Efficient Secure and Verifiable Location-Based Skyline Queries over Encrypted Data

Zuan Wang, Xiaofeng Ding, Hai Jin
BDTS, SCTS, CGCL,
School of Computer Science and Technology,
Huazhong University of Science and Technology
{zuanwang,xfding,hjin}@hust.edu.cn

Pan Zhou
Hubei Engineering Research Center on Big Data Security,
School of Cyber Science and Engineering,
Huazhong University of Science and Technology
panzhou@hust.edu.cn

ABSTRACT

Supporting secure location-based services on encrypted data that is outsourced to cloud computing platforms remains an ongoing challenge for efficiency due to expensive ciphertext calculation overhead. Furthermore, since the clouds may not be trustworthy or even malicious, data security and result authenticity has caused huge concerns. Unfortunately, little work can enable query efficiency, dataset confidentiality and result authenticity to be commendably guaranteed. In this paper, we demonstrate the potential of supporting secure and verifiable location-based skyline queries (SVLSQ). First, we devise a novel and unified structure, named semi-blind R-tree (SR-tree), which protects the query unlinkability. Based on SR-tree, we propose an authenticated data structure, named secure and verifiable scope R-tree (SVSR-tree). Then, we develop several secure protocols based on SVSR-tree to accelerate the query efficiency and reduce the size of verification objects. Our method avoids compromising the privacy of datasets, queries, results and access patterns. Meanwhile, it authenticates the soundness and completeness of the skyline results while preserving privacy. Finally, we analyze the complexity and security of SVLSQ. Findings from the performance evaluation illustrate that SVLSQ is a dramatically efficient method in terms of query (no less than 3 orders of magnitude faster than other solutions) and verification.

PVLDB Reference Format:
Zuan Wang, Xiaofeng Ding, Hai Jin and Pan Zhou. Efficient Secure and Verifiable Location-Based Skyline Queries over Encrypted Data. PVLDB, 15(9): 1822 - 1834, 2022.
doi:10.14778/3538598.3538605

1 INTRODUCTION

Outsourcing data to cloud platforms allows data owners to use tremendous computing power and massive storage capacity. However, in order to offer efficient location-based services in this fashion, data confidentiality and result authenticity are the primary issues. Concretely, the cloud cannot be trusted since it may capture and deduce the sensitive content of data. Meanwhile, the cloud may actually be compromised or even malicious rather than be semi-honest [39]. In this case, it could send incorrect results to clients for program glitches or commercial interests, where users have no ability to determine whether results are authentic. Hence, guaranteeing data confidentiality and result authenticity becomes a crucial and urgent issue to be tackled.

Proceedings of the VLDB Endowment, Vol. 15, No. 9 ISSN 2150-8097.
doi:10.14778/3538598.3538605

Point	Price
P_1	6
P_2	5
P_3	3
P_4	4
P_5	4
P_6	7
P	8

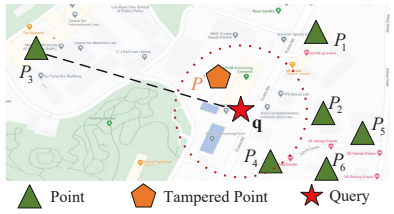


Figure 1: Sample of the skyline query

In this paper, we focus on one of the most prevalent location-based services, as called secure and verifiable location-based skyline queries (SVLSQ) on encrypted datasets (e.g., restaurants, hotels, and plazas). As shown in Fig. 1, we provide an example on how to choose a suitable hotel, where $P = \{\mathcal{P}_1, ..., \mathcal{P}_6\}$ indicates a 2-dimensional dataset, namely: location and price, and $\boldsymbol{q}$ is a query with user's location. We employ the skyline query to calculate 1NN records based on the weight of each attribute so that it filters hotels in advance for tourists with different preferences.

Thus, the tourist obtains real skyline results $\{\mathcal{P}_3, \mathcal{P}_4\}$ for $\boldsymbol{q}$ from Fig. 1. However, the cloud may return tampered results $\{\mathcal{P}_3, \mathcal{P}_2, \mathcal{P}'\}$ due to commercial factors, e.g., $\mathcal{P}_2$ is a hotel sponsoring the cloud platform and $\mathcal{P}'$ is a forged hotel with high consumer prices to serve as a foil to $\mathcal{P}_2$. Obviously, the user cannot validate the reliability of results. Therefore, our SVLSQ provides a verification mechanism to guarantee the authenticity of results in two ways [4, 21, 38]: *i*) soundness, i.e., no tampered results (e.g., $\mathcal{P}_2$ and $\mathcal{P}'$) and *ii*) completeness, i.e., no discarded results (e.g., $\mathcal{P}_4$). In addition, without security guarantee, the content such as dataset P, query $\boldsymbol{q}$ and result $\mathcal{R}$ may be leaked to the cloud server. To this end, our SVLSQ also aims to protect four widely adopted aspects [9, 18, 24]: *i*) *data privacy*: the dataset P; *ii*) *skyline result privacy*: the skyline result $\mathcal{R}$; *iii*) *query privacy*: the query $\boldsymbol{q}$; and *iv*) *access patterns privacy*: the positions of results in P, which include spatial and non-spatial attributes of the data points.

Recently, there have been a number of solutions to tackle the problem of data confidentiality in privacy-preserving queries, but they cannot guarantee the above requirements completely [27, 31, 37]. Some works [7, 36, 42] use AES (a symmetric encryption), OPE [32] or ORE [6, 20] to encrypt datasets and then implement secure queries. However, calculations in this ciphertext domain, such as distance, cannot be achieved, which is a necessary procedure involved in location-based services. Moreover, these works [7, 36, 42] cannot guarantee the requirement of access patterns privacy

Table 1: Summary of representative approaches

Approach	Data Privacy	Query Privacy	Result Privacy	Verifiable	Supporting Skyline	Access Patterns Privacy	Distance Calculation
Wu [38]	✓	✓	✓	✓	×	–	–
Cui [9]	✓	✓	✓	✓	×	–	–
Choi [7]	✓	✓	✓	×	×	–	–
Elmehdwi [11]	✓	✓	✓	×	×	–	–
Liu [24, 25]	✓	✓	✓	×	✓	✓	✓
Wang [36]	✓	✓	✓	×	✓	×	×
Liu [27]	×	✓	✓	×	✓	×	✓
Zeighami[42]	✓	✓	✓	×	✓	×	×
Our work	✓	✓	✓	✓	✓	✓	✓
'✓' denotes the method meets the requirement; '×' denotes it violates the requirement; '–' represents it is ignored for not supporting skyline.							

without oblivious RAM structures [34]. As indicated in [17], when attackers launch an inference attack with several prior information about the data, the query can be recovered if attackers know the positions of results among P. Hence, approaches based on the partial homomorphic encryption (PHE) [1] are proposed to solve secure queries [11, 25]. However, as shown in [11], the efficiency of secure kNN query is too poor to be practical. Similarly, the efficiency of secure skyline queries [25] using Paillier [29] (a PHE) is also low on large-scale datasets without parallelism.

Besides, the above works assume that the cloud is semi-honest. That is, it may return real results. In fact, it may be malicious enough to return tampered results for program glitches, security vulnerabilities, and commercial interests. However, little work has so far focused on secure and verifiable skyline queries. Recently, in [9], the authors process a secure kNN query based on a PHE. Another approach [38] addresses secure range queries using the secure and verifiable tree, but some sensitive information other than results is exposed to users during the verification process. Furthermore, neither of them are appropriate for the skyline computation. For demonstration purposes, we exhibit several representative approaches in Table 1. It is worth notice that none of the published solutions can be sufficiently efficient to address the problem about location-based skyline queries while simultaneously provide verifiability (i.e., to guarantee the soundness and completeness of the results) and four aspects of privacy as mentioned above.

Inspired by this, our ongoing goal is to devise an efficient solution that guarantees all privacy requirements to process location-based skyline queries on static or infrequently updated datasets (e.g., hotels and carparks).[1] However, there are several key differences that render existing techniques inapplicable to SVLSQ. First, most existing secure skyline query techniques [25, 36, 42] cannot provide result verificaton for clients. Second, most existing variants of R-tree indexes (e.g., Merkle R-tree [40, 41]) and their query methods cannot guarantee the privacy of datasets, results, queries and access patterns simultaneously. Third, one naive method is to directly build an R-tree index with encryption. However, this method cannot preserve the query unlinkability, i.e., the cloud can track visiting paths of two queries to determine whether they are from the same query. Moreover, this unsophisticated method cannot provide result verification without privacy leakage since the verification object decrypted to the client contains original contents of the dataset.

[1]The problem of guaranteeing the freshness of query results is beyond the scope of our study. We cannot provide queries over the most up-to-date outsourced dataset.

Clearly, there are two key technical challenges that need to be addressed. 1) *How do we devise an authenticated data structure (ADS) for secure skyline queries while guaranteeing data confidentiality and result authenticity?* To this end, we firstly propose a unified index structure called semi-blind R-tree (SR-tree) to preserve the query unlinkability. Due to the semi-blind structure, the positions of results among P are hidden from the cloud. Then, according to SR-tree, we build the secure and verifiable R-tree (SVR-tree) index using Paillier, ORE and hash function. After that, we propose a skyline searching algorithm named basic and verifiable location-based skyline queries (BVLSQ) based on SVR-tree. However, in BVLSQ, the dominance operation drags down the query efficiency, and the size of verification objects (VOs) is too large partly due to the ciphertext of ORE. Furthermore, although ORE protects contents of the data (in VOs), the ordering of some data is leaked to clients. These lead to the next technical challenge. 2) *How do we further accelerate the efficiency, optimize the communication bandwidth and enhance data security?* We observe that the dominance relationships of points can be pre-computed by the data owner such that the query and verification time could be further reduced. To decrease the communication burden as caused by the ciphertext of ORE, we devise a novel leaf node structure, which takes into account data confidentiality during the query and verification. In general, with the semi-blind idea of SR-tree as the cornerstone, we propose a newly developed secure and verifiable scope R-tree (SVSR-tree) merely using Paillier and hash function, which stores the encrypted data objects and verification information (instead of encrypted data points themselves). After that, we present secure and verifiable location-based skyline queries (SVLSQ) protocol, where the positions of results could be preserved from the cloud.

Using intervals instead of points to represent data objects makes SVR-tree and SVSR-tree variants of R-tree indexes [12]. However, in contrast to the initially proposed techniques, our approach allow us to (1) efficiently support secure skyline queries in the ciphertext domain, (2) verify the result authenticity without privacy leakage, and (3) preserve the query unlinkability. Furthermore, none of those existing techniques could commendably guarantee query efficiency, data confidentiality and result authenticity simultaneously.

Observations from extensive experiments show that the query performance of our proposed SVLSQ (BVLSQ) is no less than 3 orders of magnitude faster than the other approaches from [24, 25]. Moreover, compared with SVR-tree, the VO size belonging to SVSR-tree is reduced notably.

To summarize, we present our contributions as follows.

- As far as we are aware, this is the first work to study the secure and verifiable location-based skyline queries over encrypted data, which preserves the aforementioned privacy requirements and guarantees the authenticity of results while achieving considerable efficiency.
- We propose a novel unified index structure named semi-blind R-tree (SR-tree), which could protect the query unlinkability. Furthermore, we devise an SVR-tree and present the BVLSQ protocol to process the query efficiently and reduce the size of verification objects dramatically.
- Combining the properties of location, we also devise another ADS named SVSR-tree based on SR-tree. Then, we propose the SVLSQ protocol which is more efficient and adaptive at constructing VOs for smaller sizes.

2 PRELIMINARIES AND PROBLEM FORMULATION

2.1 Cryptographic Tools

Paillier Cryptosystem [29]. In order to ensure the semantic security of the ciphertext for distance calculation, the Paillier cryptosystem serves as the encryption scheme. Paillier encrypts the plaintext x_1 by the public key pk (denoted by $[\![x_1]\!]$) and decrypts the ciphertext m_1 by the private key sk (denoted by $D_{sk}(x_1)$), where $[\![\cdot]\!]$ and $D_{sk}(\cdot)$ respectively represent encryption and decryption functions. It also has additive homomorphic properties as follows:

$$D_{sk}([\![x_0]\!] \times [\![x_1]\!] \bmod N^2) = x_0 + x_1 \bmod N, \tag{1}$$

$$D_{sk}([\![x_0]\!]^{x_1} \bmod N^2) = x_0 \times x_1 \bmod N, \tag{2}$$

where $x_0, x_1 \in \mathbb{Z}_N$, N is the product of two large primes.

