Published in IEEE Transactions on Dependable and Secure Computing (2023) 20 (7), 3358-3370.
DOI: 10.1109/TDSC.2022.3205670

Forward/Backward and Content Private DSSE for Spatial Keyword Queries

Xiangyu Wang, *Member, IEEE*, Jianfeng Ma, *Member, IEEE*, Ximeng Liu, *Senior Member, IEEE*, Yinbin Miao, *Member, IEEE*, Yang Liu, and Robert H. Deng, *Fellow, IEEE*

Abstract—Spatial keyword queries are attractive techniques that have been widely deployed in real-life applications in recent years, such as social networks and location-based services. However, existing solutions neither support dynamic update nor satisfy the privacy requirements in real applications. In this article, we investigate the problem of Dynamic Searchable Symmetric Encryption (DSSE) for spatial keyword queries. First, we formulate the definition of DSSE for spatial keyword queries (namely, DSSE_{SKQ}) and extend the DSSE leakage functions to capture the leakages in DSSE_{SKQ}. Then, we present a practical DSSE_{SKQ} construction based on geometric prefix encoding inverted-index and encrypted bitmap. Rigorous security analysis proves that our construction can achieve not only forward/backward privacy but content privacy as well, which can resist the most existing leakage-abuse attacks. Evaluation results using real-world datasets demonstrate the efficiency and feasibility of our construction. Comparative analysis reveals that our construction outperforms state-of-the-art schemes in terms of privacy and performance, e.g., our construction is $175\times$ faster than existing schemes with only 51% server storage cost.

Index Terms—Spatial keyword queries, dynamic searchable symmetric encryption, location-based services, forward/backward privacy, content privacy

1 Introduction

Nowadays, spatial keyword data services have brought great convenience to people's daily life. Diverse applications based on spatial keyword services such as social networks (e.g., *Facebook, Twitter*), location-based services (e.g., *Google Maps*), and location-based games (e.g., *PokÃ©mon Go*) are used by billions of users day by day. To cope with the storage and query challenges of massive *spatio-textual* data, several *spatio-textual* databases, such as Oracle Spatial[1] and PostGIS,[2] are proposed. In *spatio-textual* databases, Boolean Range Queries (BRQ) [1] is a typical type of spatial keyword query, which takes both the spatial geometric range and textual keyword into account. Given a geometric range query R and a set of textual keywords $\mathcal{W}_Q$, a BRQ search user aims to retrieve from a database all objects that are located in R and contain all keywords in $\mathcal{W}_Q$.

A mushrooming number of data owners are motivated to outsource their data services to the cloud for cost-saving and service flexibility. Unfortunately, the cloud server is not fully trusted as a third party, which means the privacy of the data and services outsourced to the cloud should be protected [2], [3], [4], [5], [6], [7], [8]. To solve this problem, Cui et al. [2] proposed several privacy-preserving BRQ schemes on encrypted data. They proposed a Bloom filter compression encoding method and constructed a Bloom filter R-tree index for BRQ. Then, they introduced the Asymmetric Scalar Product-preserving Encryption (ASPE) [6] to encrypt and search the tree structure. However, Li et al. [9] pointed out that the ASPE is insecure even in the Ciphertext-Only Attack (COA) model, which means the ASPE-based solutions cannot ensure query and data privacy. Next, Wang et al. [3] presented a BRQ scheme on encrypted data with strong security. They encoded the spatial data and textual keywords into gray code and bitmap, respectively, and searched the encoded objects through hidden vector encryption [10]. After that, Wang et al. [11] improved the search efficiency by introducing Hilbert curve and Bloom filter hierarchical tree. Recently, Tong et al. [12] considered the access control problem and proposed a privacy-preserving BRQ scheme with temporal access control. Unfortunately, the above schemes ignore the important requirement of dynamic update, which seriously hinders the practical application of *spatio-textual* data services in reality. For instance, an active user of social networks may frequently update her/his interests and locations.

How to construct efficient BRQ constructions on an encrypted database that support dynamic update is still an unsolved issue.

1. https://www.oracle.com/database/
2. http://www.postgis.org

- *Xiangyu Wang, Jianfeng Ma, Yinbin Miao, and Yang Liu are with the Shaanxi Key Laboratory of Network and System Security, Xidian University, Xi'an 710071, China. E-mail: xywang_xidian@163.com, jfma@mail.xidian.edu.cn, ybmiao@xidian.edu.cn, bcds2018@foxmail.com.*
- *Ximeng Liu is with the College of Computer and Data Science, Fuzhou University, Fuzhou 350116, China. E-mail: snbnix@gmail.com.*
- *Robert H. Deng is with the School of Information Systems, Singapore Management University, Singapore 188065. E-mail: robertdeng@smu.edu.sg.*

Manuscript received 22 March 2022; revised 23 August 2022; accepted 27 August 2022. Date of publication 12 September 2022; date of current version 11 July 2023.
This work was supported in part by the National Key Research and Development Program of China under Grant 2021YFB3101100, in part by the National Natural Science Foundation of China under Grants 62202364, 62072361, 62072109, and U21A20464, in part by the Key Research and Development Program of Shaanxi under Grant 2022GY-029, and in part by China 111 Project under Grant B16037.
(Corresponding authors: Xiangyu Wang and Ximeng Liu.)
Digital Object Identifier no. 10.1109/TDSC.2022.3205670

1.1 Summary and Limitations of Prior Arts

Dynamic Searchable Symmetric Encryption. Dynamic Searchable Symmetric Encryption (DSSE) [16] allows a client to encrypt a database before outsourcing it to an untrusted server, while the encrypted database can still be searched and updated. Almost all of the DSSE schemes allow reasonable leak information about the database and queries to improve search efficiency. Especially, update operations leak some information that can be used to reveal the private information of the database or queries. In 2016, Zhang et al. [17] confirmed powerful update leakage-abuse attacks called file-injection attacks, which are able to infer search queries by inserting a limited number of documents into encrypted databases. To resist such attacks, the forward privacy introduced in [18] has received significant attention. In [19], Bost formally defined the forward privacy for DSSE and proposed a concrete forward private DSSE scheme. Moreover, Bost et al. [20] formalized backward privacy with three-level privacy leakage-resistance levels from strong to weak, denoted Type-I to Type-III, and presented several corresponding schemes. Next, Chamani et al. [21] presented several backward private DSSE schemes based on ORAM. Sun et al. [22] designed a symmetric puncturable encryption scheme to construct a practical Type-III backward private DSSE. Recently, a series of forward/backward private DSSE constructions [23], [24], [25], [26], [27], [28], [29], [30], [31] were proposed to improve privacy, efficiency and searchability.

Limitation-I. Most of the existing forward/backward private DSSE schemes primarily focus on single-keyword queries, which do not satisfy the requirements of BRQ (BRQ requires both textual keyword queries and geometric range queries).

*Geometric Range Queries on Encrypted Data.*Usually, we need *compute-then-compare* operations to perform Geometric Range Queries (GRQ). For instance, to confirm whether a point is inside a circle, we first compute the distance from the point to the circle's center and then compare the circle's radius with the distance. Homomorphic Encryption (HE) [32] is an theoretically cryptography tools that can evaluate *compute-then-compare* operations securely. Nevertheless, the HE cannot directly reveal compare results from encrypted data to the cloud, which incurs multiple round-trips between the client and the server. To achieve efficient single round-trip GRQ, Xu et al. [33] proposed a scheme based on the polynomial fitting technique and the ASPE [6]. Unfortunately, as described above, the ASPE-based solution is not strong enough to cope with real-world attacks. Wang et al. [13], [34], [35] presented several GRQ schemes with strong security based on the Predicate Encryption (PE) [36]. Their key idea is to convert GRQ into the evaluation of whether the two vectors' inner product is zero so that they can perform a search securely using PE. In [34], they achieved circular range queries by mapping a circular range to a concentric circle set. Next, they represented spatial data and search queries by Bloom filters to support arbitrary geometric range queries [35]. To improve efficiency, they proposed FastGeo [13] that leverages a two-level search scheme to check whether a point is inside a geometric range or not. The first level is the x-axis coordinate of the spatial space and the second level is the y-axis that converted to equality-vector form. Following this work, Guo et al. [37] enhanced the search efficiency by replacing the y-axis equality-vector to GeoHash. Note that the encrypted database in FastGeo can be updated by uploading a vector of the second-level index, but it does not consider forward and backward privacy. In addition, recent researchers find that the access pattern and communication volume may enable an attacker to reconstruct the spatial dataset [38], [39], [40], [41], but they are not be considered in forward and backward privacy. The ORAM-based DSSE schemes [21] can hide the access pattern but still leak the volume pattern. At the same time, ORAM brings multiple round-trips of communication, which reduce the practicability of real applications.

To address the above-mentioned issues, Kasra et al. [14] defined a security notion called content privacy. Content privacy captures the leakages that do not be considered by forward/backward privacy for GRQ, it excludes the leakage on the updated points of the encrypted databases during update and search. Then, they proposed two schemes for range queries, the first scheme ($\mathsf{Constru-I}$) is Type-II backward and content private, and the second scheme ($\mathsf{Constru-II}$) is forward and content private. Next, to improve the search efficiency in large-scale databases, Kasra et al. [15] presented a Type-II backward and content private DSSE scheme based on R-tree and secret sharing, called Geo-DRS. However, Geo-DRS introduces two non-colluding servers, which are difficult to implement in real-world applications, to avoid multiple client-server round-trips. Moreover, $\mathsf{Constru-I}$, $\mathsf{Constru-II}$ and Geo-DRS convert the GRQ to their minimum bounding rectangle to perform the query, which cannot support arbitrary geometric range queries on encrypted databases.

Limitation-II. All of the above schemes can only support either forward or backward and content privacy, which cannot provide a comprehensive response to various privacy challenges.

1.2 Our Contributions

In this work, we investigate the challenging problem of DSSE for spatial keyword queries. The main contributions of our work are listed as below:

1) We formulate the new definition of DSSE for spatial keyword queries (namely, $\mathsf{DSSE_{SKQ}}$) and extend the leakage functions in DSSE to capture privacy leakages in $\mathsf{DSSE_{SKQ}}$ (see Section 3).
2) We propose a forward/backward and content private $\mathsf{DSSE_{SKQ}}$ construction. Specifically, we first reduce the geometric range queries to range queries through the Hilbert curve [42] and cover the range queries using prefix encoding. Then, based on the prefix encoded inverted-index and homomorphic encrypted bitmap, we present a $\mathsf{DSSE_{SKQ}}$ construction with forward/backward and content privacy (see Sections 4 and 5).
3) We implement the proposed construction and evaluate it over real-world datasets. The results show that our $\mathsf{DSSE_{SKQ}}$ is practical on large-scale datasets and more efficient than the other existing schemes. Particularly, our construction is $175\times$ faster than the state-of-the-art scheme with only 51% storage cost (see Section 6).

