\ l-1 \mid\mid \mathsf{HMAC}_{K_0}(g_{l-1}), l \mid\mid \mathsf{Exp}(g_l)\}, \end{aligned} \tag{7}$$

where $\mathsf{Exp}(g_l)$ expands current grid to a bigger area which consists of the nearest nine grids as shown in Fig. 3.

For each location code $lc_u, 2 \leq u \leq l-1$, $U$ computes $p$ hashes $h_j(lc_u), 1 \leq j \leq p$. For each $h_j(lc_u), 1 \leq j \leq p$, $U$ computes $h_{K_{p+1}}(h_j(lc_u))$. The subtoken for $lc_u$ is a $p$-pair of twin locations and hashes: $\{(h_{K_{p+1}}(h_1(lc_u)), h_1(lc_u)), \cdots, (h_{K_{p+1}}(h_p(lc_u)), h_p(lc_u))\}$. Then, $O$ obtains a $((l-2)\times p)$-pair of twin locations and hashes. We denote the set by $\mathcal{T}_1$, i.e., the first part of the $\mathcal{T}$.

For the $l$th level, $U$ computes a minimum set of prefixes $\mathcal{M}_1$ for $\mathsf{Exp}(g_{il})$ and a minimum set of prefixes $\mathcal{M}_2$ for $R_i(id_i)$. Here, we require that $R_i(id_i) =$

$$\begin{cases} [id_i, id_i+1] \vee [id_i+2, id_i+3] \vee \cdots \vee \\ [id_i+2 \mid \mathsf{S}(1,n) \mid -2, id_i+2 \mid \mathsf{S}(1,n) \mid -1], \ if\ id_i\%2 = 0 \\ [id_i-1, id_i] \vee [id_i+1, id_i+2] \vee \cdots \vee \\ [id_i+2 \mid \mathsf{S}(1,n) \mid -3, id_i+2 \mid \mathsf{S}(1,n) \mid -2], \ otherwise \end{cases}$$

By doing so, we have that $\mid \mathcal{M}_2 \mid = \mid \mathsf{S}(id_1, id_n) \mid$. $U$ mixes $\mathcal{M}_1$ with $\mathcal{M}_2$ by concatenating their prefixes to obtain a mixed code set $\mathcal{MC}$. Further, $U$ prefixes each mixed code with the level number and the type. We denote the set by $\mathcal{T}_2$, i.e., the second part of the $\mathcal{T}$. Finally, $U$ submits the query token $\mathcal{T} = (\mathcal{T}_1, \mathcal{T}_2)$ to the SP. We draw the process of tree construction and token generation in Fig. 5.

### 5.4 Query Processing

After receiving $\mathcal{T}$, the SP searches $\mathcal{I}$ from up to the bottom to find leaf nodes that match $\mathcal{T}$. Specifically, the SP proceeds in two stpes. (1) SP performs query processing by checking whether $IBF[H(h_{K_{p+1}}(lc)) \oplus r)][h_j(lc)] = 1$ for at least one $j \in [1, p]$ and $(H(h_{K_{p+1}}(lc)), h_j(lc))$ is one pair in $\mathcal{T}_1$. If this match continues until the leaf level, it means there is at least one data item matches the query on the first $l - 2$ levels. (2) At a leaf node, the SP performs the similar query processing by using $\mathcal{T}_2$. If there is a match, the SP continues to search other matched leaf nodes. Finally, the SP returns the ciphertexts of match lead nodes and the proofs (IBFs of the branch-but-unmatched nodes) to the $U$.

### 5.5 Result Verification

$O$ decrypts the ciphertexts and checks whether the returned data items include the preferred data item. Next, $O$ verifies that her query does not match the IBFs in the proofs. $O$ also recomputes the value of the root from bottom to up by using the leaf IBFs and proofs. If the computed value equals to $RT$, $O$ is convinced that that results are not tampered with.

## 6 Privacy Analysis

### 6.1 Data/Index/Token Privacy

**Theorem 1.** *ROU is adaptive IND-CKA $(\mathcal{L}_1, \mathcal{L}_2)$-secure in the random oracle model, achieving data/index/token privacy.*

Due to the space limitation, please refer to our technical report for the detailed proofs.