Order-Revealing Encryption [6, 20]. To implement ciphertext comparison, we use the order-revealing encryption (ORE) scheme to encrypt data with any secret key by $Enc(\cdot)$. Then, we compare $Enc(\mathcal{P}_1)$ and $Enc(\mathcal{P}_2)$ by $compare(\cdot,\cdot)$, and it returns $\{-1, 0, 1\}$.

Cryptographic Hash Function. We generate a fixed-length string by a one-way and collision-resistant hash function $Hash(\cdot)$ (such as SHA-1) to build a new ADS. The data owner can calculate the digital signature $Sig(Hash(\mathcal{P}_1), \mathcal{K}_1)$ of $Hash(\mathcal{P}_1)$ with private key $\mathcal{K}_1$. For ease of presentation, hereafter, we replace $Sig(Hash(\cdot), \mathcal{K}_1)$ with $Sig(Hash(\cdot))$.

2.2 System Model

In the cloud environment, we adopt two collude-resistant clouds [9, 25], as shown in Fig. 2. Usually, these two clouds named data service provider (DSP) and data assistance provider (DAP) are supposed to be competitive companies, generally from prestigious enterprises like Amazon and Microsoft. Moreover, in order to complement the verification function, we add the verification object VO into the model. The specific description of the model is as follows:

(1) **Data Owner (DO)**. As a trusted entity with dataset P, the DO generates the Paillier cryptosystem's keys $< pk, sk >$ and a hash function $Hash(\cdot)$, then it builds a secure and verifiable index I and additional encrypted information Tag according to P. Next, the DO sends pk, I to DSP, and $< pk, sk >$, Tag to DAP, respectively. After successfully auditing the registration information from the requester, the DO assigns pk and $Hash(\cdot)$ to it.

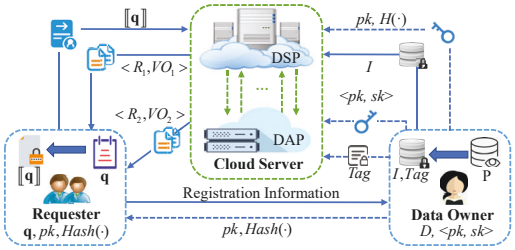


Figure 2: System model

(2) **Queries Requester (QR)**. An authorized client sends an encrypted query $[\![\boldsymbol{q}]\!]$ to DSP. After receiving query results and VOs from DSP and DAP, referred to as $< \mathcal{R}_1, VO_1 >$ and $< \mathcal{R}_2, VO_2 >$, respectively, the client further computes skyline results and checks their authenticity.

(3) **Two Cloud Servers**. The data service provider (DSP), one cloud server, processes secure and verifiable skyline queries over I and $[\![\boldsymbol{q}]\!]$ in collaboration with the data assistance provider (DAP) when receiving query request from QR. When the query finishes, DSP and DAP respectively return $< \mathcal{R}_1, VO_1 >$ and $< \mathcal{R}_2, VO_2 >$.

2.3 Security Model

In our security model, there are the following threats: *i*) since the clouds are untrustworthy, the results returned from them may not be correct; *ii*) the clouds are so curious that they collect confidential content during query processing.

To address the first threat, the client can verify query results $\mathcal{R}$ based on a secure and efficient *authenticated data structure* (ADS), which involves the following definition.

Definition 1. *(Soundness and Completeness) Given a dataset P, a query function $Query(\cdot)$ and a query $\boldsymbol{q}$, the cloud returns the result $\mathcal{R} \subseteq P$, if for $\forall \mathcal{R}_i \in \mathcal{R}$, $\mathcal{R}_i \in Query(\boldsymbol{q}, P)$, then we say $\mathcal{R}$ is sound. If for $\forall \mathcal{P}_j \in P - \mathcal{R}$, $\mathcal{P}_j \notin Query(\boldsymbol{q}, P)$, then we say $\mathcal{R}$ is complete.*

To address another threat, we devise two secure indexes and protocols to process queries with privacy guarantee. We also summarize the following privacy requirements similar to [9, 24].

(1) **Data Privacy**. The content about the original data cannot be gained by two clouds; likewise, the client knows nothing about the data except the query result $\mathcal{R}$.

(2) **Result Privacy**. No one can gain the plaintext of query results $\mathcal{R}$ except the corresponding client.

(3) **Query Privacy**. Two clouds cannot know the content of $\boldsymbol{q}$.

(4) **Indirect Privacy**. Two clouds know nothing about the positions of the results (i.e., access patterns) in the dataset. Furthermore, they cannot realize whether two queries are identical by tracking visiting paths, which is also named query unlinkability.

Note that we present the formal security definition and analysis in Section 6.2.

2.4 Problem Definition

Let $P = \{\mathcal{P}_1, ..., \mathcal{P}_n\}$ be a dataset, and each point $\mathcal{P}_i \in P (1 \le i \le n)$ is associated with two spatial attributes (denoted by $(\mathcal{P}_i[1], \mathcal{P}_i[2])$) and several non-spatial attributes (denoted by $(\mathcal{P}_i[3], ..., \mathcal{P}_i[d])$), where d denotes the number of all dimensions and d^* denotes the number of non-spatial dimensions of $\mathcal{P}_i$.

Definition 2. *(Non-Spatial Dominance [16]) Given two points $\mathcal{P}_i$ and $\mathcal{P}_j$, for any non-spatial dimension v ($3 \le v \le d$), if $\mathcal{P}_i[v] \le \mathcal{P}_j[v]$, we say $\mathcal{P}_i$ non-spatially dominates $\mathcal{P}_j$, and $\mathcal{P}_i$ is a non-spatial dominator of $\mathcal{P}_j$, denoted by $\mathcal{P}_i \preceq \mathcal{P}_j$. The set of $\mathcal{P}_j$'s non-spatial dominators is denoted as $Dom(\mathcal{P}_j)$.*

Definition 3. *(Location-Based Dominance [13]) Given a location query point $\boldsymbol{q} = (\boldsymbol{q}[1], \boldsymbol{q}[2])$ and two points $\mathcal{P}_i$ and $\mathcal{P}_j$, if (1) $\mathcal{P}_i \preceq \mathcal{P}_j$ and (2) $\mathcal{P}_i$ is closer to $\boldsymbol{q}$ than $\mathcal{P}_j$, then we say $\mathcal{P}_i$ dominates $\mathcal{P}_j$ w.r.t. $\boldsymbol{q}$, denoted by $\mathcal{P}_i \prec_{\boldsymbol{q}} \mathcal{P}_j$.*

Definition 4. *(Location-Based Skyline Query, LSQ [13]) Given a dataset P, the location-based skyline result $\mathcal{R}$ of a query point q is a subset of P, $LSQ(P, q)$, such that for each $\mathcal{P}_i \in \mathcal{R}$, $\forall \mathcal{P}_j \in P, \mathcal{P}_j \not\prec_{\boldsymbol{q}} \mathcal{P}_i$, and for each $\mathcal{P}_j \in P - \mathcal{R}$, $\exists \mathcal{P}_i \in \mathcal{R}, \mathcal{P}_i \prec_{\boldsymbol{q}} \mathcal{P}_j$.*

Definition 5. *(Secure and Verifiable Location-Based Skyline Query, SVLSQ) Given the secure index I over P, and an encrypted query $[\![\boldsymbol{q}]\!]$, SVLSQ aims to securely provide VOs and results $\mathcal{R} = LSQ(P, q)$ to the client by using I and $[\![\boldsymbol{q}]\!]$ such that for each $\mathcal{P}_i \in \mathcal{R}, \forall \mathcal{P}_j \in P$, $[\![\mathcal{P}_j]\!] \not\prec_{\boldsymbol{q}} [\![\mathcal{P}_i]\!]$, and for each $\mathcal{P}_j \in P - \mathcal{R}, \exists \mathcal{P}_i \in \mathcal{R}, [\![\mathcal{P}_i]\!] \prec_{\boldsymbol{q}} [\![\mathcal{P}_j]\!]$. According to VOs, the client is able to verify the soundness and completeness of $\mathcal{R}$.*

3 SEMI-BLIND R-TREE

To protect the query unlinkability, a novel structure named semi-blind R-tree (SR-tree) is devised. It is a height-balanced tree similar to R-tree. Concretely, the leaf node in SR-tree contains data objects referring to encrypted points in P. The common non-leaf node contains index objects in the form of $([\![MBR]\!], pointer)$, where $pointer$ is the address of lower node in SR-tree and $[\![MBR]\!]$ is an encrypted minimum bounding rectangle (MBR) which is denoted as:

$$[\![MBR]\!] = ([\![mbr_1]\!], [\![mbr_2]\!], ..., [\![mbr_d]\!]), \tag{3}$$

where $[\![mbr_i]\!]$ is a closed bounded interval $\left[[\![mbr_i^{low}]\!], [\![mbr_i^{upp}]\!]\right]$ describing the encrypted extent of each object along dimension i. We use Paillier with pk (i.e., $[\![\cdot]\!]$) to encrypt each element in mbr_i, where $[\![\cdot]\!]$ denotes the Paillier encryption function. The special non-leaf node at the penultimate level in SR-tree contains intermediate entries (called blind objects) of the form $([\![MBR]\!], \mathcal{B}(\cdot))$, where $\mathcal{B}(\cdot)$ is the function calculating its corresponding children nodes. Notice that the index object corresponds to a lower leveled node with encrypted MBR, while the object of leaf node corresponds to encrypted data point in P. We present the formal definition of SR-tree as follows.

Definition 6. *(Semi-Blind R-tree) Given the semi-blind R-tree SRT and a node N_i, SRT's level is $\{L_0, L_1, ...\}$ from bottom to top. Thereupon SRT can be formalized as:*

$$SRT = \begin{cases} \{[\![\mathcal{P}_j]\!]\}_{1 \le j \le c}, & N_i \in L_0 \\ \{([\![MBR_j]\!], pointer)\}_{1 \le j \le \theta}, & N_i \in \{L_k\}_{k \ne 0,1} \\ \{([\![MBR_j]\!], \mathcal{B}(\cdot))\}_{1 \le j \le \theta}, & N_i \in L_1 \end{cases}, \tag{4}$$

where $[\![\mathcal{P}_j]\!]$, c/θ, $[\![MBR_j]\!]$, $pointer$ and $\mathcal{B}(\cdot)$ respectively refer to an encrypted point, the capacity of a leaf or non-leaf node, an encrypted minimum bounding rectangle, the address of a node at the lower level and the function of calculating the corresponding children nodes.

Next, we illustrate how the data owner build SR-tree. A naive but effective method is to encrypt each element in R-tree with Paillier cryptosystem. However, this method fails to protect query unlinkability. To this end, the data owner needs to hide visiting path from the blind object to the data object. Concretely, the leaf nodes (for ease of understanding, a leaf node is denoted as an encrypted vector $[\![N_i]\!] = \left[[\![\mathcal{P}_1]\!]...[\![\mathcal{P}_c]\!]\right]$, where c is the capacity of leaf node) of the same parent node are structured into a set called a node bucket NB, which is denoted as $NB = \left[[\![N_1]\!]^{\mathsf{T}}...[\![N_\theta]\!]^{\mathsf{T}}\right]^{\mathsf{T}}$ and θ ($\theta \le c$) is the capacity of this parent node. For example, as shown in Fig. 3(b), leaf nodes corresponding to o_1 and o_2 at node N_6 are N_1 and N_2, respectively. So, N_1 and N_2 are structured into a node bucket NB (i.e., $NB = \left[[\![N1]\!]^{\mathsf{T}}[\![N_2]\!]^{\mathsf{T}}\right]^{\mathsf{T}}$). For each blind object, we set up an encryption vector IV, which is consisted of $[\![1]\!]$ and $[\![0]\!]$. If the blind object corresponds to the k-th node in NB, then $IV[k] = [\![1]\!]$ and $\{IV[i] = [\![0]\!]\}_{1 \le i \le \theta, i \ne k}$. Hence, in the semi-blind structure, with the encrypted vector IV and the node bucket NB as input, an encrypted leaf node $[\![N_i]\!]$ is calculated through $\mathcal{B}(\cdot)$ as follows.

$$[\![\mathcal{P}_j]\!] = \prod_{k=1}^{\theta} \mathrm{SM}(IV[k], NB_{kj}), \tag{5}$$

$$[\![N_i]\!] = \mathcal{B}(IV, NB) = \left[[\![\mathcal{P}_1]\!] \cdots [\![\mathcal{P}_j]\!] \cdots [\![\mathcal{P}_c]\!]\right], \tag{6}$$

where $\mathrm{SM}(\cdot,\cdot)$ is the secure multiplication (SM) protocol from [11].