We compare the properties of our $\mathsf{DSSE_{SKQ}}$ with related prior work in Table 1. As far as we know, our construction

TABLE 1
Comparison With Existing Schemes

Scheme	Rectangle	GRQ	BRQ	Update	Forward	Backward	Content	Security
FastGeo [13]	✓	✓	✗	✓	✗	✗	✗	IND-SCPA
ELCBFR+ [2]	✓	✓	✓	✗	N/A	N/A	✗	COA
PBRQ [3]	✓	✓	✓	✗	N/A	N/A	✗	IND-SCPA
[11]	✓	✓	✓	✗	N/A	N/A	✗	IND-SCPA
Constru-I [14]	✓	✗	✗	✓	✗	Type-II	✓	IND-CPA
Constru-II [14]	✓	✗	✗	✓	✓	✗	✓	IND-CPA
Geo-DRS [15]	✓	✗	✗	✓	✗	Type-II	✓	IND-CPA
Our $\mathsf{DSSE}_{\mathsf{SKQ}}$	✓	✓	✓	✓	✓	Type-II	✓	IND-CPA

Note. Rectangle: rectangle range query for spatial data; GRQ: geometric range query for spatial data; BRQ: boolean range query for spatial-textual data; Update: dynamic update on encrypted database; Forward: forward privacy; Backward: backward privacy; Content: content privacy.

is the first to achieve dynamic update with forward/backward and content privacy for spatial keyword queries.

2 PRELIMINARIES

In this section, we review the cryptography tools and definitions used in our construction and security analysis.

2.1 Delegatable Pseudorandom Functions

Delegatable Pseudorandom Functions (DPRF) [43] enable the delegation of the evaluation of a PRF to an untrusted proxy according to a given predicate. The predicate defines the inputs of the PRF that will be evaluated on the proxy. For a range predicate DPRF, the master key holder who keeps the secret key K can generate a delegated key K_C associated with a range predicate $C \in \mathcal{C}$, where $\mathcal{C}$ is the whole range domain. The delegated key K_C can evaluate PRF on inputs v iff $v \in C$. Here, we briefly define a DPRF $\tilde{F}: \{0,1\}^\lambda \times \mathcal{V} \to \mathcal{Z}$: ($\tilde{F}$.$\mathtt{DelKey}$, $\tilde{F}$.$\mathtt{Derive}$) as follows:

- $K_C \leftarrow \tilde{F}.\mathtt{DelKey}(K, C)$: On input a master key $K \in \{0,1\}^\lambda$ and a range predicate $C \in \mathcal{C}$, it returns a delegated key K_C.
- $y \leftarrow \tilde{F}.\mathtt{Derive}(K_C, v)$: On input a delegated key K_C and $v \in \mathcal{V}$, it returns $z \in \mathcal{Z}$ iff $v \in C$.

2.2 An Lightweight Additive Homomorphic Symmetric Encryption

Following [44], a lightweight additive homomorphic symmetric encryption Π is described as follows:

- $n \leftarrow \mathtt{Init}(1^\lambda)$: On input a security parameter λ, it returns a public size parameter n, where $n = 2^l$, l is the maximum number of files can be supportd in a scheme.
- $e \leftarrow \mathtt{Enc}(sk, m, n)$: On input a a message $m (0 \leqslant m < n)$, a size parameter n, and a random one-time secret key $sk (0 \leqslant sk < n)$, this algorithm computes a ciphertext $e = sk + m \bmod n$.
- $m \leftarrow \mathtt{Dec}(sk, e, n)$: On input a ciphertext e, a size parameter n, and an one-time secret key sk, it decrypts the message as $m = e - sk \bmod n$.
- $\widehat{e} \leftarrow \mathtt{Add}(e_1, e_2, n)$: On input two ciphertexts e_1, e_2, and a size parameter n, it returns $\widehat{e} \leftarrow e_1 + e_2 \bmod n$.

Note that for two ciphertexts $e_1 = m_1 + sk_1 \bmod n$ and $e_2 = m_2 + sk_2 \bmod n$, anybody can compute $\widehat{e} = e_1 + e_2 \bmod n$. However, only the one know $sk_1 + sk_2 \bmod n$ can decrypt $\widehat{e}$ and recover $m_1 + m_2 \bmod n$. We have

$$\mathtt{Dec}(\widehat{sk}, \widehat{e}, n) = \widehat{e} - \widehat{sk} \bmod n = m_1 + m_2 \bmod n, \tag{1}$$

where $\widehat{sk} = sk_1 + sk_2 \bmod n$.

Perfectly Security [44]. Note that if every secret key sk is used once only, Π enjoys perfect security. If the secret key sk is kept secret, for any Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$, its advantage inthe perfectly-secure game is

$$\mathbf{Adv}^{\mathsf{PS}}_{\Pi,\mathcal{A}}(\lambda) = |\Pr[\mathcal{A}(\mathtt{Enc}(sk, m_1, n)) = 1] - \Pr[\mathcal{A}(\mathtt{Enc}(sk, m_2, n)) = 1]| \leq \mathsf{negl}(\lambda), \tag{2}$$

where $\mathsf{negl}(\lambda)$ denotes a negligible function in the security parameter λ, $n \leftarrow \mathtt{Init}(1^\lambda)$, $0 \leq m_1, m_2 < n$.

2.3 Dynamic Searchable Symmetric Encryption

A DSSE scheme contains an algorithm and two client-server protocols as follows:

- $(\sigma; \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda)$: On input a security parameter λ, it returns a state σ, and an encrypted database EDB.
- $\mathcal{R} \leftarrow \mathsf{Search}(Q, \sigma; \mathsf{EDB})$: On input a query Q and a state σ, the client produces a search request to the server, the server performs the search over EDB and return result $\mathcal{R}$ to the client after this protocol.
- $(\sigma'; \mathsf{EDB}') \leftarrow \mathsf{Update}(\sigma, \mathsf{up}, O; \mathsf{EDB})$: On input a state σ, a update object O and a update operation $\mathsf{up} \in \{\mathsf{ins}, \mathsf{del}\}$. The client first generates an update request to the server. Upon receiving the request, the server inserts or deletes the object O from EDB. Finally, the state σ' and the encrypted database EDB' are updated.

Security. Except for some explicit leakage, in a DSSE scheme, an adversary should not obtain any useful information from the encrypted database and the queries. As shown in [16], the DSSE security is captured by using the real-world versus ideal-world game. Let $\mathcal{L} = (\mathcal{L}^{\mathsf{Stp}}, \mathcal{L}^{\mathsf{Srch}}, \mathcal{L}^{\mathsf{Updt}})$ be a leakage function that capture the information leaked to the adversary from Setup, Search and Update, respectively.

The adversary's task is to distinguish between the following experiments *Real* and *Ideal*.

Definition 1 (Adaptive Security of DSSE). *For any PPT adversary $\mathcal{A}$ that makes a polynomial number of queries $q(\lambda)$, we say that a DSSE scheme* Setup, Search, Update *is $\mathcal{L}$-adaptively-secure, if there exists an efficient simulator $\mathcal{S}$ such that*

$$|\Pr[\mathbf{Real}_{\mathcal{A}}(\lambda)=1]-\Pr[\mathbf{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}(\lambda)=1]| \leq \mathsf{negl}(\lambda),$$

where $\mathbf{Real}_{\mathcal{A}}$ and $\mathbf{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}$ are defined as:

- $\mathbf{Real}_{\mathcal{A}}$*: $\mathcal{A}$ chooses a database* DB*, and obtains* EDB *from* Setup*. Next, $\mathcal{A}$ adaptively runs* Search *or* Update*. Finally, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$ by observing real transcripts of all operations.*
- $\mathbf{Ideal}_{\mathcal{A},\mathcal{S},\mathcal{L}}$*: $\mathcal{A}$ chooses \tag{2} a database* DB*, and obtains* EDB *from the simulator $\mathcal{S}(\mathcal{L}^{\mathsf{Stp}}(\mathsf{DB}))$. Next, $\mathcal{A}$ runs* Search *or* Update *(simulated by $\mathcal{S}(\mathcal{L}^{\mathsf{Srch}})$ or $\mathcal{S}(\mathcal{L}^{\mathsf{Updt}})$) adaptively. Eventually, $\mathcal{A}$ outputs a bit $b \in \{0,1\}$ by observing simulated transcripts for all operations.*

2.3.1 Generic Leakage Functions in DSSE

The leakage function $\mathcal{L}$ keeps as state the query list $\mathcal{Q}$, i.e., the list of all queries issued so far. Each entry of $\mathcal{Q}$ is either a search query (u, w) on keyword w or an update query $(u, \mathsf{up}, (w, f))$, where (w, f) is a keyword/file identifier pair and u is a timestamp initially set to 0 and gradually increased with queries. The general leakage functions [19] associated with DSSE schemes can be defined as follows:

- $\mathsf{sp}(w) = \{u | (u, (u, w)\} \in \mathcal{Q}\}$ denotes the search pattern which leaks whether two search queries correspond to the same keyword w or not.
- $\mathsf{TimeDB}(w) = \{(u, f) | (u, \mathsf{ins}, (w, f)) \in \mathcal{Q} \wedge \forall u,'$ $(u,' \mathsf{del}, (w, f)) \notin \mathcal{Q}\}$ denotes the list of all documents mathcing w, excluding the deleted ones, together with the timestamp of when they were inserted in the database.
- $\mathsf{Updates}(w) = \{u | (u, \mathsf{up}, (w, f) \in \mathcal{Q})\}$ denotes the list of timestamps of each insertion/deletion operation for w.

2.3.2 Forward/backward and Content Privacy

Forward and backward privacy of DSSE were first defined in [18], and first formalized by Bost et al. [19], [20]. In this subsection, we briefly review the original definitions of forward and backward privacy.

Definition 2 (Forward Privacy [19]). *We say that an $\mathcal{L}$-adaptively-secure DSSE scheme is forward-private iff $\mathcal{L}^{\mathsf{Updt}}$ is*

$$\mathcal{L}^{\mathsf{Updt}}(\mathsf{up}, (w, f)) = \mathcal{L}'(\mathsf{up}, (f_i, u_i)),$$

where (f_i, u_i) is a set that captures all updated files as the number of keywords u_i modified in file f_i, $\mathcal{L}'$ is stateless.

Definition 3 (Type-II Backward Privacy [20]). *We say that an $\mathcal{L}$-adaptively-secure DSSE scheme is Type-II backward-private iff $\mathcal{L}^{\mathsf{Srch}}$ and $\mathcal{L}^{\mathsf{Updt}}$ is*

$$\mathcal{L}^{\mathsf{Updt}}(\mathsf{up}, (w, f)) = \mathcal{L}'(\mathsf{up}, w),$$
$$\mathcal{L}^{\mathsf{Srch}}(w) = \mathcal{L}''(\mathsf{TimeDB}(w), \mathsf{Updates}(w), \mathsf{sp}(w)),$$

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

Content privacy was first introduced by Kasra et al. [14] for spatial data. They formulated a leakage that was not captured in forward/backward privacy. In addition to update patterns, content privacy also captures the leakage of updated file identifiers in the search and update. Specifically, backward privacy leaks about the content in the search queries via access pattern ($\mathsf{TimeDB}(w)$ in Def. 3), whereas the content privacy does not leak about the content during the search.