### 6.2 Obliviousness

In the adversarial obliviousness experiment $\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi}$, a challenge query token $\mathcal{T}_b$ is returned to the adversary $\mathcal{A}$. Specifically, $\mathcal{T}_b$ consists of two parts $\mathcal{T}_{b1}$, $\mathcal{T}_{b2}$. Given that $loc_0 = loc_1$, we have $\mathcal{T}_{01} = \mathcal{T}_{11}$. For the second part, we require that $R(id'_1)$ is a customized range satisfying $\mid \mathsf{S}(R(id_1)) \mid = \mathsf{S}([id_1, id_n])$. By doing so, we have $\mid \mathcal{T}_{02} \mid = \mid \mathcal{T}_{12} \mid$ and they are indistinguishable for using secret keys and the one-way hash functions. Therefore, $\mathcal{T}_0$ and $\mathcal{T}_1$ are indistinguishable, i.e., $\Pr[\mathsf{PrivK}^{\mathsf{obl}}_{\mathcal{A},\Pi} = 1] = \frac{1}{2}$.□

### 6.3 Unlinkability

In the adversarial unlinkability experiment $\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi}$, a challenge query token $\mathcal{T}_b$ is returned to the $\mathcal{A}$. Similarly, we have $\mathcal{T}_{01} = \mathcal{T}_{11}$. Although $di_{id'_1} \in \mathsf{Query}(q_0, \mathcal{I})$, i.e., the data item $di_{id'_1}$ belongs to the previously received data items, we have also randomized $\mathcal{T}_{12}$ to make it indistinguishable from $\mathcal{T}_{02}$. Therefore, we have $\Pr[\mathsf{PrivK}^{\mathsf{unl}}_{\mathcal{A},\Pi} = 1] = \frac{1}{2}$.□

### 6.4 Exclusiveness

To prevent the SP from abandoning the preferred data item, we ask the data users to explicitly integrate a customized identity range in the query token. In this way, the SP can only return matched data items. Further, the SP has to generate proofs to prove that the claimed unmatched nodes do not match the query. In this way, the data users are convinced that the preferred data item is not abandoned.

## 7 Performance Analysis

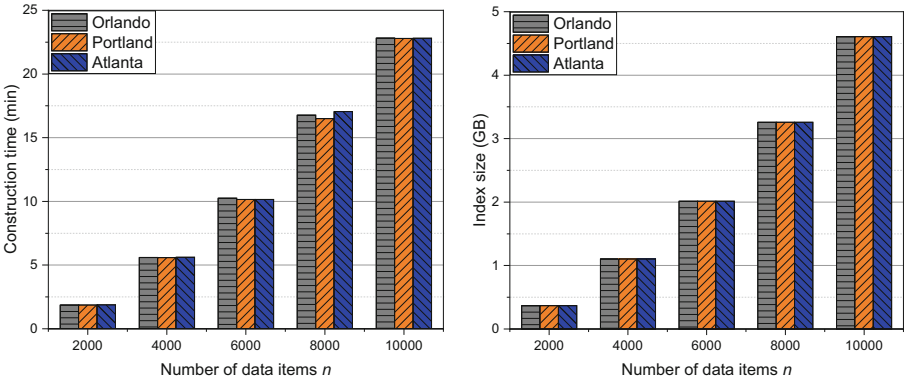
### 7.1 Experiment Settings

**Dataset**. We use the locations of three cities, i.e., Orlando, Portland, and Atlanta, from the Yelp dataset [31]. Each location has a type and two location coordinates. After preprocessing the dataset, we obtain 10,000 data items from each of the three cities and each item is in the form of $(id, t, loc)$.

**Parameters**. We vary $n$ from 2,000 to 10,000, and $k$ from 1 to 5. The false positive rate is set to 1%. The number of pseudo-random hash functions $p$ is 5. According to the false positive rate equation [13], the IBF size $m$ ranges from 1.2 to 12 KB. The lengths of secret keys, random numbers, and the symmetric key are 1024 bits, 1024 bits, and 256 bits, respectively.

**Metrics**. We evaluate the time of tree construction, token generation, query processing, and result verification. We evaluate the communication overhead of index tree, query token, results and proofs. We conduct each set of experiment over twenty times and compute the average time. Communication overhead is calculated by measuring the size of the transmitted messages. Since AES is applicable to the symmetric encryption of all schemes, we remove this part in comparison, but focus on the index, token, query, and verification.

**Setup**. We instantiate ROU on a PC server running Windows Server 2021 R2 Datacenter with a 3.7-GHz Intel(R) Core(TM) i7-8770K processor, and 32 GB RAM. We use HMAC-SHA256 as the pseudo-random function to implement the hash functions of IBF. We use AES as the symmetric encryption. We have uploaded all source codes of ROU on Github: https://github.com/UbiPLab/ROU.