Note that the data owner uploads the SR-tree except the encrypted vector IV to DSP, and the set of encrypted vector IV is uploaded to DAP as an additional encrypted information Tag. When the blind object is accessed, DSP requests DAP to obtain the corresponding IV. Before returning IV to DSP, DAP updates IV by calculating $IV = IV \times [\![0]\!]$. Then, DSP calculates the leaf node by $\mathcal{B}(\cdot)$. Owing to the probabilistic property of Paillier, the ciphertext of identical leaf nodes are different. Hence, each node is 'read' from NB blindly and the cloud cannot know its exact position in P.

4 BASIC METHOD

4.1 Secure and Verifiable R-tree

Based on SR-tree, we devise a secure and verifiable R-tree (SVR-tree) to preserve the privacy and offer verification information. Concretely, the leaf node N_i in SVR-tree contains entries in the form of $< [\![\mathcal{P}_i]\!], [\![Enc(\mathcal{P}_i)]\!], [\![\mathcal{D}_i]\!], [\![\mathcal{S}_i]\!] >$, where $[\![\mathcal{P}_i]\!]$, $[\![Enc(\mathcal{P}_i)]\!]$, $[\![\mathcal{D}_i]\!]$ and $[\![\mathcal{S}_i]\!]$ respectively refer to $\mathcal{P}_i$ encrypted by Paillier, $Enc(\mathcal{P}_i)$ (i.e., $\mathcal{P}_i$ encrypted by ORE) encrypted by Paillier, the encrypted hash value of $Enc(\mathcal{P}_i)$, i.e., $[\![\mathcal{D}_i]\!] = [\![Hash(Enc(\mathcal{P}_i))]\!]$, and the encrypted digital signature of $\mathcal{P}_i$ and $Enc(\mathcal{P}_i)$, i.e.,

$$[\![\mathcal{S}_i]\!] = [\![Sig(Hash(Hash(\mathcal{P}_i)|Hash(Enc(\mathcal{P}_i))))]\!]. \tag{7}$$

The non-leaf node N_i contains intermediate objects (i.e., o_i) of the form $< [\![MBR]\!], [\![Enc(MBR)]\!], pointer/\mathcal{B}(\cdot), [\![\mathcal{H}]\!] >$, where $[\![MBR]\!]$, $[\![Enc(MBR)]\!]$ and $[\![\mathcal{H}]\!]$ refer to MBR encrypted by Paillier, $Enc(MBR)$ (i.e., MBR encrypted by ORE) encrypted by Paillier, and the encrypted hash value of the corresponding node at lower level. In specific, $o_i.[\![\mathcal{H}]\!]$ is calculated as:

$$o_i.[\![\mathcal{H}]\!] = \begin{cases} [\![Hash(...|\mathcal{P}_z.\mathcal{D}|...)]\!], & N_i \text{ is a leaf node} \\ [\![Hash(...|o_y.MBR|o_y.\mathcal{H}|...)]\!], & N_i \text{ is a non-leaf node} \end{cases}, \tag{8}$$

where $\mathcal{P}_z$ ($z \in [1, c]$ and o_y ($y \in [1, \theta]$) are covered by N_i. Through the above calculation, the root hash value of SVR-tree is denoted

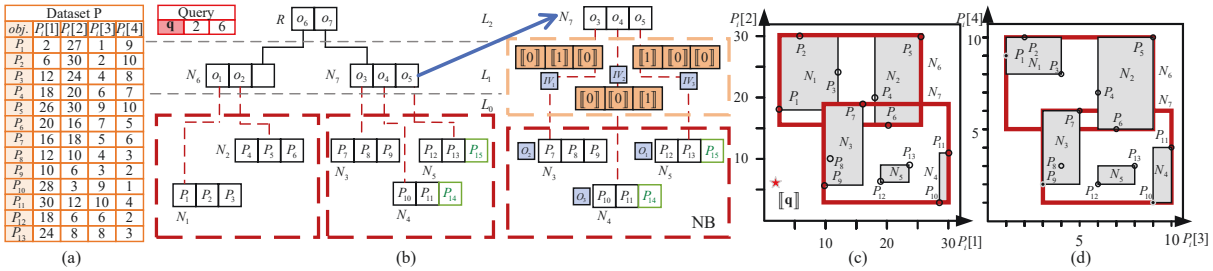


Figure 3: An example of SVR-tree index. (a) is the dataset of 4D data points. (b) is an SVR-tree, where each leaf node N_i ($1 \le i \le 5$) has 3 data objects (the objects with green are padding objects), all of which contain $< [\![\mathcal{P}_j]\!], [\![Enc(\mathcal{P}_j)]\!], [\![\mathcal{D}_j]\!], [\![\mathcal{S}_j]\!] >$ ($1 \le j \le 15$). The entries of non-leaf nodes at L_1 (we call them blind objects, such as o_1, o_2,...,o_5) contain $< [\![MBR]\!], [\![Enc(MBR)]\!], \mathcal{B}(\cdot), [\![\mathcal{H}]\!] >$ which is a semi-blind structure. The entries of non-leaf nodes at L_2 (i.e., index objects o_6 and o_7) contain $< [\![MBR]\!], [\![Enc(MBR)]\!], pointer, [\![\mathcal{H}]\!] >$. (c) and (d) respectively visualize the spatial and non-spatial information.

as $\mathcal{H}_{root}$. To guarantee the data authenticity, we need to generate an encrypted signature $[\![Sig(\mathcal{H}_{root})]\!]$.

Intuitively, as shown in Fig. 3(a), given a dataset $P = \{\mathcal{P}_1, ..., \mathcal{P}_{13}\}$, points in P, such as $\mathcal{P}_1$, $\mathcal{P}_2$ and $\mathcal{P}_3$, which are close to each other are grouped into the same leaf node N_1 (node capacity $c = 3$). Although the efficiency of R-tree is generally considered to deteriorate rapidly for high-dimensional spaces, since most location-based scenarios involve no more than five dimensions [19], thus our proposed SVR-tree still keeps high efficiency and effective in real situations. In Fig. 3(b), it is an SVR-tree with a typical semi-blind structure ($c = 3$ and $\theta = 3$). Fig. 3(c) visualizes the spatial attributes of objects and $[\![\boldsymbol{q}]\!]$, while the non-spatial dimensions of objects are vividly presented in Fig. 3(d). The shaded parts in Fig. 3(c) and 3(d) indicate that their locations are hidden from the clouds due to query unlinkability.

Since some leaf nodes may not be fully filled, to defend against inference attacks according to the number of data points in leaf nodes (e.g., DSP determines whether two queries have visited the same nodes based on the number of objects), some crafted points are generated by the data owner. By doing so, noise points are appended into nodes lacking sufficient entries, e.g., the green points in Fig. 3(b). Note that the fake points do not affect the precision of queries because a certain point will always dominate them (their values are set to be large enough). Similar to SR-tree, SVR-tree of P is constructed by the data owner. Then, SVR-tree except the encrypted vector IV is uploaded to DSP. Furthermore, an additional encrypted information Tag composed of IV and the corresponding hash value $\mathcal{H}$ is uploaded to DAP. The traversal method used to retrieve skyline results adopts the branch-and-bound paradigm, the detailed query procedure is given in Section 4.3.

Discussion. Since the leaf node is calculated from the node bucket NB rather than read from the index for each query, the cloud cannot determine whether two queries are same by tracking access paths. Therefore, DSP cannot distinguish which points in NB or P are the results (i.e., the positions of results). Although the semi-blind structure incurs additional computational cost, secure skyline queries with SVR-tree only perform dominance operations on few data points without traversing the entire datasets. Meanwhile, the number of semi-blind structure triggered is only related to that of skyline results, which is very beneficial for most applications with sparse results and massive data. For example, the semi-blind structure is only triggered 3 times for 13 tuples in Fig. 3(b).

4.2 Location-Based Secure Dominance Algorithm

The purpose of location-based secure dominance (LSDM) algorithm is to calculate the dominance relationship among the encrypted object o and result object s. The DSP has s from $\mathcal{R}$, an encrypted Euclidean distance $[\![d_s]\!]$ between s and $[\![\boldsymbol{q}]\!]$, o from I, and an encrypted Euclidean distance $[\![d_o]\!]$ between o and $[\![\boldsymbol{q}]\!]$, where $o.MBR$ (or $o.\mathcal{P}$) and $s.\mathcal{P}$ are not revealed to both clouds. The basic thought of LSDM algorithm is to determine who is closer to $\boldsymbol{q}$ from s and o. Then, the non-spatial dominance relationship is calculated by returning 1 if $s \prec_{\boldsymbol{q}} o$, otherwise, 0 is returned.

Concretely, DSP sends $[\![d_o + r]\!]$ and $[\![d_s + r]\!]$ with noise r to DAP, where $r \in \mathbb{Z}_N^*$ and is generated by pseudo-random function f. Upon receiving values, DAP decrypts them to obtain d_o' and d_s'. If $d_s' < d_o'$, DAP gets $\Psi = 1$, which means s is closer to $\boldsymbol{q}$ than o. Otherwise, DAP gets $\Psi = 0$. Then, DAP sends Ψ to DSP. Subsequently, DSP calculates $\mathbf{u}'[i-2] = s.[\![Enc(\mathcal{P})[i]]\!] \times o.[\![Enc(mbr_i^{low})]\!]^{N-1} / o.[\![Enc(\mathcal{P})[i]]\!]^{N-1}$ for $3 \le i \le d$ and sents them to DAP. Once these values are received, DAP decrypts them and returns the numerical relationship $\mathbf{u}[j] = 1$ if $D_{sk}(\mathbf{u}'[j]) \le 0$, otherwise, if $D_{sk}(\mathbf{u}'[j]) > 0$, $\mathbf{u}[j] = 0$, where $j \in [1, d-2]$. Finally, DSP calculates the dominance relationship $\Phi = \Psi \wedge \mathbf{u}[1] ... \wedge \mathbf{u}[d-2]$.

4.3 BVLSQ Protocol

To begin with, we present an important notion about the distance over all dimensions (i.e., *mindist*) from object o to query point $\boldsymbol{q}$. If o is a data object (i.e., $\mathcal{P}$), we denote the distance by $mindist(\mathcal{P}, \boldsymbol{q}) = dist(\mathcal{P}, \boldsymbol{q}) + \sum_{i=3}^{d} \mathcal{P}[i]$, where $dist(\mathcal{P}, \boldsymbol{q})$ is the spatial distance between $\mathcal{P}$ and $\boldsymbol{q}$. If o is an index object, the distance from o to $\boldsymbol{q}$ is denoted as $mindist(o, \boldsymbol{q}) = dist(o, \boldsymbol{q}) + \sum_{i=3}^{d} o.mbr_i^{low}$, where $dist(o, \boldsymbol{q})$ is the minimum spatial distance between o and $\boldsymbol{q}$. Then, we demonstrate a basic method named BVLSQ based on SVR-tree to help DSP to prune dominated index objects to decrease the size of VO, while guaranteeing the ability to verify the results.

