Range search on encrypted spatial data with dynamic updates

Shabnam Kasra Kermanshahi [a,*], Rafael Dowsley [b], Ron Steinfeld [b], Amin Sakzad [b], Joseph Liu [b], Surya Nepal [c], Xun Yi [a] and Shangqi Lai [b]

[a] *STEM|School of Computing Technologies, RMIT University, Melbourne, Australia*
E-mails: shabnam.kasra.kermanshahi@rmit.edu.au, xun.yi@rmit.edu.au
[b] *Faculty of IT, Monash University, Melbourne, Australia*
E-mails: rafael.dowsley@monash.edu, ron.steinfeld@monash.edu, amin.sakzad@monash.edu, joseph.liu@monash.edu, shangqi.lai@monash.edu
[c] *Data 61, CSIRO, New South Wales, Australia*
E-mail: surya.nepal@data61.csiro.au

Abstract. Driven by the cloud-first initiative taken by various governments and companies, it has become a common practice to outsource spatial data to cloud servers for a wide range of applications such as location-based services and geographic information systems. Searchable encryption is a common practice for outsourcing spatial data which enables search over encrypted data by sacrificing the full security via leaking some information about the queries to the server. However, these inherent leakages could equip the server to learn beyond what is considered in the scheme, in the worst-case allowing it to reconstruct of the database. Recently, a novel form of database reconstruction attack against such kind of outsourced spatial data was introduced (Markatou and Tamassia, IACR ePrint 2020/284), which is performed using common leakages of searchable encryption schemes, i.e., access and search pattern leakages. An access pattern leakage is utilized to achieve an *order reconstruction attack*, whereas both access and search pattern leakages are exploited for the *full database reconstruction attack*. In this paper, we propose two novel schemes for outsourcing encrypted spatial data supporting dynamic range search. Our proposed schemes leverage R^+tree to partition the dataset and binary secret sharing to support secure range search. They further provide backward and content privacy and do not leak the access pattern, therefore being resilient against the above mentioned database reconstruction attacks. The evaluations and results on the real-world dataset demonstrate the practicality of our schemes, due to (a) the minimal round-trip between the client and server, and (b) the low computation and storage overhead on the client side.

Keywords: searchable encryption, range query, dynamic

1. Introduction

The information retrieval community has been studying geometric range search (GRS) for decades [1, 24] and it has a wide range of applications in geosciences, location-based services, geographical information system, geo-medical engineering, and so on. Besides its use in applications assisting in our daily life activities such as taking an Uber, finding nearby locations on Google Maps or friends on Facebook, GRS can be used in some significant emerging public health and safety applications. For instance, with the current COVID-19 outbreak, governments and researchers need to collect information

*Corresponding author. E-mail: shabnam.kasra.kermanshahi@rmit.edu.au.

0926-227X/$35.00 © 0 – IOS Press and the authors. All rights reserved

(e.g. number of the test taken, confirmed cases, death toll, etc.) in a specific geometric area. The need is the same in other emergency situations, e.g., a bushfire emergency situation.

Driven by the cloud-first policy of many companies and governments, outsourcing the spatial data to a cloud server is a common practice around the world. The cloud provides the scalable infrastructure to handle large datasets and supports on-demand access through its highly available services. Data privacy is a necessity in such scenarios. Although public cloud providers are trusted in providing their services, they cannot be fully trusted for data privacy. One obvious solution is to only store encrypted data in the cloud. However, downloading and decrypting large datasets every time a search or update operation needs to be performed is completely impractical. Hence, searchable encryption (SE) is considered as a solution to correctly perform queries (search/update) over outsourced encrypted data.

Searchable Symmetric Encryption (SSE) efficiently enables search over encrypted data at the cost of revealing some well-defined information to the server, known as the leakage. The most common SSE leakage functions are access pattern and search pattern. Access pattern leaks all file identifiers that are matching a search query. In contrast, search pattern leaks the repetition of search queries (i.e., it is possible to determine if two search tokens correspond to the same query). Exploiting SSE leakages might enable an adversary (often an honest-but-curious cloud server) to infer information about the database beyond what is considered in an SSE scheme (e.g. leakage abuse attacks [8, 26]).

Most of the existing SSE schemes that support geometric range search are designed in the static setting (i.e., updates of the database records after the setup are not possible or come at the cost of re-encryption and re-upload of the database). Although the dynamic setting provides more flexibility to the schemes and supports more real-world applications, it introduces more leakages. To capture new leakages in a dynamic setting, Bost et al. [5] introduced security notions for dynamic SSE, so called forward and backward privacy. Recently, Kasra-Kermanshahi et al. [18] showed that there might be additional leakages when dealing with geometric data that are not captured by Bost's forward and backward privacy models, and introduced a new security notion for dynamic SSE over spatial data (called content privacy) that hides the access pattern both in search and update operations.

Different cryptographic primitives have been used to support secure range search over geometric data such as order-preserving encryption (OPE), somewhat/fully homomorphic encryption, Geohash, and so on [18, 22, 30–32, 34–36]. However, due to the inherent leakages associated with geometric range search, the majority of them fail to resist the newly developed leakage abuse attacks that target SSE schemes designed for GRS [23, 27].

1.1. Our Contributions

In this paper, we propose two dynamic searchable symmetric encryption schemes to support geometric range search, Geo-DRS and Geo-DRS$^+$. The first scheme illustrates a novel approach to support geometric range search using R$^+$tree where more round trips between the client and the server are required to achieve content privacy (alternatively homomorphic encryption can be used at higher computational cost). Our Geo-DRS$^+$ scheme provides an efficient dynamic range search by leveraging R$^+$tree and secret sharing in $\mathbb{Z}_2$. Moreover, it uses two non-colluding servers to avoid multiple rounds of client-to-server interactions. Thus, it has only one round trip between the client and the servers during searches and updates, with a logarithmic number of communication rounds between the two servers. Geo-DRS$^+$ is efficient and scalable while resilient against *Full Database Reconstruction* (FDR) and *Approximate Database Reconstruction* (ADR) attacks. Our security analysis shows that Geo-DRS$^+$ is backward and content private.

It is worth noting that this paper is an extension of our work published in European Symposium on Research in Computer Security (ESORICS'21) [16]. In this version, several sections are added to facilitate the understanding of the work presented. Furthermore, we conduct experiments on a real-world dataset, demonstrating the effectiveness of Geo-DRS$^+$ in practice, and showing the significant improvement of efficiency by our design compared with state-of-the-art schemes.

1.2. Motivation and Related Works

Order Preserving Encryption (OPE) [2] is one of the most popular approaches to perform range search over encrypted data due to its efficiency. However, several studies have shown that it is possible to perform inference attacks on one-dimensional datasets using OPE leakages [11, 17, 19, 20]. The search and access pattern leakages are the most common leakages used in performing inference attacks. For example, Naveed et al. [26] used frequency analysis to perform sorting and cumulative attack. Later, Durak et al. [8] discovered two more types of attacks (*Inter-column correlation-based attacks* and *Inter+Intra-column correlation-based attack*) using OPE leakages that have not been considered by Naveed's work. Grubbs et al. [11] designed a leakage abuse attack which takes advantages of both frequency and order leakage of OPE. Grubbs's attack is faster, with a higher recovery rate in comparison with Naveed's cumulative attack. Furthermore, a passive adversary is also able to perform FDR without requiring auxiliary information, as discussed by Kellaris et al. [17].

The above discussed attacks mainly focused on one-dimensional data. Recently, Pan et al. [27] investigated data inference attacks against multi-dimensional OPE-encrypted databases. They designed a greedy and polynomial-time algorithm with approximation guarantees. The FDR attacks for geometric datasets were introduced recently by Markatou and Tamassia [23]. They utilized access pattern leakage to reconstruct the horizontal and vertical order of the points, and both access and search pattern leakages to recover the coordinates of the points.

Several studies have begun to support range search over encrypted spatial data [18, 22, 30–32, 34–36]. For example, Wang et al. [30–32] proposed several constructions for geometric range search using SSW[1] encryption [29], which is a pairing-based public-key encryption (PBKE). The main idea of these works is to enumerate all possible points and then check whether they are in the queried range. Due to the use of SSW, it is necessary to perform a pairing computation for each database point. Similarly, bilinear pairing operations are used by Zhu et al.[36] to support range search for location-based services. Both Xu et al. [34] and Zheng et al. [35] proposed an OPE-based scheme which utilizes R-tree for range search over spatial data. Luo et al. [22] used asymmetric scalar-product-preserving encryption (ASPE) [33] and a geometric transformation to achieve efficient range search. However, Li et al. [21] showed that Luo's scheme has some security flaws and cannot achieve the stated security notion. They proposed an enhanced version of Luo's scheme to overcome the security issues. However, both schemes are designed in a static setting; hence the update (insertion/deletion) of the points in the datasets is either not possible or requires re-encryption of the entire dataset. Guo et al. [12] proposed a dynamic searchable encryption scheme for geometric range search called MixGeo. They utilized Geohash and predicate symmetric searchable encryption to achieve efficient linear search and update. Although, the scheme supports update of the dataset points, there is no discussion about forward and backward privacy of the scheme as well as resilience against leakage abuse attacks.