Definition 4 (Content Privacy [14]). *We say that an $\mathcal{L}$-adaptively-secure DSSE scheme is content-private iff $\mathcal{L}^{\mathsf{Srch}}$ and $\mathcal{L}^{\mathsf{Updt}}$ is*

$$\mathcal{L}^{\mathsf{Updt}}(\mathsf{up}, (w, f)) = \mathcal{L}'(\mathsf{up}, w),$$
$$\mathcal{L}^{\mathsf{Srch}}(w) = \mathcal{L}''(w),$$

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

TABLE 2
Notations Used in Our Construction

Notation	Description
λ	Security parameter
f	File identifier
$\mathsf{p} = \{x, y\}$	Spatial point
$\mathcal{W}$	Textual keywords set
N	# of objects in a database
$Q = \{R, \mathcal{W}_Q\}$	BRQ query
R	Geometric range of spatial data
$\mathcal{H}(\cdot)$	Hilbert curve encoding
$\lvert \cdot \rvert$	# of elements in a set
$\mathcal{P}$	A prefix family
BPC	Best prefix cover of a geometric range
l	Maximum number of supported files
$\mathsf{negl}(\lambda)$	A negligible function in λ

3 Problem Formulation

Here, we define the framework of DSSE for spatial keyword queries and formulate its security. Table 2 describes the main notations used in our work.

3.1 DSSE for Spatial Keyword Queries

System Model. As shown in Fig. 1, our system model consists of two entities, a client and a server. The client may be a company or an organization that stores its *spatio-textual* datasets on the server to reduce its local cost. Each object in a *spatio-textual* dataset contains a spatial point p and a textual keywords set $\mathcal{W}$. In addition, the client also wants to perform BRQ over its outsourced *spatio-textual* dataset. The server is *honest-but-curious*, which means, it will provide services honestly but try to obtain the client's data or search queries. Thus, the client encrypts its *spatio-textual* datasets and search queries with its own secret key before submitting them to the server. Meanwhile, the server should perform the BRQ

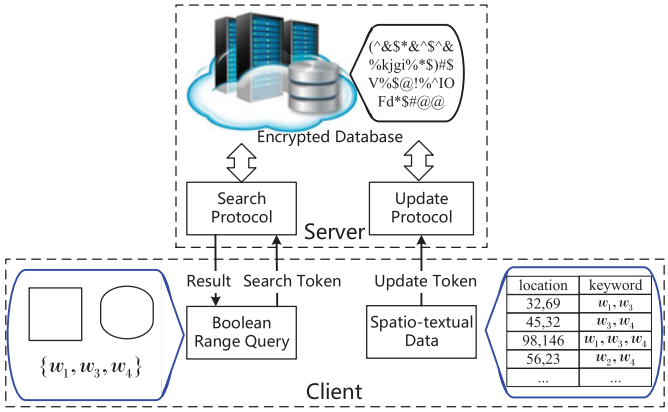


Fig. 1. System model.

search on encrypted data without decryption, and it should return the correct search results to the client.

Based on the system model as described above, we define the DSSE for spatial keyword queries (namely $\mathsf{DSSE_{SKQ}}$) as follows:

Definition 5 ($\mathsf{DSSE_{SKQ}}$). *A $\mathsf{DSSE_{SKQ}}$ scheme Σ contains one algorithm and two client-server protocols as follows:*

- $(K_\Sigma, \sigma; \mathsf{EDB}) \leftarrow \mathsf{Setup}(1^\lambda)$*: On input a security parameter λ, it ouputs a set of secret key K_Σ, a state σ, and an encrypted database* EDB.
- $\mathcal{R} \leftarrow \mathsf{Search}(K_\Sigma, Q = \{R, \mathcal{W}_Q\}, \sigma; \mathsf{EDB})$*: For a query $Q = \{R, \mathcal{W}_Q\}$, where R is a geometric range, and $\mathcal{W}_Q$ is a textual keywords set. The client generates a search request to the server, the server performs the search over* EDB *and returns results $\mathcal{R}$ to the client.*
- $(K_\Sigma, \sigma'; \mathsf{EDB}') \leftarrow \mathsf{Update}(K_\Sigma, \sigma, \mathsf{up}, O = \{\mathsf{p}, \mathcal{W}, f\}; \mathsf{EDB})$*: The client inputs $(K_\Sigma, \sigma, \mathsf{up}, O = \{\mathsf{p}, \mathcal{W}, f\})$, where $\mathsf{up} \in \{\mathsf{ins}, \mathsf{del}\}$ denotes an insertion or a deletion operation, p is a spatial loation, $\mathcal{W}$ is a set of textual keywords, and f is the identifier of the object O. The server inserts or deletes the object O from* EDB.

3.2 Leakage Functions in $\mathsf{DSSE_{SKQ}}$

Before defining the leakage functions in $\mathsf{DSSE_{SKQ}}$, we define a BRQ query $Q = (u, R, \mathcal{W}_Q) = \{u, p, w\}_{p \in \mathsf{BPC}, w \in \mathcal{W}_Q}$, where u is a timestamp, $p \in \mathsf{BPC}$ is a prefix element that belongs to the best prefix cover of a geometric range.[3] Define an update query $O = (u, \mathsf{up}, \mathsf{p}, \mathcal{W}, B) = \{u, p, w, B\}_{w \in \mathcal{W}, p \in \mathcal{P}}$, where u is a timestamp, $p \in \mathcal{P}$ is a prefix element that belongs to a prefix family[3], up is the update operation, and B is the bitmap of a list file identifier to be updated. The leakage functions associated with $\mathsf{DSSE_{SKQ}}$ can be defined as follows:

- $\mathsf{sp}'(Q) = \{u | (u, w, p)\}_{w \in \mathcal{W}_Q, p \in \mathsf{BPC}}$ denotes the search pattern repetition of search queries on $w \in \mathcal{W}_Q$ and $p \in \mathsf{BPC}$.
- $\mathsf{TimeDB}'(Q) = \{\mathsf{TimeDB}(w), \mathsf{TimeDB}(p)\}_{w \in \mathcal{W}_Q, p \in \mathsf{BPC}}$ denotes the list of all documents mathcing w, p, excluding the deleted ones, together with the timestamp of when they were inserted in the database.

3. See Section 4.1 for more details.

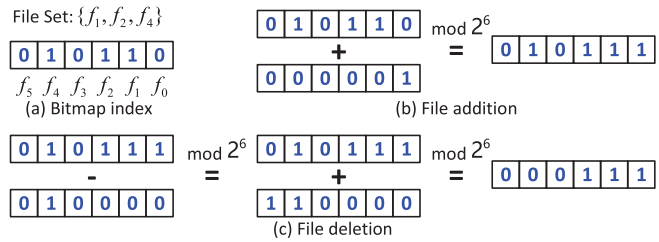


Fig. 2. Bitmap index example.

- $\mathsf{Updates}'(Q) = \{\mathsf{Updates}(w), \mathsf{Updates}(p)\}_{w \in \mathcal{W}_Q, p \in \mathsf{BPC}}$ denotes the list of timestamps of each insertion or deletion operation for w and p.

3.3 Forward/Backward and Content Privacy of $\mathsf{DSSE_{SKQ}}$

Definition 6 (Forward Privacy of $\mathsf{DSSE_{SKQ}}$). *We say that an $\mathcal{L}$-adaptively-secure DSSE scheme is forward-private iff $\mathcal{L}^{\mathsf{Updt}}$ is*

$$\mathcal{L}^{\mathsf{Updt}}(O) = \mathcal{L}'(\mathsf{up}, (f_i, u_i)),$$

where (f_i, u_i) is a set that captures all updated files as the number of keywords u_i modified in file f_i, $\mathcal{L}'$ is stateless.

Definition 7 (Type-II Backward Privacy of $\mathsf{DSSE_{SKQ}}$). *We say that an $\mathcal{L}$-adaptively-secure DSSE scheme is Type-II backward-private iff $\mathcal{L}^{\mathsf{Srch}}$ and $\mathcal{L}^{\mathsf{Updt}}$ is*

$$\mathcal{L}^{\mathsf{Updt}}(\mathsf{up}, (O, f)) = \{\mathcal{L}'(\mathsf{up}, w, p)\}_{w \in \mathcal{W}_Q, p \in \mathcal{P}},$$
$$\mathcal{L}^{\mathsf{Srch}}(w) = \mathcal{L}''(\mathsf{TimeDB}'(Q), \mathsf{Updates}'(Q), \mathsf{sp}'(Q)),$$

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

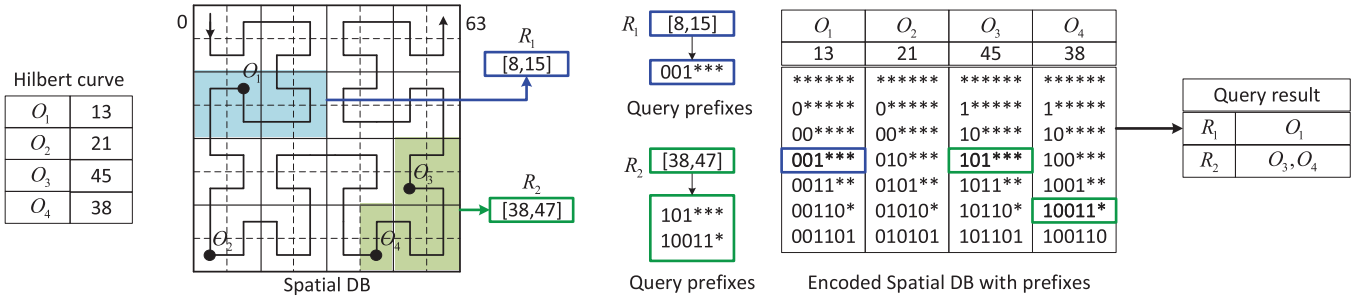
Definition 8 (Content Privacy of $\mathsf{DSSE_{SKQ}}$). *We say that an $\mathcal{L}$-adaptively-secure DSSE scheme is content-private iff $\mathcal{L}^{\mathsf{Srch}}$ and $\mathcal{L}^{\mathsf{Updt}}$ is*

$$\mathcal{L}^{\mathsf{Updt}}(\mathsf{up}, (O, f)) = \{\mathcal{L}'(\mathsf{up}, w, p)\}_{w \in \mathcal{W}_Q, p \in \mathcal{P}},$$
$$\mathcal{L}^{\mathsf{Srch}}(Q) = \{\mathcal{L}''(p, w)\}_{w \in \mathcal{W}_Q, p \in \mathsf{BPC}},$$

where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.

4 Proposed $\mathsf{DSSE_{SKQ}}$ Construction

In this section, we propose a $\mathsf{DSSE_{SKQ}}$ construction that supports BRQ and dynamic update. In particular, we first adopt a geometric prefix encoding method to convert geometric range queries into keyword queries as the basis of our $\mathsf{DSSE_{SKQ}}$. We then present our $\mathsf{DSSE_{SKQ}}$ construction with forward/backward and content privacy. The $\mathsf{DSSE_{SKQ}}$ represents file identifiers as bitmap index. To cover a *spatio-textual* database with up to l files, we set the string-length of B to l-bits. For an existing file f_i, the ith bit of B is set to 1; otherwise, the ith bit of B is set to 0. Fig. 2 gives an example of insertion and deletion of a bitmap. Specifically, Fig. 2a shows a bitmap index for a database with up to $l = 6$ files which contains files $\{f_1, f_2, f_4\}$. Fig. 2b shows an example of inserting a file f_0 to the database. Since f_0 corresponds to bit string 000001, 000001 is added to the index. Fig. 2c shows an example of file deletion operation. To delete a file f_4 from the



(a) Hilbert curve examples (b) Geometric prefix encoding and queries examples

Fig. 3. Hilbert curve and geometric prefix encoding examples.

database, we can subtract the string 010000 from the index or by adding $-(010000)_2 = (110000)_2$ to the index.