Obviously, in our BVLSQ, all data is encrypted. Therefore, under the premise of protecting data privacy, how to calculate an encrypted *mindist* (denoted as $[\![mindist]\!]$) becomes a primary problem. Since the squared spatial distance $[\![dist^2]\!]$ is easier to compute than $[\![dist]\!]$, we employ the former one to calculate. With the help of secure squared Euclidean distance (SSED) protocol [22], $[\![dist^2]\!]$

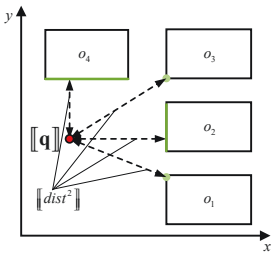


Figure 4: Squared distance $[\![dist^2]\!]$ between o and $[\![\mathbf{q}]\!]$

between $o.[\![\mathcal{P}]\!]$ and $[\![\boldsymbol{q}]\!]$ is calculated in ciphertext form without privacy leakage if o is a data object. Otherwise, if o is an index object, $[\![dist^2]\!]$ between $o.[\![MBR]\!]$ and $[\![\boldsymbol{q}]\!]$ is calculated based on the position of o relative to $\boldsymbol{q}$ using SSED (see Fig. 4). Then, DSP calculates $[\![mindist^2(\cdot)]\!]$ instead of $[\![mindist(\cdot)]\!]$ as Eq. 9.

$$[\![mindist^2(o,\boldsymbol{q})]\!]=\begin{cases}[\![dist^2(o,\boldsymbol{q})]\!]\times\prod_{i=3}^{d}\varpi_i, \text{ o is a data object}\\ [\![dist^2(o,\boldsymbol{q})]\!]\times\prod_{i=3}^{d}m_i, \text{ o is an index object}\end{cases}, \quad (9)$$

where $\varpi_i = [\![o.\mathcal{P}[i]^2]\!]$ and $m_i = [\![o.mbr_i^{low^2}]\!]$ can be calculated by the secure multiplication (SM) protocol [11].

In the sequel, we present the BVLSQ protocol. The skyline results and VOs are respectively stored in $\mathcal{R}$ and $voTree$. First, $\mathcal{R}$ is initialized to empty and the root of SVR-tree is added into $voTree$. The SMIN-Queue SQ is maintained to keep the index (data) objects to be scanned with the object of minimum $[\![mindist^2(\cdot)]\!]$ at top through the secure minimum (SMIN) protocol [11]. For each loop iteration, the top object o is popped up from SQ. Then, DSP and DAP cooperatively use LSDM to determine whether o is dominated by some point in $\mathcal{R}$ and retun the intermediate result ζ. If $\zeta = 1$, DSP discards o and keeps $(o.[\![Enc(MBR)]\!], o.[\![\mathcal{H}]\!])$ or $(o.[\![Enc(\mathcal{P})]\!], o.[\![\mathcal{D}]\!])$ in this node. Otherwise, if $\zeta = 0$, DSP processes o based on its type. In the case where o is an index object, DSP needs to calculate its children by $\mathcal{B}(\cdot)$ with input $o.IV$ from DAP if o is a blind object, where $o.IV$ is obtained by hash value $\mathcal{H}$. If o is a common index object, child objects are obtained directly from the index. For each child object o_i of o, o_i is discarded and $(o_i.[\![Enc(MBR)]\!], o_i.[\![\mathcal{H}]\!])$ is inserted into $voTree$ if o_i is dominated by some point in $\mathcal{R}$ with LSDM. Otherwise, o_i is inserted into SQ and $voTree$. In the case where o is a data object, $o.[\![\mathcal{P}]\!]$ and $(o.[\![Enc(\mathcal{P})]\!], o.[\![\mathcal{D}]\!], o.[\![\mathcal{S}]\!])$ are respectively inserted into $\mathcal{R}$ and $voTree$. The algorithm continues until SQ = $\emptyset$.

Returning $\mathcal{R}$ and $voTree$. For each encrypted element $[\![e_i]\!]$ in $\mathcal{R}$ and $voTree$, the DSP chooses a random number r_i and calculates $v_i = [\![r_i]\!] \times [\![e_i]\!]$, where $r_i \in \mathbb{Z}_N^*$ and is generated by f. Then, the random vector $\boldsymbol{r}$ composed of r_i is sent to client and $\boldsymbol{\nu}$ is sent to DAP. Next, DAP decrypts $\boldsymbol{\nu}$ as $\boldsymbol{\eta} = D_{sk}(\boldsymbol{\nu})$ and sends $\boldsymbol{\eta}$ to the client. Finally, the client calculates the result as $\boldsymbol{s} = \boldsymbol{\eta} - \boldsymbol{r}$.

Verification Processing. To verify the soundness and completeness of skyline results, the client should check three aspects as follows: *i*) any two objects in $\mathcal{R}$ are not dominated by each object; *ii*) no skyline point is tampered; *iii*) no valid skyline point is missed. For the first aspect, it is easy to determine objects in $\mathcal{R}$ are dominated by each other because the result set is plaintext. After that, the client calculates the signature $\widehat{\mathcal{S}_i} = Sig(Hash(Hash(\mathcal{P}_i)|Hash(Enc(\mathcal{P}_i))))$. Next, if calculated signature $\widehat{\mathcal{S}_i}$ matches $\mathcal{S}_i$ from $voTree$, it means the skyline point is not tampered, and vice versa. Furthermore, if the signature can be matched, it indicates that $\mathcal{P}_i$ and $Enc(\mathcal{P}_i)$ are related, that is, $Enc(\mathcal{P}_i)$ denotes that $\mathcal{P}_i$ is encrypted by the ORE algorithm, which provides a basic guarantee for subsequent completeness verification. For the last aspect, the client checks the dominance relationship according to $voTree$ from bottom to top, and calculates the signature $\widehat{Sig}(\mathcal{H}_{root})$ of the root node of $voTree$. Finally, the client compares whether $\widehat{Sig}(\mathcal{H}_{root})$ is consistent with the signature $Sig(\mathcal{H}_{root})$ returned from DSP.

5 IMPROVED METHOD

5.1 Secure and Verifiable Scope R-tree

Although BVLSQ securely calculates the skyline results over SVR-tree on the fly, it still has the following drawbacks.

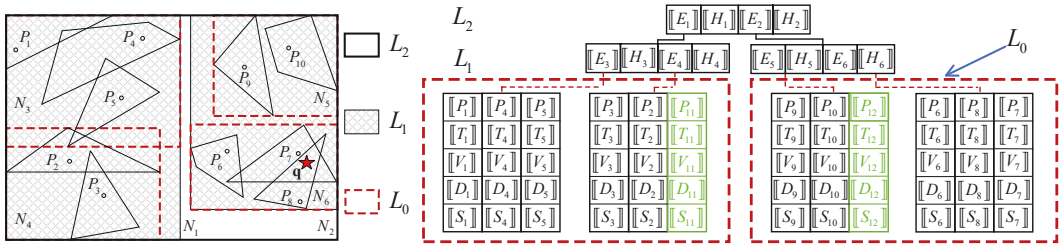
- The VO size is too large due to the ciphertext of ORE, which becomes a huge burden for communication bandwidth.
- The VO exposes the ordering of partial original data to clients due to the use of ORE encryption.
- Since some skyline points for static datasets can be pre-computed, query and verification time can be further reduced.

To overcome these drawbacks, a notion of *skyline scope* is presented (see Def. 7). Intuitively, the skyline scope of $\mathcal{P}_i$ indicates that if $\mathcal{P}_i$ is a skyline point of query $\boldsymbol{q}$, then $\boldsymbol{q}$ should be in the area defined by its skyline scope. In other words, if no object can non-spatially dominate $\mathcal{P}_i$ (i.e., $Dom(\mathcal{P}_i) = \emptyset$), $\mathcal{P}_i$ is one of the skyline points for any query $\boldsymbol{q}$, we call them non-spatial skyline points. Therefore, the data owner pre-computes and signs them ahead, which can be omitted in the following discussion. Otherwise, if $\mathcal{P}_i$ is not farther from $\boldsymbol{q}$ than any of its non-spatial dominators, then it is one of the skyline points w.r.t. $\boldsymbol{q}$. That is, the skyline scope can be calculated as a Voronoi cell of $\mathcal{P}_i$ under the set $\{\mathcal{P}_i \cup Dom(\mathcal{P}_i)\}$.

Definition 7. *(Skyline Scope) For a point $\mathcal{P}_i \in P$, we denote the skyline scope of $\mathcal{P}_i$ by $SS(\mathcal{P}_i) = \{\boldsymbol{q} | \boldsymbol{q} \in \mathbb{A} \wedge \mathcal{P}_i \in SVLSQ(P, \boldsymbol{q})\}$, where $\mathbb{A}$ is a two-dimensional plane.*

Intuitively, the naive method is to pre-compute the skyline scope and its encryption. However, such an approach is difficult to promptly construct the verification object VO, and we need to traverse all skyline scopes to search for points that meet query conditions. To this end, we devise an effective data structure to achieve the purpose of speeding up query and verification. Simultaneously, the privacy of datasets, queries, results and access patterns is guaranteed during the processes of query and verification. Therefore, the primary technical challenge is to build a secure and effective ADS.

According to the above analysis, we propose a secure and verifiable scope R-tree (SVSR-tree) index as shown in Fig. 5. The skyline scopes of all points in SVSR-tree are structured in the form of SR-tree, where skyline scopes are encrypted by Paillier algorithm. To benefit the result verification, along with the encrypted skyline scopes, some auxiliary verification information is also recorded in the leaf nodes. Concretely, each object in a leaf node stores 1) an encrypted point $[\![\mathcal{P}_i]\!]$; 2) the point's skyline scope expressed as an encrypted tuple set $[\![\mathcal{T}_i]\!] = ([\![\mathbf{t}_1]\!], ..., [\![\mathbf{t}_\kappa]\!])$, where $[\![\mathbf{t}_\kappa]\!] \in [\![\mathcal{T}_i]\!]$ is an encrypted vertex and κ is the number of vertices of the polygon formed by its skyline scope; 3) the encrypted approximate polygon $[\![\mathcal{V}_i]\!] = ([\![\mathbf{v}_1]\!], ..., [\![\mathbf{v}_\phi]\!])$ that roughly



(a) Spatial information

(b) Secure and verifiable scope R-tree hierarchy

Figure 5: An example of SVSR-tree index with node capacity $c = 3$. (a) visualizes the spatial information. (b) shows a secure and verifiable scope R-tree hierarchy, where each leaf node has 3 objects (the objects with green are padding ones) and each object contains $< [\![\mathcal{P}_i]\!], [\![\mathcal{T}_i]\!], [\![\mathcal{V}_i]\!], [\![\mathcal{D}_i]\!], [\![\mathcal{S}_i]\!] >$. Each object in the non-leaf node contains $< [\![E_i]\!], [\![\mathcal{H}_i]\!], pointer/\mathcal{B}(\cdot) >$.

covers the skyline scope from the inside, where $[\![\mathbf{v}_\phi]\!]$ is the vertex of $[\![\mathcal{V}_i]\!]$ and $\phi > \kappa$; 4) the encrypted object's digest $[\![\mathcal{D}_i]\!] = [\![Hash(Hash(\mathbf{v}_1)|...|Hash(\mathbf{v}_\phi))]\!]$; and 5) the encrypted object's digest signature $[\![\mathcal{S}_i]\!] = [\![Sig(Hash(Hash(\mathcal{P}_i)|\mathcal{D}_i))]\!]$, where "|" is a concatenation operator. Since VO constructed based on $\mathcal{T}_i$ may expose the privacy of datasets during the result verification, we set an extra component $\mathcal{V}_i$, an approximate polygon inside $\mathcal{T}_i$, whose vertices are not consistent with those of $\mathcal{T}_i$. Meanwhile, the client only needs to determine $\boldsymbol{q}$ does not fall in $\mathcal{V}_i$ of the non-result point. We use some simple approximate methods construct $\mathcal{V}_i$, such as selecting the point on the line segment A_1A_2 of $\mathcal{V}_i$ close to vertex A_1 or the point on the diagonal A_2A_5 of $\mathcal{V}_i$ close to vertex A_2.