Unlike other existing works in the area of geometric range search, Kasra-Kermanshahi et al. [18] proposed two constructions which consider forward and backward privacy. Moreover, they have defined

[1] *Shen-Shi-Waters*

Table 1
Table of Notations.

Notation	Description
$\mathcal{D}$	spatial dataset
N	number of objects in $\mathcal{D}$
ℓ	bit length of database objects (64 bits)
$[\![x]\!]$	a secret share of x over $\mathbb{Z}_2$
$\mathrm{ID}_i \in \{0,1\}^\ell$	ℓ-bit object identifier
m	maximum number of objects per leaf node
$\mathcal{E}$	encrypted dataset
$\mathcal{ST}$	search token
$\mathcal{R}$	search results

a new security notion for spatial data named content privacy. Their constructions utilize binary tree and a special type of additive symmetric homomorphic encryption (ASHE). To the best of our knowledge, only three of the state-of-the-art symmetric searchable encryption schemes that support geometric range search are presented in a dynamic setting. Only one of them, Kasra-Kermanshahi et al. [18], considered forward, backward, and content privacy. However, the constructions are not scalable as the size of the utilized binary tree grows linearly with the number of grid points in each dimension of the environment.

2. Building Blocks

2.1. Notation

Some of the notations that are used more frequently in the work are given in Table 1.

2.2. Syntax of Dynamic Symmetric Searchable Encryption

In this section, Dynamic Symmetric Searchable Encryption (DSSE) is briefly reviewed. Let $\mathrm{DB} = \{(ind_i, \mathrm{W}_i) : 1 \leqslant i \leqslant D\}$ be a database with $ind_i \in \{0,1\}^\ell, W_i \subseteq \{0,1\}^*$. Here, ind_i are document indices and W_i is a set of keywords matching document ind_i. We denote the set of keywords in DB with $W = \cup_{i=1}^{D}$ where $K = |W|$. We define $N = \sum_{i=1}^{D} |W_i|$ as the number of document/keyword pairs. We denote $\mathrm{DB}(w) = \{\mathrm{ind}_i | w \in \mathrm{W}_i\}$ as the set of documents containing keyword w. The interface between client and server involves the Setup algorithm and Search and Update protocols [5]:

- **Setup**$(DB, \lambda) \rightarrow (EDB, K, \sigma)$: The encrypted database EDB, master K, and σ as the client's state are output by this algorithm given the security parameter λ and database DB.
- **Search** $(q, \sigma, EDB) \rightarrow (ER)$: Clients and servers interact through this protocol. Given the search query q by the client, the server searches the encrypted database EDB and outputs the set of the encrypted matching results, ER.
- **Update** $(K, \sigma, op, in, EDB) \rightarrow (EDB', \sigma')$: The client inputs K, σ, and an operation op with its input $in = (ind, w)$ (an index and a set of keywords to be modified). The server inputs EDB. **Update** outputs the new version of the encrypted database and the updated client's state.

2.3. R-tree and R^+tree

R-tree was first introduced by Antonin Guttman in 1984 [13], to handle spatial data efficiently. This data structure is a height-balanced tree-structure with index records in its leaf nodes containing pointers to data objects. In this paper, we use R^+tree [28], a variation of R-tree in which overlapping rectangles in intermediate nodes are avoided. Moreover, R^+trees have better searching performance compared to R-trees [28].

We briefly review the example from [28] as shown in Figures 1, 2 and 3 to see how a R^+tree is formed (for the sake of simplicity, the values of the bounding boxes (Rect) are not mentioned in this example).

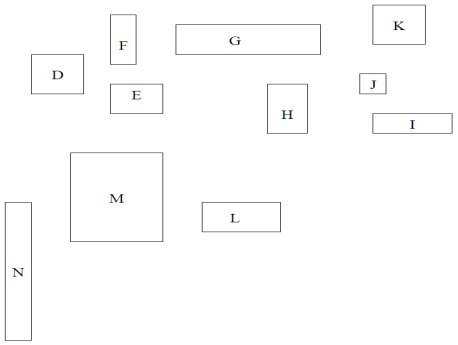

Fig. 1. The sample dataset.

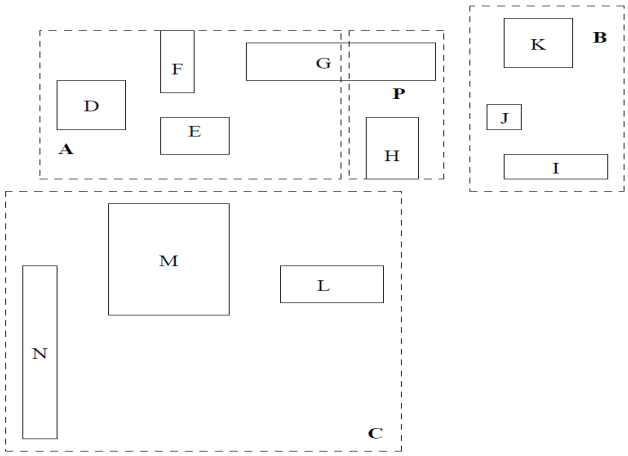

Fig. 2. The rectangles of Figure 1 grouped to form an R+tree.

In R^+trees leaf nodes consist of $(\mathrm{ID}, \mathrm{Rect})$, where ID is the object identifier and Rect represents the bounding box where the object is located. That is, $\mathrm{Rect} = (x_{min}, x_{max}, y_{min}, y_{max})$ which are the coordinates of the lower left corner and the coordinate of the upper right corner. Non-leaf nodes contain entries of the form (p, Rect), where p is the pointer to the address of the lower nodes (children nodes) and Rect covers the rectangles in the lower node's entries. A R^+tree has the following properties:

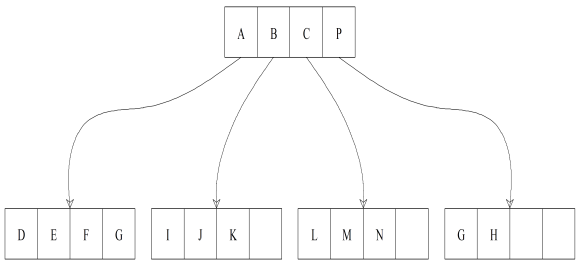

Fig. 3. The R+tree built for Figure 2.

- For each entry (p, Rect) in an intermediate node, the corresponding subtree contains a rectangle R if and only if R is covered by Rect unless R is a rectangle at a leaf node; in which case R must just overlap with Rect.
- There is no overlap in any two entries in an intermediate node.
- The root has at least two children unless it is a leaf.
- All leaves are at the same level/height.

2.4. Inverted Index

We use the inverted index to facilitate the storage and search of the dataset. For instance, to build the inverted index of the R^+tree shown in Figure 3, we first label the R^+tree nodes, where n_0 is the root and n_1 to n_4 are the leaf nodes from left to right. Then, we store the corresponding values as mentioned in Section 2.3 for each node as shown in the Table 2.

For the sake of simplicity, the bounding boxes (Rect) are not mentioned in this example.

Table 2
Inverted Index

Node Label	**Value**
n_0	$(p_{n_1}, \text{Rect}_A), (p_{n_2}, \text{Rect}_B), (p_{n_3}, \text{Rect}_C), (p_{n_4}, \text{Rect}_P)$
n_1	$(D, \text{Rect}_D), (E, \text{Rect}_E), (F, \text{Rect}_F), (G, \text{Rect}_G)$
n_2	$(I, \text{Rect}_I), (J, \text{Rect}_J), (K, \text{Rect}_K)$
n_3	$(L, \text{Rect}_L), (M, \text{Rect}_M), (N, \text{Rect}_N)$
n_4	$(G, \text{Rect}_G), (H, \text{Rect}_H)$

2.5. Secure Bitwise Comparison

This work uses secure two-party computation based on bitwise secret sharings. An additively secret sharing of $x \in \mathbb{Z}_2$ consists of two shares x_1 and x_2 chosen uniformly at random subject to the constraint that $x = x_1 + x_2 \mod 2$. The two shares are distributed to two servers, respectively. We will denote this secret sharing by $[\![x]\!]$. All secret sharing operations are modulo 2 and the modular notation is omitted for conciseness. Note that modulo 2, addition and subtraction are equivalent. Given secret sharings $[\![x]\!]$ and $[\![y]\!]$, the two servers can locally compute in a trivial way secret sharings corresponding to $z = x + y$. This

operation will be denoted by $[\![z]\!] \leftarrow [\![x]\!] + [\![y]\!]$. It is also trivial to add the constant 1 to a secret sharing, one of the servers simply adds it locally.