(a) Construction time by varying $n$ (b) Index size by varying $n$

**Fig. 6.** Performance of tree construction.

### 7.2 Index Building

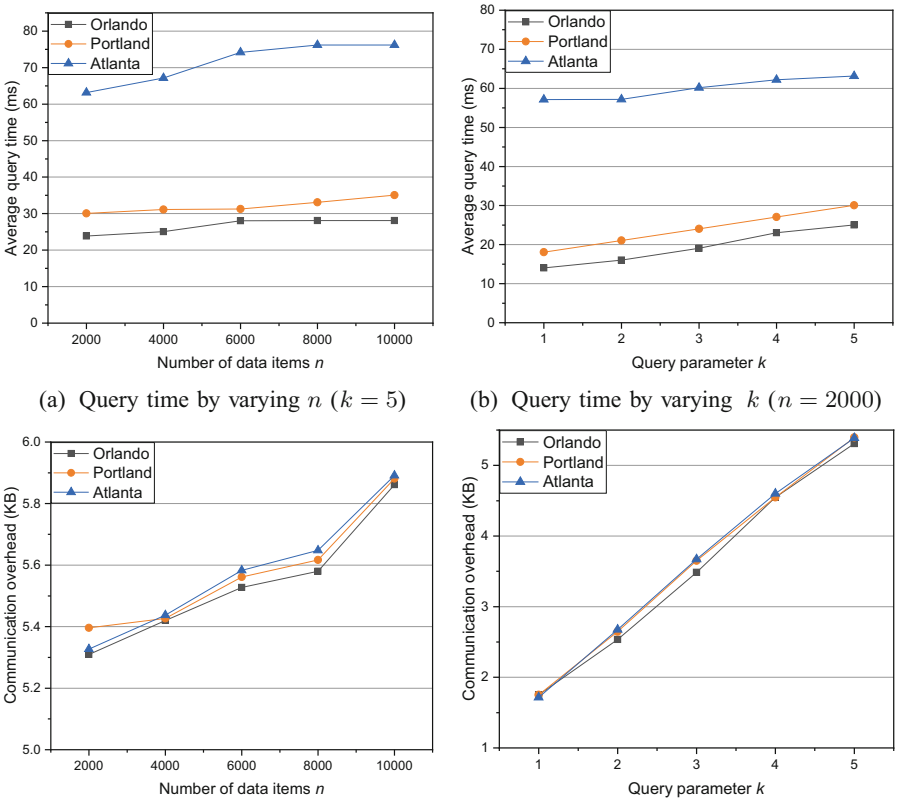
The computational cost of building an index tree as a function of $n$ is shown in Fig. 6(a). The communication overhead of uploading the index tree to the SP as shown in Fig. 6(b). It can be observed that construction time and index size grow linearly with $n$. When $n = 10,000$, it costs the data owner less than 23 ms in computing an index tree of 4.6 GB.

### 7.3 Token Generation

We assume that the data user only want to find one specific data item that was returned in her previous query. The token size is independent of $k$, but not $n$ because the size of $\mathcal{M}_2$ increases with $n$. There are two types of queries: ordinary query and repetitive query. Since the total number of prefixes in the two cases are the same, there will be no difference for their computational cost (35.1 ms) and communication overhead (77.1 KB) when $n = 2000$.

### 7.4 Query Processing

The query processing time of ROU is a function of $n$ and $k$. Figure 7(a) shows that the query processing time is in the millisecond scale. When $k = 1$ and $n = 10000$, the average query processing time for Orlando is 28 ms. The difference among the three cities are caused by the different distribution of matched data items. However, Fig. 7(b) shows that when $n = 2000$, with the $k$ increasing from 1 to 5, the query processing time does not grow much with $k$ on each of the three lines, because we have designed a customized identity range for the data owner, which may lead to less matched results. In other words, the repetitive S$k$NN query does not have to return $k$ results. In Fig. 7(c) and Fig. 7(d), the communication overhead of the SP increases with $n$ and $k$ because the IBF size increases with $n$ and the number of returned nodes increases, respectively.