For the index object, it consists of an encrypted MBR $[\![E_i]\!] = ([\![\mathbf{n}_1]\!], ..., [\![\mathbf{n}_4]\!])$ ($\mathbf{n}_j$ is the vertex of MBR), an encrypted digest $[\![\mathcal{H}_i]\!]$ of the child node, and a pointer to the node N_i at the lower level or a function $\mathcal{B}(\cdot)$ for calculating the leaf node. We calculate the encrypted digest of the object as:

$$[\![\mathcal{H}_i]\!] = [\![Hash(E_{c_1}|\mathcal{H}_{c_1}|...|E_{c_c}|\mathcal{H}_{c_c})]\!], \tag{10}$$

where E_{c_i} denotes the i-th object in N_i, and $\mathcal{H}_{c_i}$ denotes the digest of the corresponding object. Significantly, in an index object, $E_{c_i} = (\mathbf{n}_1, ..., \mathbf{n}_4)$, while in a data object, $E_{c_i} = \mathcal{V}_i$ and $\mathcal{H}_{c_i} = \mathcal{D}_i$. Therefore, the digest of each object is calculated recursively through a bottom-up way. Finally, the root hash is denoted as $\mathcal{H}_{root}$ and its digital signature $Sig(\mathcal{H}_{root})$ is generated by the data owner. Note that SVSR-tree except the encrypted vector IV is uploaded to DSP, while the additional encrypted information Tag composed of IV and the corresponding object's digest is uploaded to DAP.

Although the access path is hidden between the leaf node and its parent node, DSP obtains leaf nodes by Eq. 5 and Eq. 6. Since the data object is 'read' from NB blindly, its exact location avoids being exposed to the cloud. Meanwhile, to avoid suffering inference attacks, we consider two aspects: 1) how to hide the number of vertices in the skyline scope and approximate polygon; and 2) how to hide the number of data objects in the leaf node. For the first aspect, we set a fixed number of vertices. That is, we fill the skyline scope and approximate polygon lacking sufficient tuples by using their own vertices clockwise and circularly. For the other aspect, the data owner generates some crafted data objects with virtual elements and pads them into the leaf node lacking sufficient objects. To ensure that each query is not in skyline scopes of crafted objects, skyline scopes are devised to overlap two segments. By randomly adding some noise objects, the number of objects in the leaf nodes is consistent and at least one inveracious object is included, which further protects the query unlinkability. Moreover, to meet the fixed number of vertices, we also use the vertices of the segments clockwise and circularly to fill skyline scopes.

5.2 SVLSQ Protocol

Based on SVSR-tree, the SVLSQ protocol is simplified to determine whether the encrypted query point is in the area of the skyline scope. To protect the privacy of queries and datasets , the vertices that make up the scope are encrypted and the query point is also encrypted. Therefore, it is a huge challenge for the cloud server to calculate the results and construct the VO without decrypting them. Specifically, the challenges include the following two aspects: 1) how to build the verification object VO; 2) how to securely determine whether $\boldsymbol{q}$ is located in MBRs or skyline scopes.

To address the first challenge, VOs are indexed in the form of a tree. Hereafter, this new structure is called $voTree$, which includes the following elements: 1) an encrypted approximate polygon $[\![\mathcal{V}_i]\!]$ and an encrypted digest $[\![\mathcal{D}_i]\!]$ of the non-result object; 2) an encrypted digest $[\![\mathcal{D}_i]\!]$ and a corresponding signature $[\![\mathcal{S}_i]\!]$ of the skyline object; 3) two encrypted MBR $[\![E_i]\!]$ and digest $[\![\mathcal{H}_i]\!]$ of the traversed index object (including the pruned object). The reason why $[\![E_i]\!]$ of the pruned index object and $[\![\mathcal{V}_i]\!]$ of the non-result object are involved is for the client to easily verify whether the query is located in the corresponding MBR or skyline scope.

Lemma 1. *Given a convex polygon Ω, for each edge l, the interior points in Ω are all on the same side of the line that the edge l defines.*

To address the second challenge, we propose a secure query positioning (SQP) algorithm, the underlying idea of which comes from Lemma 1 [35]. We observe that the convex polygon vertices form an array of vertices in clockwise order, with two vertices in turn forming a sequence of segments. If the point falls inside the convex polygon, the point must be on the same side of all segment sequences. Thus, in three-dimensional space, point A and B are on the same plane, with A as the starting point and B as the end point to form the vector $\overrightarrow{AB} = (x_1, y_1, 0)$. In order to determine the position of point C relative to $\overrightarrow{AB}$, we calculate the cross product of $\overrightarrow{AB}$ and $\overrightarrow{AC}$ is $(x_1 \cdot y_2 - x_2 \cdot y_1)\vec{k}$, where the vector $\overrightarrow{AC} = (x_2, y_2, 0)$. Therefore, DSP calculates the encrypted location relationship $[\![\gamma]\!] = [\![x_1 \cdot y_2 - x_2 \cdot y_1]\!]$ using the SM protocol [11] based on the additive

Algorithm 1: SVLSQ Protocol

Input: DSP has the root of SVSR-tree $sRoot$ and $[\![q]\!]$. DAP has sk.
Output: DSP ← skyline set $\mathcal{R}$, VOs $voTree$.

1 initialize $voTree$ with $sRoot$; insert $sRoot$ into a queue Q;
2 **if** *there is no object in* Q **then**
3 return $\mathcal{R}$ and $voTree$;
4 pop up the top encrypted object o from Q;
5 **if** o *has the blind child object* **then**
6 request IVs = $\{o.IV_1, ..., o.IV_\xi\}$, IVs$'$ = $\{o.IV_1, ..., o.IV_{\theta-\xi}\}$ from DAP based on whether $[\![q]\!]$ is located in MBRs of o's children $\{o_1, .., o_\theta\}$;
7 **for** $i = 1$ *to* $size$(IVs) **do**
8 $o_i = \prod_{j=1}^{\theta} \mathrm{SM}(o.IV_i[j], o_j)$, where o_j is from $\{o_1, .., o_\theta\}$;
9 insert o_i into $voTree$;
10 $\{o_1, .., o_c\} = \mathcal{B}(o_i.IV, o_i.NB)$; request $\{M, M'\}$ from DAP based on whether $[\![q]\!]$ is located in skyline scopes of $\{o_1, .., o_c\}$;
11 **for** $k = 1$ *to* $size(M)$ **do**
12 $[\![\mathcal{P}_k]\!] = \prod_{j=1}^{c} \mathrm{SM}(M_k[j], \mathcal{P}_j)$;
13 $\mathcal{R} = \mathcal{R} \cup [\![\mathcal{P}_k]\!]$, insert $\{[\![\mathcal{D}_k]\!], [\![\mathcal{S}_k]\!]\}$ into $voTree$;
14 **for** $k = 1$ *to* $size(M')$ **do**
15 $[\![\mathcal{P}_k]\!] = \prod_{j=1}^{c} \mathrm{SM}(M_k[j], \mathcal{P}_j)$;
16 discard $\mathcal{P}$ and insert $\{[\![\mathcal{V}_k]\!], [\![\mathcal{D}_k]\!]\}$ into $voTree$;
17 **for** $i = 1$ *to* $size$(IVs$'$) **do**
18 $o_i = \prod_{j=1}^{\theta} \mathrm{SM}(o.IV_i[j], o_j)$; insert $\{[\![E_i]\!], [\![\mathcal{H}_i]\!]\}$ into $voTree$;
19 **else**
20 **for** *each child* o_i *of* o **do**
21 **if** SQP(o_i, $[\![q]\!]$) **then**
22 insert o_i into Q and $voTree$;
23 **else**
24 discard o_i and insert $\{[\![E_i]\!], [\![\mathcal{H}_i]\!]\}$ into $voTree$;
25 GOTO Line 2;

homomorphic property. Then, DSP permutes $\psi' = \pi_1([\![\gamma]\!])$ using random permutation function π_1. Next, DSP sends ψ' to DAP. Upon receiving the value, DAP decrypts ψ' and obtains ψ. Next, if each ψ_i is not greater than 0 or not less than 0, then DAP returns result $\delta = 1$ to DSP, otherwise, DAP returns $\delta = 0$ to DSP.

As shown in Alg. 1, VO and skyline results are respectively stored in $voTree$ and $\mathcal{R}$. The $voTree$ is initialized with the root $sRoot$ of SVSR-tree. Then, $sRoot$ is added into the queue Q. Subsequently, the objects in SVSR-tree are traversed with queue Q. Concretely, DSP pops up the encrypted object o from Q. If o has blind child objects $\{o_1, .., o_\theta\}$, DSP needs to confuse the order of them. Thereupon, DSP calculates an encrypted location relationship (i.e., $[\![\gamma]\!]$ in SQP) about whether $[\![q]\!]$ falls in the MBR of $o_j \in \{o_1, .., o_\theta\}$ and sends it to DAP. Next, by decrypting $[\![\gamma]\!]$, DAP can determine whether $[\![q]\!]$ is located in the MBR of o_j. If so, DAP generates an encrypted vector IV_i, where $IV_i[j] = [\![1]\!]$ and the remaining dimensions of IV_i are $[\![0]\!]$. Note that $IV_i[j] = [\![1]\!]$ indicates the j-th blind object covers $[\![q]\!]$. Then, DAP assembles all IV_i into matrix IVs, which is used to determine blind objects that cover $[\![q]\!]$. Similarly, DAP also generates a matrix IVs', which is used to determine blind objects that do not cover $[\![q]\!]$. To obtain this blind object, SM is employed to calculate an encrypted product between $IV_i[j]$ and o_j. Since all other objects except the one covers $[\![q]\!]$ will be $[\![0]\!]$, DSP sums all encrypted products to obtain o_i. Thereupon, o_i is added to $voTree$ (Lines 8-9). Similarly, DSP can obtain the blind object that does not cover $[\![q]\!]$ and insert $\{[\![E_i]\!], [\![\mathcal{H}_i]\!]\}$ into $voTree$ (Lines 18). It is noteworthy that the order of blind objects is random due to IVs and IVs', so the access path between the blind object and its child node of two queries is indistinguishable for DSP.

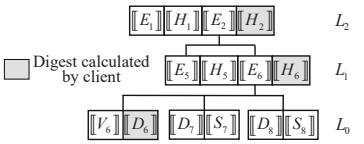


Figure 6: Verifiable object $voTree$

Subsequently, DSP needs to blindly "read" the results and non-results from $\{o_1, ..., o_c\}$ (i.e., $\{\mathcal{P}_1, ..., \mathcal{P}_c\}$ that can be calculated by $\mathcal{B}(\cdot)$ with the input $o_i.IV$ from DAP, where $o_i.IV$ is obtained according to the digest $\mathcal{H}_i$). Next, DSP provides encrypted location relationship (i.e., $[\![\gamma]\!]$ in SQP) about whether $[\![q]\!]$ falls in the child object's skyline scope to DAP. By decrypting $[\![\gamma]\!]$, DAP can determine whether $[\![q]\!]$ is located in the skyline scope of $o_j \in \{o_1, ..., o_c\}$. If so, DAP generates an encrypted vector LI, where $LI[j] = [\![1]\!]$ and the remaining dimensions of LI are $[\![0]\!]$. Otherwise, if $[\![q]\!]$ is not located in o_j's skyline scope, DAP generates an encrypted vector NLI, where $NLI[j] = [\![1]\!]$ and the remaining dimensions of NLI are $[\![0]\!]$. DAP assembles LI and NLI into matrices M and M', respectively. Subsequently, DSP obtains a vector M_k, where $M_k[j] = [\![1]\!]$ indicates that the j-th object is a result. To achieve this result, SM is employed to calculate an encrypted product between $M_k[j]$ and $\mathcal{P}_j$. Since all other objects except the one that is a skyline object will be $[\![0]\!]$, DSP sums all encrypted products to obtain a skyline point $[\![\mathcal{P}_k]\!]$. Thereupon, $[\![\mathcal{P}_k]\!]$ is added to the result $\mathcal{R}$ and $\{[\![\mathcal{D}_i]\!], [\![\mathcal{S}_i]\!]\}$ is inserted into $voTree$. Similarly, as for the non-result object, $[\![\mathcal{V}_i]\!]$ and $[\![\mathcal{D}_i]\!]$ are inserted into $voTree$. If o is an index object without blind child objects, DSP and DAP cooperatively use the SQP algorithm to examine whether $[\![q]\!]$ falls in MBRs of o's children. When accessing a child object o_i of o, if its MBR covers $[\![q]\!]$, o_i is added into Q for expansion; otherwise, o_i is discarded. Simultaneously, DSP inserts $\{[\![E_i]\!], [\![\mathcal{H}_i]\!]\}$ into $voTree$. The algorithm continues until $Q = \emptyset$.