In this work, secure multiplications of secret shared values are performed in a standard way using pre-distributed multiplications triples [3, 7], which consist of $([\![a]\!], [\![b]\!], [\![c]\!])$ for uniformly random a and b, and $c = ab$. These triples are pre-distributed by the data owner to the two computing servers. In order to improve the communication costs, a pseudorandom function (PRF) is used to generate the triples: (1) the data owner sends a key $K1$ of the PRF to $S1$ and a key $K2$ to $S2$; (2) the data owner and $S1$ use the PRF to obtain pseudorandom values $a_1, b_1, c_1 \in \mathbb{Z}_2$, while the data owner and $S2$ use the PRF to obtain pseudorandom values $a_2, b_2 \in \mathbb{Z}_2$; (3) the data owner fixes c_2 such that $c = ab$ and transmits the share c_2 to $S2$. With this optimization the communication cost for pre-distributing each multiplication triple is reduced to a single bit.

For performing secure bitwise comparison, we use the same protocol as De Cock et al. [6], which is a variant of the protocol of Garay et al. [10] using bitwise secret sharings in the field $\mathbb{Z}_2$ (a detailed description of the underlying protocol can be found in Section 4.3.3 of De Hoogh's PhD thesis [14]). For ℓ-bits values x and y, the two servers have as inputs secret shares $[\![x_i]\!]$ and $[\![y_i]\!]$ for $i \in \{0, \ldots, \ell-1\}$, where $x = \Sigma_{i=0}^{\ell-1} x_i 2^i$ and $y = \Sigma_{i=0}^{\ell-1} y_i 2^i$. The protocol **GEQ** outputs a secret shared value $[\![z]\!]$, where z is equal to 1 if $x \geqslant y$; and equal to 0 otherwise. The protocol, which uses the divide and conquer paradigm, is presented in Algorithm 1. It outputs the inverse of the output of the protocol **LT** that outputs a secret shared value $[\![z]\!]$, where z is equal to 1 if $x < y$; and equal to 0 otherwise. **LT** uses as a subprotocol **LTEQ**, which outputs $([\![z]\!], [\![w]\!])$ such that z is equal to 1 if and only if $x < y$ and w is equal to 1 if and only if $x = y$. The protocol **GEQ** has $\log \ell + 1$ rounds and needs to perform $3\ell - \log \ell - 2$ secure multiplications of values that are secret shared in $\mathbb{Z}_2$.

3. Definitions, Security Notions and Model

3.1. Syntax of Our Geometric Dynamic Range Search (Geo-DRS$^+$)

Our geometric dynamic range search (Geo-DRS$^+$) scheme consists of the following algorithms:

(1) **Setup**(DB): The first step is to generate the shares of the database records to be outsourced to the servers. This phase is run by the data owner as follows:

- **Build**.R$^+$tree$(\mathrm{DB}, m) \rightarrow (RT)$: Given a database DB and the tree parameter m (which determines the maximum number of the points in each node), this algorithm outputs a height-balanced R$^+$tree.
- **SecretShare**$(RT) \rightarrow (\mathbb{S}1, \mathbb{S}2)$: This algorithm gets the R$^+$tree as input and outputs its bitwise secret shares.

The Setup phase also generates the multiplications triples (that will be needed for the executions of Protocol **GEQ**) and the database state δ. $\mathbb{S}1$ is given to the first server and $\mathbb{S}2$ to the second server.

(2) **Search**$(\mathrm{Rect}_q/\mathbb{S}1/\mathbb{S}2)$ is a protocol between a client and the servers. To find the desirable range query Rect_q, the client secret shares the query coordinates with the servers whom run the **GEQ** protocol over their stored shares $\mathbb{S}1/\mathbb{S}2$ traversing the R$^+$tree jointly to find the minimum bounding boxes (leaf nodes) that cover the query. The servers output the shares of the result set, $\mathcal{R}1$ and $\mathcal{R}2$.

(3) **Update**$(n_i, \delta, \mathbb{S}1/\mathbb{S}2)$ is a protocol between the data owner and the servers. To insert or delete an object, the data owner should generate the new shares of the corresponding leaf node. Upon

Algorithm 1 Comparison Protocols

GEQ$(\ell, [\![x_\ell]\!], \ldots, [\![x_1]\!], [\![y_\ell]\!], \ldots, [\![y_1]\!])$
Input: ℓ, $[\![x_\ell]\!], \ldots, [\![x_1]\!], [\![y_\ell]\!], \ldots, [\![y_1]\!]$
Output: $[\![z]\!]$

return $[\![z]\!] \leftarrow 1 + \mathbf{LT}(\ell, [\![x_\ell]\!], \ldots, [\![x_1]\!], [\![y_\ell]\!], \ldots, [\![y_1]\!])$ // z is equal to 1 if $x \geqslant y$; and equal to 0 otherwise

LT$(k, [\![x_k]\!], \ldots, [\![x_1]\!], [\![y_k]\!], \ldots, [\![y_1]\!])$
Input: k, $[\![x_k]\!], \ldots, [\![x_1]\!], [\![y_k]\!], \ldots, [\![y_1]\!]$
Output: $[\![z]\!]$

if $k = 1$ **then**
  return $[\![z]\!] \leftarrow [\![y_1]\!] + [\![y_1]\!][\![x_1]\!]$
else
  $k' \leftarrow \lfloor k/2 \rfloor$
  $([\![a]\!], [\![b]\!]) \leftarrow \mathbf{LTEQ}(k - k', [\![x_k]\!], \ldots, [\![x_{k'+1}]\!], [\![y_k]\!], \ldots, [\![y_{k'+1}]\!])$
  $[\![c]\!] \leftarrow \mathbf{LT}(k', [\![x_{k'}]\!], \ldots, [\![x_1]\!], [\![y_{k'}]\!], \ldots, [\![y_1]\!])$
  return $[\![z]\!] \leftarrow [\![a]\!] + [\![b]\!][\![c]\!]$ // z is equal to 1 if $x < y$; and equal to 0 otherwise
end if

LTEQ$(k, [\![x_k]\!], \ldots, [\![x_1]\!], [\![y_k]\!], \ldots, [\![y_1]\!])$
Input: k, $[\![x_k]\!], \ldots, [\![x_1]\!], [\![y_k]\!], \ldots, [\![y_1]\!]$
Output: $[\![z]\!], [\![w]\!]$

if $k = 1$ **then**
  $[\![z]\!] \leftarrow [\![y_1]\!] + [\![y_1]\!][\![x_1]\!]$
  $[\![w]\!] \leftarrow 1 + [\![x_1]\!] + [\![y_1]\!]$
  return $([\![z]\!], [\![w]\!])$
else
  $k' \leftarrow \lfloor k/2 \rfloor$
  $([\![a]\!], [\![b]\!]) \leftarrow \mathbf{LTEQ}(k - k', [\![x_k]\!], \ldots, [\![x_{k'+1}]\!], [\![y_k]\!], \ldots, [\![y_{k'+1}]\!])$
  $([\![c]\!], [\![d]\!]) \leftarrow \mathbf{LTEQ}(k', [\![x_{k'}]\!], \ldots, [\![x_1]\!], [\![y_{k'}]\!], \ldots, [\![y_1]\!])$
  $[\![z]\!] \leftarrow [\![a]\!] + [\![b]\!][\![c]\!]$
  $[\![w]\!] \leftarrow [\![b]\!][\![d]\!]$
  return $([\![z]\!], [\![w]\!])$ // z is equal to 1 if and only if $x < y$ and w is equal to 1 if and only if $x = y$.
end if

receiving the shares, the servers update their stored shares $\mathbb{S}1/\mathbb{S}2$ by replacing them with the new shares. At the end, the servers update the dataset state to $\delta + 1$.

Remark. Note that, in our model we assume that the data owner sets m large enough according to the size of the environment such that the insertion of new objects would not require node splitting (see [28] for more details). Thus, to add/delete an object only the corresponding leaf nodes would be updated. Moreover, even if the number of objects in a leaf node become larger than m, the data owner can proceed with splitting the corresponding leaf node and updating the encrypted records accordingly.

3.2. Generic Dynamic SSE Leakage Functions

The leakage function $\mathcal{L}$ keeps as state the query list $\mathcal{Q}$, i.e., the list of all queries issued so far. The entries are (t, w) for a search query on keyword w, or $(\mathsf{t}, op, (w, ind))$ for an update query, where t is the timestamp, w is the search keyword, $op \in \{Add, Del\}$ denoting the operation, and ind is a list of file identifiers to be updated. According to Bost [5] the general leakage functions associated with dynamic SSE schemes are the following:

- $\mathrm{sp}(w) = \{\mathsf{t} : (\mathsf{t}, w) \in \mathcal{Q}\}$ is the search pattern which leaks if two search queries correspond to the same keyword w.
- $\mathrm{UpHist}(w)$ is a history which outputs the list of all updates on keyword w. Each element of this list is an update query tuple $q_u = (\mathsf{t}, op, (w, ind))$.
- $\mathrm{TimeDB}(w)$ is the list of all documents matching w, excluding the deleted ones, together with the timestamp of when they were inserted in the database.
- $\mathrm{Updates}(w)$ is the list of timestamps of updates on w.
- $\mathrm{DelHist}(w)$ is the deletion history of w, which is the list of timestamps for all deletion operations together with the timestamp of the inserted entry it removed.

