Efficient and Privacy-Preserving Spatial Keyword Similarity Query over Encrypted Data

Songnian Zhang, Suprio Ray, *Member, IEEE,* Rongxing Lu, *Fellow, IEEE,* Yunguo Guan, Yandong Zheng, and Jun Shao, *Senior Member, IEEE*

Abstract—As a popular and practical query type in location-based services, the spatial keyword query has been extensively studied in both academia and industry. Meanwhile, with the growing demand for data privacy, many privacy-preserving spatial keyword query schemes have been proposed to deal with queries over encrypted data. However, none of the existing schemes preserve access pattern privacy, and the recent research illustrates that leaking such privacy may incur inference attacks and thus disclose sensitive information. In addition, most existing schemes only consider the boolean keyword search, which is not quite practical and flexible in real-world applications. To address the above issues, in this paper, we propose two privacy-preserving spatial keyword similarity query schemes that can preserve full and partial access pattern privacy, respectively. First, we present a basic privacy-preserving spatial keyword similarity query scheme (PPSKS) by integrating a secure set membership test (SSMT) technique with secure circuits. After that, to improve performance, we propose a tree-based scheme (PPSKS+) by employing a new index called FR-tree together with a predicate encryption technique that can encrypt FR-tree. Formal security analysis shows that: i) our proposed schemes can protect outsourced data, query requests, and query results; ii) our PPSKS scheme can hide full access patterns, while the PPSKS+ scheme preserves m-access pattern privacy. Extensive experiments are also conducted, and the results indicate that our tree-based PPSKS+ scheme is much more efficient, almost two orders of magnitude better than our linear search PPSKS scheme in performing queries.

Index Terms—Spatial keyword similiarity query, Privacy preservation, Secure circuits, Bloom filter, Lagrange interpolation.

1 Introduction

The proliferation of the mobile Internet drives the widespread use of location-based services (LBS), especially the spatial keyword query services offered by a slew of commercial applications, e.g., Yelp and Google Map. Due to its broad utility in LBS, the spatial keyword query has been extensively investigated in both academia and industry [1]–[5]. One of the real-life examples is the POI (point of interest) recommendation system, in which a user can enjoy the services by requesting with a spatial query range and keywords to a service provider. Assume that the service provider is equipped with a spatial keyword database containing POI locations and feature keywords, e.g., a restaurant with location (39.95, -82.99) and a set of keywords $\{\text{coffee}, \text{beef}, \text{pizza}\}$. By performing the spatial keyword query, the service provider can retrieve the POIs satisfying the following two conditions: i) the POIs' locations fall inside the spatial query range; and ii) the POIs' keywords match the query keywords.

Meanwhile, with increasing concerns about data privacy, performing queries over encrypted data has attracted considerable attention. As a quite practical query type, processing spatial keyword queries over encrypted data is naturally an important research topic, and several privacy-preserving spatial keyword query schemes have been proposed [6]–[10]. However, existing schemes have two issues: i) most of them [6]–[9] only consider the boolean keyword match, i.e., a data record's keywords must exactly contain all query keywords, which is not practical and flexible enough in real-world applications. Although the work [10] studied the keyword similarity, it adopted the Euclidean distance as the metric, which is more suitable for the vector space characterized by the fixed vector length instead of the scenario with dynamic keyword set sizes [11]; and ii) none of the existing schemes protect access pattern privacy [12], i.e., the information about which data records satisfy the query conditions. As reported in [13], [14], leaking access patterns may incur inference attacks and thus disclose sensitive information.

Aiming at the above two issues, we focus on privacy-preserving spatial keyword similarity query schemes by considering keyword similarity and protecting the access pattern privacy. Notably, since the Jaccard similarity is very popular in measuring the keyword set similarity [15], [16], and this work exact considers such a scenario, it is adopted in this paper. However, the privacy-preserving spatial keyword similarity query scheme is more challenging to develop than the privacy-preserving boolean spatial keyword query scheme. This is because the latter only involves the equality test for keywords, while the former contains a typical *compute-then-compare* operation in dealing with keywords, which is recognized as challenging when securing this operation in a single-server model [9], [17]. Furthermore, in addition to the data and query privacy, our proposed schemes have a stricter security goal, i.e., preserving the privacy of access patterns, which makes it even more

- *S. Zhang, S. Ray, R. Lu, and Y. Guan are with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada (e-mail: szhang17@unb.ca, sray@unb.ca, rlu1@unb.ca, yguan4@unb.ca).*
- *Y. Zheng is with the State Key Laboratory of Integrated Services Networks, Xidian University, Xi'an, 710071, China (e-mail: zhengyandong@xidian.edu.cn).*
- *J. Shao is with School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, 310018, China (e-mail: chn.junshao@gmail.com).*

difficult to design the privacy-preserving spatial keyword similarity query schemes. Besides these, protecting access patterns will inevitably introduce the performance issue. Therefore, devising an efficient scheme while preserving the privacy of access patterns is a challenging problem that we have to tackle.

In this paper, we first propose a privacy-preserving spatial keyword similarity query scheme, named as PPSKS, to securely determine range constraint, compute keyword similarity, and hide access patterns. The main idea is to map the spatial data into bloom filters and then encrypt them with fully homomorphic encryption (FHE). Then, the operator can obtain an encrypted flag to determine whether a data record satisfies the spatial query range. However, a problem arises, i.e., "how do we compute encrypted flags from the encrypted bloom filters?". To tackle it, we design a Lagrange interpolation-based approach and propose a secure set membership test (SSMT) scheme to make it possible. For the keyword similarity, we surprisingly found that we can transform the encrypted flags (obtained by our SSMT scheme) into an encrypted bit sequence, which can represent the number of intersecting elements of two keyword sets, by designing a secure partial addition circuit. Based on this observation, we can then use the secure addition and secure comparison circuits to obtain an encrypted flag that can determine whether these two keyword sets are similar. With these encrypted flags, we can compute (not select) the encrypted query results and thus hide access patterns.

To improve performance, we further propose a tree-based privacy-preserving spatial keyword similarity query scheme, PPSKS+, in which we introduce an index called FR-tree and modify a predicate encryption technique [18] allowing the operator to search over the encrypted FR-tree. Note that the modified predicate encryption technique can protect the conjunctive privacy discussed in [9], which refers to the information about the spatial constraint or the keyword constraint mismatches when a data record is not picked. Specifically, our paper has the following contributions:

- First, we propose a novel secure set membership test (SSMT) scheme that can securely determine whether an element belongs to a set or not. In our SSMT scheme, we combine the bloom filter technique and Lagrange interpolation function such that it can protect the privacy of the element, the set, and the decision result. Note that our SSMT scheme can also be applied to other privacy-preserving schemes that seek the fully secure set membership test in a single-server model.
- Second, we propose a privacy-preserving spatial keyword similarity query scheme, PPSKS, to securely retrieve the data records that satisfy the query constraints. In our PPSKS scheme, we first transform the spatial query problem into the set membership test, which makes it possible to use our SSMT scheme. In addition, we observe that secure circuits can also be introduced to securely calculate Jaccard similarity. It is worth noting that we are the first to consider the Jaccard similarity and access patterns in privacy-preserving spatial keyword query schemes.
- Third, we propose a tree-based PPSKS scheme, denoted as PPSKS+, by employing an FR-tree index and a predicate encryption technique. In order to further improve performance, we present a vector bucketing technique to split a large vector into sub-vectors. Theoretically, this technique can reduce the computational overheads in key generation, data outsourcing, and token generation phases.
- Finally, we formally analyze the security of our proposed schemes and demonstrate that our proposed schemes can attain our privacy goals. Besides, we conduct extensive experiments to evaluate our proposed schemes, and the results illustrate that our tree-based PPSKS+ scheme can significantly reduce the computational costs of the linear search scheme (PPSKS).

The remainder of this paper is organized as follows. In Section 2, we introduce our system model, security model, and design goal. Then, we review the preliminaries in Section 3. After that, we present our PPSKS and PPSKS+ schemes in Section 4, followed by security analysis and performance evaluation in Section 5 and Section 6, respectively. Finally, we discuss some related works in Section 7 and draw our conclusion in Section 8.

2 Models and Design Goal

In this section, we formalize our system model, security model, and identify our design goal.

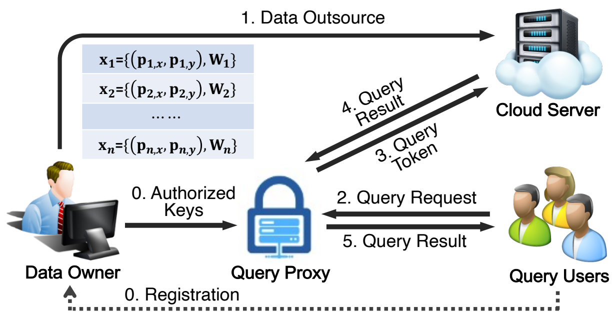


Fig. 1. System model under consideration

2.1 System Model

In our system model, we consider a typical outsourcing model, which is comprised of four entities: a data owner $\mathcal{O}$, a powerful cloud server $\mathcal{C}$, a query proxy $\mathcal{P}$, and multiple query users $\mathcal{U} = \{u_1, u_2, \cdots\}$, as shown in Fig. 1.

Data Owner $\mathcal{O}$: In our system model, the data owner $\mathcal{O}$ has a spatial keyword dataset $\mathcal{X} = \{\mathtt{x}_i = \{(\mathtt{p}_{i,x}, \mathtt{p}_{i,y}), \mathtt{W}_i\} \mid 1 \leq i \leq n\}$, where $(\mathtt{p}_{i,x}, \mathtt{p}_{i,y})$ is the spatial location of the data point, and $\mathtt{W}_i$ is the keyword set. In order to make full use of the dataset, the data owner $\mathcal{O}$ offers the spatial keyword similarity (SKS) query services to the query users. However, since $\mathcal{O}$ may not be powerful in storage and computing, it tends to outsource the dataset $\mathcal{X}$ and the SKS query services to a cloud. Meanwhile, to ensure privacy, the data owner $\mathcal{O}$ generates secret keys and encrypts the outsourced data before uploading them to the cloud.

Cloud Server $\mathcal{C}$: In our system, the cloud server $\mathcal{C}$ is considered as powerful in storage and computing. It receives the outsourced dataset from the data owner $\mathcal{O}$ and provides the SKS query services to the query users by leveraging the received dataset.

Query Proxy $\mathcal{P}$: In order to manage the generated keys, the data owner $\mathcal{O}$ can deploy a query proxy $\mathcal{P}$ and authorize the generated keys to $\mathcal{P}$. In our system, $\mathcal{P}$ is sitting between the cloud server $\mathcal{C}$ and query users and can provide the query token generation and query result decryption services to the query users. It is worth noting that we can also make the data owner $\mathcal{O}$ undertake the tasks of $\mathcal{P}$ as adopted in [6], [19]. In order to allow $\mathcal{O}$ to go offline after initialing the whole system, we deploy $\mathcal{P}$ in our system.

Query Users $\mathcal{U} = \{u_1, u_2, \cdots\}$: In our system, query users $\mathcal{U}$ should first register to the data owner $\mathcal{O}$ and obtain the authorized keys. After that, $\mathcal{U}$ can enjoy the SKS query services through the query proxy $\mathcal{P}$.

2.2 Security Model

In our security model, the data owner $\mathcal{O}$ is considered to be *trusted* because it initializes the whole system. For the query users, we consider the registered ones to be *honest*, i.e., they will honestly follow the proposed scheme. However, in our model, the cloud server $\mathcal{C}$ and the query proxy $\mathcal{P}$ are considered as *semi-honest* [20], which indicates that they will sincerely follow the proposed schemes but are curious to learn some private information. For the cloud server $\mathcal{C}$, it may attempt to infer the outsourced data, query requests, and query results. Regarding the query proxy $\mathcal{P}$, it may be interested in the query requests and query results received from the query users $\mathcal{U}$ and cloud server $\mathcal{C}$, respectively. We assume that there is no collusion between any two entities of the cloud server $\mathcal{C}$, the query proxy $\mathcal{P}$, and query users $\mathcal{U}$ [9]. It is reasonable since they are strictly regulated, and their reputation will be damaged when collusive behavior is detected. Note that, as this work mainly focuses on privacy computation techniques, other active attacks, e.g., authentication and verification issues, are beyond the scope of this paper and will be discussed in our future work.

2.3 Design Goal

In this work, our goal is to present privacy-preserving and efficient SKS query schemes. In particular, the following objectives should be attained.

- Privacy Preservation: The basic requirement of our proposed scheme is privacy preservation. First, we should protect the privacy of outsourced data, query requests, and query results against the cloud server $\mathcal{C}$. Then, we should protect the privacy of query requests and query results against the query proxy $\mathcal{P}$.

In addition, our proposed scheme should hide access patterns for the cloud server $\mathcal{C}$, which indicates $\mathcal{C}$ has no idea about which data records are returned as the query results. Assume $\mathcal{C}$ knows query distribution as background knowledge. If the access patterns are leaked, $\mathcal{C}$ can learn the access distribution of outsourced data over time and thus infer query contents by correlating the query distribution to the data access distribution. Once the content of an encrypted query is known, $\mathcal{C}$ may further infer outsourced data retrieved by this query [12], [14]. Therefore, hiding access patterns is significant in privacy preservation.

- Efficiency: Achieving the privacy requirements will inevitably incur additional costs. As a result, we also aim to minimize the computational costs when performing the privacy-preserving SKS queries.

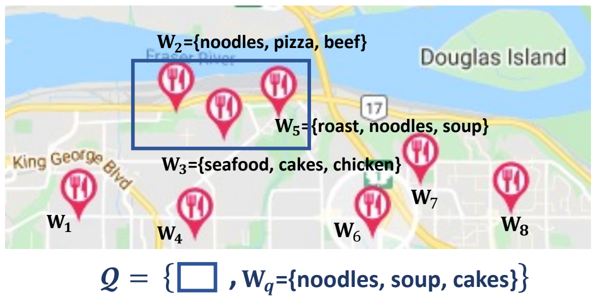

W2={noodles, pizza, beef}
Douglas Island
Fraser River
17
W5={roast, noodles, soup}
King George Blvd
W3={seafood, cakes, chicken}
W7
W1
W4
W6
W8
Q = { , Wq={noodles, soup, cakes}}

Fig. 2. An example of SKS query (the background map was extracted from Google maps).

3 Preliminaries

In this section, we first define the spatial keyword similarity (SKS) queries. Then, we introduce bloom filter, fully homomorphic encryption, and secure circuits, which will be used in our proposed scheme.

3.1 Spatial Keyword Similarity Queries

The spatial keyword queries have been extensively investigated in both academic and industrial communities due to its wide applications [1]–[3]. Given a spatial keyword query $\mathcal{Q} = \{(\mathtt{R}_{q,x}, \mathtt{R}_{q,y}), \mathtt{W}_q\}$, where $(\mathtt{R}_{q,x} = [\mathtt{p}_{q,xl}, \mathtt{p}_{q,xu}], \mathtt{R}_{q,y} = [\mathtt{p}_{q,yl}, \mathtt{p}_{q,yu}])$ are the ranges of x and y dimensions, and $\mathtt{W}_q$ is the query keyword set, the basic spatial keyword query retrieves the data records that satisfy: $\mathtt{p}_{i,x} \in \mathtt{R}_{q,x}$, $\mathtt{p}_{i,y} \in \mathtt{R}_{q,y}$, and $\mathtt{W}_q \subseteq \mathtt{W}_i$ [1]. However, the exact matching of the keyword sets lacks flexibility and practicality. For example, the basic spatial keyword queries would not return the data records if there is only one element of a query keyword set mismatch (while all other elements are matched). To make the spatial keyword query more flexible and practical, the SKS queries are defined to measure the similarity of keyword sets [5].

Definition 1 (Spatial Keyword Similarity Queries)**.** *Given a spatial keyword dataset $\mathcal{X}$, a query request $\mathcal{Q}$, and a similarity threshold τ, the spatial keyword similarity (SKS) queries return the data records in $\mathcal{X}$ satisfying:*

i) spatial constraint, i.e., $\mathtt{p}_{i,x} \in \mathtt{R}_{q,x}$, $\mathtt{p}_{i,y} \in \mathtt{R}_{q,y}$;

ii) keyword constraint, i.e., $\mathsf{sim}(\mathtt{W}_q, \mathtt{W}_i) \geq \tau$.

Same as [5], [15], [16], here we use the Jaccard similarity for the keyword set similarity:

$$\mathsf{sim}(\mathtt{W}_q, \mathtt{W}_i) = \frac{|\mathtt{W}_q \cap \mathtt{W}_i|}{|\mathtt{W}_q \cup \mathtt{W}_i|}. \quad (1)$$

A simple example of the SKS query is shown in Fig. 2, in which $\mathtt{W}_i$ $(1 \leq i \leq 8)$ is the keyword set of the data record $\mathtt{x}_i = \{(\mathtt{p}_{i,x}, \mathtt{p}_{i,y}), \mathtt{W}_i\}$. Given a spatial query $\mathcal{Q}$ and a Jaccard similarity threshold $\tau = 2/5$, $\mathtt{x}_5$ is returned as the query result. First, it is clear that $\{\mathtt{x}_2, \mathtt{x}_3, \mathtt{x}_5\}$ satisfy the spatial constraint. Then, since the Jaccard similarity of $\{\mathtt{x}_2, \mathtt{x}_3, \mathtt{x}_5\}$ is $\{1/5, 1/5, 1/2\}$, respectively, only $\mathtt{x}_5$ satisfies the keyword constraint.

3.2 Bloom Filter

A bloom filter (BF) can determine whether an element exists in a given set [21]. In general, given a set S, the bloom filter represents all elements in S using an array of η bits, denoted as $\mathsf{BF}[t], 1 \le t \le \eta$. Initially, all bits in the array are set to 0. Then, with γ independent hash functions $\{h_1, \cdots, h_\gamma\}$, each element $\mathtt{x} \in \mathsf{S}$ is mapped to the array, i.e., $\mathsf{BF}[h_i(\mathtt{x})] = 1$. Given an element $\mathtt{x}'$, to determine whether $\mathtt{x}' \in \mathsf{S}$ or not, we can check whether all $\mathsf{BF}[h_i(\mathtt{x}')]$ $(1 \le i \le \gamma)$ are set to 1. If not, $\mathtt{x}'$ must be not a member of S. If yes, $\mathtt{x}'$ is in S with a high probability. Obviously, a bloom filter may yield *false positive*. Assuming the size of the set S is μ, i.e., $\mu = |\mathsf{S}|$, the *false positive* probability is:

$$f_p = (1 - (1 - 1/\eta)^{\gamma\mu})^\gamma \approx (1 - e^{-\gamma\frac{\mu}{\eta}})^\gamma. \tag{2}$$

Given μ and η, the value of γ that minimizes the *false positive* probability is: $\gamma = \ln 2 \cdot (\eta/\mu)$. In this case, $f_p \approx (1/2)^\gamma$. If f_p, γ, and μ are given, we could obtain $\eta = \gamma \cdot \mu / \ln 2$, which is the optimal size of the bloom filter array.