(a) Query time by varying $n$ ($k = 5$)

(b) Query time by varying $k$ ($n = 2000$)

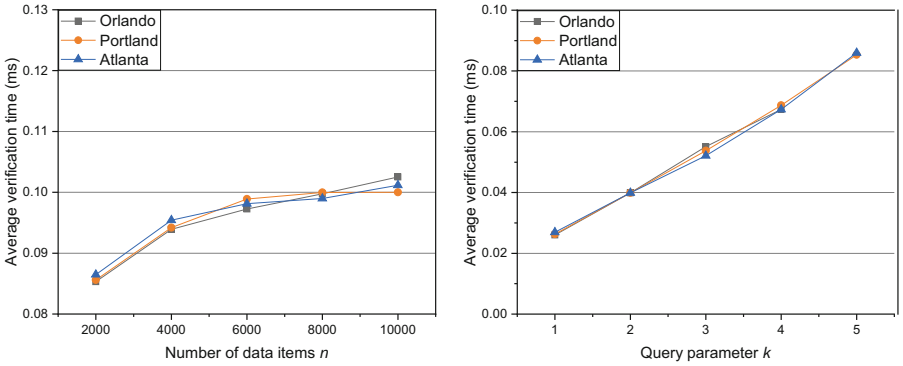
(c) Communication overhead by varying $n$ ($k = 5$)

(d) Communication overhead by varying $k$ ($n = 2000$)

**Fig. 7.** Performance of query processing.

### 7.5 Result Verification

After the SP returns the results and proofs to the data user, the data user verifies the results by recomputing the root's hash value from the received hash values. The result verification time, as shown in Fig. 8, corresponds to the results and proofs returned by the CS. It costs the data user (in Orlando) 0.08 ms and 0.1 ms when $k = 1, n = 2000$ and $k = 1, n = 10000$, respectively. We attribute this advantage to the exclusiveness of the repetitive query.




(a) Verification time by varying $n$ (b) Verification time by varying $k$

**Fig. 8.** Performance of result verification.

**Table 1.** Comparison of computational costs and communication overhead.

| Computational costs | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Scheme | Index building (min) | | Token generation (ms) | | Query processing (ms) | | | |
| | | | | | $n = 2000$ | | $n = 10000$ | |
| | $n = 2000$ | $n = 10000$ | $n = 2000$ | $n = 10000$ | $k = 1$ | $k = 5$ | $k = 1$ | $k = 5$ |
| SecEQP [11] | 1.82 | 22.5 | 33.08 | 512.37 | 16.06 | 28.07 | 17.06 | 30.08 |
| ServeDB [15] | 0.34 | 2.02 | 32.02 | 511.38 | 21.06 | 73.19 | 26.07 | 88.23 |
| ROU | 1.87 | 22.8 | 35.1 | 516.37 | 14.04 | 25.04 | 16.04 | 28.08 |
| Communication overhead | | | | | | | | |
| Scheme | Index building (GB) | | Token generation (KB) | | Query processing (KB)[1] | | | |
| | | | | | $n = 2000$ | | $n = 10000$ | |
| | $n = 2000$ | $n = 10000$ | $n = 2000$ | $n = 10000$ | $k = 1$ | $k = 5$ | $k = 1$ | $k = 5$ |
| SecEQP [11] | 0.37 | 4.60 | 41.25 | 43.12 | n/a | | | |
| ServeDB [15] | 0.31 | 1.86 | 20.14 | 21.18 | 17.99 | 84.56 | 18.35 | 88.49 |
| ROU | 0.37 | 4.60 | 77.10 | 116.19 | 1.75 | 5.31 | 2.03 | 5.86 |
| 1: messages for result verification | | | | | | | | |

### 7.6 Comparison

We compare ROU with existing work, i.e., SecEQP [11] and ServeDB [15], which are constructed upon the same techniques. We add the type and data item identity into their schems by using privacy-preserving range query. We record the comparison results in Table 1. In **index building**, SecEQP and ServeDB also build an index tree. The cost of SecEQP is similar to ours for using multiple coordinate systems. The cost of ServeDB is lower only uses a Bloom filter as an index, thereby involving less computation time and communication overhead. In

**token generation**, the two comparison schemes have a slightly smaller cost for not mixing the location codes and identity prefixes. ROU's token size is large for using mix indexes. In **query processing**, ROU's average query time is smaller because the data user has a specific requirement on data item, thus cutting off many search paths when the CS is searching on the index tree. Comparison results show that ROU exhibits practical efficiency.