4.1 The Geometric Prefix Encoding Method

To construct forward/backward and content private DSSE for spatial keyword queries, our key idea is to reduce the BRQ to multiple single-keyword queries, so that we can support BRQ according to forward/backward and content private inverted-index. As we described in Section 1.1, BRQ requires both geometric range queries and textual keyword queries. Textual keyword queries can be supported by single-keyword queries. Hence, the main challenge is: *How to reduce geometric range queries to multiple single-keyword queries?* To address this issue, we adopt the geometric prefix encoding method [11]. Before describing the proposed method, we first introduce the Hilbert curve used in our method as follows.

Hilbert Space-Filling Curve. Space-filling curves are promising techniques [42] to map multi-dimensional spatial space to one-dimensional space. Since its clustering property, we use the classic Hilbert curve [42] as the building block. Hilbert curve ia a continuous Space-filling curve, if two data points are close to each other in the Hilbert curve, they are also close to each other in d-dimensional space. This property makes it beneficial to improve the performance of geometric prefix coding method. The basic element of a Hilbert curve is a U-shape, which can fill 2×2 square grid. To make the space division more accurate, we can represent this 2×2 grid as 4×4 grid and traverse the grid at a high level following 2×2 grid pattern. Denote h to be the order of the Hilbert curve, which means each space will be divided in half for h times. For a d-dimensional spatial space, an order h Hilbert curve will partition the whole space into 2^{dh} regions and represents each region as a dh-bits value. In Fig. 3a, we give examples of the Hilbert curve, where the original dimension is 2 and the order is 3. A user generates a query range as $R_1 = [8, 15]$ or $R_2 = [38, 47]$ if he/she wants to obtain all objects inside R_1 or R_2. Thus, we can transforme any geometric range queries on d-dimensional space into range queries on one-dimensional space. In the rest of this paper, we use $\mathcal{H}(\cdot)$ to denote the Hilbert encoding of a point or a range.

Geometric Prefix Encoding Method. We now formally present geometric prefix encoding method, including two algorithms $\mathsf{PreCode}$, $\mathsf{PreCover}$ as follows:

- $\mathcal{P} \leftarrow \mathsf{PreCode}(\mathsf{p})$: Given a spatial point p, it generates the Hilbert curve $\mathcal{H}(\mathsf{p})$ and outputs the corresponding prefix family $\mathcal{P}$. For a ω-bits Hilbert curve $\mathcal{H}(\mathsf{p}) = \beta_1\beta_2\cdots\beta_\omega$, we denote its prefix family as $\mathcal{P} = \{\beta_1\beta_2\cdots\beta_\omega, \beta_1\beta_2\cdots\beta_{\omega-1}*, \ldots, \beta_1 * \cdots *, * * \cdots *\}$, where the family size is $\omega + 1$ and ith prefix elements is $\beta_1\beta_2\cdots\beta_{\omega-i+1} * \cdots *$.
- $\mathsf{BPC} \leftarrow \mathsf{PreCover}(R)$: Given a geometric range query R, this algorithm first converts it into one-dimensional range queries $\mathcal{H}(R)$, and then outputs their Best Prefix Cover (BPC). For an one-dimensional range query $[x_{min}, x_{max}]$, the BPC is the minimum set of prefix elements that cover the range. For example, for a 4-bits spatial database, the search query of [4,7] is generated as $\mathsf{BPC} = \{01 * *\}$. For any spatial data X and a query range $[x_{min}, x_{max}]$, $X \in [x_{min}, x_{max}]$ iff $\mathsf{PreCode}(X) \cap \mathsf{BPC}([x_{min}, x_{max}]) \neq \emptyset$. For instance, for a 4-bits spatial database, the prefix family of 5 is $\mathcal{P} = \{0101, 010*, 01 * *, 0 * **, * * **\}$. We have $\mathcal{P} \cap \mathsf{BPC} = 01 * *$, thus 5 falls in range [4,7].

Let x_{min} and x_{max} be two ω-bits spatial data, the number of prefix elements in $\mathsf{PreCover}([x_{min}, x_{max}])$ is at most $2\omega - 2$. We can now reduce the geometric range queries to a multiple single-keyword queries according to the Hilbert curve and prefix encoding of spatial data. For the reader's understanding, we provide an example as follows.

Algorithm 1. Update Protocol

$\mathsf{Update}(K_\Sigma, O = \{\mathsf{p}, \mathcal{W}\}, B, \sigma; \mathsf{EDB} = \{\mathsf{SDB}, \mathsf{KDB}\})$:

@ Client:

$\mathcal{P} \leftarrow \mathsf{PreCode}(\mathsf{p})$;
foreach $p \in \mathcal{P}$ **do**
  $K_p || K'_p \leftarrow F_{K_\Sigma}(p), c \leftarrow \mathbf{T}[p]$;
  if $c = \perp$ **then** $c \leftarrow -1$;
  $T^p_{c+1} \leftarrow \tilde{F}.\mathtt{Derive}(K_p, C_{c+1}), \mathbf{T}[p] \leftarrow c + 1$;
  $UT^p_{c+1} \leftarrow H_1(K_p, ' T^p_{c+1})$;
  $sk^p_{c+1} \leftarrow H_2(K_p, ' c + 1)$;
  $e^p_{c+1} \leftarrow \mathtt{Enc}(sk^p_{c+1}, B, n)$;
  Send (UT^p_{c+1}, e^p_{c+1}) to the server.
foreach $w \in \mathcal{W}$ **do**
  $K_w || K'_w \leftarrow F_{K_\Sigma}(w), c \leftarrow \mathbf{T}[w]$;
  if $c = \perp$ **then** $c \leftarrow -1$;
  $T^w_{c+1} \leftarrow \tilde{F}.\mathtt{Derive}(K_w, C_{c+1}), \mathbf{T}[w] \leftarrow c + 1$;
  $UT^w_{c+1} \leftarrow H_1(K_w, ' T^w_{c+1})$;
  $sk^w_{c+1} \leftarrow H_2(K_w, ' c + 1)$;
  $e^w_{c+1} \leftarrow \mathtt{Enc}(sk^w_{c+1}, B, n)$;
  Send (UT^w_{c+1}, e^w_{c+1}) to the server.

@ Server:

foreach $p \in \mathcal{P}$ **do** $\mathsf{SDB}[UT^p_{c+1}] \leftarrow e^p_{c+1}$;
foreach $w \in \mathcal{W}$ **do** $\mathsf{KDB}[UT^w_{c+1}] \leftarrow e^w_{c+1}$;

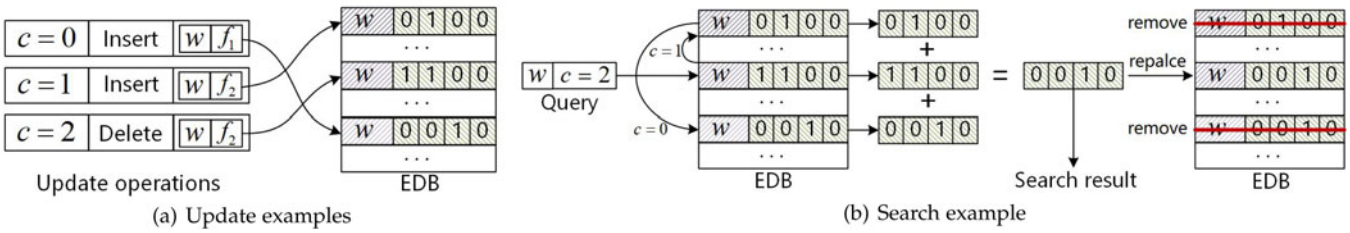


Fig. 4. Update and search examples of $\mathsf{DSSE}_{\mathsf{SKQ}}$.

Algorithm 2. Search Protocol

Search($K_\Sigma, Q = \{R, \mathcal{W}_Q\}, \sigma$; EDB $= \{$SDB, KDB$\}$):

@ Client:

BPC $\leftarrow$ PreCover(R);
foreach $w \in \mathcal{W}_Q$ **do**
  $K_w || K'_w \leftarrow F_{K_\Sigma}(w), c \leftarrow \mathbf{T}[w]$;
  if $c = \perp$ **then return** $\emptyset$;
  $ST^w \leftarrow \tilde{F}.\mathtt{DelKey}(K_w, C_c)$;
  Send $\{K'_w, ST^w, c\}$ to the server.
foreach $p \in$ BPC **do**
  $K_p || K'_p \leftarrow F_{K_\Sigma}(p), c \leftarrow \mathbf{T}[p]$;
  if $c = \perp$ **then return** $\emptyset$;
  $ST^p \leftarrow \tilde{F}.\mathtt{DelKey}(K_p, C_c)$;
  Send $\{K'_p, ST^p, c\}$ to the server.

@ Server:

$Sum_{\mathcal{P}} \leftarrow 0$;
foreach $\{K'_p, ST^p, c\}$ **do**
  $Sum^p_e \leftarrow 0$;
  for $i = c$ **to** 0 **do**
    $T^p_i \leftarrow \tilde{F}.\mathtt{Derive}(ST^p, i)$;
    $UT^p_i \leftarrow H_1(K'_p, T^p_i)$;
    $e^p_c \leftarrow$ KDB$[UT^p_i]$;
    if $e^p_c = \emptyset$ **then break**;
    else $Sum^p_e \leftarrow \mathtt{Add}(Sum^p_e, e^p_c, n)$;
    KDB$[UT^p_i] \leftarrow \emptyset$;
  KDB$[UT^p_c] \leftarrow Sum^p_e$;
  $Sum_{\mathcal{P}} \leftarrow \mathtt{Add}(Sum_{\mathcal{P}}, Sum^p_e, n)$;
foreach $\{K'_w, ST^w, c\}$ **do**
  $Sum^w_e \leftarrow 0$;
  for $i = c$ **to** 0 **do**
    $T^w_i \leftarrow \tilde{F}.\mathtt{Derive}(ST^w, i)$;
    $UT^w_i \leftarrow H_1(K'_w, T^w_i)$;
    $e^w_c \leftarrow$ KDB$[UT^w_i]$;
    if $e^w_c = \emptyset$ **then break**;
    else $Sum^w_e \leftarrow \mathtt{Add}(Sum^w_e, e^w_c, n)$;
    KDB$[UT^w_i] \leftarrow \emptyset$;
  KDB$[UT^w_c] \leftarrow Sum^w_e$;
Send $Sum_{\mathcal{P}}, \{Sum^w_e\}_{w \in \mathcal{W}_Q}$ to the client.