3.3. Range Search Leakage Functions

We denote the leakage function of our Geo-DRS$^+$ scheme by $\mathcal{L}$. That is, the information which each server is allowed to learn about the dataset and the queries. This leakage function corresponds to the Setup, Search and Update of Geo-DRS$^+$; $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$.

- *Search pattern* (s): Similar to most of the existing searchable encryption schemes, our scheme leaks some information about whether any two queries are generated from the same range search or not. In our design, the servers learn if the minimum bounding boxes match the minimum bounding boxes of previous searches. That is, every time a search happens, the client will generate new random secret shares of the coordinates. So, even if the same search query happens twice or more, the random secret shares of the coordinates will be different from the point of view of any single server, and he cannot link the search queries using this part of the protocol. But the servers learns the resulting minimum bounding boxes and can compare with the respective boxes of previous search queries.
- *Number of updates* (Nu): The server learns how many updates are performed on the dataset but he cannot recognize the type of the update (insertion, deletion, modification) and also on which point the update is performed. Therefore, the update does not leak any information about the dataset and the search queries.
- *Range search size* (rs): the server learns which minimum bounding boxes cover the range for each search query.
- *Range update size* (ru): the server learns which minimum bounding boxes cover the range for each update query.
- *R^+tree structure (R^+)*: The structure of R$^+$tree is leaked to the servers.

Therefore, Geo-DRS$^+$ leakage consists of $\mathcal{L}^{Stp}(\mathcal{D}) = R^+$, $\mathcal{L}^{Srch}(r) = (\mathsf{s}, \mathsf{rs})$, and $\mathcal{L}^{Updt}(op, \mathrm{ID}_i) = (\mathsf{ru}, \mathsf{Nu})$.

3.4. Security Notions and Definitions

Kasra-Kermanshahi et al. [18] introduced a new security notion for spatial data called content privacy. They formulated a leakage that was not captured in previous definitions such as forward/backward privacy [4, 5]. In short, there should be no leakage on updated points neither in the search phase nor during the update. Content privacy and backward privacy (Type-II) have some common properties: both protect the content and do not leak anything about the documents' identifiers in the update queries. However, backward privacy (Type-II) leaks information about the content in the search queries via the access pattern.

Backward privacy (Type-II) reveals all of the information contained in Backward privacy (Type-I) [2] and also reveals when all updates over the search keyword happened without their content.

Definition 3.1 (Backward Security with Update Pattern)**.** *A $\mathcal{L}$-adaptively-secure SSE scheme is update pattern revealing backward-secure if, and only if, the search and update leakage functions $\mathcal{L}^{Srch}$, $\mathcal{L}^{Updt}$ can be written as: $\mathcal{L}^{Updt}(op, w, ind) = \mathcal{L}'(op, w)$ and $\mathcal{L}^{Srch}(w) = \mathcal{L}''(TimeDB(w), Updates(w), sp(w))$, where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless.*

Definition 3.2 (Content Privacy for Spatial Dataset)**.** *A $\mathcal{L}$-adaptively-secure SSE scheme is content-private if, and only if, the search and update leakage functions $\mathcal{L}^{Srch}$, $\mathcal{L}^{Updt}$ can be written as: $\mathcal{L}^{Updt}(op, r, P) = \mathcal{L}'(op, r)$ and $\mathcal{L}^{Srch}(r) = \mathcal{L}''(r)$ where $\mathcal{L}'$ and $\mathcal{L}''$ are stateless. Here, r represents a range of coordinates and a point identifier is denoted by P.*

3.5. Security Model

The security model of the proposed constructions is formulated using two games; $\texttt{REAL}^{\Sigma}_{\mathcal{A}}(\lambda)$ and $\texttt{IDEAL}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$, for a security parameter λ. The former is executed using our Geo-DRS$^+$ scheme (denoted by Σ), whereas the latter is simulated using the leakage of our scheme as defined in Section 3.3. The leakage is parameterised by a function $\mathcal{L} = (\mathcal{L}^{Stp}, \mathcal{L}^{Srch}, \mathcal{L}^{Updt})$, which describes what information is leaked to the adversary $\mathcal{A}$. If the adversary $\mathcal{A}$ cannot distinguish these two games, then we can say that there is no leakage beyond what is defined in the leakage function. These games can be formally defined as followed;

- $\texttt{REAL}^{\Sigma}_{\mathcal{A}}(\lambda)$: On input a dataset chosen by the adversary $\mathcal{A}$, it outputs the shares of the R$^+$tree nodes by using **Setup**(DB) to $\mathcal{A}$. The adversary can repeatedly perform search and update queries. The game outputs the results generated by running **Search**(Rect$_q$/$\mathbb{S}1$/$\mathbb{S}2$) and **Update**($n_i, \delta, \mathbb{S}1/\mathbb{S}2$) to $\mathcal{A}$. Eventually, $\mathcal{A}$ outputs a bit.
- $\texttt{IDEAL}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)$: On input a database chosen by $\mathcal{A}$, it outputs the shares of R$^+$tree nodes to the adversary $\mathcal{A}$ by using a simulator $\mathcal{S}(\mathcal{L}^{Stp})$. Then, it simulates the results for search queries using the leakage function $\mathcal{S}(\mathcal{L}^{Srch})$ and uses $\mathcal{S}(\mathcal{L}^{Updt})$ to simulate the results for update queries. Eventually, $\mathcal{A}$ outputs a bit.

Definition 3.3. *The scheme Σ is $\mathcal{L}$-adaptively-secure if for every PPT adversary $\mathcal{A}$, there exists an efficient simulator $\mathcal{S}$ such that* $|\Pr[\texttt{REAL}^{\Sigma}_{\mathcal{A}}(\lambda) = 1] - \Pr[\texttt{IDEAL}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda) = 1]| \leqslant negl(\lambda)$.

4. Dynamic Secure Range Search on Encrypted Spatial Data

This section first presents the Geo-DRS scheme to address the challenge of secure range search on spatial data in a dynamic manner. Figure 4 demonstrates the overview of Geo-DRS scheme. This base scheme imposes a logarithmic number of communication rounds between the client and the server to perform the search. One possible solution to avoid this communication overhead is to store the R$^+$tree structure from root to the leaf nodes on the client side and put the rest on the server. However, this is not desirable as it contradicts the main goal of outsourcing the data and also is not appropriate for resource

[2] the document identifiers matching the issued search keyword when they were inserted, and the total number a_w of updates over the search keyword

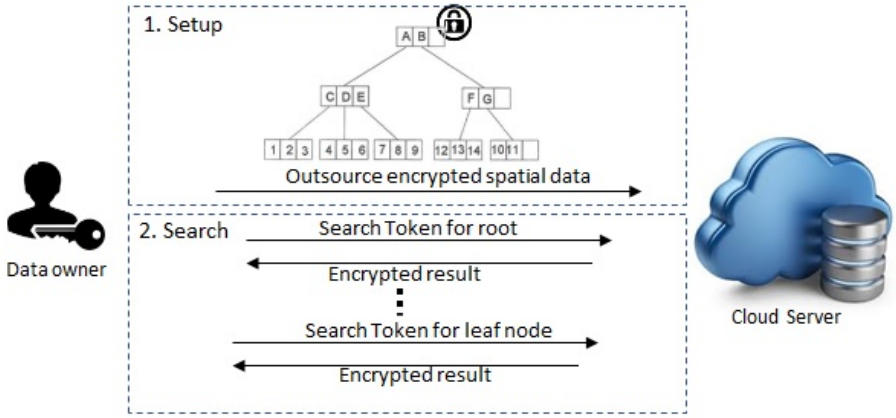

Fig. 4. The system model of Geo-DRS scheme

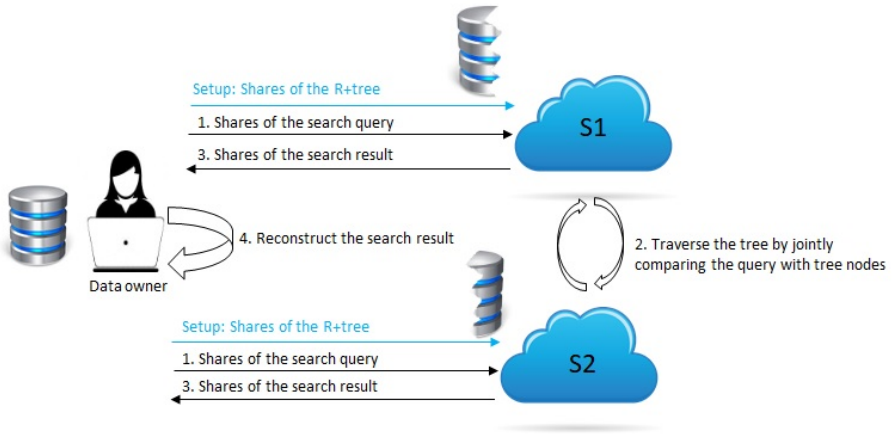

Fig. 5. The system model of Geo-DRS$^+$ scheme

constrained devices. Therefore, we design Geo-DRS$^+$, an enhanced version of the Geo-DRS scheme in which the single-server model of Geo-DRS is replaced with a two non-colluding server model, see Figure 5. This enables us to shift the communication between the client and a server to the communication between the two non-colluding servers. To enable the servers to perform secure computation over the outsourced data and achieve backward and content privacy, we utilize binary secret sharing in Geo-DRS$^+$.