3.3 Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) is a popular cryptographic primitive that can support computations through encrypted data [22]. Due to its nice homomorphic properties, it is widely used to design searchable encryption schemes [23], [24]. Typically, an FHE scheme satisfies two homomorphic properties: i) Homomorphic addition: $\mathsf{E}(m_1)+\mathsf{E}(m_2) \to \mathsf{E}(m_1+m_2)$; ii) Homomorphic multiplication: $\mathsf{E}(m_1)\cdot\mathsf{E}(m_2) \to \mathsf{E}(m_1\cdot m_2)$, where m_1 and m_2 are two plaintexts, and $\mathsf{E}(m_1)$ and $\mathsf{E}(m_2)$ are the corresponding FHE ciphertexts. In our proposed scheme, we employ the FHE scheme as cryptography primitive and exploit its addition and multiplication homomorphic properties. Since our proposed schemes can be constructed on any FHE scheme, here we do not show the algorithms of FHE and refer readers to [25], [26] for details.

3.4 Secure Circuits

Assume there are two non-negative integers $\{x, y\}$ of the same bit length, and the corresponding encrypted bit sequences are:

$$\mathsf{E}(\vec{x}) = (\mathsf{E}(x^j)|_{j=l}^0) = \big(\mathsf{E}(x^l), \mathsf{E}(x^{l-1}), \cdots, \mathsf{E}(x^0)\big);$$
$$\mathsf{E}(\vec{y}) = (\mathsf{E}(y^j)|_{j=l}^0) = \big(\mathsf{E}(y^l), \mathsf{E}(y^{l-1}), \cdots, \mathsf{E}(y^0)\big),$$

where $x^j, y^j \in \{0,1\}$, $j = l, \cdots, 1, 0$, and l is the most significant bit position.

- Secure addition circuit [27]. Given $\mathsf{E}(\vec{x})$ and $\mathsf{E}(\vec{y})$, the secure addition circuit, denoted as *Sadd*, can compute $\mathsf{E}(\vec{z}) = (\mathsf{E}(z^j)|_{j=l}^0) = \textit{Sadd}\big(\mathsf{E}(\vec{x}), \mathsf{E}(\vec{y})\big)$ satisfying $z = x + y$, where z is an integer, and $\mathsf{E}(\vec{z})$ is its encrypted bit sequence. The main idea of *Sadd* is to derive logical expressions from the truth table of the addition circuit and then adopt the homomorphic properties of FHE to execute the derived logical expressions over ciphertexts. See details in [27].
- Secure comparison circuit [28]. Given $\mathsf{E}(\vec{x})$ and $\mathsf{E}(\vec{y})$, the secure comparison circuit, denoted as *Scom*, can output $\mathsf{E}(z) = \textit{Scom}\big(\mathsf{E}(\vec{x}), \mathsf{E}(\vec{y})\big) = \mathsf{E}(1)$ if $x < y$, otherwise $\mathsf{E}(z) = \mathsf{E}(0)$. The main idea of *Scom* is to identify the most significant differing bit of two bit sequences, and we can formalize it as follows, seeing the detailed analysis in [28]:

$$\mathsf{E}(z) = \sum_{j=0}^{l} \Big((\mathsf{E}(x^j) < \mathsf{E}(y^j)) \prod_{j<t<l} (\mathsf{E}(x^t) = \mathsf{E}(y^t)) \Big),$$

TABLE 1
Notations used in our proposed scheme

Notation	Definition
(pk, sk)	the public key and secret key of FHE
mk	the master key held by the cloud server $\mathcal{C}$
ssk_i	a shared key of query user u_i, $ssk_i = \mathsf{H}(mk, id_i)$
ss_i	a session key at timestamp ts, $ss_i = \mathsf{H}(ssk_i, ts)$
μ	the number of elements to be put into bloom filter
γ	the number of hash functions of bloom filter
η	the length of bloom filter
BF_*	the bloom filter corresponding to $*$
φ	the maximum size of keyword sets
t_*	the size of the corresponding set
w	the bit length of an integer
$\mathtt{x}_i$	a spatial keyword data point, $1 \le i \le n$
$(\mathtt{p}_{i,x}, \mathtt{p}_{i,y})$	the spatial location of the data point $\mathtt{x}_i$
$\mathtt{W}_i$	the keyword set of the data point $\mathtt{x}_i$
$(\mathtt{R}_{q,x}, \mathtt{R}_{q,y})$	the spatial rectangle of query $\mathcal{Q}$
$\mathtt{W}_q$	the keyword set of query $\mathcal{Q}$
(τ_1, τ_2)	two integers represent the threshold τ
$\vec{x}$	the bit sequence of x
Φ	the cardinality of intersection, $\Phi = \lvert\mathtt{W}_q \cap \mathtt{W}_i\rvert$
k	the number of query results
$\mathtt{v}_j$	a vector with n elements, $1 \le j \le k$
π	the permutation function
BM_*	a bitmap of the corresponding keyword set
ρ	the number of unique keywords in the dataset
$(\mathtt{T}_x, \mathtt{T}_y)$	the upper bound of x and y dimensions
d	the number of dummy values
κ	the length of a sub-vector
ξ	the number of sub-vectors
M	the invertible random matrix

where $\big(\mathsf{E}(x^j) < \mathsf{E}(y^j)\big) = \mathsf{E}(y^j \cdot (x^j + 1))$ and $\big(\mathsf{E}(x^t) = \mathsf{E}(y^t)\big) = \mathsf{E}(y^t + x^t + 1)$. Note that, we need to calculate $z = z \mod 2$ to ensure $z \in \{0, 1\}$.

- Secure multiplication circuit. Given $\mathsf{E}(\vec{x})$ and $\mathsf{E}(\vec{y})$, the secure multiplication circuit, denoted as *Smul*, can compute $\mathsf{E}(\vec{z}) = (\mathsf{E}(z^j)|_{j=2l+1}^0) = \textit{Smul}\big(\mathsf{E}(\vec{x}), \mathsf{E}(\vec{y})\big)$ satisfying $z = x \cdot y$. It can be achieved by calculating *Sadd* with l times:

$$\mathsf{E}(\vec{z}) = \sum_{j=0}^{l} \big(\underbrace{\mathsf{E}(0), \cdots, \mathsf{E}(0)}_{l+1-j}, \mathsf{E}(y^j) \cdot \mathsf{E}(\vec{x}), \underbrace{\mathsf{E}(0), \cdots, \mathsf{E}(0)}_{j} \big).$$

Note that here the sum symbol indicates performing the secure addition circuit, *Sadd*.

Now, we give examples to illustrate the above secure circuits. If we set $x = 5$ and $y = 7$, we would have $\mathsf{E}(\vec{x}) = (\mathsf{E}(0), \mathsf{E}(1), \mathsf{E}(0), \mathsf{E}(1))$ and $\mathsf{E}(\vec{y}) = (\mathsf{E}(0), \mathsf{E}(1), \mathsf{E}(1), \mathsf{E}(1))$. Thus, $\textit{Sadd}\big(\mathsf{E}(\vec{x}), \mathsf{E}(\vec{y})\big) = (\mathsf{E}(1), \mathsf{E}(1), \mathsf{E}(0), \mathsf{E}(0)) = \mathsf{E}(\vec{12})$, $\textit{Scom}\big(\mathsf{E}(\vec{x}), \mathsf{E}(\vec{y})\big) = \mathsf{E}(1)$, and $\textit{Smul}\big(\mathsf{E}(\vec{x}), \mathsf{E}(\vec{y})\big) = (\mathsf{E}(0), \mathsf{E}(0), \mathsf{E}(1), \mathsf{E}(0), \mathsf{E}(0), \mathsf{E}(0), \mathsf{E}(1), \mathsf{E}(1)) = \mathsf{E}(\vec{35})$.

4 Our Proposed Schemes

In this section, we first propose a novel secure set membership test (SSMT) scheme, which servers as the building block. Then, we present our privacy-preserving spatial keyword similarity query scheme, PPSKS. Finally, we carefully design a tree-based PPSKS scheme, denoted as PPSKS+, to achieve a sub-linear search efficiency. Before delving into the details, we provide a notation table (Table 1) to describe the main notations used in our proposed schemes.

4.1 Secure Set Membership Test Scheme

Given a set S with μ elements, our SSMT scheme determines whether an element $\mathtt{x} \in \mathsf{S}$ or not in a fully secure manner, namely, without leaking any information, including the set S, the element x, and the information about whether $x \in S$ or not. Our key idea is to map all elements in S into a bloom filter, denoted as BF_s, and map x into another bloom filter, denoted as BF_x, using the same hash functions and keeping the same length as BF_s. Then, we encrypt each bit in these bloom filters with FHE, denoted as $E(BF_s)$ and $E(BF_x)$, respectively, and calculate the inner product of these two encrypted bloom filters. Finally, the Lagrange interpolation function is used to output $E(1)$ if $x \in S$, otherwise $E(0)$. We formally describe our SSMT scheme as follows.

- SSMT.Setup(λ): Given a security parameter λ, the setup algorithm outputs an FHE key pair (pk, sk), where pk is the public key, and sk is the secret key. Then, it chooses γ independent hash functions $\mathcal{H} = \{h_1, h_2, \cdots, h_\gamma\}$.
- SSMT.Interpolation(γ): Given the number of hash functions γ, the interpolation algorithm chooses a large prime number p and constructs a polynomial function $f(x)$ using Lagrange interpolation at nodes $\{(0,0), (1,0), (2,0), \cdots, (\gamma-1, 0), (\gamma, 1)\}$:

$$f(x) = a_0 + a_1x + a_2x^2 + \cdots a_\gamma x^\gamma \mod p.$$

- SSMT.Enc(S, $pk, \mathcal{H}$): On input of a set S, the public key pk, and a set of hash functions $\mathcal{H}$, the encryption algorithm maps all elements in S into a bloom filter BF_s, encrypts each bit in BF_s with the public key pk, and outputs $E(BF_s)$ as the result, i.e., $E(BF_s) = \text{SSMT.Enc}(S, pk, \mathcal{H})$.
- SSMT.Token(x, $pk, \mathcal{H}$): On input of an element x, the public key pk, and a set of hash functions $\mathcal{H}$, the token generation algorithm maps x into a bloom filter BF_x, encrypts each bit in BF_x with the public key pk, and outputs $E(BF_x)$ as the result, i.e., $E(BF_x) = \text{SSMT.Token}(x, pk, \mathcal{H})$.
- SSMT.Check ($E(BF_s), E(BF_x), f(x)$): Given $E(BF_s)$, $E(BF_x)$, and $f(x)$, the check algorithm determines whether $x \in S$ with the following two steps:

Step-1. Conduct the inner product operation between $E(BF_s)$ and $E(BF_x)$ and output $E(\sigma)$ as the result:

$$\begin{aligned} E(\sigma) = E(BF_s) \circ E(BF_x) &= \sum_{t=1}^{\eta} \Big(E(BF_s[t]) \cdot E(BF_x[t]) \Big) \\ &= E\Big(\sum_{t=1}^{\eta} (BF_s[t] \cdot BF_x[t]) \Big), \end{aligned} \tag{3}$$

where η is the length of both bloom filters.

Step-2. Calculate $E(\theta)$ as the result of the check algorithm by integrating $f(x)$ and $E(\sigma)$:

$$E(\theta) = f(E(\sigma)) = E(f(\sigma)).$$

Correctness. We say our SSMT scheme is correct when it outputs $E(\theta) = E(1)$ if $x \in S$, otherwise $E(\theta) = E(0)$.

Proof. From Eq. (3), we know that σ is one of the values in the set $\{0, 1, 2, \cdots, \gamma\}$. Only when $\sigma = \gamma$, we have $x \in S$ according to the definition of bloom filter (see Section 3.2). Since our polynomial function $f(x)$ is interpolated at nodes $\{(0,0), (1,0), (2,0), \cdots, (\gamma-1, 0), (\gamma, 1)\}$, we have $f(\gamma) = 1$ and $f(x) = 0$ for $x = 0, 1, \cdots \gamma - 1$. Therefore, from *Step 2*, we have $x \in S \Leftrightarrow \sigma = \gamma \Leftrightarrow E(\theta) = E(f(\gamma)) = E(1)$. □

In Fig. 3, we give an example to determine whether a set $S = \{x, y\}$ contains x and x′ separately. Assume the

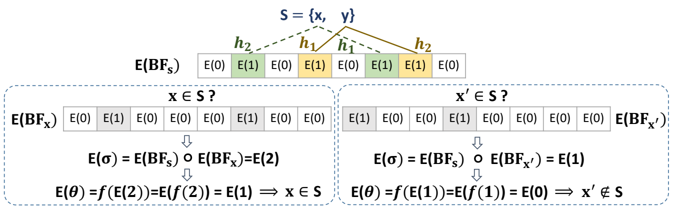

S = {x, y}
h2
h1
h1
h2
E(BFs)
E(0) E(1) E(0) E(1) E(0) E(1) E(1) E(0)
x ∈ S ?
E(BFx)
E(0) E(1) E(0) E(0) E(0) E(1) E(0) E(0)
E(σ) = E(BFs) ∘ E(BFx)=E(2)
E(θ) =f(E(2))=E(f(2)) = E(1) ⟹ x ∈ S
x′ ∈ S ?
E(1) E(0) E(0) E(1) E(0) E(0) E(0) E(0)
E(BFx′)
E(σ) = E(BFs) ∘ E(BFx′) = E(1)
E(θ) =f(E(1))=E(f(1)) = E(0) ⟹ x′ ∉ S

Fig. 3. An example of our SSMT scheme, in which the length of bloom filters is 8, i.e., $\eta = 8$, the number of hash functions is 2, i.e., γ=2, and $f(x)$ is interpolated at nodes $\{(0,0), (1,0), (2,1)\}$.

bloom filter's length is 8, and two hash functions are used to map elements to a bloom filter. In this case, $f(x)$ can be interpolated at nodes $\{(0,0), (1,0), (2,1)\}$. If we assume $p = 71$, we could obtain $f(x) = 0 + 35x + 36x^2 \mod 71$. As shown in Fig. 3, $x \in S$ due to $E(\theta) = E(1)$, while $x' \notin S$ as $E(\theta) = E(0)$.

Remark. To securely determine whether an element is in a set, existing schemes [6], [9] leak the decision information about $x \in S$ or $x \notin S$. However, our proposed scheme can hide this information by integrating Lagrange interpolation function into FHE encrypted bloom filters. We argue that, if a *compute-then-compare* operation, which is hard to be securely implemented on a single-server model [17], can be converted into the problem of set membership test, our SSMT scheme can provide a fully secure solution.

4.2 Basic Construction: Our PPSKS Scheme

4.2.1 *Overview of Our PPSKS Scheme*

Recalling the definition of the SKS query (Definition 1), since the spatial constraint is to check whether a value belongs to a range, it can be securely achieved by our SSMT scheme. However, it is still challenging to deal with the keyword constraint. This is because we need to calculate the cardinality of intersection and union of two sets and compare their quotient with $\tau \in [0, 1]$. To address it, we transform the keyword constraint into the following inequality:

$$\begin{aligned} \text{sim}(W_q, W_i) \geq \tau \Rightarrow \text{sim}(W_q, W_i) &\geq \frac{\tau_1}{\tau_2} \\ \xrightarrow{\Phi = |W_q \cap W_i|} \frac{\Phi}{|W_q| + |W_i| - \Phi} &\geq \frac{\tau_1}{\tau_2} \\ \Rightarrow (\tau_1 + \tau_2)\Phi &\geq \tau_1|W_q| + \tau_1|W_i|, \end{aligned} \tag{4}$$

where τ_1 and τ_2 are two integers that can represent τ. Consequently, the keyword constraint is transformed into calculating Φ (set membership test) and determining whether the inequality (Eq. (4)) holds. Although it is simple to securely compute Φ with our SSMT scheme, it is non-trivial to determine Eq. (4) without leaking the values in the inequality. To tackle it, our idea is to convert the result of SSMT into a bit sequence and then introduce secure circuits (Section 3.4). In this way, we can securely determine whether Eq. (4) holds or not. If yes, we can obtain $E(1)$, otherwise $E(0)$.

From the above analysis, we know that the SSMT scheme will be used to securely compute Φ for our PPSKS scheme. However, the bloom filter length η will negatively affect the performance of our PPSKS scheme due to the inner product operation in the SSTM scheme (Eq. (3)). As discussed in Section 3.2, we can improve the efficiency of our PPSKS scheme by reducing the number of elements μ mapped into a bloom filter. In this paper, we employ a prefix encoding technique [29] to reduce μ, leading to a smaller η.

Given a value x and a range R, the prefix encoding technique [30] can encode them into two sets: $\mathcal{F}(x)$ and $\mathcal{S}(R)$, respectively. $\mathcal{F}(x) = \{x_1x_2\cdots x_w, x_1x_2\cdots x_{w-1}*, \cdots, x_1*\cdots*, **\cdots*\}$ is a set containing $w+1$ elements by continually replacing x's bit with $*$, where w is the bit length of x. $\mathcal{S}(R)$ is a set generated by extracting the minimum set of prefix elements covering the range R. For example, if we assume $x = 6$ and $R = [3, 7]$, $\mathcal{F}(6) = \{110, 11*, 1**, ***\}$ and $\mathcal{S}([3, 7]) = \{011, 1**\}$. Therefore, determining $x \in R$ is converted into checking whether $\mathcal{F}(x) \cap \mathcal{S}(R) = \emptyset$ or not. If yes, $x \notin R$, otherwise $x \in R$. As demonstrated in [30], the number of elements in $\mathcal{S}(R)$ is at most $2w-2$. Thus, we can reduce μ from all integers in the range of R to $2w-2$.

4.2.2 Description of Our PPSKS Scheme

Based on the above transformation, prefix encoding technique, and our SSMT scheme, we construct our PPSKS scheme, which is comprised of five phases: 1) system initialization; 2) data outsourcing; 3) token generation; 4) search; 5) data recovery.

System Initialization. In our PPSKS scheme, the data owner $\mathcal{O}$ initializes the whole system. First, $\mathcal{O}$ employs SSMT.setup(λ) to generate the FHE key pair (pk, sk) and γ hash functions $\mathcal{H} = \{h_1, h_2, \cdots, h_\gamma\}$. Then, $\mathcal{O}$ generates a master key mk. Next, $\mathcal{O}$ chooses a secure symmetric key encryption $\mathsf{SE}()$, e.g., AES-256, and a secure hash function $\mathsf{H}()$. After that, the data owner $\mathcal{O}$ has the following tasks:

• $\mathcal{O}$ authorizes sk to the query proxy $\mathcal{P}$ and mk to the cloud server $\mathcal{C}$, as shown in Fig. 4.