## 8 Conclusions

In this work, we have located the repetitive query in S$k$NN and have proposed a repetitive, oblivious, and unlinkable query processing scheme over encrypted data on cloud. The novelty of ROU is in realizing repetitive query by mixing customized privacy-preserving range querying with S$k$NN query. We formally define and prove the privacy of ROU. By carefully designing the index building and token generation, we achieve repetitive query in an oblivious and unlinkable manner. We implement ROU and evaluate its performance on a desktop server and a real-world dataset. The experimental results show that ROU achieves practical efficiency.

**Acknowledgment.** The work described in this paper is supported by National Natural Science Foundation of China (NSFC) under the grant No. 62002094 and Anhui Provincial Natural Science Foundation under the grant No. 2008085MF196. It is partially supported by EU LOCARD Project under Grant H2020-SU-SEC-2018-832735.

## References

1. Liu, X., He, K., Yang, G., Susilo, W., Tonien, J., Huang, Q.: Broadcast authenticated encryption with keyword search. In: Baek, J., Ruj, S. (eds.) ACISP 2021. LNCS, vol. 13083, pp. 193–213. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-90567-5_10
2. Luo, Y., Jia, X., Fu, S., Xu, M.: pRide: privacy-preserving ride matching over road networks for online ride-hailing service. IEEE Trans. Inf. Forensics Secur. (TIFS) **14**(7), 1791–1802 (2019)
3. Zhu, L., Li, M., Zhang, Z., Qin, Z.: ASAP: an anonymous smart-parking and payment scheme in vehicular networks. IEEE Trans. Dependable Secure Comput. (TDSC) **17**(4), 703–715 (2020). https://doi.org/10.1109/TDSC.2018.2850780
4. Zhu, X., Ayday, E., Vitenberg, R.: A privacy-preserving framework for outsourcing location-based services to the cloud. IEEE Trans. Dependable Secure Comput. (TDSC) **18**(1), 384–399 (2021)
5. Damodaran, A., Rial, A.: Unlinkable updatable databases and oblivious transfer with access control. In: Liu, J.K., Cui, H. (eds.) ACISP 2020. LNCS, vol. 12248, pp. 584–604. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-55304-3_30
6. Li, M., Chen, Y., Zheng, S., Hu, D., Lal, C., Conti, M.: Privacy-preserving navigation supporting similar queries in vehicular networks. IEEE Trans. Dependable Secure Comput. (TDSC), **99**(2), 1–11. https://doi.org/10.1109/TDSC.2020.3017534

7. Danger within: defending cloud environments against insider threats (2018). https://www.cloudcomputing-news.net/news/2018/may/01/danger-within-defending-cloud-environments-against-insider-threats
8. 7 Most Infamous Cloud Security Breaches (2017). https://blog.storagecraft.com/7-infamous-cloud-security-breaches
9. Yao, B., Li, F., Xiao, X.: Secure nearest neighbor revisited. In: Proceeding 29th IEEE International Conference on Data Engineering (ICDE), April, pp. 733–744, Brisbane, Australia (2013)
10. Elmehdwi, Y., Samanthula, B.K., Jiang, W.: Secure k-nearest neighbor query over encrypted data in outsourced environments. In: Proceeding IEEE 30th International Conference on Data Engineering (ICDE), pp. 664–675, Chicago, USA (2014)
11. Lei, X., Liu, A. X., Li, R., Tu, G.-H.: SecEQP: a secure and efficient scheme for SkNN query problem over encrypted geodata on cloud. In: Proceeding 35th IEEE International Conference on Data Engineering (ICDE), April, pp. 662–673, Macao, China (2019)
12. Cui, N., Yang, X., Wang, B., Li, J., Wang, G.: SVkNN: efficient secure and verifiable k-nearest neighbor query on the cloud platform. In: Proceeding 36th IEEE International Conference on Data Engineering (ICDE), April, pp. 253–264, Dallas, USA (2020)
13. Li, R., Liu, A., Wang, A. L., Bruhadeshwar, B.: Fast range query processing with strong privacy protection for cloud computing. In: Proceeding 40th International Conference on Very Large Data Bases (VLDB), September, pp. 1953–1964, Hangzhou, China (2014)
14. Li, R., Liu, A.X.: Adaptively secure conjunctive query processing over encrypted data for cloud computing. In: Proceeding IEEE 33rd International Conference on Data Engineering (ICDE), April, pp. 697–708, San Diego, USA (2017)
15. Wu, S., Li, Q., Li, G., Yuan, D., Yuan, X., Wang, C.: ServeDB: secure, verifiable, and efficient range queries on outsourced database. In: Proceeding IEEE 35th International Conference on Data Engineering (ICDE), April, pp. 626–637, Macao, China (2019)
16. Chen, Y., Li, M., Zheng, S., Hu, D., Lal, C., Conti, M.: One-time, oblivious, and unlinkable query processing over encrypted data on cloud. In: Meng, W., Gollmann, D., Jensen, C.D., Zhou, J. (eds.) ICICS 2020. LNCS, vol. 12282, pp. 350–365. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-61078-4_20
17. Poutinsev, F.: Unfair search engine ranking results (2021). https://honestproscons.com/unfair-search-engine-ranking-results. Honest Pros and Cons (HPC)
18. Paillier, P.: Public-key cryptosystems based on composite degree residuosity classes. In: Stern, J. (ed.) EUROCRYPT 1999. LNCS, vol. 1592, pp. 223–238. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-48910-X_16
19. Liu, A.X., Chen, F.: Collaborative enforcement of firewall policies in virtual private networks. In: Proceeding 27th ACM Symposium on Principles of Distributed Computing (PODC), August, pp. 95-104, Toronto, Canada (2008)
20. Cao, Y., Xiao, Y., Xiong, L., Bai, L., Yoshikawa, M.: Protecting spatiotemporal event privacy in continuous location-based services. IEEE Trans. Knowl. Data Eng. (TKDE) **33**(8), 3141–3154 (2021)
21. Canetti, R., Feige, U., Goldreich, O., Naor, M.: Adaptively secure multi-party computation. In: Proceeding 28th ACM Symposium on Theory of Computing (STOC), May, pp. 639–648, Philadelphia, USA (1996)