4.1. Geo-DRS Scheme

To explain the ideas underlying our main construction (Geo-DRS+), we first describe the details of the Geo-DRS scheme in Algorithm 2. This scheme consists of three main algorithms: SETUP, SEARCH, UPDATE.

- SETUP: The data owner proceeds as follows:
  - ∗ On input the dataset $\mathcal{D}$, security parameter λ and the tree parameter m, she partitions the environment and builds a height-balanced R$^+$tree.

* Encrypt each of the tree nodes and outsource it to the server.

- SEARCH: The protocol is executed between the client and the server as follows:
  * Client: Given the desired range query $\text{Rect}_q = ([x_{LL}(q), x_{UR}(q)], [y_{LL}(q), y_{UR}(q)])$, the client generates the search token $\mathcal{ST}$ for the tree root and sends it to the server. Upon receiving the corresponding result $\mathcal{R}$ from the server, he decrypts it to find the next node in the R^+tree and continues this procedure to reach the desirable object.
  * Server: Given the encrypted dataset $\mathcal{E}$ and the search token $\mathcal{ST}$, it outputs $\mathcal{R}$ which contains the ciphertext of the nodes corresponding to the issued search token.
- UPDATE[3]: The data owner and the server perform the following protocol:
  * Data Owner: Given the update query $\mathcal{Q}_u = \text{ID}_i$, whether it is an insertion or a deletion, they first perform the SEARCH protocol so that the data owner finds the corresponding leaf node, n_i. Then, the data owner re-encrypts n_i and sends the re-encryption to the server.
  * Server: The server replaces the corresponding entry for n_i with the given value from the data owner and updates the encrypted dataset $\mathcal{E}$ state.

4.2. *Geo-DRS$^+$: Optimised Geometric Dynamic Range Search*

In our model, we use a R^+tree to categorise the data before creating the inverted index. We applied the technique of De Cock et al. [6] with the secret sharing of [10] in the field $\mathbb{Z}_2$ to perform the secure search. The protocols for the setup, search and update work as follows (Figure 6 illustrates the details of Geo-DRS$^+$ scheme):

- **Setup($\mathcal{D}$):** This algorithm is performed by the data owner that inputs the the spatial dataset $\mathcal{D}$. He first partitions the environment to build the R^+tree. Then he creates bitwise secret sharings of the inverted index based on each node in the tree, and sends the sets of shares $\mathbb{S}1$ and $\mathbb{S}2$ to S1 and S2, respectively. He also pre-distributes to the servers the multiplications triples that will be needed for the executions of the **GEQ** protocol.[4]
- **Search($\textbf{Rect}_q/\mathbb{S}1/\mathbb{S}2$):** This protocol is executed by the client and the servers. On an input query $\text{Rect}_q = ([x_{LL}(q), x_{UR}(q)], [y_{LL}(q), y_{UR}(q)])$, the client generates bitwise secret sharings of those coordinates and send the set of shares $\mathcal{ST}1$ and $\mathcal{ST}2$ to the corresponding servers. Given the shares of the search token and of the inverted index, the servers S1 and S2 jointly perform the search and return shares of the results, $(\mathcal{R}1, \mathcal{R}2)$, to the client. Finally, the client reconstructs the results, $\mathcal{R}$.
- **Update($n_i, \delta, \mathbb{S}1/\mathbb{S}2$):** This protocol is executed between the data owner and the servers. To update (i.e., insertion/deletion) an object in the outsourced dataset, the data owner should update the corresponding leaf node. That is, it first updates the object and then generates the new shares of that leaf node. As the entire entry for the leaf node is getting updated the servers would not learn which particular object is being updated. To update the leaf node n_i, the data owner generates the corresponding shares $\mathcal{U}1$ and $\mathcal{U}2$ for the servers. Given such shares, the servers update their shares by replacing them with the new shares. Finally, the servers update the dataset state to $\delta + 1$.

[3] It is also possible to use additive homomorphic encryption to perform the update at the server side (e.g. update in [18]), here we want to show only a basic scenario.

[4] The data owner can initially distribute some reasonable number of multiplication triples, and once the servers are about to run out of triples, they can request more triples to the data owner.

Algorithm 2 Geo-DRS Construction

Setup$(\lambda, \mathcal{D}, m)$

Input: $\lambda, \mathcal{D}, m$

Output: $\mathcal{E}, K$

The data owner partitions $\mathcal{D}$
$RT \leftarrow R^{+}tree(\mathcal{D}, m)$ // generates a R^{+}tree where each node has m entries, filling up the empty spaces with dummy values
Append each partition of size m of $\mathrm{ID}_i \in \mathcal{D}$ to the corresponding leaf node
Initialize $UT \leftarrow \emptyset$ indexed by nodes' Tag
$K_s, K_t \xleftarrow{\$} \{0,1\}^{\lambda}$, $\mathcal{E} \leftarrow \{\}$, $\delta \leftarrow 1$ // state of $\mathcal{E}$
for all $n_i \in RT$ **do**
    $K_i \leftarrow F(K_s, n_i)$ // F is a pseudo-random function (PRF)
    $Tag_i \leftarrow F(K_t, n_i)$
    for $j = 1, ..., m$ **do**
        for $\mathsf{R_j}(n_i) \in RT$ **do** // $\mathsf{R_j}(n_i)$ is the records associated with a node
            $e_i \leftarrow Enc_{K_i}(\mathsf{R_j}(n_i))$
            $UT(Tag_i) \leftarrow e_i$
        end for
    end for
end for
Append UT to $\mathcal{E}$
return $\mathcal{E}, K_s, K_t$ // The data owner stores $K = (K_s, K_t)$ and the identifier of the root of RT which is n_0, and sends $\mathcal{E}$ to the server

Search$(\mathbf{Rect}_q, \mathcal{E})$

Input: $\mathrm{Rect}_q = ([x_{LL}(q), x_{UR}(q)], [y_{LL}(q), y_{UR}(q)]), n_0, \mathcal{E}$

Output: $\mathcal{R}$

The client starts from the root n_0 and performs **Sch**(n_0)
Returns $\mathcal{R}$ as the list of objects which overlap with the queried rectangle

Sch(n_i)

The client computes $Tag_i \leftarrow F(K_t, n_i)$
$K_i \leftarrow F(K_s, n_i)$
Send Tag_i to the server
The server computes $e_i \leftarrow UT(Tag_i)$ and send e_i to the client
if n_i is a leaf node **then**
    $\mathcal{R} = \{R_j(n_i) = (\mathrm{ID}_o, rect_o)\}_{j=1,...,m}^{o \in \{1,...,N\}} \leftarrow Dec_{K_i}(e_i)$
    return $\mathcal{R}$
else
    $\mathcal{R} = \{R_j(n_i) = (n_c, rect_c)\}_{j=1,...,m}^{c \in \{1,...,n\}} \leftarrow Dec_{K_i}(e_i)$
    for $j = 1, ..., m$ **do**
        // Check if Rect_q collides with $rect_{n_c}$
        if $((y_{LL}(n_i) \leqslant y_{UR}(q) \leqslant y_{UR}(n_i))$ OR $(y_{LL}(n_i) \leqslant y_{LL}(q) \leqslant y_{UR}(n_i)))$ AND $((x_{LL}(n_i) \leqslant x_{LL}(q) \leqslant x_{UR}(n_i))$ OR $(x_{LL}(n_i) \leqslant x_{UR}(q) \leqslant x_{UR}(n_i)))$ **then**
            Sch(n_c)
        end if
    end for
    return $\mathcal{R}$
end if

Update(ID$_i; \mathcal{E}, \delta$**)**

Input: $\mathrm{ID}_i; \mathcal{E}, \delta$

Output: $\mathcal{E}, \delta$

if $op = Add$ **then**
    The data owner append ID_i to $R_j(n_i)$
else
    The data owner replaces ID_i with dummy value in $R_j(n_i)$
end if
$Tag_i \leftarrow F(K_t, n_i)$
$e_i' \leftarrow Enc_{K_i}(\mathsf{R_j}(n_i))$ and send (Tag_i, e_i') to the server
The server finds $e_i \leftarrow UT(Tag_i)$ and replace e_i with e_i' in $\mathcal{E}$
$\delta \leftarrow \delta + 1$
return $\mathcal{E}, \delta$

Setup($\mathcal{D}$)

1. The data owner generates the R$^+$tree, $RT \leftarrow R^+tree(\mathcal{D}, m)$
2. For each $n_i \in RT$ and for $j \in \{0, ..., \ell - 1\}$ (where ℓ is the bit-length used to represent the coordinates), the data owner secret shares between S1 and S2 the values ($[\![x^j_{LL}(n_i)]\!]$, $[\![y^j_{LL}(n_i)]\!]$, $[\![x^j_{UR}(n_i)]\!]$, $[\![y^j_{UR}(n_i)]\!]$). Let $\mathbb{S}1$ denote the shares of S1, and $\mathbb{S}2$ those of S2.
3. The data owner sets the state $\delta = 0$
4. The data owner generates the multiplication triples that are necessary to execute the **GEQ** protocol and sends the respective shares of the triples, as well as δ, to S1 and S2.