• When a query user u_i registers to the system with his/her identity id_i, $\mathcal{O}$ generates a shared key $ssk_i = \mathsf{H}(mk, id_i)$ for u_i and authorizes $\{ssk_i, \mathcal{H}\}$ to u_i.

Finally, the data owner $\mathcal{O}$ publishes $\{pk, \mathsf{SE}(), \mathsf{H}(), \gamma\}$. Note that, in our proposed schemes, all data transmissions are via secure channels.

Data Outsourcing. Assume the data owner $\mathcal{O}$ has a dataset $\mathcal{X} = \{\mathtt{x}_i = \{(\mathtt{p}_{i,x}, \mathtt{p}_{i,y}), \mathtt{W}_i\} \mid 1 \le i \le n\}$, where $(\mathtt{p}_{i,x}, \mathtt{p}_{i,y})$ is the location information and scaled into integers. Each keyword in $\mathtt{W}_i$ is also encoded into an integer. For each data record $\mathtt{x}_i = \{(\mathtt{p}_{i,x}, \mathtt{p}_{i,y}), \mathtt{W}_i\}$, the data owner $\mathcal{O}$ prepares the outsourced data with the following steps:

• *Step-1.* $\mathcal{O}$ generates $\mathcal{F}(\mathtt{p}_{i,x})$ and $\mathcal{F}(\mathtt{p}_{i,y})$ with the prefix encoding technique (see details in Section 4.2.1). Then, by invoking the SSMT.Enc algorithm, $\mathcal{O}$ constructs two encrypted bloom filters, denoted as $\mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,x}})$ and $\mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,y}})$, where $\mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,x}}) = \mathsf{SSMT.Enc}(\mathcal{F}(\mathtt{p}_{i,x}), pk, \mathcal{H})$ and $\mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,y}}) = \mathsf{SSMT.Enc}(\mathcal{F}(\mathtt{p}_{i,y}), pk, \mathcal{H})$.

• *Step-2.* Regarding the keyword set $\mathtt{W}_i$, $\mathcal{O}$ constructs an encrypted bloom filter $\mathsf{E}(\mathsf{BF}_{\mathtt{W}_i}) = \mathsf{SSMT.Enc}(\mathtt{W}_i, pk, \mathcal{H})$. Then, $\mathcal{O}$ encodes the value $|\mathtt{W}_i|$ (the number of keywords) into its bit sequence format and encrypts each bit as an FHE ciphertext. We denote the encrypted bit sequence as $\mathsf{E}(|\vec{\mathtt{W}}_i|)$.

• *Step-3.* $\mathcal{O}$ encrypts each data record $\{(\mathtt{p}_{i,x}, \mathtt{p}_{i,y}), \mathtt{W}_i\}$ as $\{(\mathsf{E}(\mathtt{p}_{i,x}), \mathsf{E}(\mathtt{p}_{i,y})), \mathsf{E}(\mathtt{W}_i)\}$, where $\mathsf{E}(\mathtt{W}_i)$ indicates encrypting each keyword in $\mathtt{W}_i$ into an FHE ciphertext. Notably, $\mathcal{O}$ will pad $\mathsf{E}(0)$ to make each $\mathsf{E}(\mathtt{W}_i)$ have φ elements, where $\varphi = \arg\max_{i\in[1,n]}(|\mathtt{W}_i|)$.

Finally, the data owner $\mathcal{O}$ outsources

$$\begin{aligned}\mathsf{E}(\mathtt{x}_i) = \{&\mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,x}}), \mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,y}}), \mathsf{E}(\mathsf{BF}_{\mathtt{W}_i}),\\ &(\mathsf{E}(\mathtt{p}_{i,x}), \mathsf{E}(\mathtt{p}_{i,y})), \mathsf{E}(\mathtt{W}_i), \mathsf{E}(|\vec{\mathtt{W}}_i|)\}\end{aligned} \tag{5}$$

to the cloud server $\mathcal{C}$, as shown in Fig. 4. To ensure the consistency of the bloom filter length, $\mathcal{O}$ sets:

$$\begin{cases} \eta_{\mathtt{p}_x} = \lceil \arg\max\limits_{i\in[1,n]} \left((2w_{\mathtt{p}_{i,x}} - 2)\cdot\gamma/\ln 2\right)\rceil \rightarrow \text{ length of } \mathsf{BF}_{\mathtt{p}_{i,x}} \\ \eta_{\mathtt{p}_y} = \lceil \arg\max\limits_{i\in[1,n]} \left((2w_{\mathtt{p}_{i,y}} - 2)\cdot\gamma/\ln 2\right)\rceil \rightarrow \text{ length of } \mathsf{BF}_{\mathtt{p}_{i,y}} \\ \eta_{\mathtt{W}} = \lceil \varphi\cdot\gamma/\ln 2\rceil \rightarrow \text{ length of } \mathsf{BF}_{\mathtt{W}_i}, \end{cases} \tag{6}$$

where $w_{\mathtt{p}_{i,x}}$ and $w_{\mathtt{p}_{i,y}}$ are the bit length of $\mathtt{p}_{i,x}$ and $\mathtt{p}_{i,y}$, respectively. Finally, $\mathcal{O}$ publishes $\{\eta_{\mathtt{p}_x}, \eta_{\mathtt{p}_y}, \eta_{\mathtt{W}}\}$.

Token Generation. When a query user u_i launches an SKS query, i.e., $\mathcal{Q} = \{\mathtt{R}_q, \mathtt{W}_q\}$ and $\{\tau_1, \tau_2\}$, where $\mathtt{R}_q = (\mathtt{R}_{q,x} = [\mathtt{p}_{q,xl}, \mathtt{p}_{q,xu}], \mathtt{R}_{q,y} = [\mathtt{p}_{q,yl}, \mathtt{p}_{q,yu}])$, the query token can be generated as follows.

• *Step-1.* u_i encodes $\mathtt{R}_{q,x}$ and $\mathtt{R}_{q,y}$ into $\mathcal{S}(\mathtt{R}_{q,x})$ and $\mathcal{S}(\mathtt{R}_{q,y})$ respectively by using the prefix encoding technique. Then, u_i maps each element in $\mathcal{S}(\mathtt{R}_{q,x})$ into a bloom filer and generates a set of bloom filters $\mathsf{BF}_{\mathtt{R}_{q,x}} = \{\mathsf{BF}_{\mathtt{R}_{q,x,1}}, \mathsf{BF}_{\mathtt{R}_{q,x,2}}, \cdots, \mathsf{BF}_{\mathtt{R}_{q,x,t_x}}\}$, where t_x is the number of elements in $\mathcal{S}(\mathtt{R}_{q,x})$, and each bloom filter has the length $\eta_{\mathtt{p}_x}$. After that, u_i chooses a set of random values, denoted as $\mathtt{r}_x$, which has $t_x\cdot\eta_{\mathtt{p}_x}$ random positive integers, i.e., $|\mathtt{r}_x| = t_x\cdot\eta_{\mathtt{p}_x}$. Next, u_i adds the random values of $\mathtt{r}_x$ into the corresponding element in $\mathsf{BF}_{\mathtt{R}_{q,x}}$. We denote the new bloom filter set as $\mathsf{BF}'_{\mathtt{R}_{q,x}}$.

Similarly, u_i can generate $\mathsf{BF}_{\mathtt{R}_{q,y}}$ that has t_y bloom filters, and each of them has the length $\eta_{\mathtt{p}_y}$. After adding a set of random values, denoted as $\mathtt{r}_y$ ($|\mathtt{r}_y| = t_y\cdot\eta_{\mathtt{p}_y}$), into $\mathsf{BF}_{\mathtt{R}_{q,y}}$, u_i can obtain a new bloom filter set $\mathsf{BF}'_{\mathtt{R}_{q,y}}$.

• *Step-2.* Regarding the query keyword set $\mathtt{W}_q$, u_i constructs a bloom filter for each keyword in $\mathtt{W}_q$. Assuming there are $t_{\mathtt{w}}$ keywords in $\mathtt{W}_q$, i.e., $t_{\mathtt{w}} = |\mathtt{W}_q|$, u_i generates $\mathsf{BF}_{\mathtt{W}_q} = \{\mathsf{BF}_{\mathtt{W}_{q,1}}, \mathsf{BF}_{\mathtt{W}_{q,2}}, \cdots, \mathsf{BF}_{\mathtt{W}_{q,t_{\mathtt{w}}}}\}$, and each bloom filter has the length $\eta_{\mathtt{W}}$. Then, u_i chooses $t_{\mathtt{w}}\cdot\eta_{\mathtt{W}}$ random positive integers, organized as $\mathtt{r}_{\mathtt{w}} = (r_{1,1}, r_{1,2}, \cdots, r_{1,\eta_{\mathtt{W}}}, \cdots, r_{t_{\mathtt{w}},1}, r_{t_{\mathtt{w}},2}, \cdots, r_{t_{\mathtt{w}},\eta_{\mathtt{W}}})$, and adds them into each bit of $\mathsf{BF}_{\mathtt{W}_q}$. We denote the new bloom filters as $\mathsf{BF}'_{\mathtt{W}_q} = \{\mathsf{BF}'_{\mathtt{W}_{q,1}}, \mathsf{BF}'_{\mathtt{W}_{q,2}}, \cdots, \mathsf{BF}'_{\mathtt{W}_{q,t_{\mathtt{w}}}}\}$.

Besides, u_i encodes $t_{\mathtt{w}}\cdot\tau_1$, τ_1, and $\tau_1+\tau_2$ into bit sequences, denoted as $\vec{\tau}_t$, $\vec{\tau}_1$, and $\vec{\tau}_2$, where the lengths of $\{\vec{\tau}_t, \vec{\tau}_1, \vec{\tau}_2\}$ are $\{l_t, l_1, l_2\}$. Next, u_i generates three random sets: $\mathtt{r}_{\tau_t} = (r_{\tau_t,1}, r_{\tau_t,2}, \cdots, r_{\tau_t,l_t})$, $\mathtt{r}_{\tau_1} = (r_{\tau_1,1}, r_{\tau_1,2}, \cdots, r_{\tau_1,l_1})$, and $\mathtt{r}_{\tau_2} = (r_{\tau_2,1}, r_{\tau_2,2}, \cdots, r_{\tau_2,l_2})$ and adds them into $\vec{\tau}_t$, $\vec{\tau}_1$, and $\vec{\tau}_2$, respectively. The randomized bit sequences are denoted as $\{\vec{\tau}'_t, \vec{\tau}'_1, \vec{\tau}'_2\}$.

• *Step-3.* With the authorized ssk_i, u_i first computes a session key $ss_i = \mathsf{H}(ssk_i, ts)$, where ts is the timestamp. Then, using the session key ss_i, u_i encrypts the random sets $\{\mathtt{r}_x, \mathtt{r}_y, \mathtt{r}_{\mathtt{w}}, \mathtt{r}_{\tau_t}, \mathtt{r}_{\tau_1}, \mathtt{r}_{\tau_2}\}$ into:

$$\mathtt{SER} = \mathsf{SE}_{ss_i}(\mathtt{r}_x||\mathtt{r}_y||\mathtt{r}_{\mathtt{w}}||\mathtt{r}_{\tau_t}||\mathtt{r}_{\tau_1}||\mathtt{r}_{\tau_2})$$

and sends the following encoded query request:

$$\{\mathsf{BF}'_{\mathtt{R}_{q,x}}, \mathsf{BF}'_{\mathtt{R}_{q,y}}, \mathsf{BF}'_{\mathtt{W}_q}, \vec{\tau}'_t, \vec{\tau}'_1, \vec{\tau}'_2, \mathtt{SER}, id_i, ts\}$$

to the query proxy $\mathcal{P}$. Here, since we add random values into each bloom filter and bit sequence, it can prevent $\mathcal{P}$ from inferring the original query request $\mathcal{Q}$ and $\{\tau_1, \tau_2\}$.

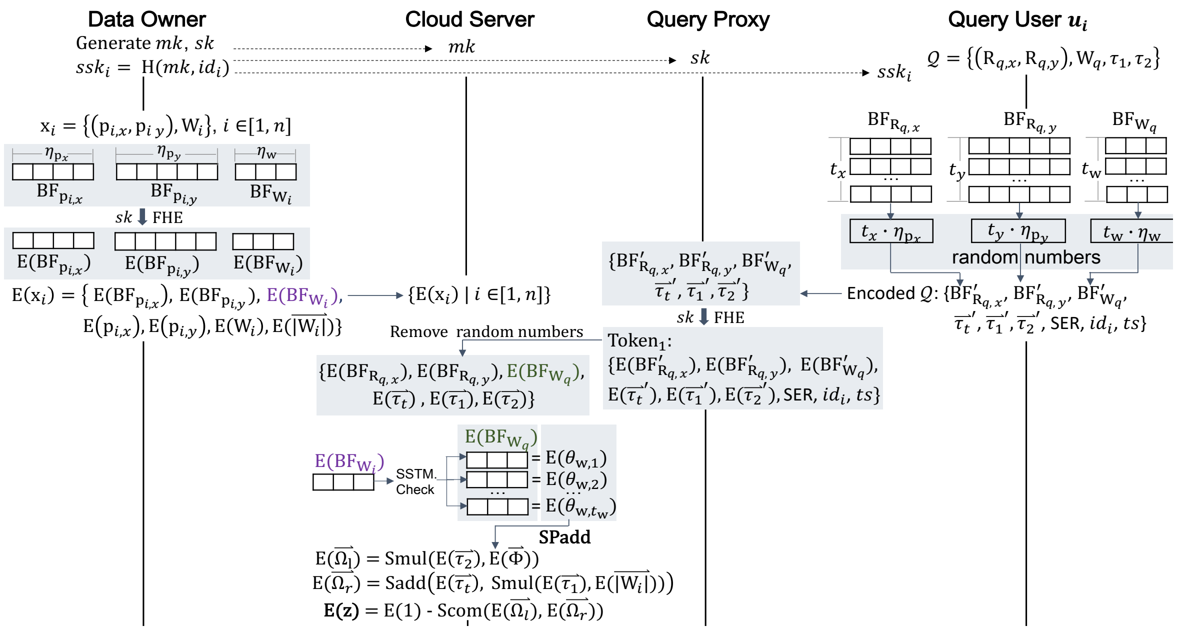

Data Owner
Cloud Server
Query Proxy
Query User u_i
Generate mk, sk
ssk_i = H(mk, id_i)
mk
sk
ssk_i
$Q = \{(R_{q,x}, R_{q,y}), W_q, \tau_1, \tau_2\}$
$x_i = \{(p_{i,x}, p_{i,y}), W_i\}, i \in [1, n]$
$BF_{p_{i,x}}$ $BF_{p_{i,y}}$ BF_{W_i}
sk FHE
$E(BF_{p_{i,x}})$ $E(BF_{p_{i,y}})$ $E(BF_{W_i})$
$E(x_i) = \{E(BF_{p_{i,x}}), E(BF_{p_{i,y}}), E(BF_{W_i}), E(p_{i,x}), E(p_{i,y}), E(W_i), E(\overrightarrow{|W_i|})\}$
$\{E(x_i) \mid i \in [1, n]\}$
$BF_{R_{q,x}}$ $BF_{R_{q,y}}$ BF_{W_q}
$t_x \cdot \eta_{p_x}$ $t_y \cdot \eta_{p_y}$ $t_w \cdot \eta_w$
random numbers
Encoded Q: $\{BF'_{R_{q,x}}, BF'_{R_{q,y}}, BF'_{W_q}, \overrightarrow{\tau_t}', \overrightarrow{\tau_1}', \overrightarrow{\tau_2}'$, SER, id_i, $ts\}$
sk FHE
Token$_1$: $\{E(BF'_{R_{q,x}}), E(BF'_{R_{q,y}}), E(BF'_{W_q}), E(\overrightarrow{\tau_t}'), E(\overrightarrow{\tau_1}'), E(\overrightarrow{\tau_2}')$, SER, id_i, $ts\}$
Remove random numbers
$\{E(BF_{R_{q,x}}), E(BF_{R_{q,y}}), E(BF_{W_q}), E(\overrightarrow{\tau_t}), E(\overrightarrow{\tau_1}), E(\overrightarrow{\tau_2})\}$
SSTM. Check
$= E(\theta_{w,1})$ $= E(\theta_{w,2})$ $= E(\theta_{w,t_w})$
SPadd
$E(\overrightarrow{\Omega_l})$ = Smul(E($\overrightarrow{\tau_2}$), E($\overrightarrow{\Phi}$))
$E(\overrightarrow{\Omega_r})$ = Sadd(E($\overrightarrow{\tau_t}$), Smul(E($\overrightarrow{\tau_1}$), E($\overrightarrow{|W_i|}$)))
E(z) = E(1) - Scom(E($\overrightarrow{\Omega_l}$), E($\overrightarrow{\Omega_r}$))

Fig. 4. The main process of PPSKS, in which SER contains the encrypted random numbers generated by query user u_i and can be decrypted by $\mathcal{C}$ with mk. Notably, on the cloud server $\mathcal{C}$ side, we only take the calculation of $\mathsf{E}(z)$ as an example, in which $z = 1$ indicates the query keyword set and the data keyword set satisfy the keyword constraint, i.e., Eq. (4) holds.

• *Step-4.* Upon receiving the encoded query request, $\mathcal{P}$ can encrypt bloom filters and bit sequences into $\{\mathsf{E}(\mathsf{BF}'_{\mathtt{R}_{q,x}}), \mathsf{E}(\mathsf{BF}'_{\mathtt{R}_{q,y}}), \mathsf{E}(\mathsf{BF}'_{\mathtt{W}_q}), \mathsf{E}(\vec{\tau}'_t), \mathsf{E}(\vec{\tau}'_1), \mathsf{E}(\vec{\tau}'_2)\}$ with the FHE secret key sk. Then, $\mathcal{P}$ forwards the query token:

$$\begin{aligned}\mathtt{Token}_1 = \{&\mathsf{E}(\mathsf{BF}'_{\mathtt{R}_{q,x}}), \mathsf{E}(\mathsf{BF}'_{\mathtt{R}_{q,y}}), \mathsf{E}(\mathsf{BF}'_{\mathtt{W}_q}),\\ &\mathsf{E}(\vec{\tau}'_t), \mathsf{E}(\vec{\tau}'_1), \mathsf{E}(\vec{\tau}'_2), \mathtt{SER}, id_i, ts\},\end{aligned} \tag{7}$$

to the cloud server $\mathcal{C}$. See the process in Fig. 4.