@ Client:

$Sum^{\mathcal{P}}_{sk} \leftarrow 0$;
for $p \in$ BPC **do**
  for $i = c$ **to** 0 **do**
    $sk^p_i \leftarrow H_2(K'_p, i)$;
    $Sum^{\mathcal{P}}_{sk} \leftarrow Sum^{\mathcal{P}}_{sk} + sk^p_i \bmod n$;
$B_{\mathcal{R}} \leftarrow \mathtt{Dec}(Sum^{\mathcal{P}}_{sk}, Sum_{\mathcal{P}}, n)$;
for $w \in \mathcal{W}_Q$ **do**
  $Sum^w_{sk} \leftarrow 0$;
  for $i = c$ **to** 0 **do**
    $sk^w_i \leftarrow H_2(K'_w, i)$;
    $Sum^w_{sk} \leftarrow Sum^w_{sk} + sk^w_i \bmod n$;
  $B_{\mathcal{R}} \leftarrow B_{\mathcal{R}} \& \mathtt{Dec}(Sum^w_{sk}, Sum^w_e, n)$;

Example. In Fig. 3b, we give examples of GRQ on the spatial database using geometric prefix encoding method. The example spatial database consists of four spatial objects. Each spatial object is encoded by geometric prefix encoding method. To query all objects in R_1 or R_2, a serach user encodes query area using Hilbert curve and generates the corresponding prefixes as $\{001 * **\}$ or $\{101 * **, 10011*\}$. Since $001 * ** \in \mathcal{P}(\mathcal{H}(O_1))$, $101 * ** \in \mathcal{P}(\mathcal{H}(O_3))$, and $10011* \in \mathcal{P}(\mathcal{H}(O_4))$, we have $O_1 \in R_1$ and $O_3, O_4 \in R_2$.

4.2 Our $\mathsf{DSSE}_{\mathsf{SKQ}}$ Construction

Overview of Our $\mathsf{DSSE}_{\mathsf{SKQ}}$. In our $\mathsf{DSSE}_{\mathsf{SKQ}}$, we use inverted-index to store the spatial keyword data. Specifically, the server keeps two maps SDB and KDB, which store keyword/identifier pairs and prefix/identifier pairs, independently. The prefixes are generated by the above-mentioned geometric prefix encoding method, and each identifier is represented by the bitmap. We employ DPRF to generate update tokens so that the update operation does not leak information about previous queries. When the client wants to update an object, he/she generates update tokens according to the DPRF. The server can perform searches over newly updated objects iff he/she obtains the newest delegated key of DPRF, which ensures that previously issued queries cannot be executed on newly inserted objects (i.e., forward privacy). To support backward and content privacy, we introduce the lightweight additive homomorphic symmetric encryption that can update files by adding bitmaps on encrypted data. Therefore, only the client can obtain the final query results without revealing the content of historical update operations, which does not leak information about the file identifiers.

Details of Our $\mathsf{DSSE}_{\mathsf{SKQ}}$. We now formally present our $\mathsf{DSSE}_{\mathsf{SKQ}}$ construction. In particular, our construction uses a DPRF $\tilde{F}$, a 2λ-bit PRF F, two keyed hash founctions H_1, H_2, and a lightweight additive homomorphic symmetric encryption scheme Π. Denote C_c be the range predicate $[0, c]$.

- Setup(1^λ): For a security parameter λ, it first randomly generates a master secret key K_Σ. Next, it initializes the public parameter n. The encrypted database EDB is initialized by two empty sets KDB, SDB, which store keyword and spatial prefix, respectively.
- Update$(K_\Sigma, O = \{\mathsf{p}, \mathcal{W}\}, B, \sigma;$ EDB$\})$: To update an object $O = \{\mathsf{p}, \mathcal{W}\}$, the client first obtains prefixes of p according to PreCode. Then, the client generates secret keys for every $p \in \mathcal{P}$ (or $w \in \mathcal{W}$) by F. Next, the client increases the update counter c and generates token T^p_{c+1} (or T^w_{c+1}) by $\tilde{F}$. The update token UT^p_{c+1} (or UT^w_{c+1}) is derived from the token by H_1. The bitmap is encrypted by Enc according to H_2. Finally, on the server-side, the SDB stores the prefix token/bitmap pairs, the KDB stores keyword token/bitmap pairs. Fig. 4a

shows examples to update a keyword w 3 times. At the first time, the client inserts w/f_1 into the EDB. The client first sets the update counter c as $c = 0$ and generates update token (data location in the EDB) according to DPRF and c. Next, the clients sets the bitmap of f_1 as 0010 and encrypts it using Enc. Finally, the client submits update token/encrypted bitmap pair to the server. The second update is similar to the first update, note that its update token is different from the first time since the update counter changed to $c = 1$. The third update is a deletion operation, thus the bitmap is set to 1100.

• Search$(K_\Sigma, Q = \{R, \mathcal{W}\}, \sigma; \mathsf{EDB})$: Given a query $Q = \{R, \mathcal{W}\}$, the client first obtains the best prefix cover BPC of R according to PreCover. Then, the client generates secret keys for every $p \in \mathcal{P}$ (or $w \in \mathcal{W}$), and derives search token ST^p (or ST^w) from $\tilde{F}$. Finally, the client sends $\{K'_p, ST^p, c\}$ and $\{K'_w, ST^w, c\}$ to the server. Upon receiving the search tokens, the server obtains index locations according to $\tilde{F}$ and H_1. Then, the server computes the updated bitmaps by Add and returns the ciphertexts of bitmaps $Sum_\mathcal{P}$, $\{Sum_e^w\}_{w\in\mathcal{W}}$ to the client.[4] The client decrypts the $Sum_\mathcal{P}$ to obtain all files matching R. The plaintext of Sum_e^w is the bitmap of all files matching $w \in \mathcal{W}$. To obtain all files matching BRQ Q, the server computes the intersection of geometric range query bitmap and every keyword query bitmap. As the example shown in Fig. 4b, the client needs to search the previously updated keyword w. She/he first generates search token UT_c^w according to w and the corresponding counter $[0, c]$. Next, the server backtracks all previously updated bitmaps of w according to the UT_c^w and c. The search result is the sum of all bitmaps and the server replaces the bitmap stored in $\mathsf{EDB}[UT_c^w]$ as the search result and removes all previously updated bitmaps to save the storage space.

Batch Updates. We can slightly modify the update protocol to support batch updates on files. For a list of files, we can update a single keyword/bitmap (or prefix/bitmap) pair for every unique keyword (or prefix), where the bitmap covers all update objects. For instance, referring to the example described in Fig. 2a, if a client wants to insert two files f_0, f_3 containing keyword w_1, she/he can generate an update corresponding to w_1/B, where $B = 001001$. As for deletion operation, if a client wants to delete two files f_4, f_2 from the bitmap in Fig. 2a, she/he can generate an update bitmap as $B = 101100$.

Arbitrary Boolean Queries for Textual Keywords. We can also slightly modify the search protocol to support arbitrary Boolean queries for textual keywords. In the Search protocol, the client decrypts every bitmap B_w corresponding to every textual keyword $w \in \mathcal{W}_Q$ and performs "AND" operations with spatial query result $B_\mathcal{R}$, which obtains all files that located in R and contain all keywords in $\mathcal{W}_Q$. Since the bitmap of every keyword is independent, the client can perform Boolean operations on the bitmaps of every textual keyword before performing "AND" operations with the spatial query result $B_\mathcal{R}$. In this way, the final result is all files that are located in R and satisfy the Boolean query rule of textual keywords. For example, if the search query is $Q = \{R, \{w_1 \text{ OR } w_2\}\}$, which means the search user wants to find all files that are located in R and contain w_1 or w_2. Suppose that $B_{w_1} = 00100$, $B_{w_2} = 01110$, and $B_\mathcal{R} = 11110$, the client first computes $B_\mathcal{W} = B_{w_1}$ OR $B_{w_2} = 01110$, and then computes $B_\mathcal{R} = B_\mathcal{R} \& B_\mathcal{W} = 01110$, which satisfy the "OR" Boolean queries for textual keywords.

4. Note that, to save the storage space, the locations are set to empty once they are searched if they are not the newest location. Only the newest location will be set to the final bitmap.

Algorithm 3. $\mathbb{G}_2$ and $\tilde{\mathbb{G}}_2$

Setup(1^λ):
1 $bad \leftarrow 0, n \leftarrow \mathtt{Init}(1^\lambda), K_\Sigma \xleftarrow{\$} \{0,1\}^\lambda$;
2 $\mathbf{T}, \mathsf{KDB}, \mathsf{SDB} \leftarrow \emptyset$;
3 **return** $(K_\Sigma, \mathbf{T}, n; \mathsf{EDB} = \{\mathsf{KDB}, \mathsf{SDB}\})$.

Update$(K_\Sigma, O = \{\mathsf{p}, \mathcal{W}\}, B, \sigma; \mathsf{EDB} = \{\mathsf{SDB}, \mathsf{KDB}\})$:
@ Client:
1 $\mathcal{P} \leftarrow \mathsf{PreCode}(\mathsf{p})$;
2 **foreach** $p \in \mathcal{P}$ **do**
3 $K_p || K'_p \leftarrow \mathbf{K}(p), c \leftarrow \mathbf{T}[p]$;
4 $T^p_{c+1} \leftarrow \tilde{F}.\mathtt{Derive}(K_p, c+1)$;
5 $\mathbf{T}[p] \leftarrow (T^p_0, T^p_1, \ldots, T^p_{c+1}, c+1)$;
6 $UT^p_{c+1} \leftarrow \{0,1\}^\lambda$;
7 **if** $\mathbf{H}_1(k'_p, T^p_{c+1}) \neq \perp$ **then**
8 $bad \leftarrow 1; UT^p_{c+1} \leftarrow \mathbf{H}_1(K'_p, T^p_{c+1})$
9 $\mathbf{UT}[p, c+1] \leftarrow UT^p_{c+1}$;
10 $sk^p_{c+1} \leftarrow H_2(K_p, ' c+1)$;
11 $e^p_{c+1} \leftarrow \mathtt{Enc}(sk^p_{c+1}, B, n)$;
12 Send (UT^p_{c+1}, e^p_{c+1}) to the server.
@ Server:
13 **foreach** $p \in \mathcal{P}$ **do**
14 $\mathsf{SDB}[UT^p_{c+1}] \leftarrow e^p_{c+1}$

Search$(K_\Sigma, Q = \{R, \mathcal{W}\}, \sigma; \mathsf{EDB} = \{\mathsf{SDB}, \mathsf{KDB}\})$:
@ Client:
1 $\mathsf{BPC} \leftarrow \mathsf{PreCover}(R)$;
2 **foreach** $p \in \mathsf{BPC}$ **do**
3 $K_p || K'_p \leftarrow \mathbf{K}(p), (T^p_0, T^p_1, \ldots, T^p_c, c) \leftarrow \mathbf{T}[p]$;
4 **for** $i = 0$ **to** c **do**
5 $\mathbf{H}_1(K'_p, T^p_i) \leftarrow \mathbf{UT}[p, i]$
6 $ST^p \leftarrow \tilde{F}.\mathtt{DelKey}(K_p, C_c)$;
7 Send $\{K'_p, ST^p, c\}$ to the server.
@ Server:
8 $Sum_\mathcal{P} \leftarrow 0$;
9 **foreach** $\{K'_p, ST^p, c\}$ **do**
10 $Sum^p_e \leftarrow 0$;
11 **for** $i = c$ **to** 0 **do**
12 $T^p_i \leftarrow \tilde{F}.\mathtt{Derive}(ST^p, i)$;
13 $UT^p_i \leftarrow H_1(K'_p, T^p_i)$;
14 $e^p_c \leftarrow \mathsf{KDB}[UT^p_i]$;
15 **if** $e^p_c = \emptyset$ **then break**;
16 **else** $Sum^p_e \leftarrow \mathtt{Add}(Sum^p_e, e^p_c, n)$;
17 $\mathsf{KDB}[UT^p_i] \leftarrow \emptyset$;
18 $\mathsf{KDB}[UT^p_c] \leftarrow Sum^p_e$;
19 $Sum_\mathcal{P} \leftarrow \mathtt{Add}(Sum_\mathcal{P}, Sum^p_e, n)$;
20 Send $Sum_\mathcal{P}$ to the client.