Search(**Rect**$_q$; /$\mathbb{S}1$/$\mathbb{S}2$)

1. The client $\mathcal{C}$ secret shares the query coordinates ($[\![x^j_{LL}(q)]\!]$, $[\![y^j_{LL}(q)]\!]$, $[\![x^j_{UR}(q)]\!]$, $[\![y^j_{UR}(q)]\!]$) between the servers S1 and S2.
2. S1 and S2 run the protocol **GEQ** over the shared query coordinates and R$^+$tree nodes starting from the root to determine which children need to be searched.
If the following condition hold (the output of comparison is "1") then S1 and S2 traverse the R$^+$tree via that node until they reach the leaf node
$((y_{LL}(n_i) \leqslant y_{UR}(q) \leqslant y_{UR}(n_i))$ OR $(y_{LL}(n_i) \leqslant y_{LL}(q) \leqslant y_{UR}(n_i)))$ AND $((x_{LL}(n_i) \leqslant x_{LL}(q) \leqslant x_{UR}(n_i))$ OR $(x_{LL}(n_i) \leqslant x_{UR}(q) \leqslant x_{UR}(n_i)))$.
3. Once S1 and S2 reach the leaf nodes they output their shares of the result set, $\mathcal{R}1$ and $\mathcal{R}2$.
4. The client $\mathcal{C}$ reconstructs the results $\mathcal{R}$ using the given shares $\mathcal{R}1$ and $\mathcal{R}2$.

Update($n_i, \delta, \mathbb{S}1/\mathbb{S}2$)

1. The data owner sends the new shares of the entries to be updated, $\mathcal{U}1$ and $\mathcal{U}2$, to the servers S1 and S2, respectively.
2. S1 and S2 replace their old shares with the new ones and update the state $\delta \leftarrow \delta + 1$.

Fig. 6. Geo-DRS$^+$ scheme

5. Security analysis

In our construction, each search result is a share of a list associated with a leaf node and client is the one who reconstructs the final result using these shares. To insert or delete an object within a list, the client generates the new shares of the list and the servers will replace the old shares with the new ones. Thus, 1) there is no leakage regarding the content of the dataset (object's identifier), 2) it is impossible to distinguish which object was being updated, 3) the search queries do not leak matching objects after they have been deleted. As a result, our construction is content and backward private as proved below.

Theorem 5.1. *Let $\mathcal{L}$ denote the leakage function of our Geo-DRS$^+$ scheme as defined in section 3.3. Our constructed Geo-DRS$^+$ is $\mathcal{L}$-adaptively-secure, if the protocol of De Cock et al.(we call it π_s) [6] is secure. Let Σ represents Geo-DRS$^+$, and $\mathcal{A}$ be the adversary (the honest-but-curious server)[5], who breaks the security of Σ. Suppose $\mathcal{A}$ make at most $q_u > 0$ update queries. One can construct an algorithm $\mathcal{B}$ that can break the UC-security of De Cock et al. [6] protocol by running $\mathcal{A}$ as a subroutine with non-negligible probability if $\log_2 q_s + \ell \geqslant \lambda$, for security parameter λ.*

[5] who follows the protocol instructions correctly, but try to learn additional information

Proof

The proof proceeds using a hybrid argument, by game hopping, starting from the real-world game $\texttt{REAL}^{\Sigma}_{\mathcal{A}}(\lambda)$.

- **Game G_0:** This game is exactly the same as the real world security game $\texttt{REAL}^{\Sigma}_{\mathcal{A}}(\lambda)$. Hence, we have

$$\mathbb{P}\left[\texttt{REAL}^{\Sigma}_{\mathcal{A}}(\lambda)=1\right]=\mathbb{P}\left[G_0=1\right].$$

- **Game G_1:** In this game, we pick random values instead of the output of π_s as a share of a search query and store it in a table to be reused if same query is issued. The advantage of the adversary in distinguishing between G_0 and G_1 is exactly the same as advantage for π_s. Thus, we can build a reduction $\mathcal{B}$ which is able to distinguish between π_s and a truly random function.

$$|\mathbb{P}\left[G_0=1\right]-\mathbb{P}\left[G_1=1\right]|\leqslant \mathrm{Adv}^{\pi_s}_{\mathcal{S}_{\pi_s},\mathcal{B}}(\lambda).$$

- **Game G_2:** To update (delete/insert) an object from the list associated to a leaf node on the R$^+$tree, this game replaces the shares of the leaf node with random shares. For update token, it uses the leakage to learn which node should be updated. The adversary $\mathcal{A}$ cannot distinguish the real shares from the truly random shares. Suppose $\mathcal{A}$ makes at most $q_u > 0$ update queries, then we have

$$|\mathbb{P}\left[G_2=1\right]-\mathbb{P}\left[G_1=1\right]|\leqslant \frac{1}{q_u\cdot 2^{\ell}}.$$

- **Simulator.** We can simulate the $\texttt{IDEAL}$ game like Game G_2. Let $\mathcal{S}_{\pi_s}$ be the simulator for De Cock et al. [6] protocol; then we construct a simulator $\mathcal{S}$ for our construction to perform the search. The algorithm $\mathcal{B}$ uses $\mathcal{S}_{\pi_s}$ to construct the simulator $\mathcal{S}$ in order to answer the queries issued by $\mathcal{A}$. We just need to use $\mathcal{S}_{\pi_s}$ for $\mathcal{A}_{\pi_s}$, to construct $\mathcal{S}$ for $\mathcal{A}$. We have that

$$|\mathbb{P}\left[\texttt{REAL}^{\Sigma}_{\mathcal{A}}(\lambda)=1\right]-\mathbb{P}\left[\texttt{IDEAL}^{\Sigma}_{\mathcal{A},\mathcal{S}}(\lambda)=1\right]|\leqslant \mathrm{Adv}^{\pi_s}_{\mathcal{S}_{\pi_s},\mathcal{B}}(\lambda)+\frac{1}{q_u\cdot 2^{\ell}}.$$

  For the update, simulator $\mathcal{S}$ works the same as G_1 without knowing the content (objects' identifiers). The simulator only uses ru to identify the bounding box of the update query and not the object's identifier. Therefore, it can simulate the attacker's view using only $\mathcal{L}^{Updt}$.

As a result, our construction satisfies content and backward privacy as the search leakage does not include TimeDB(w) or Updates(w). □

6. Performance Evaluation

We consider that the dataset objects are represented in a metre scale where coordinate values are 64 bits ($\ell = 64$). To compare the queried coordinate value with the bounding box coordinates in each level of the R$^+$tree, we require a Boolean circuit of depth $\log \ell + 1$ for ℓ-bit integers. Note that, this logarithmic-round protocol for secure integer comparison is performed between the two non-colluding servers during the search, hence no overhead to the client. For each comparison $3\ell - \log \ell - 2$ bit

Table 3a
Comparison

Scheme	Guo2019	Li 2019	Zheng 2020	Kasra-I 2020	Kasra-II 2020	Geo-DRS$^+$
Search Complexity (Server)	$O(N)$	$O(n\eta \log N)$	$O(m \log mN)$	$O(\log(2R)N)$	$O(\log(2R)N)$	$O(\ell m \log m)$
Search Complexity (Client)	$O(\theta)$	$O((n+d)\eta^2)$	$O(1)$	$O((\log R)N)$	$O((\log R)N)$	$O(1)$
Update Complexity (Server)	$O(N)$	NA	NA	$O(1)$	$O(2^t N)$	$O(1)$
Update Complexity (Client)	$O(1)$	NA	NA	$O(ktN)$	$O(1)$	$O(1)$
#client-server roundtrips (Search)	2	1	1	1	1	1
#client-server roundtrips (Update)	2	NA	NA	$O(\log R)$	1	1
Dynamic	✓	✗	✗	✓	✓	✓
Avoid Search pattern leakage	✗	✗	✗	✗	✗	✗
Avoid Access pattern leakage	✗	✗	✗	✓	✓	✓
Content privacy	✗	NA	NA	✓	✓	✓
Cryptographic primitive	Geohash and PBKE	ASPE	OPE	SE	ASHE	SS