Search. After receiving $\mathtt{Token}_1$, $\mathcal{C}$ first calculates the session key ss_i with the authorized master key mk, i.e., $ss_i = \mathsf{H}(\mathsf{H}(mk, id_i), ts)$. Then, $\mathcal{C}$ can recover $\{\mathtt{r}_x, \mathtt{r}_y, \mathtt{r}_\mathtt{w}, \mathtt{r}_{\tau_t}, \mathtt{r}_{\tau_1}, \mathtt{r}_{\tau_2}\}$ from SER with ss_i. Next, $\mathcal{C}$ exploits the homomorphic properties of FHE to remove these random sets $\{\mathtt{r}_x, \mathtt{r}_y, \mathtt{r}_\mathtt{w}, \mathtt{r}_{\tau_t}, \mathtt{r}_{\tau_1}, \mathtt{r}_{\tau_2}\}$ from $\{\mathsf{E}(\mathsf{BF}'_{\mathtt{R}_{q,x}}), \mathsf{E}(\mathsf{BF}'_{\mathtt{R}_{q,y}}), \mathsf{E}(\mathsf{BF}'_{\mathtt{W}_q}), \mathsf{E}(\vec{\tau}'_t), \mathsf{E}(\vec{\tau}'_1), \mathsf{E}(\vec{\tau}'_2)\}$ and get:

$$\{\mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,x}}), \mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,y}}), \mathsf{E}(\mathsf{BF}_{\mathtt{W}_q}), \mathsf{E}(\vec{\tau}_t), \mathsf{E}(\vec{\tau}_1), \mathsf{E}(\vec{\tau}_2)\}.$$

After that, the cloud server $\mathcal{C}$ can compute an encrypted flag set $\mathsf{E}(\mathtt{S}_f) = \{\mathsf{E}(\mathtt{f}_i) \mid i \in [1, n]\}$, in which $\mathtt{f}_i = 1$ if the data record $\mathtt{x}_i$ satisfies the query $\mathcal{Q}$, otherwise $\mathtt{f}_i = 0$. The concrete process is as follows.

• *Step-1.* $\mathcal{C}$ invokes SSMT.Interpolation(γ) to generate a polynomial function $f(x)$.

• *Step-2.* $\mathcal{C}$ first checks whether $\mathtt{p}_{i,x} \in \mathtt{R}_{q,x}$ with the SSMT.Check algorithm. For each bloom filter $\mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,x,j}})$ in $\mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,x}})$, where $j \in [1, t_x]$, the cloud server $\mathcal{C}$ obtains:

$$\mathsf{E}(\theta_{x,j}) = \mathsf{SSMT.Check}\left(\mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,x}}), \mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,x,j}}), f(x)\right).$$

Afterward, $\mathcal{C}$ computes $\mathsf{E}(\theta_x) = \sum_{j=1}^{t_x} \mathsf{E}(\theta_{x,j})$ to represent whether $\mathtt{p}_{i,x} \in \mathtt{R}_{q,x}$. If yes, $\mathsf{E}(\theta_x) = \mathsf{E}(1)$, otherwise $\mathsf{E}(0)$. This is because, for the prefix encoding technique, there is only one common element for $\mathcal{F}(x)$ and $\mathcal{S}(R)$ when $x \in R$, i.e., $|\mathcal{F}(x) \cap \mathcal{S}(R)| = 1$. Similarly, $\mathcal{C}$ can obtain $\mathsf{E}(\theta_y)$ to indicate whether $\mathtt{p}_{i,y} \in \mathtt{R}_{q,y}$ or not by computing $\sum_{j=1}^{t_y} \mathsf{SSMT.Check}\left(\mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,y}}), \mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,y,j}}), f(x)\right)$.

Algorithm 1 Calculating $\mathsf{E}(\vec{\Phi})$

```
Input: An encrypted data, $\mathsf{E}(\mathtt{x}_i)$. An encrypted query token, $\mathtt{Token}_1$.
       The maximum bit length of secure circuits $l_{\max}$;
Output: The number of intersecting keywords, $\mathsf{E}(\vec{\Phi})$;
 1: $\mathsf{E}(\vec{\Phi}) \leftarrow \{\mathsf{E}(0)|^{0}_{j=l_{\max}-1}\}$;
 2: for each $\mathsf{E}(\mathsf{BF}_{\mathtt{W}_{q,j}})$ in $\mathsf{E}(\mathsf{BF}_{\mathtt{W}_q})$ do
 3:     $\mathsf{E}(\theta_{\mathtt{w},j}) \leftarrow \mathsf{SSMT.Check}\left(\mathsf{E}(\mathsf{BF}_{\mathtt{W}_i}), \mathsf{E}(\mathsf{BF}_{\mathtt{W}_{q,j}}), f(x)\right)$;
 4:     $\mathsf{E}(\vec{\Phi}) \leftarrow SPadd\left(\mathsf{E}(\vec{\Phi}), \mathsf{E}(\theta_{\mathtt{w},j}), l_{\max}\right)$;

 5: $SPadd(\mathsf{E}(\vec{x}), \mathsf{E}(\theta), l_{\max})$
 6:     Ecarry $\leftarrow \mathsf{E}(\theta)$;
 7:     for $j \leftarrow 0$ to $l_{\max}-1$ do
 8:         Esum $\leftarrow \mathsf{E}(x^j)$ + Ecarry;   Emul $\leftarrow \mathsf{E}(x^j) \cdot$ Ecarry;
 9:         $\mathsf{E}(z^j) \leftarrow$ Esum + 2 · Emul · $\mathsf{E}(-1)$
10:         Ecarry $\leftarrow$ Emul;
11:     return $\mathsf{E}(\vec{z}) \leftarrow (\mathsf{E}(z^j)|^{0}_{j=l_{\max}-1})$;
```

• *Step-3.* Regarding the keyword set, $\mathcal{C}$ first obtains $\mathsf{E}(\theta_{\mathtt{w},j}) = \mathsf{SSMT.Check}\left(\mathsf{E}(\mathsf{BF}_{\mathtt{W}_i}), \mathsf{E}(\mathsf{BF}_{\mathtt{W}_{q,j}}), f(x)\right)$, where $j \in [1, t_\mathtt{w}]$. After calculating $\mathsf{E}(\Phi) = \sum_{j=1}^{t_\mathtt{w}} \mathsf{E}(\theta_{\mathtt{w},j})$, $\mathcal{C}$ needs to determine whether Eq. (4) holds or not. If yes, $\mathsf{E}(1)$ is produced, otherwise $\mathsf{E}(0)$. However, since all values in Eq. (4) are encrypted, it is hard to directly execute the comparison. Our solution is to introduce secure circuits by providing $\mathsf{E}(|\vec{\mathtt{W}}_i|)$ and $\{\mathsf{E}(\vec{\tau}_t), \mathsf{E}(\vec{\tau}_1), \mathsf{E}(\vec{\tau}_2)\}$. Nevertheless, there is still a gap between $\mathsf{E}(\Phi)$ and $\mathsf{E}(\vec{\Phi})$ as the prerequisite of using secure circuits is to convert $\mathsf{E}(\Phi)$ into its bit sequence format $\mathsf{E}(\vec{\Phi})$ without decryption. To tackle it, we devise a secure partial addition circuit, denoted as *SPadd*, and illustrate the conversion in Algorithm 1.

With $\mathsf{E}(\vec{\Phi})$, $\mathcal{C}$ can obtain the left value of Eq. (4), denoted as Ω_l ($\Omega_l = (\tau_1 + \tau_2) \cdot \Phi$), using *Smul*, i.e., $\mathsf{E}(\vec{\Omega}_l) = Smul\left(\mathsf{E}(\vec{\tau}_2), \mathsf{E}(\vec{\Phi})\right)$, where $\vec{\tau}_2$ is the bit sequence of $\tau_1 + \tau_2$. Then, with $\mathsf{E}(\vec{\tau}_1)$, $\mathsf{E}(\vec{\tau}_t)$, and $\mathsf{E}(|\vec{\mathtt{W}}_i|)$, $\mathcal{C}$ can obtain the right value of Eq. (4), denoted as Ω_r ($\Omega_r = \tau_1|\mathtt{W}_q| + \tau_1|\mathtt{W}_i|$), using *Smul* and *Sadd*, i.e., $\mathsf{E}(\vec{\Omega}_r) = Sadd\left(\mathsf{E}(\vec{\tau}_t), Smul\left(\mathsf{E}(\vec{\tau}_1), \mathsf{E}(|\vec{\mathtt{W}}_i|)\right)\right)$, where $\vec{\tau}_t = \tau_1 \cdot t_\mathtt{w} = \tau_1 \cdot |\mathtt{W}_q|$. Finally, $\mathcal{C}$ uses *Scom* to check Eq. (4), i.e., $\mathsf{E}(z) =$

Scom$\big(\mathsf{E}(\vec{\Omega}_l), \mathsf{E}(\vec{\Omega}_r)\big)$. If Eq. (4) holds, $z = 1$, otherwise $z = 0$. We depict the process of computing $\mathsf{E}(z)$ in Fig. 4. Note that here we need to flip z by computing $\mathsf{E}(z) = \mathsf{E}(1 - z)$ since Eq. (4) requires "$\geq$", while *Scom* ensures "$<$".

• *Step-4.* $\mathcal{C}$ calculates $\mathsf{E}(\mathtt{f}_i) = \mathsf{E}(\theta_x) \cdot \mathsf{E}(\theta_y) \cdot \mathsf{E}(z) = \mathsf{E}(\theta_x \cdot \theta_y \cdot z)$, where $\mathsf{E}(\theta_x)$ and $\mathsf{E}(\theta_y)$ have been calculated in *Step-2*. Then, $\mathcal{C}$ sends $\mathsf{E}(\mathtt{S}_f) = \{\mathsf{E}(\mathtt{f}_i) \mid i \in [1, n]\}$ to the query proxy $\mathcal{P}$ after applying permutation π on $\mathsf{E}(\mathtt{S}_f)$. In Fig. 5, we give an example of $\mathsf{E}(\mathtt{S}_f)$ with $n = 5$, in which we assume the second and fifth data records satisfy the query $\mathcal{Q}$.

• *Step-5.* $\mathcal{P}$ first recovers $\mathtt{S}_f = \{\mathtt{f}_i \mid i \in [1, n]\}$ with the secret key sk. If there are k data records satisfying the query $\mathcal{Q}$, $\mathcal{P}$ constructs k encrypted vectors $\{\mathsf{E}(\mathtt{v}_1), \mathsf{E}(\mathtt{v}_2), \cdots, \mathsf{E}(\mathtt{v}_k)\}$, where $\mathsf{E}(\mathtt{v}_j) = \{\mathsf{E}(v_{j,1}), \mathsf{E}(v_{j,2}), \cdots, \mathsf{E}(v_{j,n})\}$, $j \in [1, k]$. For an encrypted vector $\mathsf{E}(\mathtt{v}_j)$, only one element is $\mathsf{E}(1)$, and others are $\mathsf{E}(0)$. After that, $\mathcal{P}$ forwards $\{\mathsf{E}(\mathtt{v}_j) \mid j \in [1, k]\}$ to the cloud server $\mathcal{C}$. In the example of Fig. 5, since there are two data records satisfying the query $\mathcal{Q}$, the query proxy $\mathcal{P}$ generates two encrypted vectors $\{\mathsf{E}(\mathtt{v}_1), \mathsf{E}(\mathtt{v}_2)\}$. If $k = 0$, $\mathcal{P}$ returns $\perp$ to the query user u_i.

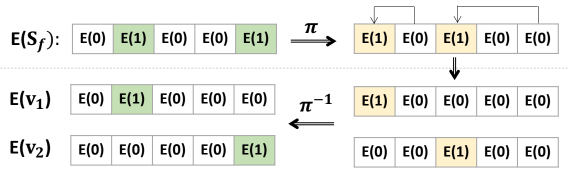


Fig. 5. An example of constructing $\{\mathsf{E}(\mathtt{v}_j) \mid j \in [1, k]\}$, in which there are five data records, and the second and fifth ones satisfy the query $\mathcal{Q}$.

• *Step-6.* Upon receiving $\{\mathsf{E}(\mathtt{v}_j) \mid j \in [1, k]\}$, the cloud server $\mathcal{C}$ applies the inverse permutation π^{-1} on each vector $\mathsf{E}(\mathtt{v}_j)$ and then calculates the k encrypted results as follows:

$$\begin{cases} \mathsf{E}(\mathtt{p}_{j,x}) = \sum_{i=1}^{n} \big(\mathsf{E}(\mathtt{v}_{j,i}) \cdot \mathsf{E}(\mathtt{p}_{i,x})\big) = \mathsf{E}(\sum_{i=1}^{n} \big(\mathtt{v}_{j,i} \cdot \mathtt{p}_{i,x})\big) \\ \mathsf{E}(\mathtt{p}_{j,y}) = \sum_{i=1}^{n} \big(\mathsf{E}(\mathtt{v}_{j,i}) \cdot \mathsf{E}(\mathtt{p}_{i,y})\big) = \mathsf{E}(\sum_{i=1}^{n} \big(\mathtt{v}_{j,i} \cdot \mathtt{p}_{i,y})\big) \\ \mathsf{E}(\mathtt{W}_j) = \sum_{i=1}^{n} \big(\mathsf{E}(\mathtt{v}_{j,i}) \cdot \mathsf{E}(\mathtt{W}_i)\big) = \mathsf{E}(\sum_{i=1}^{n} \big(\mathtt{v}_{j,i} \cdot \mathtt{W}_i)\big), \end{cases} \quad (8)$$

where $j \in [1, k]$. After that, $\mathcal{C}$ chooses $2 + \varphi$ random positive integers: $\mathtt{r}_j = (r_{j,x}, r_{j,y}, r_{j,1}, r_{j,2}, \cdots, r_{j,\varphi})$ for each encrypted result, in which $r_{j,x}$ is added into $\mathsf{E}(\mathtt{p}_{j,x})$, $r_{j,y}$ is added into $\mathsf{E}(\mathtt{p}_{j,y})$, and $\{r_{j,1}, r_{j,2}, \cdots, r_{j,\varphi}\}$ are added into encrypted keywords $\mathsf{E}(\mathtt{W}_j)$. We denote the new result as $\{\mathsf{E}(\mathtt{p}'_{j,x}), \mathsf{E}(\mathtt{p}'_{j,y}), \mathsf{E}(\mathtt{W}'_j)\}$. After encrypting $\mathtt{r}_j$ with the query user's session key ss_i, the cloud server $\mathcal{C}$ sends

$$\mathtt{ERes} = \{\big(\mathsf{E}(\mathtt{p}'_{j,x}), \mathsf{E}(\mathtt{p}'_{j,y})\big), \mathsf{E}(\mathtt{W}'_j), \mathsf{SE}_{ss_i}(\mathtt{r}_j), id_i \mid j \in [1, k]\}$$

to the query proxy $\mathcal{P}$.

Data Recovery. Upon receiving $\mathtt{ERes}$, $\mathcal{P}$ first recovers $\{(\mathtt{p}'_{j,x}, \mathtt{p}'_{j,y}), \mathtt{W}'_j \mid j \in [1, k]\}$ with the FHE secret key sk. After that, $\mathcal{P}$ forwards the query result:

$$\mathtt{Res}' = \{(\mathtt{p}'_{j,x}, \mathtt{p}'_{j,y}), \mathtt{W}'_j, \mathsf{SE}_{ss_i}(\mathtt{r}_j) \mid j \in [1, k]\}$$

to the query user u_i. With $\mathtt{Res}'$, u_i first recovers $\mathtt{r}_j$ using his/her session key ss_i. Finally, u_i obtains the desired result: $\mathtt{Res} = \{(\mathtt{p}_{j,x}, \mathtt{p}_{j,y}), \mathtt{W}_j \mid j \in [1, k]\}$ by removing $\mathtt{r}_j$.

Remark. Our PPSKS scheme is the first to consider the keyword set similarity while hiding access patterns. To achieve it, there are three key points: i) transforming the Jaccard similarity into the combination of set membership test and a simple *compute-then-compare* operation; ii) designing the SSMT scheme that uses the Lagrange interpolation function to map encrypted values into $\mathsf{E}(0)$ or $\mathsf{E}(1)$; iii) devising *SPadd* to convert $\mathsf{E}(\Phi)$ into $\mathsf{E}(\vec{\Phi})$ without decryption and introducing secure circuits. Although our PPSKS scheme leaks the number of data records for the query result, it is a trivial leakage, and we will prove that it is secure against the semi-honest cloud server and query proxy in Section 6.

4.3 Tree-based Construction: Our PPSKS+ Scheme

4.3.1 Overview of Our PPSKS+ Scheme

Although our PPSKS scheme can hide access patterns among n data records, its performance is limited to be linear to the dataset size. To improve efficiency, we propose a tree-based privacy-preserving SKS query scheme, denoted as PPSKS+, to attain a sublinear efficiency. Specifically, we first design an index that we call FR-tree by modifying R*-tree [31]. Then, we encrypt the FR-tree by adopting a predicate encryption technique for non-leaf nodes and Eq. (5) for leaf nodes. Finally, we employ a vector bucketing technique to improve the efficiency of our PPSKS+ scheme.

Build FR-tree over plaintexts. Same as R*-tree [31], our FR-tree groups nearby spatial data and represents them with their *minimum bounding rectangle* (MBR) in the higher level of the tree, as shown in Fig. 6. Different from R*-tree, however, our FR-tree counts the keyword frequency of the given dataset $\mathcal{X}$ and sorts them with the decreasing order according to the frequencies. Then, for each keyword set $\mathtt{W}_i$, we can build a bitmap, which is denoted as BM_i and has the length ρ as the same as the number of unique keywords in the dataset. Next, our FR-tree merges BM_i from bottom to top. For instance, in Fig. 6, BM_9 and BM_{10} are merged into a new bitmap $\{1, 1, 0, 1, 0, 1, 0, 1, 0\}$. In this way, we can effectively filter out the data records whose keyword sets do not satisfy the constraint of Jaccard similarity [32]. The details of our FR-tree are shown as follows.

• Root node. The root node must intersect with a spatial keyword query in both spatial and keyword domains. Therefore, we ignore the root node and store nothing in it.

• Non-leaf nodes. Each non-leaf node represents an MBR containing two components: i) the spatial rectangle of MBR $\mathtt{R} = (\mathtt{R}_x, \mathtt{R}_y)$, where $\mathtt{R}_x = [\mathtt{p}_{xl}, \mathtt{p}_{xu}]$ and $\mathtt{R}_y = [\mathtt{p}_{yl}, \mathtt{p}_{yu}]$; ii) the merged bitmap BM.