As illustrated in Figs. 5(a) and 5(b), we take an example to show how SVLSQ works with node capacity $c = 3$ and $[\![q]\!]$. Assume that $[\![q]\!]$ is covered by $[\![\mathcal{P}_7]\!]$ and $[\![\mathcal{P}_8]\!]$. The root node is visited first, followed by its children objects o_1 and o_2. Since o_2 covers $[\![q]\!]$ while o_1 does not, o_2 is inserted into Q and $voTree$, while $\{[\![E_1]\!], [\![\mathcal{H}_1]\!]\}$ is inserted into $voTree$. Next, o_2 is popped up from Q. As o_2 has blind children objects, the DSP calculates his children objects o_5 and o_6. Therefore, the DAP generates $IVs = [[[\![0]\!][\![1]\!]]^\mathsf{T}]^\mathsf{T}$ and $IVs' = [[[\![1]\!][\![0]\!]]^\mathsf{T}]^\mathsf{T}$. The DSP blindly obtains o_6 that is inserted into $voTree$ according to IVs, while the DSP blindly obtains o_5 whose $\{[\![E_5]\!], [\![\mathcal{H}_5]\!]\}$ is into $voTree$ according to IVs'. After that, the DSP calculates o_6's children $[\![\mathcal{P}_6]\!]$, $[\![\mathcal{P}_7]\!]$ and $[\![\mathcal{P}_8]\!]$. Since $[\![\mathcal{P}_7]\!]$ and $[\![\mathcal{P}_8]\!]$ cover $[\![\mathbf{q}]\!]$ while $[\![\mathcal{P}_6]\!]$ does not, the DAP only generates $M = [[[\![0]\!][\![1]\!][[\![0]\!]]^\mathsf{T}, [[\![0]\!][\![0]\!][[\![1]\!]]^\mathsf{T}]^\mathsf{T}$ and $M' = [[[\![1]\!][\![0]\!][[\![0]\!]]^\mathsf{T}]^\mathsf{T}$ Thus, $[\![\mathcal{P}_7]\!]$ and $[\![\mathcal{P}_8]\!]$ are the final skyline results. Moreover, the DSP inserts $\{[\![\mathcal{D}_7]\!], [\![\mathcal{S}_7]\!]\}$ and $\{[\![\mathcal{D}_8]\!], [\![\mathcal{S}_8]\!]\}$ into $voTree$, whereas $\{[\![\mathcal{V}_6]\!], [\![\mathcal{D}_6]\!]\}$ is inserted into $voTree$. As shown in Fig. 6, it is a verifiable object $votree$, where $[\![\mathcal{H}_2]\!]$, $[\![\mathcal{H}_5]\!]$ and $[\![\mathcal{D}_6]\!]$ are omitted to reduce the communication overhead.

Returning $\mathcal{R}$ and $voTree$. The return method is the same as BVLSQ, so we omitted here for space limitation.

Verification Processing. The verification aspects to be examined by the client are the same as BVLSQ. For the first aspect, it is

easy to check the objects in $\mathcal{R}$ are dominated by each other under the plaintext. Next, the client examines the second aspect using the signature of each object in $\mathcal{R}$. Concretely, the client knows the skyline point $(\mathcal{P}_i, \mathcal{D}_i)$. Therefore, it can calculate the signature $\widehat{\mathcal{S}_i} = Sig(Hash(Hash(\mathcal{P}_i)|\mathcal{D}_i))$. Next, if the calculated signature $\widehat{\mathcal{S}_i}$ matches $\mathcal{S}_i$ from $voTree$, it means the skyline point is not tampered, and vice versa. After that, the client checks whether approximate polygons of the non-results and MBRs of the pruned index objects cover $\boldsymbol{q}$. Note that the approximate polygon will not expose the content of original data, because its vertices are different from those of the skyline scope. To maintain platform reputation and revenue, DSP avoids discarding or tampering with results since the values of approximate polygons are not known by DSP and this malicious behavior will be detected with high probability. Simultaneously, the client compares whether the calculated signature $\widehat{Sig}(\mathcal{H}_{root})$ is consistent with $Sig(\mathcal{H}_{root})$ returned from DSP.

Discussion. Since SVSR-tree builds an index by pre-computing skyline scopes, the query efficiency of SVLSQ is more efficient than that of BVLSQ (our baseline), and SVLSQ is adept at constructing VOs for smaller sizes. Furthermore, with a simple data partitioning, we implement a parallel version (see Section 7.3) for SVLSQ whereas this method does not work for BVLSQ. However, compared to SVR-tree, SVSR-tree takes more time to be constructed. Meanwhile, since SVR-tree uses intervals to describe the extent of each point along all dimensions, BVLSQ can be well extended to general skyline queries such as dynamic skyline queries [25] and reverse skyline queries [10]. On the contrast, SVSR-tree utilizes the nature of location-based data, which is not suitable for general skyline queries.

6 COMPLEXITY AND SECURITY ANALYSIS

6.1 Complexity Analysis

We exhibit the computational load of SVLSQ as follows. For SQP, it requires 6κ encryptions and 4κ decryptions to obtain the encrypted position relationship $[\![\gamma]\!]$. It also requires κ decryptions to obtain the value ψ. Hence, it leads to 6κ encryptions and 5κ decryptions in total. Then, suppose that n denotes the number of data objects and λ_s is the number of skyline results without considering non-spatial skyline points. Since the height of SVSR-tree is at most $\lceil \log_c n \rceil - 1$, SVLSQ approximately requires $O(\lambda_s \kappa \log_c n)$ encryptions and $O(\lambda_s \kappa \log_c n)$ decryptions. As for BVLSQ, $[\![mindist^2(\cdot)]\!]$ requires $3d$ encryptions and d decryptions for one object. Thereupon BVLSQ approximately requires $O(dn)$ encryptions and $O(dn)$ decryptions. Clearly, as n increases, SVLSQ is more efficient than BVLSQ and the efficiency gap becomes more pronounced. Only when n is small enough ($n < 1000, c = 3, d = 4$), the efficiency of BVLSQ will be close to or even better than that of SVLSQ.

For communication complexity, it takes 3 $\|N\|$ bits to execute SM, where $\|N\|$ denotes the bitsize of N. Hence, it takes $3\kappa\,\|N\|+1$ bits to run SQP. Based on the above assumption, it takes $O(\lambda_s \kappa \log_c n\,\|N\|)$ bits for SVLSQ. Similarly, it takes $O(dn\,\|N\|)$ bits for BVLSQ.

6.2 Security Analysis

First, we give the following formal definition of the leakage collection $\mathcal{L}(P, \boldsymbol{q}) = (\mathsf{dim}, \mathsf{num})$.

• Dimension of a point (dim). $\mathsf{dim} = |\mathcal{P}_i|$, where $\mathcal{P}_i \in P$ and $|x|$ is denoted as the size of x.

• Numbers of elements in ψ that are greater than 0, less than 0, and equal to 0 (num). Let $X_1(\psi_i) = \begin{cases} 1, & \psi_i \geq 0 \\ 0, & \psi_i < 0 \end{cases}$, $X_2(\psi_i) = \begin{cases} 1, & \psi_i \leq 0 \\ 0, & \psi_i > 0 \end{cases}$, $X_3(\psi_i) = \begin{cases} 1, & \psi_i = 0 \\ 0, & \psi_i \neq 0 \end{cases}$. Then, num is an array such that for $1 \leq j \leq 3$, $\mathsf{num}[j] = \sum_{1 \leq i \leq \kappa} X_j(\psi_i)$.

Then, the security of SVLSQ is formally defined by the standard simulation-based model that has been proverbially used in secure computation [9, 22, 23].

Definition 8. *(Security) Let λ be a security parameter and Π be the SVLSQ protocol. Let $\mathcal{S}$ be a simulator and $\mathcal{L}$ be the leakage collection. We define two games $\mathsf{Real}_{\mathcal{A}}$ and $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}$ as follows.*

- $\mathsf{Real}_{\mathcal{A}}(\lambda)$*: $\mathcal{A}$ chooses a dataset P. The game generates key pairs (sk, pk) of Paillier. Then it calculates and gives the secure index I to $\mathcal{A}$. After that, $\mathcal{A}$ outputs a polynomial number of queries: $Qs = \{\boldsymbol{q}_1, ..., \boldsymbol{q}_q\}$. Next, the game generates an encrypted query $\boldsymbol{\tau}_i = [\![\boldsymbol{q}_i]\!]$ and gives it to $\mathcal{A}$. Next, $\mathcal{A}$ takes I as input to get the encrypted results by running Π. Finally, $\mathcal{A}$ returns $b \in \{0, 1\}$ which is output by the game.*
- $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$*: Given $\mathcal{L}$, the index I^* is simulated by $\mathcal{S}$. Subsequently, I^* is sent to $\mathcal{A}$. Then, $\mathcal{A}$ outputs a polynomial number of queries: $Qs = \{\boldsymbol{q}_1, ..., \boldsymbol{q}_q\}$. $\mathcal{S}$ simulates an encrypted query $\boldsymbol{\tau}_i^*$ for each query $\boldsymbol{q}_i \in Qs$ and sends it to $\mathcal{A}$. After that, $\mathcal{A}$ obtains the encrypted results by running Π. Finally, $\mathcal{A}$ returns $b \in \{0, 1\}$ which is output by the game.*

Π is $\mathcal{L}$-secure if for all probabilistic polynomial time (PPT) adversaries $\mathcal{A}$, there exists a PPT simulator $\mathcal{S}$ such that

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

where $negl(\lambda)$ denotes a negligible function.

With the above definition, we give the following theorems.

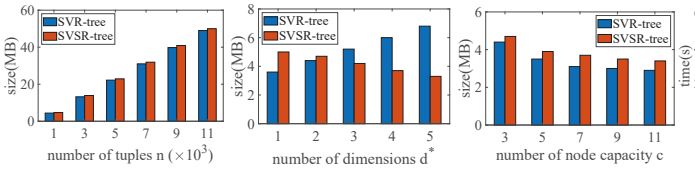
Theorem 1. *The SVLSQ protocol is $\mathcal{L}$-secure if f and π_1 are pseudo-random, as well as Paillier is semantically secure.*

Proof. We build a polynomial-time simulator $\mathcal{S}$ such that for any probability polynomial time (PPT) adversary $\mathcal{A}$, the outputs of $\mathsf{Real}_{\mathcal{A}}(\lambda)$ and $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$ are computationally indistinguishable. Hence, we describe the real view $\mathsf{Real}_{\mathcal{A}}(\lambda)$ and the simulated view $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$ as follows.

In the game $\mathsf{Real}_{\mathcal{A}}(\lambda)$, given the inputs I and $[\![\boldsymbol{q}_i]\!]$, the DSP determines which skyline scopes cover $[\![\boldsymbol{q}_i]\!]$. Then, the encrypted points 'read' from NB blindly are added to the results. Finally, the DSP outputs the results by the experiment.

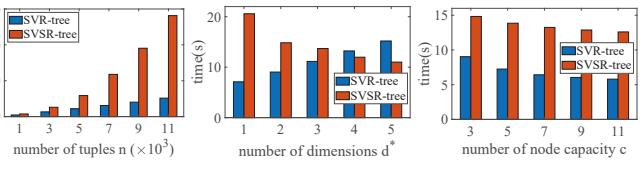
In the game $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$, given dim and num, $\mathcal{S}$ simulates the input I^* and $[\![\boldsymbol{q}_i^*]\!]$ such that for each object $o^* \in I^*$, $|o^*| = \mathsf{dim}$ and $\sum_{1 \leq i \leq \kappa} X_j(\psi_i^*) = \mathsf{num}[j]$ with $1 \leq j \leq 3$, where ψ^* is the intermediate result w.r.t. $[\![\boldsymbol{q}_i^*]\!]$ and o^*. By doing so, $\mathcal{S}$ builds the simulated input I^* and $[\![\boldsymbol{q}_i^*]\!]$. Then, it runs the SVLSQ protocol to output the results by the experiment.