SE: Symmetric Encryption; ASHE: Additive Symmetric Homomorphic Encryption; PBPKE: Pairing-based Public Key Encryption; OPE: Order Preserving Encryption; Geohash:public domain geocoding system [25]; ASPE: Asymmetric Scalar-product-Preserving Encryption; SS: Secret Sharing R: Radius of the circle query; t: Bit length of coordinates (x and y); N: Number of the data points in the dataset; N_{deg}: highest degree of a term in the used fitted polynomial θ: size of Bloom filter; n: number of the matching result; k: number of update point; T_{exp} exponentiation time in token generation of SSW; η: Plain-text vector size; d: number of dimensions; ℓ: Bit length of database objects (64 bits)

Table 3b
Comparison

Scheme	Zhu 2015	Wang 2015	Wang 2016	Luo 2017	Wang 2017	Xu 2019
Search Complexity (Server)	$O(RNT_pT_{mul})$	$O(R^2N)$	$O(\theta N)$	$O(N\delta d')$	$O(2^t)$	$O(Nt^2N_{deg}^3)$
Search Complexity (Client)	$O(1)$	$O(RT_{exp})$	$O(2^{2t}T_{exp})$	$O(\delta d')$	$O(R^2 2^t T_{exp})$	$O(N_{deg}^4 t^2)$
Update Complexity (Server)	NA	NA	NA	NA	$O(2^t N)$	$O(1)$
Update Complexity (Client)	NA	NA	NA	NA	$O(1)$	$O(kt)$
# client-server roundtrips (Search)	3	1	2	δ	δ	1
# client-server roundtrips (Update)	NA	NA	NA	NA	NA	1
Dynamic	✗	✗	✗	✗	✗	✓
Avoid Search pattern leakage	✗	✗	✗	✗	✗	✗
Avoid Access pattern leakage	✗	✗	✗	✗	✗	✗
Content privacy	NA	NA	NA	NA	NA	✗
Cryptographic primitive	PBKE	PBKE	PBKE	ASPE	PBKE	OPE

multiplications are required. Therefore, the size of the circuit is 184 secure multiplication with the depth of 7.

Our scheme requires the pre-distribution of random binary multiplication triples by the data owner to the servers in the setup phase which are needed for the secure comparisons during the search. This enables the servers to perform the search without further online interaction with the data owner. With the optimization explained in section 2.5, the communication cost for pre-distributing each multiplication triple is a single bit. To compare the search query with each bonding box, four comparisons are required. As mentioned earlier each comparison costs less than 3ℓ secure multiplications in $\mathbb{Z}_2$. Therefore, the overall search complexity in the worst-case scenario is $4m \log m \times 3\ell = 12\ell m \log m$ multiplications in $\mathbb{Z}_2$. Here, m is the maximum number of entries that can fit in each node in the tree. The number of roundtrips between the two servers is $\log m(\log \ell + 1)$ as the four comparisons of the search query with

Table 4
Memory Cost

Memory cost \ Number of records	1000	2000	3000	4000
max 30 objects per node	114 MB	454 MB	5.14 GB	11.43 GB
max 40 objects per node	85 MB	451 MB	9.13 GB	20.88 GB
max 50 objects per node	82 MB	333 MB	4.5 GB	7.96 GB

Table 5
Performance (m=50)

Performance \ Number of records	1000	2000	3000	4000
Search time	128.75 ms	154.7 ms	168.46 ms	180.1 ms
Communication cost (Client-Server)	16 bytes	16 bytes	16 bytes	16 bytes
Communication cost (Server-Server)	556 KB	834 KB	1.1 MB	1.1 MB

each bonding box can be performed in parallel. Finally, to perform the update the client should generate new shares for the leaf node to be updated. There is only one round of communication to send these values to the servers. Moreover, the server only require to replace the current value of a leaf node with the updated values.

Table 3a and Table 3b illustrate the comparison between our Geo-DRS$^+$ scheme with the state-of-the-art schemes supporting spatial range queries of encrypted data from different aspects. Except our scheme and Wang-2017, the search complexity on the server side in all of the existing related works is linearly dependent to the number of data points/records in the database. The token generation (search on client side) complexity is constant only in Geo-DRS$^+$, Zhu-2015, and Zheng-2020, whereas in the rest of the related works it varies from scheme to scheme and depends on different factors such as radius of the circle query, bit length of coordinates, and number of data points/records in the database.

Beside of our Geo-DRS$^+$ scheme, about half of the proposed schemes for geometric range search are presented in the dynamic setting, the rest have limited application as the update of the database cost the re-encryption and re-uploading the entire database. Among the dynamic schemes in this domain only our construction, Xu-2019, and Kasra-II-2020 have only one round of communication between the client and the server for search and update queries.

In terms of the leakages, the search pattern is inherent and unavoidable in all of the discussed schemes. Both constructions of Kasra-2020 and Geo-DRS$^+$ support content privacy as they are not leaking the access pattern. More importantly the access pattern leakage is required to perform the order reconstruction attack, whereas both access and search pattern leakages are exploited for the full database reconstruction attack [23].

7. Implementation and Experimental Results

This section presents the experimental evaluation on the performance of the proposed constructions. All algorithms were implemented in Java (Nodejs v10.10.0, Typescript v3.4.3) on a 64-bit machine with 3.1GHz Intel®Core(i5) processor 8GB RAM and 256GB SSD. We implemented PRF evaluations with SHA-256. We conduct experiments on real-world datasets seqFISH+ [9] and STSCC [15] with 5000 records.

Table 6
Performance (m=40)

Performance \ Number of records	1000	2000	3000	4000
Search time	180.5 ms	194.56 ms	223.4 ms	240.8 ms
Communication cost (Client-Server)	16 bytes	16 bytes	16 bytes	16 bytes
Communication cost (Server-Server)	667 KB	667 KB	1.1 MB	1.1 MB

Table 7
Performance (m=30)

Performance \ Number of records	1000	2000	3000	4000
Search time	215.36 ms	228.18 ms	250.1 ms	269.9 ms
Communication cost (Client-Server)	16 bytes	16 bytes	16 bytes	16 bytes
Communication cost (Server-Server)	501 KB	501 KB	834 KB	834 KB

The update operations for insertion and deletion perform the same. Thus, the cost of setup/update is fixed at 5.74 ms and 12.75 ms at client and server, respectively. The size of the encrypted database is affected by two parameters; the maximum number of entries per node and the total number of records in the dataset. That is, the distribution of the objects in the environment will result in different height of tree structures based on the limit on the maximum number of objects per node. As shown in Table 4, the encrypted dataset requires 114 MB of memory at the server for a dataset with 1000 records while the maximum number of objects per node is 30. The larger dataset with 4000 records requires 7.96 GB of memory if the maximum number of objects per node is set to 50.

We have tested the search time (both at client and server) as well as the communication overhead between client and server and between the two servers. The results for different settings are given in Table 5, 6, and 7.

The results indicate that while the increase in the number of records naturally increases the search time, the decrease in the maximum number of objects per node has the same effect. The reason for the increase in the search time, in this case, is that there are more round trips required to complete the search. For instance, as shown in Figure 9 (similarly in figure 7 and 8) for the same number of objects per node (50), the search time increases from 128.75 ms to 180.1 ms when increasing the number of the dataset records from 1000 to 4000. On the other hand, when the number of dataset records is fixed, for example at 4000, the overall search time increases from 180.1 ms to 269.9 ms by decreasing the number of objects per node in the tree.

As shown in t Table 5, 6, and 7, the communication cost between the client and the server is constant at 16 bytes which is the size of the token. However, the communications between the two servers vary from 501 KB (1000 records, 30 objects per node) to 1.1 MB (4000 records, 40/50 objects per node).