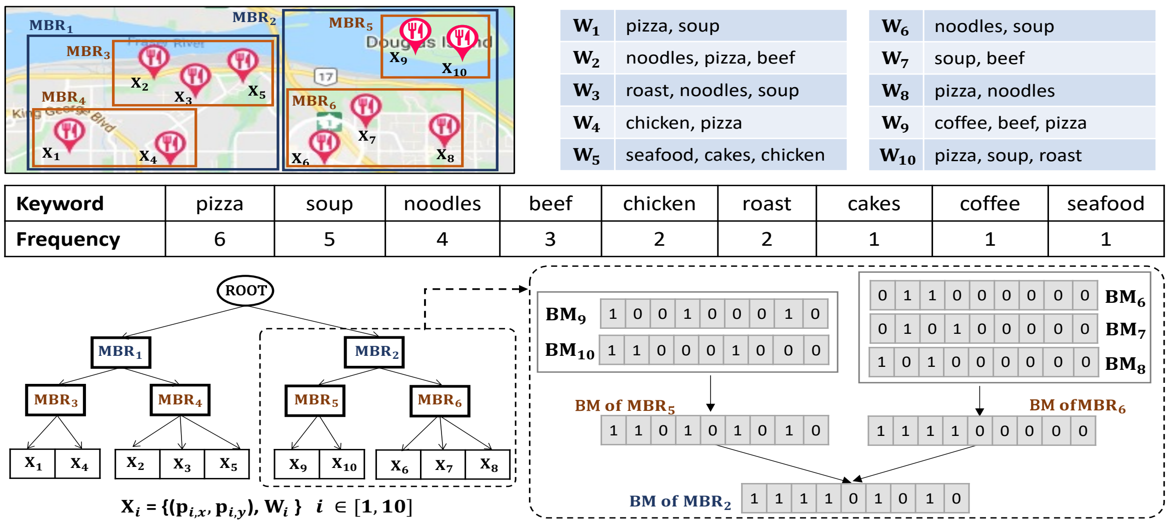
• Leaf nodes. Each leaf node represents a spatial keyword data record $\mathtt{x}_i = \{(\mathtt{p}_{i,x}, \mathtt{p}_{i,y}), \mathtt{W}_i\}$. Note that since the bitmap of $\mathtt{W}_i$ is only for generating merged bitmaps, we do not store it in the corresponding leaf node.

Given a query $\mathcal{Q} = \{\mathtt{R}_q, \mathtt{W}_q\}$ and an MBR=$\{\mathtt{R}, \mathsf{BM}\}$, if $\mathtt{R}_q$ intersects with $\mathtt{R}$, and the inner product of the bitmap BM_q (generated by $\mathtt{W}_q$) and the merged bitmap BM is larger than 0, we need to access the MBR's children. Formally, the condition of accessing MBR's children is to check:

$$(\mathtt{R} \cap \mathtt{R}_q \neq \varnothing) \wedge (\mathsf{BM} \circ \mathsf{BM}_q > 0). \quad (9)$$

Since our keyword constraint is the Jaccard similarity, we can modify the above condition and adopt a more reasonable and stricter condition:

$$(\mathtt{R} \cap \mathtt{R}_q \neq \varnothing) \wedge (\mathsf{BM} \circ \mathcal{E}(\mathsf{BM}_q) > 0), \quad (10)$$



MBR₁
MBR₂
MBR₃
MBR₄
MBR₅
MBR₆
x₁ x₂ x₃ x₄ x₅ x₆ x₇ x₈ x₉ x₁₀
ROOT
BM₉ 1 0 0 1 0 0 0 1 0
BM₁₀ 1 1 0 0 0 1 0 0 0
0 1 1 0 0 0 0 0 0 BM₆
0 1 0 1 0 0 0 0 0 BM₇
1 0 1 0 0 0 0 0 0 BM₈
BM of MBR₅ 1 1 0 1 0 1 0 1 0
BM of MBR₆ 1 1 1 1 0 0 0 0 0
BM of MBR₂ 1 1 1 1 0 1 0 1 0
X_i = {(p_{i,x}, p_{i,y}), W_i} i ∈ [1, 10]

W_1	pizza, soup	W_6	noodles, soup
W_2	noodles, pizza, beef	W_7	soup, beef
W_3	roast, noodles, soup	W_8	pizza, noodles
W_4	chicken, pizza	W_9	coffee, beef, pizza
W_5	seafood, cakes, chicken	W_{10}	pizza, soup, roast

Keyword	pizza	soup	noodles	beef	chicken	roast	cakes	coffee	seafood
Frequency	6	5	4	3	2	2	1	1	1

Fig. 6. An example of FR-tree, in which the keyword dictionary size is 9, i.e., $\rho = 9$.

where $\mathcal{E}(\mathsf{BM}_q)$ means removing the first $t_{\mathtt{w}}-(\lfloor(1-\tau)t_{\mathtt{w}}\rfloor+1)$ elements in $\mathtt{W}_q$ before mapping $\mathtt{W}_q$'s elements into BM_q. See details in [32]. Here, $t_{\mathtt{w}} = |\mathtt{W}_q|$, and τ is a similarity threshold.

Encrypt FR-tree. In our FR-tree, each leaf node is encrypted as Eq. (5). For encrypting the MBRs of non-leaf nodes, we propose a predicate encryption technique by modifying the hyper-rectangle intersection predicate encryption used in [18].

On the one hand, with regard to the spatial rectangle, our main idea is based on the following transformation:

$$\mathtt{R}\cap\mathtt{R}_q\neq\varnothing \Leftrightarrow \begin{cases}(\mathtt{p}_{xl}\in[1,\mathtt{p}_{q,xu}])\wedge(\mathtt{p}_{xu}\in[\mathtt{p}_{q,xl},\mathtt{T}_x])\\ (\mathtt{p}_{yl}\in[1,\mathtt{p}_{q,yu}])\wedge(\mathtt{p}_{yu}\in[\mathtt{p}_{q,yl},\mathtt{T}_y]),\end{cases} \tag{11}$$

where $\mathtt{T}_x$ and $\mathtt{T}_y$ are the upper bound of x and y dimensions, respectively. Consequently, we can convert $\mathtt{R}$ into a vector $\mathtt{V}_{\mathtt{R}}$ and encrypt $\mathtt{V}_{\mathtt{R}}$ into the corresponding ciphertext $\mathtt{CV}_{\mathtt{R}}$ by using the matrix encryption, i.e., $\mathtt{CV}_{\mathtt{R}} = \mathtt{V}_{\mathtt{R}}\mathsf{M}$, where M is a random invertible matrix. Given a query rectangle $\mathtt{R}_q$, we convert it into a vector $\mathtt{V}_{\mathtt{R}_q}$ and encrypt $\mathtt{V}_{\mathtt{R}_q}$ into $\mathtt{CV}_{\mathtt{R}_q} = \mathtt{V}_{\mathtt{R}_q}(\mathsf{M}^{-1})^T$. Here T means transposition. Finally, we can check whether $\mathtt{CV}_{\mathtt{R}}\circ\mathtt{CV}_{\mathtt{R}_q} = \mathtt{V}_{\mathtt{R}}\circ\mathtt{V}_{\mathtt{R}_q} > 0$ or not. If yes, it means $\mathtt{R}\cap\mathtt{R}_q\neq\varnothing$, otherwise $\mathtt{R}_q\cap\mathtt{R}=\varnothing$.

On the other hand, we can determine whether the keyword condition is satisfied by computing $\mathtt{CBM}\circ\mathtt{CBM}_{\mathtt{q}}$, where $\mathtt{CBM} = \mathsf{BM}\cdot\mathsf{M}$ and $\mathtt{CBM}_{\mathtt{q}} = \mathcal{E}(\mathsf{BM}_q)\cdot(\mathsf{M}^{-1})^T$. If $\mathtt{CBM}\circ\mathtt{CBM}_{\mathtt{q}} = \mathsf{BM}\circ\mathcal{E}(\mathsf{BM}_q) > 0$, the keyword condition is satisfied, otherwise not.

However, we cannot directly use the above approach to determine whether an MBR's children should be accessed. That is because the above approach leaks the privacy about which constraint (spatial or keyword) leads Eq. (10) to fail. Here, we term it as conjunctive privacy. Such a kind of privacy leakage has been penetratingly analyzed in [9]. To tackle it, we propose a random mask solution by constructing an integrated vector $\mathtt{V}_{\mathtt{R},\mathtt{BM}}$ with the size of $2(\mathtt{T}_x+\mathtt{T}_y)+1+\rho+d$, where d is the number of dummy values, and encrypting it into $\mathtt{CV}_{\mathtt{R},\mathtt{BM}} = \mathtt{V}_{\mathtt{R},\mathtt{BM}}\cdot\mathsf{M}$. Specifically,

$$\mathtt{V}_{\mathtt{R},\mathtt{BM}} = (\mathtt{r}_{\mathtt{s}}\alpha(\mathtt{R}_x), \mathtt{r}_{\mathtt{s}}\alpha(\mathtt{R}_y), -\mathtt{r}_{\mathtt{s}}, \mathtt{r}_{\mathtt{k}}\mathsf{BM}, -\mathtt{r}_1, -\mathtt{r}_2, \cdots, -\mathtt{r}_d),$$

where $\mathtt{r}_{\mathtt{s}}, \mathtt{r}_{\mathtt{k}}$, and $\mathtt{r}_i$ $(i\in[1,d])$ are random real numbers satisfying $\mathtt{r}_{\mathtt{s}} > \mathtt{r}_{\mathtt{k}} > \sum_{i=1}^{d}\mathtt{r}_i$. If we let $*\in\{x,y\}$, we have:

$$\alpha(\mathtt{R}_*)_{j\in[1,2\mathtt{T}_*]} = \begin{cases}1 & \text{if } j=\mathtt{p}_{*l} \text{ or } j=\mathtt{p}_{*u}+\mathtt{T}_*\\ 0 & \text{Otherwise.}\end{cases} \tag{12}$$

Given a query $\mathcal{Q}$ containing $\mathtt{R}_q$ and $\mathcal{E}(\mathsf{BM}_q)$, it is encoded as a $2(\mathtt{T}_x+\mathtt{T}_y)+1+\rho+d$ dimensional vector $\mathtt{V}_{\mathtt{R}_q,\mathtt{BM}_q}$ and then encrypted into $\mathtt{CV}_{\mathtt{R}_q,\mathtt{BM}_q} = \mathtt{V}_{\mathtt{R}_q,\mathtt{BM}_q}\cdot(\mathsf{M}^{-1})^T$.

$$\mathtt{V}_{\mathtt{R}_q,\mathtt{BM}_q} = (\mathtt{r}_{\mathtt{q}}\beta(\mathtt{R}_{q,x}), \mathtt{r}_{\mathtt{q}}\beta(\mathtt{R}_{q,y}), 4\mathtt{r}_{\mathtt{q}}, \mathtt{r}_{\mathtt{t}}\mathcal{E}(\mathsf{BM}_{\mathtt{q}}), \mathtt{r}'_1, \mathtt{r}'_2, \cdots, \mathtt{r}'_d),$$

where $\mathtt{r}_{\mathtt{q}}, \mathtt{r}_{\mathtt{t}}$, and $\mathtt{r}'_i$ $(i\in[1,d])$ are random real numbers satisfying $\mathtt{r}_{\mathtt{q}} > \mathtt{r}_{\mathtt{t}}\cdot\phi > \sum_{i=1}^{d}\mathtt{r}'_i$, and ϕ is the number of element 1 in $\mathcal{E}(\mathsf{BM}_{\mathtt{q}})$. If we let $*\in\{x,y\}$, we have:

$$\beta(\mathtt{R}_{q,*})_{j\in[1,2\mathtt{T}_*]} = \begin{cases}1 & \text{if } j\in[1,\mathtt{p}_{q,*u}]\\ & \text{or } j\in[\mathtt{p}_{q,*l}+\mathtt{T}_*, 2\mathtt{T}_*]\\ 0 & \text{Otherwise.}\end{cases} \tag{13}$$

With $\mathtt{CV}_{\mathtt{R},\mathtt{BM}}$ and $\mathtt{CV}_{\mathtt{R}_q,\mathtt{BM}_q}$, we can determine whether Eq. (10) holds or not by computing $\mathtt{CV}_{\mathtt{R},\mathtt{BM}}\circ\mathtt{CV}_{\mathtt{R}_q,\mathtt{BM}_q}$. If $\mathtt{CV}_{\mathtt{R},\mathtt{BM}}\circ\mathtt{CV}_{\mathtt{R}_q,\mathtt{BM}_q} > 0$, Eq. (10) holds, and we need to access the MBR's children, otherwise it does not hold, and we should ignore the MBR's children. In this way, when we decide not to access an MBR's children, the information about which constraint does not hold is preserved. Next, we show the correctness proof of our random mask solution as follows.

Proof.

$$\begin{aligned}\mathtt{CV}_{\mathtt{R},\mathtt{BM}}\circ\mathtt{CV}_{\mathtt{R}_q,\mathtt{BM}_q} &= \mathtt{CV}_{\mathtt{R},\mathtt{BM}}\cdot(\mathtt{CV}_{\mathtt{R}_q,\mathtt{BM}_q})^T\\ &= (\mathtt{V}_{\mathtt{R},\mathtt{BM}}\cdot\mathsf{M})\cdot\Big(\mathtt{V}_{\mathtt{R}_q,\mathtt{BM}_q}\cdot(\mathsf{M}^{-1})^T\Big)^T\\ &= \mathtt{V}_{\mathtt{R},\mathtt{BM}}\cdot(\mathtt{V}_{\mathtt{R}_q,\mathtt{BM}_q})^T = \mathtt{V}_{\mathtt{R},\mathtt{BM}}\circ\mathtt{V}_{\mathtt{R}_q,\mathtt{BM}_q}\\ &= \mathtt{r}_{\mathtt{s}}\mathtt{r}_{\mathtt{q}}\Big(\alpha(\mathtt{R}_x)\circ\beta(\mathtt{R}_{q,x})+\alpha(\mathtt{R}_y)\circ\beta(\mathtt{R}_{q,y})-4\Big)\\ &\quad+\mathtt{r}_{\mathtt{k}}\mathtt{r}_{\mathtt{t}}\Big(\mathsf{BM}\circ\mathcal{E}(\mathsf{BM}_{\mathtt{q}})\Big)-\sum_{i=1}^{d}\mathtt{r}_i\mathtt{r}'_i\end{aligned} \tag{14}$$

From Eq. (11), Eq. (12), and Eq. (13), we know that, if $\mathtt{R}\cap\mathtt{R}_q\neq\varnothing$, we have $\alpha(\mathtt{R}_x)\circ\beta(\mathtt{R}_{q,x})+\alpha(\mathtt{R}_y)\circ\beta(\mathtt{R}_{q,y}) = 4$, otherwise $\Big(\alpha(\mathtt{R}_x)\circ\beta(\mathtt{R}_{q,x})+\alpha(\mathtt{R}_y)\circ\beta(\mathtt{R}_{q,y})\Big)\in[0,3]$. Regarding the keyword condition, if it is satisfied, we have $\mathsf{BM}\circ\mathcal{E}(\mathsf{BM}_{\mathtt{q}}) >$

0, otherwise it is 0. Since $\mathtt{r_s r_q} > \mathtt{r_k r_t} \cdot \phi > \sum_{i=1}^{d} \mathtt{r}_i \mathtt{r}'_i$, *iff* Eq. (10) holds, we have $\mathtt{CV}_{\mathtt{R,BM}} \circ \mathtt{CV}_{\mathtt{R_q,BM_q}} > 0$. □

Vector Bucketing Technique. In our random mask solution, we need to construct the vectors $\mathtt{V}_{\mathtt{R,BM}}$ and $\mathtt{V}_{\mathtt{R_q,BM_q}}$ with the size of $2(\mathtt{T}_x + \mathtt{T}_y) + 1 + \rho + d$. In some cases, $\mathtt{T}_x$, $\mathtt{T}_y$, or ρ may be large, leading to efficiency deterioration in generating random matrix and encrypting these vectors. To tackle this issue, we propose a vector bucketing technique to split a vector into several sub-vectors, each of which has the size of κ. Totally, there are $\xi = \lceil \frac{2(\mathtt{T}_x+\mathtt{T}_y)+1+\rho+d}{\kappa} \rceil$ sub-vectors:

$$\begin{aligned} &\mathtt{V}_{\mathtt{R,BM}} = (\mathtt{V}^1_{\mathtt{R,BM}}, \mathtt{V}^2_{\mathtt{R,BM}}, \cdots, \mathtt{V}^{\xi}_{\mathtt{R,BM}}), \\ &\mathtt{V}_{\mathtt{R_q,BM_q}} = (\mathtt{V}^1_{\mathtt{R_q,BM_q}}, \mathtt{V}^2_{\mathtt{R_q,BM_q}}, \cdots, \mathtt{V}^{\xi}_{\mathtt{R_q,BM_q}}). \end{aligned} \tag{15}$$

Note that the last sub-vector will be padded with 0 if its size is less than κ, and the permutation technique can be used here to swap the positions of these sub-vectors. Then, we add at least 2 random real numbers into each sub-vector, i.e., $\tilde{\mathtt{V}}^j_{\mathtt{R,BM}} = (\mathtt{V}^j_{\mathtt{R,BM}}, r^j_{\mathtt{x},1}, r^j_{\mathtt{x},1}, r^j_{\mathtt{x},2}, r^j_{\mathtt{x},2})$ and $\tilde{\mathtt{V}}^j_{\mathtt{R_q,BM_q}} = (\mathtt{V}^j_{\mathtt{R_q,BM_q}}, r^j_{\mathtt{q},1}, -r^j_{\mathtt{q},1}, r^j_{\mathtt{q},2}, -r^j_{\mathtt{q},2})$, where $j \in [1, \xi]$. After that, we generate ξ random invertible matrices $\{\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_\xi\}$ and encrypt sub-vectors as follows:

$$\begin{aligned} &\widetilde{\mathtt{CV}}_{\mathtt{R,BM}} = (\tilde{\mathtt{V}}^1_{\mathtt{R,BM}} \cdot \mathsf{M}_1, \tilde{\mathtt{V}}^2_{\mathtt{R,BM}} \cdot \mathsf{M}_2, \cdots, \tilde{\mathtt{V}}^{\xi}_{\mathtt{R,BM}} \cdot \mathsf{M}_\xi), \\ &\widetilde{\mathtt{CV}}_{\mathtt{R_q,BM_q}} = (\tilde{\mathtt{V}}^1_{\mathtt{R_q,BM_q}} \cdot (\mathsf{M}_1^{-1})^T, \tilde{\mathtt{V}}^2_{\mathtt{R_q,BM_q}} \cdot (\mathsf{M}_2^{-1})^T, \cdots, \tilde{\mathtt{V}}^{\xi}_{\mathtt{R_q,BM_q}} \cdot (\mathsf{M}_\xi^{-1})^T). \end{aligned} \tag{16}$$

Similarly, we can determine whether Eq. (10) holds or not by computing $\widetilde{\mathtt{CV}}_{\mathtt{R,BM}} \circ \widetilde{\mathtt{CV}}_{\mathtt{R_q,BM_q}}$. Since we have $\widetilde{\mathtt{CV}}_{\mathtt{R,BM}} \circ \widetilde{\mathtt{CV}}_{\mathtt{R_q,BM_q}} = \tilde{\mathtt{V}}_{\mathtt{R,BM}} \circ \tilde{\mathtt{V}}_{\mathtt{R_q,BM_q}} = \mathtt{V}_{\mathtt{R,BM}} \circ \mathtt{V}_{\mathtt{R_q,BM_q}}$, the vector bucketing technique can ensure the correctness of the predicate encryption according to Eq. (14). In this way, we can improve the performance in generating random invertible matrices and encrypting vectors. First, in the process of generating a random matrix, most of time is taken in calculating its inverse matrix. Although we need to generate ξ random invertible matrices, generating a $(\kappa + 4) \times (\kappa + 4)$ random invertible matrice is more efficient than generating a $(2\mathtt{T}_x + 2\mathtt{T}_y + 1 + \rho + d) \times (2\mathtt{T}_x + 2\mathtt{T}_y + 1 + \rho + d)$ random invertible matrix. Second, the vector bucketing technique can reduce the multiplication and addition operations from around $(2\mathtt{T}_x + 2\mathtt{T}_y + 1 + \rho + d)^2$ to $\xi \cdot (\kappa + 4)^2$.