5 Security Analysis

In our $\mathsf{DSSE}_{\mathsf{SKQ}}$, the adversary cannot obtain the final file identifiers from the homomorphically encrypted bitmaps, which achieves content privacy. Content privacy provides more privacy preservation for forward/backward privacy. The Update with content privacy only leaks the number of keywords $|\mathcal{W}|$ and the prefix family size $|\mathcal{P}|$. As for Search, it leaks the search pattern of every search query Q. Besides,

the adversary can know update time for every $p \in \mathsf{BPC}$ and $w \in \mathcal{W}_Q$. However, the adversary cannot learn $\mathsf{TimeDB}'(Q)$ anymore since the content privacy hides the access pattern, which means our $\mathsf{DSSE}_{\mathsf{SKQ}}$ provides better privacy than Type-II backward privacy.

Theorem 1 (Forward/backward and content privacy of $\mathsf{DSSE}_{\mathsf{SKQ}}$). *Let F be a secure pseudo-random function, $\tilde{F}$ be a secure DPRF function, H_1, H_2 be two hash functions modeled as Random Oracles (RO), and $\Pi = (\mathtt{Init}, \mathtt{Enc}, \mathtt{Dec}, \mathtt{Add})$ be a perfectly secure additive homomorphic symmetric encryption. $\mathsf{DSSE}_{\mathsf{SKQ}}$ is $\mathcal{L}_{\mathsf{FB}}$-adaptively forward/backward and content private with $\mathcal{L}_{\mathsf{FB}} = \{\mathcal{L}^{\mathsf{Updt}}, \mathcal{L}^{\mathsf{Srch}}\}$ defined as*

$$\mathcal{L}^{\mathsf{Updt}}(\mathsf{ins}, O) = \mathcal{L}(|\mathcal{P}| + |\mathcal{W}|),$$
$$\mathcal{L}^{\mathsf{Srch}}(Q) = (\mathsf{sp}'(Q), \mathsf{Updates}'(Q)).$$

Proof. We use a hybrid argument that derives several games from the real-world game $\mathbf{Real}_{\mathcal{A}}(\lambda)$ to complete the proof proceeds. Since the proof of spatial data and textual keywords are similar, we got rid of code about textual keywords.

Game $\mathbb{G}_0$: It is the real-world game $\mathbf{Real}^{fb}_{\mathcal{A}}$

$$\Pr[\mathbf{Real}^{fb}_{\mathcal{A}}(\lambda) = 1] = \Pr[\mathbb{G}_0 = 1].$$

Game $\mathbb{G}_1$: Instead of calling the PRF F, this game stores a table $\mathbf{K}$ to map the query on p to $F(p)$. Every time a new spatial prefix is used, the table picks a new random output and stores the prefix/output pair. Hence, The adversary $\mathcal{A}$ cannot distinguish between $\mathbb{G}_0$ and $\mathbb{G}_1$ unless he/she break the security of PRF F. We can build a reduction $\mathcal{B}_1$ and have

$$\Pr[\mathbb{G}_0 = 1] = \Pr[\mathbb{G}_1 = 1] \leq \mathbf{Adv}^{\mathsf{prf}}_{F,\mathcal{B}_1}.$$

Game $\mathbb{G}_2$: This game stores update tokens produced by DPRF $\tilde{F}$ in the map $\mathbf{UT}$. Besides, nstead of calling the hash function H_1, it picks and stores random strings in the map $\mathbf{H}_1$. Algorithm 3 formally describes $\mathbb{G}_2$, and the additional boxed lines describes the intermediate game $\tilde{\mathbb{G}}_2$. Every time the Search protocol is called, the random oracle H_1 generates random tokens such that $H_1(K'_p, T^p_c) = \mathsf{UT}[p, c]$. In addition, in order to exactly program H_1 when it is queried by the adversary on a valid (K'_p, T^p_c) couple, $\tilde{\mathbb{G}}_2$ and $\mathbb{G}_2$ make some bookkeeping of the tokens T^p_c. Thus, H_1's behaviors in $\tilde{\mathbb{G}}_2$ and $\mathbb{G}_1$ are perfectly indistinguishable:

$$\Pr[\tilde{\mathbb{G}}_2 = 1] = \Pr[\mathbb{G}_1 = 1].$$

H_1 will produce random result if $\mathbf{H}_1$ does not include the tuple (K'_p, T^p_c), but if the update token T^p_c gets a collision with another token, H_1 will return the corresponding value of the equivalent token and set a flag *bad* to 1. Thus, the advantage of distinguishing between $\mathbb{G}_2$ and $\tilde{\mathbb{G}}_2$ is smaller than the probability that the $bad \leftarrow 1$ occurs in $\tilde{\mathbb{G}}_2$

$$|\Pr[\mathbb{G}_2 = 1] - \Pr[\tilde{\mathbb{G}}_2 = 1]| \leq \Pr[bad \leftarrow 1 \ in \ \tilde{\mathbb{G}}_2].$$

Note that $bad \leftarrow 1$ occurs in $\tilde{\mathbb{G}}_2$ iff an adversary breaks the one-wayness of DPRF $\tilde{F}$. Also, for an adversary who makes ℓ queries to the RO, the probability that H_1 was called on (K'_p, T^p_c) is $\ell \cdot 2^{-\lambda}$ since T^p_c is uniformly random. By constructing a reduction $\mathcal{B}_2$ from a distinguisher $\mathcal{A}$ inserting $\mathcal{N}$ prefix/identifier pairs in the database, we have

$$|\Pr[\mathbb{G}_1 = 1] - \Pr[\mathbb{G}_2 = 1]| = |\Pr[\tilde{\mathbb{G}}_2] - \Pr[\mathbb{G}_2]| \leq \mathcal{N} \cdot \mathbf{Adv}^{\mathsf{dprf}}_{\tilde{F},\mathcal{B}_2}(\lambda) + \frac{\mathcal{N}\ell}{2^\lambda}.$$

Game $\mathbb{G}_3$. The H_2 is modeled as a RO in this game. If the key K'_p keeps secret, for an adversary $\mathcal{A}$ who makes ℓ queries, we have

$$|\Pr[\mathbb{G}_2 = 1] - \Pr[\mathbb{G}_3 = 1]| \leq \frac{\ell}{2^\lambda}.$$

Game $\mathbb{G}_4$. We replace the bitmap B by the bitmap of all zeros. The adversary $\mathcal{A}$ cannot distinguish between $\mathbb{G}_4$ and $\mathbb{G}_3$ unless he/she break the perfect security of Π, so we can build a reduction $\mathcal{B}_3$ and have

$$|\Pr[\mathbb{G}_3 = 1] - \Pr[\mathbb{G}_4 = 1]| \leq \mathbf{Adv}^{\mathsf{ps}}_{\Pi,\mathcal{B}_3}(\lambda).$$

Simulator. The ideal world simulator is shown in Algorithm 4, we get rid of a part of the code in Algorithm 3 that does not influence the view of the adversary. For the BRQ query Q, we use the first timestamp $\widehat{Q} \leftarrow \min\mathsf{sp}'(Q)$.

Next, we will show that $\mathbb{G}_4$ and *Simulator* $\mathcal{S}$ are indistinguishable. For Update, the indistinguishability is obvious since each update in $\mathbb{G}_4$ is a new random string. As for Search, for each p generated from $\widehat{Q}$, we program H_1 and H_2 according to **UT** and **sk**, respectively. Eventually, the pairs (p, i) are mapped to the global update count u, which helps us to map the values randomly chosen in the Update to the corresponding values in the Search. Hence, we have

$$\Pr[\mathbb{G}_5 = 1] = \Pr[\mathbf{Ideal}^{fb}_{\mathcal{A},\mathcal{S},\mathcal{L}_{\mathsf{FB}}}(\lambda) = 1].$$

Conclusion. By combining all of the above games, the advantage of any PPT adversary attacking our scheme is

$$|\Pr[\mathbf{Real}^{fb}_{\mathcal{A}}(\lambda) = 1] - \Pr[\mathbf{Ideal}^{fb}_{\mathcal{A},\mathcal{S},\mathcal{L}_{\mathsf{FB}}}(\lambda)] = 1|$$
$$\leq \mathbf{Adv}^{\mathsf{prf}}_{F,\mathcal{B}_1} + \mathcal{N} \cdot \mathbf{Adv}^{\mathsf{dprf}}_{\tilde{F},\mathcal{B}_2}(\lambda) + \mathbf{Adv}^{\mathsf{ps}}_{\Pi,\mathcal{B}_3}(\lambda) + \frac{(\mathcal{N}+1)\ell}{2^\lambda},$$

which completes the proof. □

6 Performance Analysis

In this section, we analyze the performance of the proposed construction from both theoretical and experimental aspects.

6.1 Theoretical Analysis

Here, we analyze the theoretical computation and communication complexity of our construction. In Update, the client generates update tokens for every prefix $p \in \mathcal{P}$ and every keyword $w \in \mathcal{W}$, the computation cost is $\mathcal{O}(|\mathcal{P}| + |\mathcal{W}|)$. Since each update token (UT, e) is $(2\lambda + l)$-bits, the communication cost is $\mathcal{O}((|\mathcal{P}| + |\mathcal{W}|)(2\lambda + l))$. In Search, for every prefix $p \in \mathsf{BPC}$ and keyword $w \in \mathcal{W}_Q$, the client generates a search

TABLE 3
Theoretically Performance Comparison

Scheme	Update		Search		EDB Storage
	Computational	Communication	Computational	Communication	
ELCBFR+	$\mathcal{O}(2M(N+\log N))$	$\mathcal{O}(2M(N+\log N)\lambda)$	$\mathcal{O}(2M\log N)$	$\mathcal{O}(2M\lambda)$	$\mathcal{O}(2M(N+\log N)\lambda)$
PBRQ-Q	$\mathcal{O}(T(N+\log N))$	$\mathcal{O}(T(N+\log N)\lambda)$	$\mathcal{O}(Tk\log N)$	$\mathcal{O}(Tk\lambda)$	$\mathcal{O}(T(N+\log N)\lambda)$
Constu – I	$\mathcal{O}(tN)$	$\mathcal{O}(tN)$	$\mathcal{O}(N\log R')$	$\mathcal{O}(N\log R')$	$\mathcal{O}(2^tN)$
Constu – II	$\mathcal{O}(2^tN)$	$\mathcal{O}(2^tN)$	$\mathcal{O}(N\log R')$	$\mathcal{O}(N\log R')$	$\mathcal{O}(2^tN)$
Ours	$\mathcal{O}(\lvert\mathcal{P}\rvert+\lvert\mathcal{W}\rvert)$	$\mathcal{O}((\lvert\mathcal{P}\rvert+\lvert\mathcal{W}\rvert)(2\lambda+l))$	$\mathcal{O}(a_w(\lvert\mathsf{BPC}\rvert+\lvert\mathcal{W}_Q\rvert))$	$\mathcal{O}((1+\lvert\mathcal{W}_Q\rvert)l)$	$\mathcal{O}((2^{\lvert\mathcal{P}\rvert}+n_w)(2\lambda+l))$

Note. *N: number of data in dataset; M: the size of a Bloom filter in [2]; T, k: the size of Gray code and bitmap, and the number of query token in [3]; R,' t: side length of a square query, and bit length of coordinates (x and y) in [14]; a_w, n_w: number of new updates, number of unique keyword in dataset. Note that $|\mathcal{W}|, |\mathcal{W}_Q|, |\mathcal{P}|, |\mathsf{BPC}| \ll M, T, 2^t, R'$.*

token. For every new update of p and w, the server computes the encrypted bitmap according to the search tokens, the computation cost is $\mathcal{O}(a_w(|\mathsf{BPC}| + |\mathcal{W}_Q|))$. The search result is an encrypted bitmap for the geometirc range query and $|\mathcal{W}_Q|$ encrypted bitmaps for keyword queries, the communication cost is $\mathcal{O}((1+|\mathcal{W}_Q|)l)$. Since the maxmium number of prefixes is $2^{|\mathcal{P}|}$, the maxmium storage cost of a one-time updated EDB is at most $\mathcal{O}((2^{|\mathcal{P}|}+n_w)(2\lambda+l))$, where n_w is the number of unique keyword. Besides, every time the client updates a new object, the EDB stores the newly updated tokens in new locations, which incurs $\mathcal{O}(a_w(|\mathcal{P}|+|\mathcal{W}|)(2\lambda+l))$ storage cost. Note that the additional storage cost caused by newly updated objects is much smaller than $\mathcal{O}((2^{|\mathcal{P}|}+n_w)(2\lambda+l))$ and the additional storage will be removed when a search query is performed, the storage cost is still $\mathcal{O}((2^{|\mathcal{P}|}+n_w)(2\lambda+l))$ for a frequently queried database.