8. Conclusion

We first proposed a dynamic scheme for secure range search over spatial data and then extend it to a more efficient (in terms of client storage and round trips between client and server) version which we named Geo-DRS$^+$. In terms of security and data privacy, Geo-DRS$^+$ scheme has backward and content privacy. As Geo-DRS$^+$ does not leak access pattern and does not rely on OPE, it is resilient against

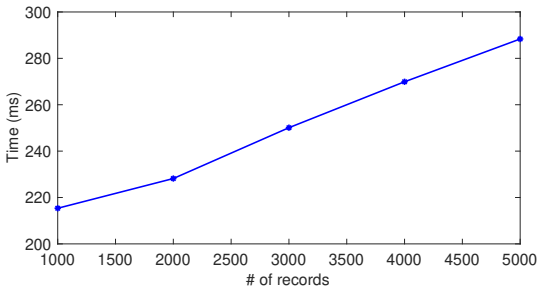

Fig. 7. Search time of Geo-DRS$^+$ scheme for $m = 30$)

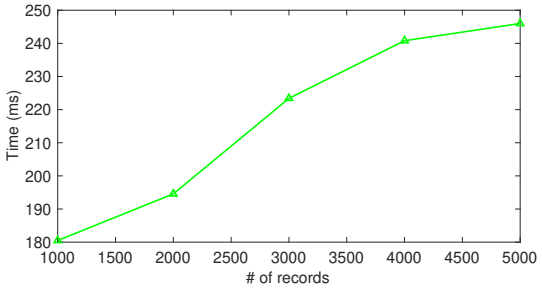

Fig. 8. Search time of Geo-DRS$^+$ scheme for $m = 40$)

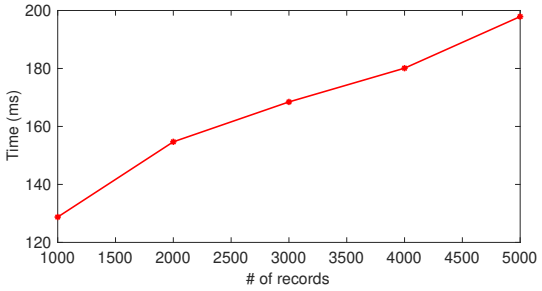

Fig. 9. Search time of Geo-DRS$^+$ scheme for $m = 50$)

recently developed ADR and FDR attacks targeting the searchable encryption schemes supporting geometric range search. The comparisons between Geo-DRS$^+$ and state-of-the-art schemes indicates that it is more appealing in practice due to lower computation and communication overhead.

References

[1] P. K. Agarwal, J. Erickson, et al. Geometric range searching and its relatives. *Contemporary Mathematics*, 223:1–56, 1999.

[2] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Order preserving encryption for numeric data. In *Proceedings of the 2004 ACM SIGMOD*, pages 563–574. ACM, 2004.

[3] D. Beaver. Commodity-based cryptography (extended abstract). In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 446–455, 1997.

[4] R. Bost. $\sigma o \varphi o \varsigma$: Forward secure searchable encryption. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1143–1154. ACM, 2016.

[5] R. Bost, B. Minaud, and O. Ohrimenko. Forward and backward private searchable encryption from constrained cryptographic primitives. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1465–1482. ACM, 2017.

[6] M. D. Cock, R. Dowsley, C. Horst, R. Katti, A. Nascimento, W.-S. Poon, and S. Truex. Efficient and private scoring of decision trees, support vector machines and logistic regression models based on pre-computation. *IEEE TDSC*, 16(2):217–230, March 2019.

[7] R. Dowsley. *Cryptography Based on Correlated Data: Foundations and Practice*. PhD thesis, Karlsruhe Institute of Technology, Germany, 2016.

[8] F. B. Durak, T. M. DuBuisson, and D. Cash. What else is revealed by order-revealing encryption? In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1155–1166. ACM, 2016.

[9] C.-H. L. Eng, M. Lawson, Q. Zhu, R. Dries, N. Koulena, Y. Takei, J. Yun, C. Cronin, C. Karp, G.-C. Yuan, et al. Transcriptome-scale super-resolved imaging in tissues by rna seqfish+. *Nature*, 568(7751):235–239, 2019.

[10] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *International Workshop on Public Key Cryptography*, pages 330–342. Springer, 2007.

[11] P. Grubbs, M. Lacharité, B. Minaud, and K. G. Paterson. Learning to reconstruct: Statistical learning theory and encrypted database attacks. In *2019 IEEE Symposium on Security and Privacy*, pages 1067–1083, 2019.

[12] R. Guo, B. Qin, Y. Wu, R. Liu, H. Chen, and C. Li. Mixgeo: efficient secure range queries on encrypted dense spatial data in the cloud. In *Proceedings of the International Symposium on Quality of Service*, pages 1–10, 2019.

[13] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the 1984 ACM SIGMOD International Conference on Management of Data*, SIGMOD '84, pages 47–57, New York, NY, USA, 1984. ACM.

[14] S. Hoogh, de. *Design of large scale applications of secure multiparty computation: secure linear programming*. PhD thesis, Department of Mathematics and Computer Science, 2012.

[15] A. L. Ji, A. J. Rubin, K. Thrane, S. Jiang, D. L. Reynolds, R. M. Meyers, M. G. Guo, B. M. George, A. Mollbrink, J. Bergenstråhle, et al. Multimodal analysis of composition and spatial architecture in human squamous cell carcinoma. *Cell*, 182(2):497–514, 2020.

[16] S. Kasra-Kermanshahi, R. Dowsley, R. Steinfeld, A. Sakzad, J. K. Liu, S. Nepal, and X. Yi. Geo-drs: Geometric dynamic range search on spatial data with backward and content privacy. In E. Bertino, H. Shulman, and M. Waidner, editors, *Computer Security – ESORICS 2021*, pages 24–43, Cham, 2021. Springer International Publishing.

[17] G. Kellaris, G. Kollios, K. Nissim, and A. O'neill. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC*, pages 1329–1340. ACM, 2016.

[18] S. K. Kermanshahi, S.-F. Sun, J. K. Liu, R. Steinfeld, S. Nepal, W. F. Lau, and M. Au. Geometric range search on encrypted data with forward/backward security. *IEEE Transactions on Dependable and Secure Computing*, pages 1–20, 2020.

[19] E. M. Kornaropoulos, C. Papamanthou, and R. Tamassia. Data recovery on encrypted databases with k-nearest neighbor query leakage. In *2019 IEEE Symposium on Security and Privacy, San Francisco, CA, USA, May 19-23, 2019*, pages 1033–1050, 2019.

[20] M.-S. Lacharité, B. Minaud, and K. G. Paterson. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 297–314. IEEE, 2018.

[21] X. Li, Y. Zhu, J. Wang, and J. Zhang. Efficient and secure multi-dimensional geometric range query over encrypted data in cloud. *Journal of Parallel and Distributed Computing*, 131:44–54, 2019.

[22] Y. Luo, S. Fu, D. Wang, M. Xu, and X. Jia. Efficient and generalized geometric range search on encrypted spatial data in the cloud. In *2017 IEEE/ACM 25th International Symposium on Quality of Service (IWQoS)*, pages 1–10. IEEE, 2017.

[23] E. A. Markatou and R. Tamassia. Database reconstruction attacks in two dimensions. Cryptology ePrint Archive, Report 2020/284, 2020. https://eprint.iacr.org/2020/284.

[24] J. Matoušek. Geometric range searching. *ACM Computing Surveys (CSUR)*, 26(4):422–461, 1994.

[25] G. M. Morton. A computer oriented geodetic data base and a new technique in file sequencing. Technical report, IBM, 1966.

[26] M. Naveed, S. Kamara, and C. V. Wright. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 644–655. ACM, 2015.

[27] Y. Pan, A. Efrat, M. Li, B. Wang, H. Quan, J. Mitchell, J. Gao, and E. Arkin. Data inference from encrypted databases: A multi-dimensional order-preserving matching approach. *arXiv:2001.08773*, 2020.

[28] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects. Technical report, University of Maryland, 1987.

[29] E. Shen, E. Shi, and B. Waters. Predicate privacy in encryption systems. In *Theory of Cryptography, 6th Theory of Cryptography Conference, TCC 2009, San Francisco, CA, USA, March 15-17, 2009. Proceedings*, pages 457–473, 2009.

[30] B. Wang, M. Li, and H. Wang. Geometric range search on encrypted spatial data. *IEEE Transactions on Information Forensics and Security*, 11(4):704–719, 2016.

[31] B. Wang, M. Li, H. Wang, and H. Li. Circular range search on encrypted spatial data. In *2015 IEEE CNS*, pages 182–190. IEEE, 2015.

[32] B. Wang, M. Li, and L. Xiong. Fastgeo: Efficient geometric range queries on encrypted spatial data. *IEEE TDSC*, 16(2):245–258, 2019.

[33] W. K. Wong, D. W.-l. Cheung, B. Kao, and N. Mamoulis. Secure knn computation on encrypted databases. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, pages 139–152, 2009.

[34] G. Xu, H. Li, Y. Dai, K. Yang, and X. Lin. Enabling efficient and geometric range query with access control over encrypted spatial data. *IEEE Transactions on Information Forensics and Security*, 14(4):870–885, 2019.

[35] Z. Zheng, J. Shen, and Z. Cao. Practical and secure circular range search on private spatial data. Cryptology ePrint Archive, Report 2020/242, 2020. https://eprint.iacr.org/2020/242.

[36] H. Zhu, R. Lu, C. Huang, L. Chen, and H. Li. An efficient privacy-preserving location-based services query scheme in outsourced cloud. *IEEE Transactions on Vehicular Technology*, 65(9):7729–7739, 2015.