4.3.2 Description of Our PPSKS+ Scheme

Based on the above FR-tree, random mask solution, and vector bucketing technique, we construct our PPSKS+ scheme, which is also comprised of five phases: 1) system initialization; 2) data outsourcing; 3) token generation; 4) search; 5) data recovery.

System Initialization. In our PPSKS+ scheme, the system initialization phase is similar to that of our PPSKS scheme. The only difference is that the data owner $\mathcal{O}$ needs to generate ξ random invertible matrices $\mathcal{M} = \{\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_\xi\}$, each of which has the size of $(\kappa + 4) \times (\kappa + 4)$, and authorizes them to the query proxy $\mathcal{P}$.

Data Outsourcing. The data owner $\mathcal{O}$ first builds the FR-tree over the dataset $\mathcal{X} = \{\mathtt{x}_i = \{(\mathtt{p}_{i,x}, \mathtt{p}_{i,y}), \mathtt{W}_i\} \mid 1 \le i \le n\}$. Here, we denote the FR-tree as Γ. Then, $\mathcal{O}$ encrypts Γ as $\mathsf{E}(\Gamma)$. Specifically, for each non-leaf node, $\mathcal{O}$ constructs $\mathtt{V}_{\mathtt{R,BM}}$ and generates encrypted vector $\widetilde{\mathtt{CV}}_{\mathtt{R,BM}}$ using the vector bucketing technique. For each leaf node, $\mathcal{O}$ generates $\mathsf{E}(\mathtt{x}_i)$ according to Eq. (5) and pads the last level's MBR to have m leaf nodes with the values outside the possible queries, where m is the maximum number of children. After that, $\mathcal{O}$ outsources $\mathsf{E}(\Gamma)$ to the cloud server $\mathcal{C}$. Besides, $\mathcal{O}$ sends the parameters $\{\mathtt{T}_x, \mathtt{T}_y, d, \kappa\}$ and the sorted keyword dictionary to the registered query users.

Token Generation. First, the query user u_i constructs the vector $\tilde{\mathtt{V}}_{\mathtt{R_q,BM_q}} = (\tilde{\mathtt{V}}^1_{\mathtt{R_q,BM_q}}, \tilde{\mathtt{V}}^2_{\mathtt{R_q,BM_q}}, \cdots, \tilde{\mathtt{V}}^{\xi}_{\mathtt{R_q,BM_q}})$ according to the query request $\mathcal{Q}$. For each $\tilde{\mathtt{V}}^j_{\mathtt{R_q,BM_q}}$ $(1 \le j \le \xi)$, u_i chooses two random real numbers $\{r^j_1, r^j_2\}$ and two random vectors $\{\mathtt{VR}^j_1, \mathtt{VR}^j_2\}$ satisfying $\tilde{\mathtt{V}}^j_{\mathtt{R_q,BM_q}} = r^j_1 \cdot \mathtt{VR}^j_1 + r^j_2 \cdot \mathtt{VR}^j_2$. Then, u_i further chooses random real numbers $\{r^j_{1,\varepsilon}, r^j_{2,\varepsilon} \mid \varepsilon = 1, 2, 3, 4\}$ and two random vectors $\{\mathtt{VR}^j_3, \mathtt{VR}^j_4\}$. After that, u_i generates $\mathtt{VQ}^j = \{\mathtt{VQ}^j_{1,1}, \mathtt{VQ}^j_{1,2}, \mathtt{VQ}^j_{2,1}, \mathtt{VQ}^j_{2,2}\}$, where

$$\begin{cases} \mathtt{VQ}^j_{1,1} = r^j_{1,1} \cdot \mathtt{VR}^j_1 + r^j_{1,2} \cdot \mathtt{VR}^j_3; \\ \mathtt{VQ}^j_{1,2} = r^j_{1,3} \cdot \mathtt{VR}^j_1 + r^j_{1,4} \cdot \mathtt{VR}^j_3; \\ \mathtt{VQ}^j_{2,1} = r^j_{2,1} \cdot \mathtt{VR}^j_2 + r^j_{2,2} \cdot \mathtt{VR}^j_4; \\ \mathtt{VQ}^j_{2,2} = r^j_{2,3} \cdot \mathtt{VR}^j_2 + r^j_{2,4} \cdot \mathtt{VR}^j_4. \end{cases} \tag{17}$$

After generating the session key $ss_i = \mathsf{H}(ssk_i, ts)$, u_i sends $\{\mathtt{VQ}^j, \mathsf{SE}_{ss_i}(r^j_1 || r^j_2 || r^j_{1,\varepsilon} || r^j_{2,\varepsilon}) \mid j \in [1, \xi], \varepsilon \in [1, 4]\}$ and $\{\mathsf{BF}'_{\mathtt{R}_{q,x}}, \mathsf{BF}'_{\mathtt{R}_{q,y}}, \mathsf{BF}'_{\mathtt{W}_q}, \vec{t}'_\tau, \vec{\tau}'_1, \vec{\tau}'_2, \mathtt{SER}\}$ (generated in *Step-3* of the PPSKS's token generation phase) to the query proxy $\mathcal{P}$.

Upon receiving them, $\mathcal{P}$ first generates $\mathtt{Token}_1$ with Eq. (7). Then, $\mathcal{P}$ generates $\mathtt{Token}_2$ with the authorized matrices $\mathcal{M} = \{\mathsf{M}_1, \mathsf{M}_2, \cdots, \mathsf{M}_\xi\}$ as follows.

$$\begin{aligned} \mathtt{Token}_2 = \{&\mathtt{VQ}^j \cdot (\mathsf{M}_j^{-1})^T, \mathsf{SE}_{ss_i}(r^j_1 || r^j_2 || r^j_{1,\varepsilon} || r^j_{2,\varepsilon}), \\ &id_i, ts \mid j \in [1, \xi], \varepsilon \in [1, 4]\}, \end{aligned}$$

where $\mathtt{VQ}^j \cdot (\mathsf{M}_j^{-1})^T = \{\mathtt{VQ}^j_{1,1} \cdot (\mathsf{M}_j^{-1})^T, \mathtt{VQ}^j_{1,2} \cdot (\mathsf{M}_j^{-1})^T, \mathtt{VQ}^j_{2,1} \cdot (\mathsf{M}_j^{-1})^T, \mathtt{VQ}^j_{2,2} \cdot (\mathsf{M}_j^{-1})^T\}$. Next, $\mathcal{P}$ forwards query tokens $\{\mathtt{Token}_1, \mathtt{Token}_2\}$ to the cloud server $\mathcal{C}$.

Search. After receiving the query tokens, with the master key mk, the cloud server $\mathcal{C}$ first calculates the session key $ss_i = \mathsf{H}(\mathsf{H}(mk, id_i), ts)$ and then recovers random sets $\{\mathtt{r}_x, \mathtt{r}_y, \mathtt{r}_\mathtt{w}, \mathtt{r}_{\tau_t}, \mathtt{r}_{\tau_1}, \mathtt{r}_{\tau_2}\}$ from $\mathtt{SER}$ and $\{r^j_1, r^j_2, r^j_{1,\varepsilon}, r^j_{2,\varepsilon}\}$ from $\mathsf{SE}_{ss_i}(r^j_1 || r^j_2 || r^j_{1,\varepsilon} || r^j_{2,\varepsilon})$. For $\mathtt{Token}_1$, $\mathcal{C}$ obtains $\{\mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,x}}), \mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,y}}), \mathsf{E}(\mathsf{BF}_{\mathtt{W}_q}), \mathsf{E}(\vec{t}_\tau), \mathsf{E}(\vec{\tau}_1), \mathsf{E}(\vec{\tau}_2)\}$ by removing random sets. For $\mathtt{Token}_2$, $\mathcal{C}$ recovers $\tilde{\mathtt{V}}^j_{\mathtt{R_q,BM_q}} \cdot (\mathsf{M}_j^{-1})^T$ using $\mathtt{VQ}^j \cdot (\mathsf{M}_j^{-1})^T$ and $\{r^j_1, r^j_2, r^j_{1,\varepsilon}, r^j_{2,\varepsilon}\}$ as follows.

$$\begin{aligned} \tilde{\mathtt{V}}^j_{\mathtt{R_q,BM_q}} \cdot (\mathsf{M}_j^{-1})^T = &\, r^j_1 \cdot \frac{r^j_{1,4} \cdot \mathtt{VQ}^j_{1,1} \cdot (\mathsf{M}_j^{-1})^T - r^j_{1,2} \cdot \mathtt{VQ}^j_{1,2} \cdot (\mathsf{M}_j^{-1})^T}{r^j_{1,1} \cdot r^j_{1,4} - r^j_{1,3} \cdot r^j_{1,2}} \\ &+ r^j_2 \cdot \frac{r^j_{2,4} \cdot \mathtt{VQ}^j_{2,1} \cdot (\mathsf{M}_j^{-1})^T - r^j_{2,2} \cdot \mathtt{VQ}^j_{2,2} \cdot (\mathsf{M}_j^{-1})^T}{r^j_{2,1} \cdot r^j_{2,4} - r^j_{2,3} \cdot r^j_{2,2}} \\ &\xrightarrow{\text{from Eq. (17)}} = (r^j_1 \cdot \mathtt{VR}^j_1 + r^j_2 \cdot \mathtt{VR}^j_2) \cdot (\mathsf{M}_j^{-1})^T = \tilde{\mathtt{V}}^j_{\mathtt{R_q,BM_q}} \cdot (\mathsf{M}_j^{-1})^T. \end{aligned}$$

Then, $\mathcal{C}$ can generate $\widetilde{\mathtt{CV}}_{\mathtt{R_q,BM_q}}$, as shown in Eq. (16). Next, $\mathcal{C}$ traverses the encrypted tree $\mathsf{E}(\Gamma)$ by checking whether $\widetilde{\mathtt{CV}}_{\mathtt{R,BM}} \circ \widetilde{\mathtt{CV}}_{\mathtt{R_q,BM_q}} > 0$ or not. If yes, $\mathcal{C}$ accesses the MBR's children. When navigating to leaf nodes, $\mathcal{C}$ executes the same steps as the search phase in the PPSKS scheme to obtain the encrypted results: $\mathtt{ERes}$. Note that, in our PPSKS scheme, $\mathcal{C}$ needs to calculate $\mathtt{ERes}$ among the whole dataset. While, in our PPSKS+ scheme, $\mathcal{C}$ only needs to calculate $\mathtt{ERes}$ within the leaf nodes under an MBR. See the detailed search process in Algorithm 2.

Data Recovery. In our PPSKS+ scheme, the data recovery phase is the same as that in the PPSKS scheme.

Remark. The basic idea of our PPSKS+ scheme is to build an encrypted FR-tree and make the cloud server search over the tree. Since R*-tree has fewer overlaps than the original R-tree, we choose it as the building block of our FR-tree. When traversing the encrypted FR-tree, we use the predicate encryption that allows the cloud server to quickly navigate to the last level's MBR. After that, the cloud server can calculate the encrypted results among the leaf nodes in the MBR. As a result, our PPSKS+ scheme can achieve better efficiency than our PPSKS scheme while hiding access patterns among m encrypted data records.

5 Security Analysis

Following our design goal in privacy preservation, in this section, we will demonstrate that i) our proposed schemes (PPSKS and PPSKS+) can protect the privacy of outsourced data, query requests, query results, and access patterns against the cloud server $\mathcal{C}$; ii) our proposed schemes can preserve the privacy of query requests and query results against the query proxy $\mathcal{P}$. Since our PPSKS and PPSKS+ schemes are built on the SSMT scheme, we will first prove the security of our SSMT scheme and then discuss the privacy preservation of our PPSKS and PPSKS+ schemes.

First of all, we would briefly review the security model for securely realizing an ideal functionality in the presence of the static semi-honest adversary [33]. In our security model, since the cloud server $\mathcal{C}$ and the query proxy $\mathcal{P}$ are semi-honest, we will separately prove that our schemes are secure against $\mathcal{C}$ and $\mathcal{P}$. If we denote $\mathcal{I}$ as the instance of $\mathcal{C}$ and $\mathcal{P}$, i.e., $\mathcal{I} \in \{\mathcal{C}, \mathcal{P}\}$, we have the following models:

Real world model: The real world execution of a scheme Π takes place in $\mathcal{I}$ and an adversary $\mathcal{A}$, who corrupts $\mathcal{I}$. Assuming that x is the input of the scheme Π over $\mathcal{I}$, and y is the auxiliary input, the execution of Π under $\mathcal{A}$ in the real world model is defined as:

$$\text{REAL}_{\Pi,\mathcal{A},y}(x) \overset{def}{=} \{\text{Output}^{\Pi}(x), \text{View}^{\Pi}(x), y\},$$

in which $\text{Output}^{\Pi}(x)$ is the output of the execution of Π with the input x on $\mathcal{I}$, and $\text{View}^{\Pi}(x)$ is the view of $\mathcal{I}$ during an execution of Π with the input x.

Ideal world model: In the ideal world execution, $\mathcal{I}$ interacts with the ideal functionality $\mathcal{F}$ for a function f. Here, the execution of f under simulator $\texttt{Sim}$ in the ideal world model on input x and auxiliary input y is defined as:

$$\text{IDEAL}_{\mathcal{F},\texttt{Sim},y}(x) \overset{def}{=} \{f(x), \texttt{Sim}(x, f(x)), y\}.$$

Definition 2 (Security against semi-honest adversary)**.** *Let $\mathcal{F}$ be a deterministic functionality and Π be a scheme in $\mathcal{I}$. We say that Π securely realizes $\mathcal{F}$ if there exists $\texttt{Sim}$ of PPT (Probabilistic Polynomial Time) transformations (where $\texttt{Sim} = \texttt{Sim}(\mathcal{A})$) such that for semi-honest PPT adversary $\mathcal{A}$, for x and y, for $\mathcal{I}$ holds:*

$$REAL_{\Pi,\mathcal{A},y}(x) \overset{c}{\approx} IDEAL_{\mathcal{F},\texttt{Sim},y}(x)$$

where $\overset{c}{\approx}$ denotes computational indistinguishability.

5.1 The security of SSMT scheme

In this subsection, with Definition 2, we will prove that our SSMT scheme achieves indistinguishability under Chosen-Plaintext Attacks (IND-CPA).

Algorithm 2 PPSKS+ Search over Encrypted Data

```
Input: Encrypted FR-tree, $\texttt{E}(\Gamma)$ and query tokens, $\{\texttt{Token}_1, \texttt{Token}_2\}$;
Output: A set containing encrypted data records, $\texttt{ERes}$;
 1: $\{\mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,x}}), \mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,y}}), \mathsf{E}(\mathsf{BF}_{\mathtt{W}_q}), \mathsf{E}(\vec{\tau}_t), \mathsf{E}(\vec{\tau}_1), \mathsf{E}(\vec{\tau}_2)\} \leftarrow \texttt{Token}_1$;
 2: $\widetilde{\texttt{CV}}_{\texttt{R}_\texttt{q},\texttt{BM}_\texttt{q}} \leftarrow \texttt{Token}_2$;
 3: Initialize an MBR set $\mathsf{S}_{\text{MBR}} \leftarrow \varnothing$; $f(x) \leftarrow \mathsf{SSMT.Interpolation}(\gamma)$
 4: SearchOnFRtree($\texttt{E}(\Gamma)$.root, $\widetilde{\texttt{CV}}_{\texttt{R}_\texttt{q},\texttt{BM}_\texttt{q}}$)
 5: for each MBR in $\mathsf{S}_{\text{MBR}}$ do
 6:     for each leaf node $\mathsf{E}(\mathtt{x}_i)$, $i \in [1, m]$, under MBR do
 7:         $\mathsf{E}(\theta_x) \leftarrow \sum_{j=1}^{t_x} \mathsf{SSMT.Check}\left(\mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,x}}), \mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,x,j}}), f(x)\right)$
 8:         $\mathsf{E}(\theta_y) \leftarrow \sum_{j=1}^{t_y} \mathsf{SSMT.Check}\left(\mathsf{E}(\mathsf{BF}_{\mathtt{p}_{i,y}}), \mathsf{E}(\mathsf{BF}_{\mathtt{R}_{q,y,j}}), f(x)\right)$
 9:         $\mathsf{E}(z) \leftarrow \mathsf{SSMT.Check}(\mathsf{E}(\mathsf{BF}_{\mathtt{W}_i}), \mathsf{E}(\mathsf{BF}_{\mathtt{W}_{q,j}}), f(x))$ and secure
            circuits: SPadd, Sadd, Scom, Smul;
10:         $\mathsf{E}(\mathtt{f}_i) \leftarrow \mathsf{E}(\theta_x) \cdot \mathsf{E}(\theta_y) \cdot \mathsf{E}(z) = \mathsf{E}(\theta_x \cdot \theta_y \cdot z)$;
11:     $\{\mathsf{E}(\mathtt{v}_j) \mid j \in [1, k]\} \leftarrow \{\mathsf{E}(\mathtt{f}_i) \mid i \in [1, m]\}$
12:     $\{\mathsf{E}(\mathtt{p}'_{j,x}), \mathsf{E}(\mathtt{p}'_{j,y}), \mathsf{E}(\mathtt{W}'_j)\} \leftarrow$ Eq. (8) and random integers;
13:     $\texttt{ERes}$.add($\{\mathsf{E}(\mathtt{p}'_{j,x}), \mathsf{E}(\mathtt{p}'_{j,y}), \mathsf{E}(\mathtt{W}'_j)\}$);

14: function SearchOnFRtree(node, $\widetilde{\texttt{CV}}_{\texttt{R}_\texttt{q},\texttt{BM}_\texttt{q}}$)
15:     if node is non-leaf node then
16:         $\widetilde{\texttt{CV}}_{\texttt{R},\texttt{BM}} \leftarrow$ encrypted vector of node.MBR;
17:         if $\widetilde{\texttt{CV}}_{\texttt{R},\texttt{BM}} \circ \widetilde{\texttt{CV}}_{\texttt{R}_\texttt{q},\texttt{BM}_\texttt{q}} > 0$ then
18:             if node is last level's non-lead node then
19:                 $\mathsf{S}_{\text{MBR}}$.add(node.MBR);
20:             else
21:                 for each childNode of node do
22:                     SearchOnFRtree(childNode, $\widetilde{\texttt{CV}}_{\texttt{R}_\texttt{q},\texttt{BM}_\texttt{q}}$);
```

Theorem 1. *The SSMT scheme is IND-CPA secure if the used fully homomorphic encryption scheme (FHE) is IND-CPA secure.*

Proof. For the work process of the simulator, $\texttt{Sim}$ first randomly chooses a set S', an element $\mathtt{x}'$, and γ independent hash functions. Then, $\texttt{Sim}$ simulates $\mathcal{A}$ as follows: i) it generates an encrypted bloom filter $\mathsf{E}(\mathsf{BF}_{\mathsf{s}'})$ for the set S'; ii) it generates an encrypted bloom filter $\mathsf{E}(\mathsf{BF}_{\mathsf{x}'})$ for the element $\mathtt{x}'$; iii) with γ, it builds a polynomial function $f'(x)$; iv) it calculates $\mathsf{E}(\sigma') = \mathsf{E}(\mathsf{BF}_{\mathsf{s}'}) \circ \mathsf{E}(\mathsf{BF}_{\mathsf{x}'})$ and $\mathsf{E}(\theta') = \mathsf{E}(f(\sigma'))$. Finally, $\texttt{Sim}$ outputs $\{\mathsf{E}(\theta')\}$ and $\{\mathsf{E}(\mathsf{BF}_{\mathsf{s}'}), \mathsf{E}(\mathsf{BF}_{\mathsf{x}'}), f'(x), \mathsf{E}(\sigma')\}$ as $\mathcal{A}$'s ideal view. In the real execution, $\mathcal{A}$ receives $\{\mathsf{E}(\mathsf{BF}_{\mathsf{s}}), \mathsf{E}(\mathsf{BF}_{\mathsf{x}}), f(x)\}$ and calculates $\{\mathsf{E}(\sigma), \mathsf{E}(\theta)\}$. Obviously, distinguishing the real and ideal views is equivalent to breaking the FHE ciphertexts. Therefore, our proposed SSMT scheme is IND-CPA secure if the employed FHE is IND-CPA secure. □

5.2 The privacy preservation of PPSKS and PPSKS+ schemes against the cloud server

Before analyzing the security of our PPSKS and PPSKS+ schemes against the cloud server $\mathcal{C}$, we first define the leakages of our PPSKS and PPSKS+ schemes as $\mathcal{L}_1$ and $\mathcal{L}_2$, respectively.