22. Boldyreva, A., Chenette, N., O'Neill, A.: Order-preserving encryption revisited: improved security analysis and alternative solutions. In: Rogaway, P. (ed.) CRYPTO 2011. LNCS, vol. 6841, pp. 578–595. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22792-9_33
23. Kamara, S., Papamanthou, C., Roeder, T.: Dynamic searchable symmetric encryption. In: Proceeding 19th ACM Conference on Computer and Communications Security (CCS), October, pp. 965–976, Raleigh, USA (2012)
24. Cash, D., et al.: Dynamic searchable encryption in very-large databases: data structures and implementation. In: Proceeding 21st Annual Network and Distributed System Security Symposium (NDSS), February, pp. 1-16, San Diego, USA (2014)
25. Li, M., Chen, Y., Lal, C., Conti, M., Alazab, M., Hu, D.: Eunomia: anonymous and secure vehicular digital forensics based on blockchain. IEEE Trans. Dependable Secure Comput. (TDSC), 1 (2021). https://doi.org/10.1109/TDSC.2021.3130583
26. Li, M., Zhu, L., Zhang, Z., Lal, C., Conti, M., Alazab, M. : User-defined privacy-preserving traffic monitoring against n-by-1 jamming attack. IEEE/ACM Trans. Networking (TON), p. 1 (2022). https://doi.org/10.1109/TNET.2022.3157654
27. Li, M., Zhu, L., Zhang, Z., Lal, C., Conti, M., Alazab, M.: Anonymous and verifiable reputation system for E-commerce platforms based on blockchain. IEEE Trans. Network Serv. Manag. (TNSM) **18**(4), 4434–4449 (2021). https://doi.org/10.1109/TNSM.2021.3098439
28. Li, M., Hu, D., Lal, C., Conti, M., Zhang, Z.: Blockchain-enabled secure energy trading with verifiable fairness in industrial internet of things. IEEE Trans. Ind. Inf. (TII) **16**(10), 6564–6574 (2020). https://doi.org/10.1109/TII.2020.2974537
29. Li, M., Zhu, L., Zhang, Z., Lal, C., Conti, M., Martinelli, F.: Privacy for 5G-supported vehicular networks. IEEE Open J. Commun. Soc. (OJ-COMS), **2**, 1935–1956 (2021). https://doi.org/10.1109/OJCOMS.2021.3103445
30. Szydlo, M.: Merkle tree traversal in log space and time. In: Proceeding 10th International Conference on the Theory and Applications of Cryptographic Techniques (Eurocrypt), May, pp. 541–554, Interlaken, Switzerland (2004)
31. Yelp Open Dataset. https://www.yelp.com/dataset