Based on the simulator $\mathcal{S}$, no PPT adversary can distinguish the output of $\mathsf{Ideal}_{\mathcal{A},\mathcal{S}}(\lambda)$ from that of $\mathsf{Real}_{\mathcal{A}}(\lambda)$ since the outputs of them are encrypted by Paillier that is semantically secure. Meanwhile, since f and π_1 are pseudo-random, intermediate results in SVLSQ are obscured and indistinguishable to $\mathcal{A}$. Due to the semi-blind structure, $\mathcal{A}$ cannot track the position of data points based on



(a) $d^* = 2, c = 3$ (b) $n = 1000, c = 3$ (c) $n = 1000, d^* = 2$

Figure 7: Memory consumption cost



(a) $d^* = 2, c = 3$ (b) $n = 1000, c = 3$ (c) $n = 1000, d^* = 2$

Figure 8: Index construction time

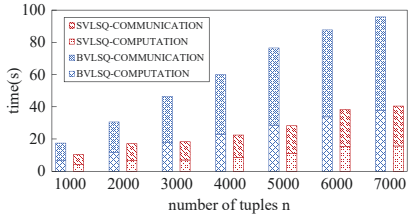


Figure 9: Total time overhead of skyline query ($d^* = 2$, $c = 3$)

the access path, which protects the indirect privacy. Since verification objects do not contain contents of the original data, the client only knows the results during the verification procedure. Overall, $\mathcal{A}$ cannot distinguish views in games Real and Ideal. □

Theorem 2. *If the signature $\widehat{\mathcal{S}}$ rebuilt from each object in $\mathcal{R}$ can match the signature $\mathcal{S}$ from VO, the results of SVLSQ are sound; if the query $\boldsymbol{q}$ is outside the range of skyline scopes of all non-result objects, the results of SVLSQ are complete.*

Proof. Since $Sig(\cdot)$ can resist forged data, the soundness of the results is effectively guaranteed by signature matching. For the completeness, $Sig(\mathcal{H}_{root})$ guarantees that all non-result objects are not discarded and not tampered with. Therefore, examining whether the skyline scopes of non-result objects cover $\boldsymbol{q}$ guarantees that the results are complete. These two scenarios are easy to demonstrate based on VO (details see Verification Processing). Therefore, the results of SVLSQ are sound and complete. □

7 PERFORMANCE EVALUATION

7.1 Experiment Setup

Our protocols are implemented with Java (JDK 1.7) on a PC (Intel Core I5-8400 2.8GHz, Windows 10). We also implemented the communication using sockets, so it can be run on two machines without modification to evaluate the communication overhead. To highlight the efficiency, we use BVLSQ as a baseline. Moreover, we compare our protocols with another two solutions, namely basic secure skyline computation (BSSP) and fully secure skyline computation (FSSP) [24, 25], which are the only two methods for location-based skyline queries without result verification.

Datasets. We produced correlated (CORR), independent (INDE), and anti-correlated (ANTI) datasets similar to work [24][2]. Furthermore, we adopted the real-world dataset about hotels (HOTE) from the version of Kaggle[3]. We extracted the latitude and longitude attributes as spatial dimensions, and the remaining 5 attributes such as Hospitality, Facilities, Cleanliness, Value for Money and Food as non-spatial dimensions.

[2]All dimensions satisfy the distribution corresponding to this dataset.
[3]https://www.kaggle.com/PromptCloudHQ/hotels-on-makemytrip

Table 2: Index building on three datasets ($n = 1000$, $d = 2$, $c = 3$)

Index	datasets	CORR	INDE	ANTI
R-tree	s_1 (KB)	472.1	470.4	472.8
	s_2 (ms)	9	7	29
SVR-tree	s_1 (MB)	4.46	4.45	4.47
	s_2 (s)	9.01	8.99	9.03
SVSR-tree	s_1 (MB)	4.85	4.79	4.72
	s_2 (s)	18.27	17.02	14.83
	s_3	321332	243525	103402

Table 3: Total number of non-spatial dominators on ANTI

$d^*(n = 1000, c = 3)$	$d^* = 1$	$d^* = 2$	$d^* = 3$	$d^* = 4$	$d^* = 5$
s_3 **of SVSR-tree**	499500	103402	39889	17876	11915

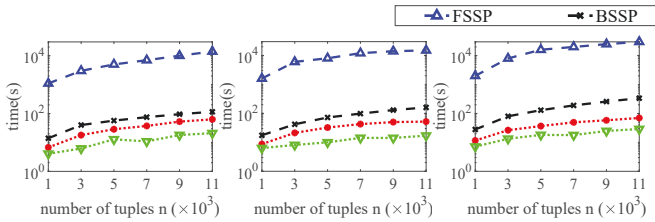
Parameter Settings. We evaluate the efficiency of the algorithms through varying the number n of tuples from 1000 to 11000, the non-spatial dimension d^* of the point from 1 to 5 and node capacity c from 3 to 11. We also fix the key bitsize of Paillier and noise as 512. Note that, we randomly choose the query points and report the performance by averaging the values of repeated measurements. Meanwhile, the computation time refers to the total computation time of DSP and DAP for one query.

7.2 Secure and Verifiable Index Construction

SVR-tree is constructed based on R-tree [12]. Apart from encrypting data and adding verification information, the DO structures leaf nodes belonging to the same parent node into a node bucket NB that is pointed by a pointer from an object at a higher level, and construct its corresponding encrypted vector IV. As for SVSR-tree, the DO first calculates $Dom(\mathcal{P}_j)$ by traversing the whole dataset P for $\mathcal{P}_i \in P$. Then, a Voronoi cell of each object $\mathcal{P}_i$ under the set $\{\mathcal{P}_i \cup Dom(\mathcal{P}_i)\}$ is calculated using a branch and bound approach from [3, 13]. Subsequently, SVSR-tree is constructed based on the idea of SR-tree.

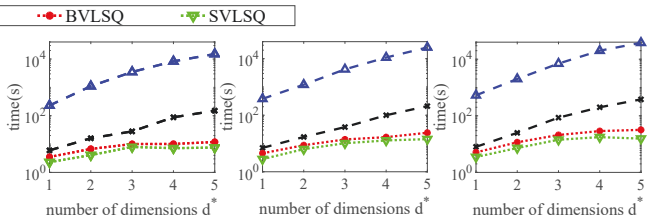
As Table 2 shows, s_1 represents the memory consumption cost of the index, s_2 represents the construction time of the index, and s_3 represents the total number of non-spatial dominators of all objects. We observe that for SVR-tree, the impact of data distribution on index construction is extremely slight. However, constructing SVSR-tree on CORR (ANTI) takes the most (least) time, and it has the most (least) memory cost. This is because CORR has the greatest number (i.e., s_3) of non-spatial dominators, which increases the cost to construct the scopes. We also observe that the memory cost and construction time of our indexes are more than those of the traditional R-tree, which are mainly sacrificed for security and verification reasons. However, this is only a one-time cost.

Next, we choose the ANTI dataset to evaluate the impact of other parameters on memory consumption cost. From Figs. 7(a) and 7(c),



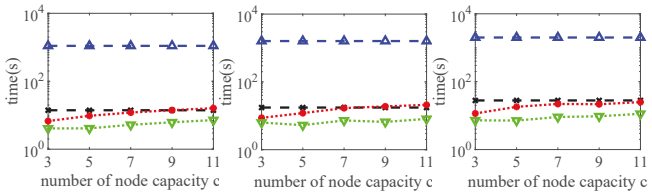
(a) CORR (b) INDE (c) ANTI

Figure 10: Query effect of n ($d^* = 2$ and $c = 3$)



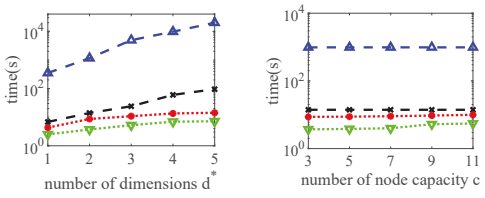
(a) CORR (b) INDE (c) ANTI

Figure 11: Query effect of d^* ($n = 1000$ and $c = 3$)



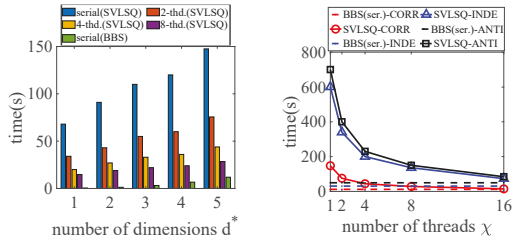
(a) CORR (b) INDE (c) ANTI

Figure 12: Query effect of c ($n = 1000$ and $d^* = 2$)



(a) $n = 1000, c = 3$ (b) $n = 1000, d^* = 2$

Figure 13: Query effect of d^* (or c) on HOTE



(a) perspective of d^* on CORR (b) perspective of $\chi(d^* = 5)$

Figure 14: Serial vs. multi-threading ($n = 100000, c = 3$)

SVSR-tree requires more memory than SVR-tree. This is because SVSR-tree stores additional information, such as skyline scopes and approximate polygons. Meanwhile, the memory cost of two secure indexes increases linearly with n increasing. However, as c increases, their memory cost decreases sub-linearly in Fig. 7(c). This is because as c continues to increase, both the height of the tree and the number of nodes will decrease. Along with varying d^*, Fig. 7(b) shows that when $d^* \geq 3$, SVSR-tree requires less memory cost than SVR-tree. This is mainly because SVSR-tree is built based on two spatial dimensions. Furthermore, as Table 3 shows, the memory cost of SVSR-tree decreases with d^* increasing, this is because the number of non-spatial dominators is gradually becomes small (thus constructing the scope is cheaper). As expected , we draw similar conclusions on construction time in Fig. 8.

7.3 Secure Query Processing

Efficiency of Communication. To reflect the real communication overhead, DSP and DAP were run on different machines. Fig. 9 shows the total time cost of computation and communication w.r.t. SVLSQ and BVLSQ under varying n on CORR. We observe that the communication time is slightly more than half of the total time.

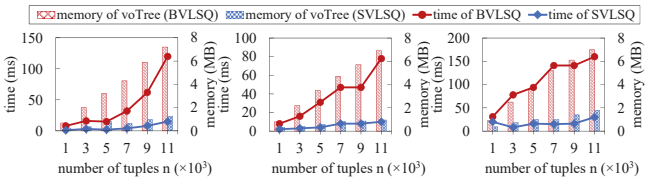
Efficiency of Computation. Along with respectively varying n, d^* and c, we evaluate the query efficiency of our protocols (including VO construction time) by comparing with BSSP and FSSP.

Fig. 10 illustrates the computation time cost (s) on different datasets by varying n. Our BVLSQ and SVLSQ are more secure than BSSP. Moreover, similar to the FSSP protocol, our protocols can protect indirect privacy. However, our protocols are more efficient than FSSP. This is because BVLSQ emloys the SVR-tree index, which reduces unnecessary computational overhead by secure pruning operations. Furthermore, SVLSQ is significantly more efficient than BVLSQ, the reason is that the computational overhead is reduced further by pre-computing datasets. Meanwhile, we also observe that as n increases, SVLSQ is very friendly to n, which is consistent with the computational complexity analysis.

Fig. 11 illustrates the computation time cost (s) on different datasets by varying d^*. We observe that BVLSQ is more efficient than BSSP. For SVLSQ, it is more efficient than BSSP and BVLSQ. Meanwhile, when $d^* > 5$, the efficiency gap between SVLSQ and BVLSQ becomes wider. This is because SVSR-tree of SVLSQ is only constructed on two-dimensional spatial attributes. From Fig. 13(a), we also observe that HOTE has a higher computation time cost than our protocols running on the CORR dataset. This is because HOTE shows weaker correlated relationships than CORR.

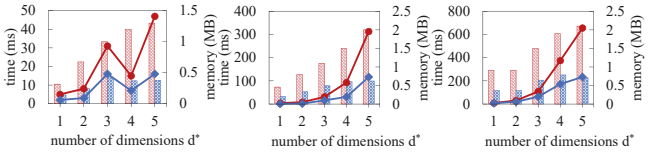
Fig. 12 illustrates the computation time cost (s) on different datasets by varying c. According to our observation, the computational time overhead increases with the number of node capacity c. The reason is that as the node capacity increases, the pruning effect of two tree indexes will become worse. Meanwhile, the efficiency of BVLSQ is also better than that of BSSP when $c \leq 7$, and SVLSQ has better query performance than other protocols. From Fig. 13(b), we draw a similar conclusion on the dataset LOCA.