- In our PPSKS scheme, since $\mathcal{C}$ can only obtain the number of query result k, the leakage $\mathcal{L}_1 = k$.
- In our PPSKS+ scheme, when searching on the non-leaf nodes, $\mathcal{C}$ knows whether the MBR's children should be accessed by checking $\widetilde{\texttt{CV}}_{\texttt{R},\texttt{BM}} \circ \widetilde{\texttt{CV}}_{\texttt{R}_\texttt{q},\texttt{BM}_\texttt{q}} > 0$. Consequently, $\mathcal{C}$ knows the inner product result between $\widetilde{\texttt{CV}}_{\texttt{R},\texttt{BM}}$ and $\widetilde{\texttt{CV}}_{\texttt{R}_\texttt{q},\texttt{BM}_\texttt{q}}$, denoted as $\texttt{dot}(\widetilde{\texttt{CV}}_{\texttt{R},\texttt{BM}}, \widetilde{\texttt{CV}}_{\texttt{R}_\texttt{q},\texttt{BM}_\texttt{q}})$. When reaching leaf nodes, since $\mathcal{C}$ executes the same operations as our PPSKS scheme to obtain the encrypted query results, it also knows k. Thus, $\mathcal{L}_2 = \{\texttt{dot}(\widetilde{\texttt{CV}}_{\texttt{R},\texttt{BM}}, \widetilde{\texttt{CV}}_{\texttt{R}_\texttt{q},\texttt{BM}_\texttt{q}}), k\}$.

Next, we show that our PPSKS and PPSKS+ schemes can achieve the privacy goals against the cloud server $\mathcal{C}$.

Theorem 2. *The PPSKS scheme securely computes* $\mathtt{ERes}$ *under* $\mathcal{L}_1$ *without leaking the outsourced data, query requests, query results, and access pattern privacy to* $\mathcal{C}$.

Proof. In our PPSKS scheme, $\mathcal{C}$ holds the encrypted outsourced data $\{\mathsf{E}(\mathtt{x}_i) \mid i \in [1,n]\}$ and the search token $\mathtt{Token}_1$. First, $\mathcal{C}$ uses the SSMT scheme to calculate the encrypted flag $\mathsf{E}(\mathtt{f}_i)$ indicating whether the corresponding data record $\mathsf{E}(\mathtt{x}_i)$ satisfies the query requests or not. Since the outsourced data and query requests are encrypted with FHE, and the SSMT scheme has been proved to be IND-CPA secure, the cloud server $\mathcal{C}$ has no idea about the outsourced data, query requests, and flags $\mathtt{f}_i$. After obtaining $\mathsf{E}(\mathtt{f}_i)$, $\mathcal{C}$ can get $\{\mathsf{E}(\mathtt{v}_j) \mid j \in [1,k]\}$ and calculate k encrypted results with Eq. (8). Since all operations are conducted under FHE ciphertexts, the security of FHE can guarantee the query results are kept secret from $\mathcal{C}$ although it has $\mathcal{L}_1$. Besides, since $\mathcal{C}$ has no idea about $\mathtt{f}_i$, and the k encrypted results are calculated from n data records (see Eq. (8)), $\mathcal{C}$ only knows that there are k data records satisfying the query request and cannot infer which data records in the dataset are selected as the query results. Therefore, our PPSKS scheme can hide access patterns. □

Theorem 3. *The PPSKS+ scheme securely computes* $\mathtt{ERes}$ *under* $\mathcal{L}_2$ *without leaking the outsourced data, query requests, query results, and m-access pattern privacy to* $\mathcal{C}$.

Before proving Theorem 3, we first define *m-access pattern privacy*.

Definition 3 (*m-access pattern privacy*)**.** *Given an encrypted dataset* $\mathcal{X}$ *with* n *data items, after forming a subset* $\mathcal{X}_{(m)}$ *with* m *data items, where* $m < n$*, the* ***m-access pattern privacy*** *ensures the adversary has no idea about which data items in* $\mathcal{X}_{(m)}$ *are selected as query results.*

Proof. In our PPSKS+ scheme, $\mathcal{C}$ holds the encrypted FR-tree $\mathtt{E}(\Gamma)$ and query tokens $\{\mathtt{Token}_1, \mathtt{Token}_2\}$. First, as shown in Algorithm 2, $\mathcal{C}$ uses $\mathtt{Token}_2$ to traverse the FR-tree on non-leaf nodes, in which the predicate encryption technique motivated by [18] is adopted. In [18], such a predicate encryption technique had been proved to be selectively secure under the leakage of inner product result $\mathtt{dot}()$. Therefore, $\mathcal{C}$ cannot infer the underlying plaintexts of MBRs and $\mathtt{Token}_2$. Then, with $\mathtt{Token}_1$, $\mathcal{C}$ uses the same approach as our PPSKS scheme to calculate the encrypted query results on m leaf nodes. Therefore, $\mathcal{C}$ has no idea regarding the plaintexts of encrypted leaf nodes $\{\mathsf{E}(\mathtt{x}_i) \mid i \in [1,n]\}$, $\mathtt{Token}_1$, and query results. Besides, in our PPSKS+ scheme, we ensure that there are m leaf nodes in the last level MBR by padding dummy data records. Therefore, $\mathcal{C}$ cannot infer which data records are returned as the query requests in the m leaf nodes. Thus, our PPSKS+ scheme hides *m-access pattern privacy*. □

5.3 The privacy preservation of PPSKS and PPSKS+ schemes against the query proxy

Now, with Definition 2, we prove that the query requests and query results of our PPSKS and PPSKS+ schemes are kept secret from the query proxy $\mathcal{P}$.

Theorem 4. *The PPSKS and PPSKS+ schemes are secure against the query proxy* $\mathcal{P}$ *if the employed symmetric key encryption* $\mathsf{SE}()$*, e.g., AES-256, is secure.*

Proof. Compared to our PPSKS scheme, our PPSKS+ scheme has an extra query token $\mathtt{Token}_2$. Therefore, if we prove our PPSKS+ scheme to be secure against $\mathcal{P}$, we can also ensure the security of our PPSKS scheme. Next, we show how to construct the simulator of our PPSKS+ scheme on $\mathcal{P}$. First, $\mathtt{Sim}$ randomly chooses an SKS query $\mathcal{Q}'' = \{\mathtt{R}''_q, \mathtt{W}''_q, \tau''_1, \tau''_2\}$, random sets $\{\mathtt{r}''_x, \mathtt{r}''_y, \mathtt{r}''_{\mathtt{w}}, \mathtt{r}''_{\tau_t}, \mathtt{r}''_{\tau_1}, \mathtt{r}''_{\tau_2}\}$, and random real numbers $\{r'^j_1, r'^j_2, r'^j_{1,\varepsilon}, r'^j_{2,\varepsilon} \mid j \in [1,\xi], \varepsilon \in [1,4]\}$. Then, $\mathtt{Sim}$ simulates $\mathcal{A}$ as follows: i) it generates $\{\mathsf{BF}''_{\mathtt{R}_{q,x}}, \mathsf{BF}''_{\mathtt{R}_{q,y}}, \mathsf{BF}''_{\mathtt{W}_q}, \vec{\tau}''_t, \vec{\tau}''_1, \vec{\tau}''_2\}$ by adding $\{\mathtt{r}''_x, \mathtt{r}''_y, \mathtt{r}''_{\mathtt{w}}, \mathtt{r}''_{\tau_t}, \mathtt{r}''_{\tau_1}, \mathtt{r}''_{\tau_2}\}$ into encoded $\mathcal{Q}''$; ii) it first constructs $\tilde{\mathtt{V}}'_{\mathtt{R_q},\mathtt{BM_q}}$ according to $\mathcal{Q}''$ and then generates $\{\mathtt{VQ}'^j \mid j \in [1,\xi]\}$ following Eq. (17). Finally, $\mathtt{Sim}$ outputs $\{\mathsf{BF}''_{\mathtt{R}_{q,x}}, \mathsf{BF}''_{\mathtt{R}_{q,y}}, \mathsf{BF}''_{\mathtt{W}_q}, \vec{\tau}''_t, \vec{\tau}''_1, \vec{\tau}''_2, \mathtt{VQ}'^1, \mathtt{VQ}'^2, \cdots, \mathtt{VQ}'^\xi\}$ as $\mathcal{A}$'s ideal view. In the real execution, $\mathcal{A}$ receives $\{\mathsf{BF}'_{\mathtt{R}_{q,x}}, \mathsf{BF}'_{\mathtt{R}_{q,y}}, \mathsf{BF}'_{\mathtt{W}_q}, \vec{\tau}'_t, \vec{\tau}'_1, \vec{\tau}'_2, \mathtt{VQ}^1, \mathtt{VQ}^2, \cdots, \mathtt{VQ}^\xi\}$. In $\mathcal{A}$'s real view, all elements are added with random values: $\{\mathtt{r}_x, \mathtt{r}_y, \mathtt{r}_{\mathtt{w}}, \mathtt{r}_{\tau_t}, \mathtt{r}_{\tau_1}, \mathtt{r}_{\tau_2}\}$ and $\{r^j_1, r^j_2, r^j_{1,\varepsilon}, r^j_{2,\varepsilon}\}$. Clearly, distinguishing the real and ideal views is equivalent to obtaining these random values. However, these values are encrypted by $\mathsf{SE}()$ with the session key ss_i. Since $\mathcal{P}$ does not have ss_i, the security of $\mathsf{SE}()$ ensures that the query request of our PPSKS+ scheme is secure. Similarly, due to the random sets $\{\mathtt{r}_j \mid j \in [1,k]\}$, which are encrypted with $\mathsf{SE}()$, the query results are also secure against $\mathcal{P}$. □

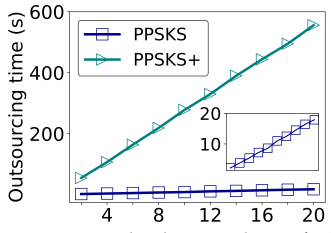
6 Performance Evaluation

In this section, we evaluate the performance of our PPSKS and PPSKS+ schemes, focusing on data outsourcing, token generation, and search phases.

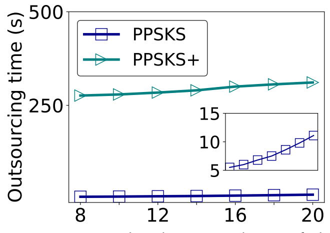
Notably, we do not compare our proposed schemes with the existing privacy-preserving spatial keyword query schemes in terms of performance. That is because this work is the first to consider the similarity of keyword sets while protecting access pattern privacy in privacy-preserving spatial keyword queries. Since the stricter security goal or more challenging functionality (both of which we have) unavoidably incur additional costs, it is unreasonable and unfair to compare the performance of schemes that have different security goals or functionalities. Nonetheless, to explore the differences between our proposed schemes and the existing schemes, we provide a detailed characteristic comparison in Table 3.

Experimental Setting: In our experiments, we use a real-world Yelp business dataset [34], denoted as **Yelp**. For each item in the dataset, we extract the location information of Florida as spatial data and the categories attribute as keyword sets. Table 2 lists the description of the **Yelp** dataset. Further, we scale the location information and have $\mathtt{T}_x = 111,135, \mathtt{T}_y = 95,102$. Therefore, $w_{\mathtt{p}_{i,x}} = w_{\mathtt{p}_{i,y}} = 17$.

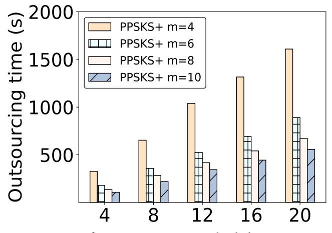
Recalling Section 3, we employ the bloom filter and FHE in our proposed schemes. For bloom filters, we will use the optimal version, i.e., following the equation $\eta = \gamma \cdot \mu / \ln 2$ to set the bloom filter's length, in which the *false positive* probability is $f_p \approx (1/2)^\gamma$. For the spatial bloom filters, $\mu = 2w_{\mathtt{p}_{i,x}} - 2 = 2w_{\mathtt{p}_{i,x}} - 2 = 32$, while $\mu = 27$ for



(a) Varying with the number of data records n ($\times 10^3$)

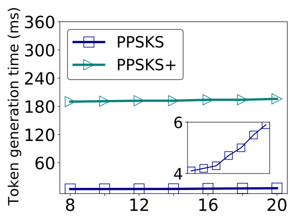


(b) Varying with the number of hash functions γ

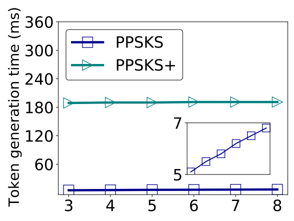


(c) Impact of maximum children varying with n ($\times 10^3$)

Fig. 7. Data outsourcing time: (a) varying with n, and setting γ=10; (b) varying with γ, and setting n=10^4; (c) varying with n, exploring the impact of maximum children of FR-tree, and setting γ=10.



(a) Varying with the number of hash functions γ



(b) Varying with the number of query keywords $|\mathtt{W}_q|$

Fig. 8. Token generation time: (a) varying with γ, and setting $|\mathtt{W}_q|$=4; (b) varying with $|\mathtt{W}_q|$, and setting γ=10.

TABLE 2
Description of the **Yelp** dataset

Name	Dataset size	Maximum size of keyword sets	Keyword dictionary size
Yelp	21,900	27	1123

the keyword bloom filter. Regarding FHE, we adopt the symmetric homomorphic encryption (SHE) used in [35], [36] due to its high efficiency. For SHE's security parameters, we set k_0 = 4096, k_1 = 80, and k_2 = 160. See detailed definitions in [35]. Note that: i) the SHE scheme also has its public-key setting [36]; ii) although SHE is a leveled homomorphic encryption scheme, we can refresh ciphertexts by using the bootstrapping protocol proposed in [35]. All of our proposed schemes were implemented with Java and executed on a machine with 16 GB memory, 3.4 GHz Intel(R) Core(TM) i7-3770 processors, and Ubuntu 16.04 OS.

6.1 Evaluation of Data Outsourcing Phase

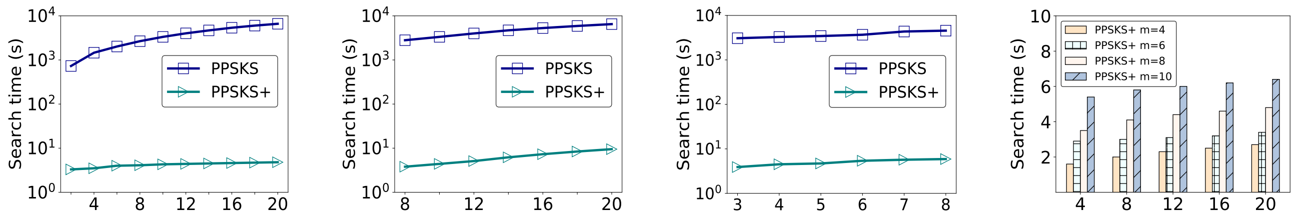
From Section 4, we know that the performance of our PPSKS and PPSKS+ schemes is affected by the number of data records n and the number of hash functions γ. Besides, the maximum number of FR-tree children has an impact on the performance of our PPSKS+ scheme due to employing the tree structure. Fig. 7 depicts the performance of our PPSKS and PPSKS+ schemes in the data outsourcing phase varying with the above parameters. From Fig. 7(a) and Fig. 7(b), we can see that the PPSKS scheme consumes less time than our PPSKS+ scheme in preparing outsourcing data. This is because, compared to the PPSKS scheme, the PPSKS+ scheme needs to additionally build FR-tree and encrypt the constructed FR-tree. It is reasonable since data outsourcing is offline and usually takes once. The tree-based scheme, PPSKS+, benefits the performance of search operations that are frequent and are the focus of the optimization. See detailed search evaluations in Section 6.3. Furthermore, Fig. 7(a) and Fig. 7(b) illustrate that the outsourcing time of the PPSKS+ scheme increases sharply with growing n, while it almost remains stable with the growth of γ. The reason is that the time cost of encrypting FR-tree's non-leaf nodes dominates that of leaf nodes. In Fig. 7(c), we evaluate the outsourcing performance of our PPSKS+ scheme under different maximum children settings. We can see that: i) the time costs increase with the growth of n for each maximum children setting. Obviously, it is due to the increase in the number of non-leaf nodes; ii) more children under an MRB leads to better outsourcing performance. This is because there are fewer non-leaf nodes if n is fixed.