Algorithm 4. Simulator $\mathcal{S}$

Update($|\mathcal{P}|$):
@ *Client:*
1 **for** 0 **to** $|\mathcal{P}|$ **do**
2 $\quad \mathbf{UT}[u] \xleftarrow{\$} \{0,1\}^\lambda$ $\mathbf{sk}[u] \leftarrow \{0,1\}^\lambda$;
3 $\quad \mathbf{e}[u] \leftarrow \mathtt{Enc}(\mathbf{sk}[u], 0\ s, n)$;
4 $\quad$ Send $\{\mathbf{UT}[u], \mathbf{e}[u]\}$ to the server;
5 $\quad u \leftarrow u+1$;

Search(sp'(Q), Updates'(Q)):
@ *Client:*
1 $\widehat{Q} \leftarrow \min$sp'$(Q)$;
2 $\widehat{\mathsf{BPC}} \leftarrow \widehat{Q}$;
3 **for** $p \in \widehat{\mathsf{BPC}}$ **do**
4 $\quad K_p || K'_p \leftarrow \mathbf{K}[p], (T^p_c, c) \leftarrow \mathbf{T}[p]$;
5 $\quad$ Parse Updates'(p) as $(u_0, u_1, \ldots, u_c)$, where Updates'$(p) \in$ Updates'$(\widehat{Q})$;
6 $\quad$ **for** $i = 0$ **to** c **do**
7 $\quad\quad$ Program H_1 s.t. $H_1(K'_p, T^p_i) \leftarrow \mathbf{UT}[u_i]$;
8 $\quad\quad$ Program H_2 s.t. $H_2(K'_p, i) \leftarrow \mathbf{sk}[u_i]$
9 $\quad$ Send $\{K'_p, ST^p, c\}$ to the server;

In Table 3, we compare our construction with existing schemes.[5] Note that the ELCBFR+ [2] and PBRQ-Q [3] are static schemes that generate the whole encrypted database during the system setup, we employ their computation and communication costs of the whole encrypted database as the cost of Update in the table. As for Constru – I, Constru – II [14], and our construction, we show the cost of a single update. We can observe that the update and search complexity of our construction only rely on the size of $\mathcal{W}, \mathcal{W}_Q, \mathcal{P}$, and BPC, which are much smaller than $T, 2^t, M, N$ in other schemes. 2^t, T and M are usually large (larger than 10^4) to guarantee query accuracy, and the N is obviously large in a large-scale dataset. Hence, the computation cost of our construction is significantly lower than that of the other schemes.

6.2 Experimental Evaluation

Implementation. We use the JAVA programming language to implement our construction. The PRF F and keyed hash functions H_1, H_2 are instantiated with HMAC-SHA256. For the DPRF $\tilde{F}$, we implement it using GGM PRF [43]. All experiments are executed on a machine with 2.6 GHz CPU and 128 GB RAM. The security parameter is set to $\lambda = 128$.

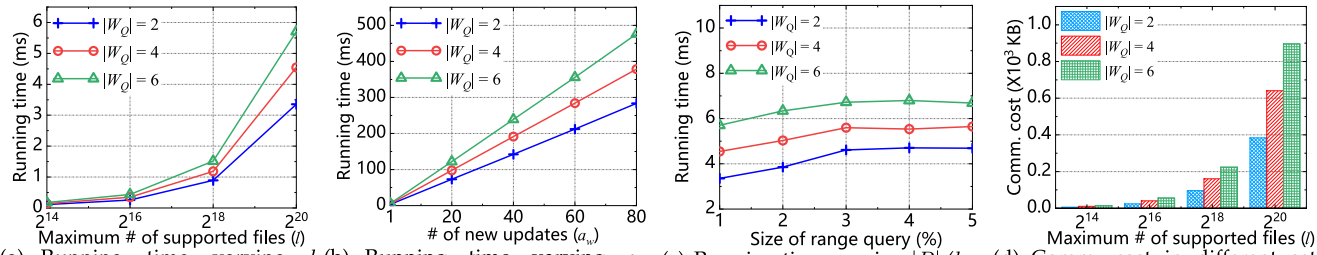
Datasets. Similar to the related spatial keyword schemes [2], [3], we use a combination of a real spatial dataset FLA[6] and a real document dataset 20 Newsgroups[7] to eveluate the proformance. The FLA contains 1,070,376 different locations. We randomly choose 3,000 unique textual keywords from the 20 Newsgroups. For each object in FLA, we attach the keywords from a 20 Newsgroups document, where each object contains 12 keywords ($|\mathcal{W}| = 12$) for average.

Performance of Update. As shown in Figs. 6a and 6b, we test the running time and communication cost of every update with different $|\mathcal{P}|$ and l, respectively, where $|\mathcal{W}| = 12$. Since the computation cost and ciphertext size of Enc grow linearly with increasing l, it is obvious that both the communication cost and running time of the update grow linearly with the increasing l. Besides, since a larger $|\mathcal{P}|$ incurs more update tokens, the running time and communication cost of the update also grow linearly with the increasing $|\mathcal{P}|$.

Performance of Search. We evaluate the search time with different l and $|W|_Q$, where $|\mathsf{BPC}| = 3$, $a_w = 1$, and plot the results in Fig. 5a. We can observe that the search time grows linearly with increasing l and $\mathcal{W}_Q$. In Search, the server performs the Add for every search token, whose running time grows linearly with increasing l. In insertion, a larger $|\mathcal{P}|$ incurs more search tokens, which raises the search time. We also evaluate the search time with different a_w and $|\mathcal{W}_Q|$ in Fig. 5b, where $|\mathsf{BPC}| = 3, l = 2^{20}$. Obviously, the running

5. In this paper, we do not compare our construction with Geo-DRS because Geo-DRS uses two non-colluding servers, which is a different model from the other works.

6. http://users.diag.uniroma1.it/challenge9
7. http://qwone.com/~jason/20Newsgroups/

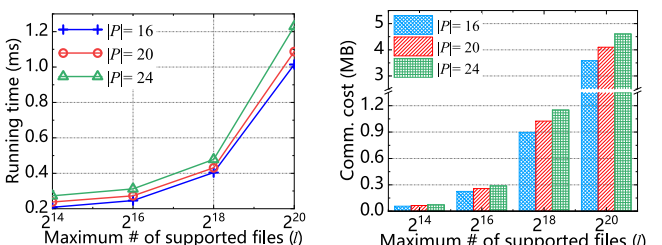


(a) Running time varying l (|BPC| = 3, $a_w = 1$) (b) Running time varying a_w (|BPC| = 3, $l = 2^{20}$) (c) Running time varying $|R|$ ($l = 2^{20}$, $a_w = 1$) (d) Comm. cost in different settings (|BPC| = 3)

Fig. 5. Performance of Search.

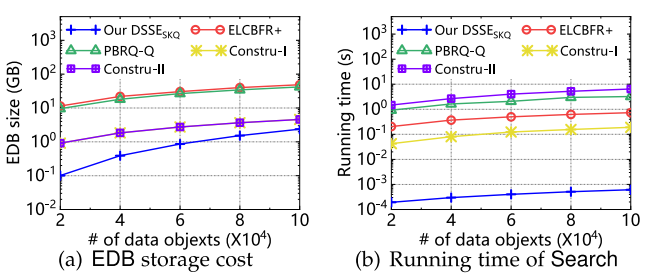
time of Search grows linearly with increasing a_w and $|\mathcal{W}_Q|$. Since every newly updated object performs insertion by Add on the server-side, larger a_w incurs more Add, which increase the computation cost. Next, we plot the running time of Search with different size of query range $|R|$ (% of the map), where $l = 2^{20}, a_w = 1$. From Fig. 5c, we can observe that the search time is not linearly increasing with the query range size. This is because the search time depends on the number of prefixes that cover the query range (i.e., |BPC|), which does not linearly increase with the query range size. Finally, we show the communacation cost of Search in Fig. 5d with different l and $|\mathcal{W}_Q|$, where |BPC| = 3. The communication cost grows linearly with increasing both l and $|\mathcal{W}_Q|$, because the number of search tokens depends on $|\mathcal{W}_Q|$ and the size of result bitmaps depends on l.

Comparative Evaluation. Here, we compare our construction with the existing schemes. Specifically, we implement the ELCBFR+ [2], PBRQ-Q [3], Constru – I and Constru – II [14] using the JAVA programming language. For ELCBFR+, we set the $m/n = 5$ and $t' = 5$ of Bloom filter, where m is the length of Bloom filter, n is the number of elements inserted into Bloom filter, and t' is the number of hash functions. As for PBRQ-Q, the maximum capacity of deepest non-leaf nodes is set to 16. For a fair comparison, we set $l = N$ in our construction, and generate our EDB using batch update in one time (i.e., $a_w = 1$). The search query is a square R with five keywords (i.e., $|\mathcal{W}_Q| = 5$), where $|R| = 5\%$. We vary the number of data objects N from 2×10^4 to 10^5 to compare the performance. Figs. 7a and 7b show the EDB storage cost and search time against data sizes. We can observe that both the EDB size and search time grow with increasing data sizes for all schemes. Besides, both the EDB storage cost and the search time of our construction outperform that of existing schemes. Specifically, the search time of our construction over 10^5 data is only 1.241 ms, which is $175\times$ faster than the most efficient existing scheme Constru – I. As for storage cost, the EDB in our construction costs 2.369 GB, which is 51.7% of the best existing scheme Constru – I. Note that ELCBFR+ and PBRQ-Q cannot support dynamic update. Although Constru – I [14] and Constru – II are able to support updates, they can only achieve either backword or forward privacy, which cannot provide a comprehensive privacy guarantee. Generally speaking, both the performance and privacy guarantee of our construction outperform those of the existing schemes.



(a) Running time of Update ($|\mathcal{W}| = 12$) (b) Comm. cost of Update ($|\mathcal{W}| = 12$)

Fig. 6. Performance of Update.



(a) EDB storage cost (b) Running time of Search

Fig. 7. Performance comparison.

7 Conclusion

In this work, we formally defined the framework and security of DSSE for spatial keyword queries and proposed a corresponding practical construction with forward/backward and content privacy. We provided detailed security analysis and performance evaluations to prove the security and efficiency of our construction. Our construction is the first to achieve both dynamic update, forward/backward and content privacy for spatial keyword queries. We would like to design more scalable DSSE schemes with more expressive queries in our future work.

References

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

[2] N. Cui, J. Li, X. Yang, B. Wang, M. Reynolds, and Y. Xiang, "When geo-text meets security: Privacy-preserving boolean spatial keyword queries," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 1046–1057.

[3] X. Wang et al., "Search me in the dark: Privacy-preserving boolean range query over encrypted spatial data," in *Proc. IEEE 39th Conf. Comput. Commun.*, 2020, pp. 2253–2262.

[4] G. Xu, H. Li, H. Ren, K. Yang, and R. H. Deng, "Data security issues in deep learning: Attacks, countermeasures, and opportunities," *IEEE Commun. Mag.*, vol. 57, no. 11, pp. 116–122, Nov. 2019.

[5] X. Wang, J. Ma, Y. Miao, X. Liu, and R. Yang, "Privacy-preserving diverse keyword search and online pre-diagnosis in cloud computing," *IEEE Trans. Serv. Comput.*, vol. 15, no. 2, pp. 710–723, Mar./Apr. 2022.

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

[7] X. Wang, J. Ma, X. Liu, and Y. Miao, "Search in my way: Practical outsourced image retrieval framework supporting unshared key," in *Proc. IEEE Conf. Comput. Commun.*, 2019, pp. 2485–2493.

[8] G. Xu, H. Li, Y. Zhang, S. Xu, J. Ning, and R. Deng, "Privacy-preserving federated deep learning with irregular users," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 2, pp. 1364–1381, Mar./Apr. 2022.

[9] R. Li, A. Liu, Y. Liu, H. Xu, and H. Yuan, "Insecurity and hardness of nearest neighbor queries over encrypted data," in *Proc. IEEE 35th Int. Conf. Data Eng.*, 2019, pp. 1614–1617.

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

[11] 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 Trans. Inf. Forensics Secur.*, vol. 16, pp. 4909–4923, 2021.

[12] Q. Tong, X. Li, Y. Miao, X. Liu, J. Weng, and R. Deng, "Privacy-preserving boolean range query with temporal access control in mobile computing," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2022.3152168.

[13] B. Wang, M. Li, and L. Xiong, "FastGeo: Efficient geometric range queries on encrypted spatial data," *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 2, pp. 245–258, Mar./Apr. 2019.

[14] S. K. Kermanshahi et al., "Geometric range search on encrypted data with forward/backward security," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 698–716, Jan./Feb. 2022.

[15] S. K. Kermanshahi et al., "Geo-DRS: Geometric dynamic range search on spatial data with backward and content privacy," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2021, pp. 24–43.

[16] S. Kamara, C. Papamanthou, and T. Roeder, "Dynamic searchable symmetric encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2012, pp. 965–976.

[17] Y. Zhang, J. Katz, and C. Papamanthou, "All your queries are belong to us: The power of file-injection attacks on searchable encryption," in *Proc. 25th USENIX Secur. Symp.*, 2016, pp. 707–720.

[18] C. B. Papamanthou, and E. Shi, "Practical dynamic searchable encryption with small leakage," in *Proc. Annu. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–15.

[19] R. Bost, ": Forward secure searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1143–1154.

[20] R. Bost, B. Minaud, and O. Ohrimenko, "Forward and backward private searchable encryption from constrained cryptographic primitives," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1465–1482.

[21] J. Chamani, D. Papadopoulos, C. Papamanthou, and R. Jalili, "New constructions for forward and backward private symmetric searchable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 1038–1055.

[22] S. Sun et al., "Practical backward-secure searchable encryption from symmetric puncturable encryption," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2018, pp. 763–780.

[23] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption with forward and stronger backward privacy," in *Proc. 24th Eur. Symp. Res. Comput. Secur.*, 2019, pp. 283–303.

[24] I. Demertzis, J. G. Chamani, D. Papadopoulos, and C. Papamanthou, "Dynamic searchable encryption with small client storage," in *Proc. 27th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2020, pp. 1–18.

[25] K. He, J. Chen, Q. Zhou, R. Du, and Y. Xiang, "Secure dynamic searchable symmetric encryption with constant client storage cost," *IEEE Trans. Inf. Forensics Secur.*, vol. 16, pp. 1538–1549, 2020.

[26] T. Chen, P. Xu, W. Wang, Y. Zheng, W. Susilo, and H. Jin, "Bestie: Very practical searchable encryption with forward and backward security," in *Proc. Eur. Symp. Res. Comput. Secur.*, 2021, pp. 3–23.

[27] S. Sun et al., "Practical non-interactive searchable encryption with forward and backward privacy," in *Proc. 28th Annu. Netw. Distrib. Syst. Secur. Symp.*, 2021, pp. 1–18.

[28] C. Zuo, S. Sun, J. K. Liu, J. Shao, and J. Pieprzyk, "Dynamic searchable symmetric encryption schemes supporting range queries with forward (and backward) security," in *Proc. 23th Eur. Symp. Res. Comput. Secur.*, 2018, pp. 228–246.

[29] X. Wang, J. Ma, X. Liu, Y. Miao, and D. Zhu, "Spatial dynamic searchable encryption with forward security," in *Proc. Int. Conf. Database Syst. Adv. Appl.*, 2020, pp. 746–762.

[30] C. Zuo, S.-F. Sun, J. K. Liu, J. Shao, J. Pieprzyk, and L. Xu, "Forward and backward private DSSE for range queries," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 328–338, Jan./Feb. 2022.

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

[32] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proc. 41st Annu. ACM Symp. Theory Comput.*, 2009, pp. 169–178.

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

[34] B. Wang, M. Li, H. Wang, and H. Li, "Circular range search on encrypted spatial data," in *Proc. IEEE Conf. Commun. Netw. Secur.*, 2015, pp. 794–795.

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

[36] E. Shen, E. Shi, and B. Waters, "Predicate privacy in encryption systems," in *Proc. 6th Theory Cryptogr. Conf.*, 2009, pp. 325–341.

[37] R. Guo et al., "LuxGeo: Efficient and secure enhanced geometric range queries," *IEEE Trans. Knowl. Data Eng.*, to be published, doi: 10.1109/TKDE.2021.3093909.

[38] G. Kellaris, G. Kollios, K. Nissim, and A. O'neill, "Generic attacks on secure outsourced databases," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2016, pp. 1329–1340.

[39] M.-S. Lacharité, B. Minaud, and K. G. Paterson, "Improved reconstruction attacks on encrypted data using range query leakage," in *Proc. IEEE Symp. Secur. Privacy*, 2018, pp. 297–314.

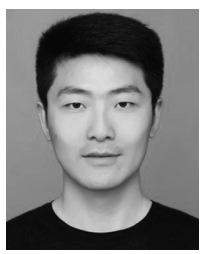
[40] P. Grubbs, M.-S. Lacharité, B. Minaud, and K. G. Paterson, "Learning to reconstruct: Statistical learning theory and encrypted database attacks," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 1067–1083.

[41] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia, "Data recovery on encrypted databases with k-nearest neighbor query leakage," in *Proc. IEEE Symp. Secur. Privacy*, 2019, pp. 1033–1050.

[42] H. Sagan, *Space-Filling Curves*, Berlin, Germany: Springer Science & Business Media, 2012.

[43] A. Kiayias, S. Papadopoulos, N. Triandopoulos, and T. Zacharias, "Delegatable pseudorandom functions and applications," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2013, pp. 669–684.

[44] C. Castelluccia, A. C. Chan, E. Mykletun, and G. Tsudik, "Efficient and provably secure aggregation of encrypted data in wireless sensor networks," *ACM Trans. Sens. Netw.*, vol. 5, no. 3, pp. 20:1–20:36, 2009.



Xiangyu Wang (Member, IEEE) received the BE and PhD degrees from Xidian University, Xi'an, China, in 2017 and 2021, respectively. He is currently an assistant professor with the School of Cyber Engineering, Xidian University, Xi'an, China. His research interests cover Big Data security, cloud security, and applied cryptography.

Jianfeng Ma (Member, IEEE) received the BS degree in mathematics from Shaanxi Normal University, Xi'an, China, in 1985, and the MS and PhD degrees in computer software and telecommunication engineering from Xidian University, Xi'an, China, in 1988 and 1995, respectively. He is currently a professor with the School of Cyber Engineering, Xidian University, Xi'an, China. He is also the director of the Shaanxi Key Laboratory of Network and System Security. His current research interests include information and network security and mobile computing systems.

Ximeng Liu (Senior Member, IEEE) received the BSc degree in electronic engineering from Xidian University, Xi'an, China, in 2010, and the PhD degree in cryptography from Xidian University, China, in 2015. Now he is the full professor with the College of Computer and Data Science, Fuzhou University. He was a research fellow with the School of Information System, Singapore Management University, Singapore. He has published more than 250 papers on the topics of cloud security and Big Data security including papers in *IEEE Transactions on Computers*, *IEEE Transactions on Information Forensics and Security*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Parallel and Distributed Systems*, *IEEE Transactions on Knowledge and Data Engineering*, *IEEE IoT Journal*, and so on. He awards "Minjiang Scholars" Distinguished Professor, "Qishan Scholars" in Fuzhou University, and ACM SIGSAC China Rising Star Award (2018). His research interests include cloud security, applied cryptography and Big Data security.

Yinbin Miao (Member, IEEE) received the BE degree with the Department of Telecommunication Engineering from Jilin University, Changchun, China, in 2011, and the PhD degree with the Department of Telecommunication Engineering from Xidian University, Xi'an, China, in 2016. He is currently an associate professor with the Department of Cyber Engineering, Xidian University, Xi'an, China. His research interests include information security and applied cryptography.

Yang Liu received the BS degree in computer science and technology from Xidian University, in 2017. He is currently working toward the PhD degree with the School of Cyber Engineering, Xidian University. His research interests cover secure multi-party computation and AI security.

Robert H. Deng (Fellow, IEEE) is AXA chair professor of cybersecurity and director of the Secure Mobile Centre, School of Information Systems, Singapore Management University (SMU). His research interests are in the areas of data security and privacy, cloud security and Internet of Things security. He received the Outstanding University Researcher Award from the National University of Singapore, Lee Kuan Yew Fellowship for Research Excellence from SMU, and Asia-Pacific Information Security Leadership Achievements Community Service Star from International Information Systems Security Certification Consortium. His professional contributions include an extensive list of positions in several industry and public services advisory boards, editorial boards, and conference committees. These include the editorial boards of *IEEE Security & Privacy Magazine*, *IEEE Transactions on Dependable and Secure Computing*, *IEEE Transactions on Information Forensics and Security*, *Journal of Computer Science and Technology*, and Steering Committee chair of the ACM Asia Conference on Computer and Communications Security.