6.2 Evaluation of Token Generation Phase

In the token generation phase, the PPSKS scheme generates one token $\mathtt{Token}_1$, while the PPSKS+ generates two tokens: $\mathtt{Token}_1$ and $\mathtt{Token}_2$. For $\mathtt{Token}_1$, it is affected by the number of hash functions γ and the number of keywords $|\mathtt{W}_q|$. Regarding $\mathtt{Token}_2$, it is affected by the number of sub-vectors ξ introduced by our vector bucketing technique. Since we will discuss the performance varying with ξ in Section 6.4, here we only evaluate the token generation performance varying with γ and $|\mathtt{W}_q|$, as shown in Fig. 8(a) and Fig. 8(b). From these two figures, we have: i) our PPSKS scheme has better performance in generating tokens. It is obvious since the PPSKS+ scheme needs to generate an additional token $\mathtt{Token}_2$; ii) the token generation time of our PPSKS scheme increases with the growth of γ and $|\mathtt{W}_q|$, while the PPSKS+ scheme presents a stable trend. This is because generating random vectors $\mathtt{VQ}^j$ ($j \in [1, \xi]$) and encrypting them into $\mathtt{Token}_2$ take more time than generating $\mathtt{Token}_1$. However, γ and $|\mathtt{W}_q|$ only have an impact on generating $\mathtt{Token}_1$; iii) although our PPSKS+ scheme is not as good as the PPSKS scheme in the performance of token generation, it is still less than 200ms for all experimental cases, validating its efficiency in this phase.

6.3 Evaluation of Search Phase

In the search phase, we explore the impact of the number of data records n, the number of hash functions γ, and the number of query keywords $|\mathtt{W}_q|$ on search efficiency, as shown in Fig. 9(a), Fig. 9(b), and Fig. 9(c), respectively. From these three figures, we can see that: i) our PPSKS+ scheme achieves at least two orders of magnitude better performance than the PPSKS scheme; ii) our PPSKS+ is at the level of second and fewer than 10 seconds in all experimental cases, which is already quite efficient in performing spatial



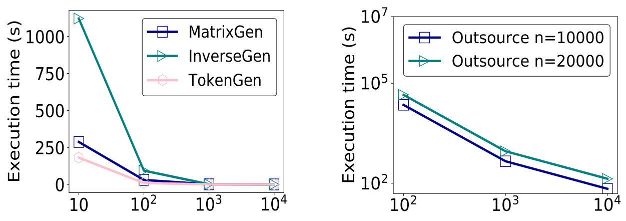
(a) Varying with the number of data records n ($\times 10^3$). (b) Varying with the number of hash functions γ (c) Varying with the number of query keywords $|\mathtt{W}_q|$ (d) Impact of maximum children varying with n ($\times 10^3$).

Fig. 9. Search time: (a) varying with n, and setting γ=10, $|\mathtt{W}_q|$=4; (b) varying with γ, and setting n=10^4, $|\mathtt{W}_q|$=4; (c) varying with $|\mathtt{W}_q|$, and setting n=10^4, γ=10; (d) varying with n, exploring the impact of maximum children of FR-tree, and setting γ=10, $|\mathtt{W}_q|$=4.

TABLE 3
Comparison with existing schemes

Schemes	**ELCBFR+** [6]	**PBRQ-L** [7]	**PBRQ-Q** [7]	**PrivSTL** [8]	**SKSE-I** [9]	**SKSE-II** [9]	**GRQ+MSSAC** [10]	**PPSKS**	**PPSKS+**
Sub-linear complexity	✔	✘	✔	✔	✘	✔	✔	✘	✔
Keyword search	Boolean	Boolean	Boolean	Boolean	Boolean	Boolean	Euclidean	Jaccard	Jaccard
Single-server model	✔	✔	✔	✘	✔	✔	✔	✔	✔
Conjunctive privacy	✔	✔	✘	✔	✔	✔	✘	✔	✔
Access pattern	✘	✘	✘	✘	✘	✘	✘	✔	◐

- Note that, ◐ indicates the PPSKS+ scheme hides *m-access pattern privacy* instead of the full access pattern privacy in PPSKS.



(a) Impact on key and token generations varying with ξ (b) Impact on data outsourcing varying with ξ

Fig. 10. Execution time of matrix generation, token generation, and data outsourcing varying with ξ: (a) impact on key and token generations; (b) impact on data outsourcing.

keyword queries over encrypted data, especially under the goals of calculating Jaccard similarity and protecting access patterns. The significant advantage of our PPSKS+ scheme stems from searching over encrypted FR-tree, allowing it to attain sublinear efficiency. In contrast, our PPSKS scheme is linear efficiency to the dataset size.

In Fig. 9(d), we explore the impact of the number of FR-tree's maximum children on search efficiency, in which we vary n and observe the search performance under different maximum children settings. We can see that the dataset size n has a minor impact on the search efficiency, whereas it is a bit significant for the number of maximum children. That is because it is quite efficient for our PPSKS+ scheme to make search decisions by computing inner products at non-leaf nodes, but it is relatively expensive in calculating search results at leaf nodes. Therefore, fewer children under an MBR can lead to better search performance.

As we employ the bloom filter technique in our schemes, accuracy is another important measurement. In fact, it is going to be a trade-off between accuracy and efficiency, i.e., the larger γ will improve accuracy since $f_p \approx (1/2)^\gamma$, but it will increase the computational costs, as shown in Fig. 7(b), Fig. 8(a), and Fig. 9(b). In our experiments, we set the maximum value of γ as 20, i.e., $f_p \approx 0.0001\%$, which is acceptable for many real-world applications. Since the number of hash functions γ has an insignificant effect on computational costs in data outsourcing and token generation phases, we can further improve accuracy by enlarging γ. Regarding the search phase, when $\gamma = 20$ ($f_p \approx 0.0001\%$), the performance of our PPSKS+ scheme is still at the level of second and fewer than 10 seconds.

6.4 Performance Gain of Vector Bucketing Technique

As the theoretical analysis in Section 4.3.1, the vector bucketing technique can improve the performance in two aspects: i) generating matrix and its inversion; ii) encrypting the non-leaf node of FR-tree and the query token ($\mathtt{Token}_2$). In this section, we evaluate the performance improvement of the vector bucketing technique in key (matrix and its inversion) generation, token generation, and data outsourcing. Since the token generation only involves one encryption operation for each query, we plot the performance of key generation and token generation in one figure, as shown in Fig. 10(a). In the figure, we vary ξ from 10 to 10^4 to observe the performance of matrix generation **MatrixGen**, inverse matrix calculation **InverseGen**, and token generation **TokenGen**. We can see that the vector bucketing technique sharply reduces the execution time for all of these operations, especially for the expensive operation of **InverseGen**. We have analyzed the reasons in the description of the vector bucketing technique, seeing details in Section 4.3.1. Regarding the data outsourcing, Fig. 10(b) depicts the execution time varying with ξ, in which we select $n = 10,000$ and $n = 20,000$ as the number of outsourced data records, and vary ξ from 10^2 to 10^4. We exclude the case of $\xi = 10$ because it takes dozens of hours to outsource the corresponding datasets. Obviously, the vector bucketing technique significantly improves the computational costs in preparing the outsourced data and makes our proposed scheme available and efficient.

7 Related work

Due to the wide applications in LBS, several privacy-preserving spatial keyword query schemes have been designed to preserve the privacy of outsourced data and query requests. In 2019, Cui et al. [6] proposed a privacy-preserving boolean spatial keyword query scheme EL-CBFR+ based on the matrix encryption and linear transformation method. It can retrieve the objects satisfying: i) their locations fall inside a query rectangle; ii) their keyword sets contain all query keywords. Aiming at the same query type, Wang et al. [7] adopted symmetric-key hidden vector encryption, gray code, and bitmap encoding technique to protect the data and query privacy. The designed schemes are denoted as PBRQ-L and PBRQ-Q. Recently, Wang et al. [9] further presented two privacy-preserving schemes, SKSE-I and SKSE-II, for boolean spatial keyword queries, in which hidden vector encryption was employed as the cryptographic primitive, and bloom filter technique was used to construct their schemes. By integrating bilinear map, RSA encryption, and linear encryption, Huang et al. [8] proposed a privacy-preserving scheme PrivSTL for the spatio-temporal keyword query that supports an additional temporal condition for the boolean spatial keyword query. Different from the above schemes that only consider the boolean keyword search, our proposed schemes are able to support the keyword set similarity over encrypted data, which is more challenging but more practical than the boolean keyword search.

Considering keyword set similarity, Song et al. [10] proposed a privacy-preserving spatial keyword similarity query scheme, denoted as GRQ+MSSAC, using matrix encryption. However, this scheme adopted the Euclidean distance to measure the keywords similarity, which is not as popular as the Jaccard metric (used in our proposed schemes) for measuring the similarity of keyword sets [16]. In addition, this scheme did not consider the conjunctive privacy discussed in [9], namely, it leaks the information about the spatial condition or the keyword condition mismatches the query request.

In addition, none of the aforementioned privacy-preserving schemes protect access pattern privacy, and our proposed schemes are the first to consider such privacy while providing spatial keyword similarity queries at the same time. In order to clearly show the differences between our proposed schemes and the existing schemes in this topic, we compare their characteristics in Table 3.

8 Conclusion

In this paper, we have proposed privacy-preserving spatial keyword similarity query schemes, in which PPSKS is a linear search scheme, and PPSKS+ can achieve the sub-linear search efficiency. In this domain, we are the first to consider Jaccard similarity for keywords sets and simultaneously protect access patterns. Specifically, we first designed an SSMT scheme using the bloom filter technique, FHE, and Lagrange interpolation function. Based on the SSMT scheme and secure circuits, we constructed our basic scheme, PPSKS. Then, we devised the FR-tree index and proposed a modified predicate encryption technique that allows us to search over encrypted FR-tree without leaking conjunctive privacy. With these components, we presented our tree-based construction, PPSKS+, which is efficient in performing the spatial keyword similarity queries. Finally, we formally analyzed the security of our proposed schemes and conducted extensive experiments to explore their performance. For future work, we will further improve the performance of our PPSKS+ scheme.

Acknowledgements

This research was supported in part by NSERC Discovery Grants (04009, 03787, RGPIN-2022-03244).

References

[1] L. Chen, G. Cong, C. S. Jensen, and D. Wu, "Spatial keyword query processing: An experimental evaluation," *Proceedings of the VLDB Endowment*, vol. 6, no. 3, pp. 217–228, 2013.

[2] T. Lee, J.-w. Park, S. Lee, S.-w. Hwang, S. Elnikety, and Y. He, "Processing and optimizing main memory spatial-keyword queries," *Proceedings of the VLDB Endowment*, vol. 9, no. 3, pp. 132–143, 2015.

[3] G. Cong and C. S. Jensen, "Querying geo-textual data: Spatial keyword queries and beyond," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 2207–2212.

[4] A. Mahmood and W. G. Aref, "Query processing techniques for big spatial-keyword data," in *Proceedings of the 2017 ACM International Conference on Management of Data*, 2017, pp. 1777–1782.

[5] P. Tampakis, D. Spyrellis, C. Doulkeridis, N. Pelekis, C. Kalyvas, and A. Vlachou, "A novel indexing method for spatial-keyword range queries," in *17th International Symposium on Spatial and Temporal Databases*, 2021, pp. 54–63.

[6] 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 *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. IEEE, 2019, pp. 1046–1057.

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

[8] Q. Huang, J. Du, G. Yan, Y. Yang, and Q. Wei, "Privacy-preserving spatio-temporal keyword search for outsourced location-based services," *IEEE Transactions on Services Computing*, 2021.

[9] X. Wang, J. Ma, F. Li, X. Liu, Y. Miao, and R. H. Deng, "Enabling efficient spatial keyword queries on encrypted data with strong security guarantees," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4909–4923, 2021.

[10] F. Song, Z. Qin, L. Xue, J. Zhang, X. Lin, and X. Shen, "Privacy-preserving keyword similarity search over encrypted spatial data in cloud computing," *IEEE Internet of Things Journal*, 2021.

[11] S. Kabir, C. Wagner, T. C. Havens, and D. T. Anderson, "A similarity measure based on bidirectional subsethood for intervals," *IEEE Transactions on Fuzzy Systems*, vol. 28, no. 11, pp. 2890–2904, 2020.

[12] Z. Zhang, K. Wang, W. Lin, A. W.-C. Fu, and R. C.-W. Wong, "Practical access pattern privacy by combining pir and oblivious shuffle," in *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019, pp. 1331–1340.

[13] M. S. Islam, M. Kuzu, and M. Kantarcioglu, "Access pattern disclosure on searchable encryption: ramification, attack and mitigation." in *Ndss*, vol. 20. Citeseer, 2012, p. 12.

[14] G. Kellaris, G. Kollios, K. Nissim, and A. O'neill, "Generic attacks on secure outsourced databases," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016, pp. 1329–1340.

[15] J. Peng, H. Wang, J. Li, and H. Gao, "Set-based similarity search for time series," in *Proceedings of the 2016 International Conference on Management of Data*, 2016, pp. 2039–2052.

[16] D. Amagata, S. Tsuruoka, Y. Arai, and T. Hara, "Feat-sksj: fast and exact algorithm for top-k spatial-keyword similarity join," in *Proceedings of the 29th International Conference on Advances in Geographic Information Systems*, 2021, pp. 15–24.

[17] B. Wang, M. Li, and L. Xiong, "Fastgeo: Efficient geometric range queries on encrypted spatial data," *IEEE transactions on dependable and secure computing*, vol. 16, no. 2, pp. 245–258, 2017.

[18] S. Zhang, S. Ray, R. Lu, Y. Zheng, Y. Guan, and J. Shao, "Towards efficient and privacy-preserving user-defined skyline query over single cloud," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[19] X. Zhu, E. Ayday, and R. Vitenberg, "A privacy-preserving framework for outsourcing location-based services to the cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 18, no. 1, pp. 384–399, 2019.
[20] J. Brickell and V. Shmatikov, "Privacy-preserving graph algorithms in the semi-honest model," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2005, pp. 236–252.
[21] S. Geravand and M. Ahmadi, "Bloom filter applications in network security: A state-of-the-art survey," *Computer Networks*, vol. 57, no. 18, pp. 4047–4064, 2013.
[22] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, "Fast homomorphic evaluation of deep discretized neural networks," in *Annual International Cryptology Conference*. Springer, 2018, pp. 483–512.
[23] M. Kim, H. T. Lee, S. Ling, B. H. M. Tan, and H. Wang, "Private compound wildcard queries using fully homomorphic encryption," *IEEE Transactions on Dependable and Secure Computing*, vol. 16, no. 5, pp. 743–756, 2017.
[24] X. Liu, R. H. Deng, K.-K. R. Choo, Y. Yang, and H. Pang, "Privacy-preserving outsourced calculation toolkit in the cloud," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 5, pp. 898–911, 2018.
[25] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption." *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.
[26] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.
[27] S. Zhang, S. Ray, R. Lu, Y. Zheng, Y. Guan, and J. Shao, "Ppaq: Privacy-preserving aggregate queries for optimal location selection in road networks," *IEEE Internet of Things Journal*, 2022.
[28] G. S. Çetin, Y. Doröz, B. Sunar, and E. Savaş, "Depth optimized efficient homomorphic sorting," in *International Conference on Cryptology and Information Security in Latin America*. Springer, 2015, pp. 61–80.
[29] R. Li, A. X. Liu, H. Xu, Y. Liu, and H. Yuan, "Adaptive secure nearest neighbor query processing over encrypted data," *IEEE Transactions on Dependable and Secure Computing*, 2020.
[30] P. Gupta and N. McKeown, "Algorithms for packet classification," *IEEE Network*, vol. 15, no. 2, pp. 24–32, 2001.
[31] N. Beckmann, H.-P. Kriegel, R. Schneider, and B. Seeger, "The r*-tree: An efficient and robust access method for points and rectangles," in *Proceedings of the 1990 ACM SIGMOD international conference on Management of data*, 1990, pp. 322–331.
[32] S. Chaudhuri, V. Ganti, and R. Kaushik, "A primitive operator for similarity joins in data cleaning," in *22nd International Conference on Data Engineering (ICDE'06)*. IEEE, 2006, pp. 5–5.
[33] O. Goldreich, *Foundations of cryptography: volume 2, basic applications*. Cambridge university press, 2009.
[34] "Yelp dataset," https://www.kaggle.com/yelp-dataset/yelp-dataset, 2020.
[35] Y. Zheng, R. Lu, Y. Guan, J. Shao, and H. Zhu, "Efficient and privacy-preserving similarity range query over encrypted time series data," *IEEE Transactions on Dependable and Secure Computing*, 2021.
[36] Y. Guan, R. Lu, Y. Zheng, S. Zhang, J. Shao, and G. Wei, "Toward privacy-preserving cybertwin-based spatiotemporal keyword query for its in 6g era," *IEEE Internet of Things Journal*, vol. 8, no. 22, pp. 16 243–16 255, 2021.

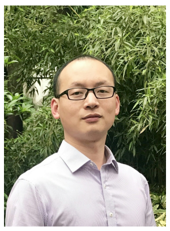
Songnian Zhang received his M.S. degree from Xidian University, China, in 2016 and he is currently pursuing his Ph.D. degree in the Faculty of Computer Science, University of New Brunswick, Canada. His research interest includes cloud computing security, big data query and query privacy.

Suprio Ray (Member, IEEE) is an Associate Professor with the Faculty of Computer Science, University of New Brunswick, Fredericton, Canada. He received a Ph.D. degree from the Department of Computer Science, University of Toronto, Canada, in 2015. His research interests include big data and database management systems, run-time systems for scalable data science, provenance and privacy issues in big data and query processing on modern hardware. E-mail: sray@unb.ca

Rongxing Lu (Fellow, IEEE) is a Mastercard IoT Research Chair, a University Research Scholar, an associate professor at the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. He is a Fellow of IEEE. His research interests include applied cryptography, privacy enhancing technologies, and IoT-Big Data security and privacy. He has published extensively in his areas of expertise, and was the recipient of 9 best (student) paper awards from some reputable journals and conferences. Currently, Dr. Lu serves as the Chair of IEEE ComSoc CIS-TC (Communications and Information Security Technical Committee), and the founding Co-chair of IEEE TEMS Blockchain and Distributed Ledgers Technologies Technical Committee (BDLT-TC).

Yunguo Guan is a PhD student of the Faculty of Computer Science, University of New Brunswick, Canada. His research interests include applied cryptography and game theory.

Yandong Zheng received the M.S. degree from the Department of Computer Science, Beihang University, China, in 2017, and received the Ph.D. degree from the Department of Computer Science, University of New Brunswick, Canada, in 2022. Since 2022, she has been an Associate Professor with the School of Cyber Engineering, Xidian University. Her research interest includes cloud computing security, big data privacy, and applied privacy.



Jun Shao (Senior Member, IEEE) received the Ph.D. degree from the Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, China, in 2008. He was a Post-Doctoral Fellow with the School of Information Sciences and Technology, Pennsylvania State University, Pennsylvania, PA, USA, from 2008 to 2010. He is currently a Professor with the School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou, China. His current research interests include network security and applied cryptography.