Scalability. We also evaluate SVLSQ on larger datasets ($n = 100000$, up to 5M). Meanwhile, to further improve the performance, we implement a multi-threading version using the data partitioning on server (Intel Xeon E5-2670 2.6GHz, 16 threads). Specifically, we pre-compute non-dominators of each object on the entire dataset P. Then, we divide P into 10 sub-datasets to respectively build SVSR-trees. Each SVSR-tree is assigned to an idle thread whose results are skyline points without further computation. As shown in Fig. 14(a), as χ increases, the computation time cost drastically decreases with $n = 100000, c = 3$. Compared to BBS [30] that is IO optimal for skyline queries (without encryption) where the data is indexed by an R-tree, our serial version is only an order of magnitude slower (due to ciphertext calculation) when $d^* = 5$. Moreover, Fig. 14(b)



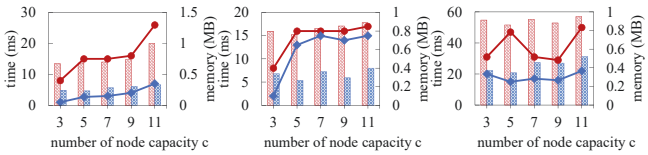
(a) CORR (b) INDE (c) ANTI

Figure 15: Verification effect of n ($d^* = 2$ and $c = 3$)



(a) CORR (b) INDE (c) ANTI

Figure 16: Verification effect of d^* ($n = 1000$ and $c = 3$)



(a) CORR (b) INDE (c) ANTI

Figure 17: Verification effect of c ($n = 1000$ and $d^* = 2$)

indicates the efficiency of our multi-threading version gradually approaches BBS with an increasing number of threads χ.

7.4 Result Verification

In Fig. 15, SVLSQ requires less memory cost for $voTree$ than BVLSQ since it provides less information for result verification. That is, SVLSQ does not involve non-spatial skyline points and its $voTree$ only includes 2-dimensional verification objects. For $voTree$ of BVLSQ, it includes d-dimensional verification objects. Meanwhile, SVLSQ takes less verification time than BVLSQ. As expected, the memory cost of $voTree$ in both of them increases with the number of n, and their verification time also increases roughly with the increase of n. From Fig. 16, we observe that the verification time of both BVLSQ and SVLSQ increases significantly, almost exponential, when d^* grows. The memory cost of $voTree$ in BVLSQ increases approximately linearly with the number of dimensions d^*. Inconsistent with the growth trend of memory cost of SVSR-tree, $voTree$ memory overhead of SVLSQ increases roughly with the number of d^*. The reason is that as d^* increases, more objects covering the query point are inserted into $voTree$, which takes up more memory cost. From Tabs. 4 and 5 (s_4 and s_5 separately represent verification time and memory cost for $voTree$), we observe similar conclusions on the real-world dataset LOCA. As illustrated in Fig. 17, the $voTree$ memory overhead and verification time of two protocols become roughly larger with the increase of c, which is reasonable since $voTree$ has more leaf nodes composed of dominated objects.

8 RELATED WORK

The skyline query algorithm named block nested loop first proposed by Börzsönyi *et al.* [2]. After that, a number of approaches have been demonstrated, such as the nearest neighbor (NN) [19], the sort filter skyline [8], the branch-and-bound skyline (BBS) [30], etc.

Table 4: Verification time/memory cost on HOTE ($d^* = 2$)

$n(c = 3)$	1000	1500	2000	2500	3000
BVLSQ, s_4(ms)/s_5(MB)	8/0.26	10/0.31	11/0.35	14/0.43	16/0.47
SVLSQ, s_4(ms)/s_5(MB)	2/0.20	4/0.29	5/0.30	7/0.36	6/0.38

Table 5: Verification time/memory cost on HOTE ($n = 1000$)

$d^*(c = 3)$	1	2	3	4	5
BVLSQ, s_4(ms)/s_5(MB)	6/0.13	8/0.26	26/0.45	45/0.63	56/0.73
SVLSQ, s_4(ms)/s_5(MB)	2/ 0.09	2/0.20	13/ 0.29	24/ 0.38	27/ 0.41

After that, a number of works begin to extend the skyline query, and integrate with location-based services. Considering the interaction of the spatial points with their dominance relationship, Huang *et al.* [16] proposed an efficient location-based skyline query protocol for moving clients. Zheng *et al.* [43] focused on the query processing and result verification of location-based skyline queries over static objects. Sharifzadeh *et al.* [33] exploited a variation of the spatial skyline queries issue from a geometric perspective. Lo *et al.* [28] focused on reducing communication cost and presented a method for authenticating spatial skyline queries. However, the privacy is generally understudied.

Considering data security, some secure skyline query protocols have recently proposed. Liu *et al.* [26] proposed a secure skyline query solution on multiple encrypted databases. Hua *et al.* [15] developed a secure skyline computation method to implement an online medical diagnosis. Then, Hua *et al.* [14] continued to study secure skyline queries on the data encrypted by Paillier cryptosystem and developed a medical primary diagnosis framework. Chen *et al.* [5] studies the problem about how to verify skyline query results, but this method exposed the data privacy. Liu *et al.* [25] proposed a fully secure skyline query protocol (FSSP) over encrypted data, which is not efficient because of expensive calculation. Wang *at al.* [36] utilized the ORE cryptography to devise a dynamic skyline computation framework. Zeighami *at al.* [42] focused on a secure and efficient approach to calculate dynamic skyline according to result materialization. Neither of these two works [36, 42] enables the distance calculation under ciphertext. Our solutions are inspired by these works, but address the problem of secure and verifiable location-based skyline queries.

9 CONCLUSION

We focus on the issue of secure and verifiable location-based skyline queries with the secure authenticated data structure. To preserve the privacy of datasets, queries, skyline results and access patterns, we have presented two solutions for skyline queries and VO construction: one is BVLSQ based on SVR-tree and another one is SVLSQ based on SVSR-tree. As is extensively illustrated in experiments, SVLSQ significantly outperforms existing methods. However, as it needs more time to construct the ADS, this solution is more functional for static datasets. In the future research, we plan to investigate verification issues about secure skyline queries with frequent data updates.

ACKNOWLEDGMENTS

This work was supported by NSF of China (62172179, 61972448, 61772215) and CCF-Huawei Innovation Research Plan (CCF-Huawei DBIR2021006B). Xiaofeng Ding is the corresponding author.

REFERENCES

[1] Abbas Acar, Hidayet Aksu, A Selcuk Uluagac, and Mauro Conti. 2018. A survey on homomorphic encryption schemes: Theory and implementation. *Comput. Surveys* 51, 4 (2018), 1–35.

[2] Stephan Borzsony, Donald Kossmann, and Konrad Stocker. 2001. The skyline operator. In *Proceedings of the ICDE*. IEEE, 421–430.

[3] Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2013. A safe zone based approach for monitoring moving skyline queries. In *Proceedings of the International Conference on Extending Database Technology*. 275–286.

[4] Qian Chen, Haibo Hu, and Jianliang Xu. 2013. Authenticating top-k queries in location-based services with confidentiality. In *Proceedings of the VLDB Endowment*, Vol. 7. VLDB Endowment, 49–60.

[5] Wenxin Chen, Mengjun Liu, Rui Zhang, Yanchao Zhang, and Shubo Liu. 2016. Secure outsourced skyline query processing via untrusted cloud service providers. In *Proceedings of the INFOCOM*. IEEE, 1–9.

[6] Nathan Chenette, Kevin Lewi, Stephen A Weis, and David J Wu. 2016. Practical order-revealing encryption with limited leakage. In *International Conference on Fast Software Encryption*. Springer, 474–493.

[7] Sunoh Choi, Gabriel Ghinita, Hyo-Sang Lim, and Elisa Bertino. 2014. Secure knn query processing in untrusted cloud environments. *IEEE Trans. on Knowledge and Data Engineering* 26, 11 (2014), 2818–2831.

[8] Jan Chomicki, Parke Godfrey, Jarek Gryz, and Dongming Liang. 2003. Skyline with presorting. In *Proceedings of the ICDE*, Vol. 3. 717–719.

[9] Ningning Cui, Xiaochun Yang, Bin Wang, Jianxin Li, and Guoren Wang. 2020. SVkNN: Efficient secure and verifiable k-nearest neighbor query on the cloud platform. In *Proceedings of the ICDE*. IEEE, 253–264.

[10] Evangelos Dellis and Bernhard Seeger. 2007. Efficient computation of reverse skyline queries. In *Proceedings of the VLDB Endowment*. 291–302.

[11] Yousef Elmehdwi, Bharath K Samanthula, and Wei Jiang. 2014. Secure k-nearest neighbor query over encrypted data in outsourced environments. In *Proceedings of the ICDE*. IEEE, 664–675.

[12] Antonin Guttman. 1984. R-trees: A dynamic index structure for spatial searching. In *Proceedings of the ACM SIGMOD*. 47–57.

[13] Arif Hidayat, Muhammad Aamir Cheema, Xuemin Lin, Wenjie Zhang, and Ying Zhang. 2021. Continuous monitoring of moving skyline and top-k queries. *The VLDB Journal* (2021), 1–24.

[14] Jiafeng Hua, Guozhen Shi, Hui Zhu, et al. 2018. CAMPS: Efficient and privacy-preserving medical primary diagnosis over outsourced cloud. *Information Sciences* (2018).

[15] Jiafeng Hua, Hui Zhu, Fengwei Wang, Ximeng Liu, Rongxing Lu, Hao Li, and Yeping Zhang. 2018. CINEMA: Efficient and privacy-preserving online medical primary diagnosis with skyline query. *IEEE Internet of Things Journal* 6, 2 (2018), 1450–1461.

[16] Zhiyong Huang, Hua Lu, Beng Chin Ooi, and Anthony KH Tung. 2006. Continuous skyline queries for moving objects. *IEEE Transactions on Knowledge and Data Engineering* 18, 12 (2006), 1645–1658.

[17] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access pattern disclosure on searchable encryption: ramification, attack and mitigation. In *Proceedings of the NDSS*. Citeseer.

[18] Hyeong-Il Kim, Hyeong-Jin Kim, and Jae-Woo Chang. 2019. A secure kNN query processing algorithm using homomorphic encryption on outsourced database. *Data & Knowledge Engineering* 123 (2019), 101602.

[19] Donald Kossmann, Frank Ramsak, and Steffen Rost. 2002. Shooting stars in the sky: An online algorithm for skyline queries. In *Proceedings of the VLDB Endowment*. Elsevier, 275–286.

[20] Kevin Lewi and David J Wu. 2016. Order-revealing encryption: New constructions, applications, and lower bounds. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*. 1167–1178.

[21] Xin Lin, Jianliang Xu, and Haibo Hu. 2011. Authentication of location-based skyline queries. In *Proceedings of the ACM CIKM*. 1583–1588.

[22] An Liu, Kai Zhengy, Lu Liz, Guanfeng Liu, Lei Zhao, and Xiaofang Zhou. 2015. Efficient secure similarity computation on encrypted trajectory data. In *Proceedings of the ICDE*. IEEE, 66–77.

[23] Chang Liu, Liehuang Zhu, Xiangjian He, and Jinjun Chen. 2021. Enabling privacy-preserving shortest distance queries on encrypted graph data. *IEEE Transactions on Dependable and Secure Computing* 18, 1 (2021), 192–204. https://doi.org/10.1109/TDSC.2018.2880981

[24] Jinfei Liu, Juncheng Yang, Li Xiong, and Jian Pei. 2017. Secure skyline queries on cloud platform. In *Proceedings of the ICDE*. IEEE, 633–644.

[25] Jinfei Liu, Juncheng Yang, Li Xiong, and Jian Pei. 2019. Secure and efficient skyline queries on encrypted data. *IEEE Trans. on Knowledge and Data Engineering* 31, 7 (2019), 1397–1411.

[26] Ximeng Liu, Kim-Kwang Raymond Choo, Robert H Deng, Yang Yang, and Yinghui Zhang. 2018. PUSC: Privacy-preserving user-centric skyline computation over multiple encrypted domains. In *Proceedings of the TrustCom/BigDataSE*. IEEE, 958–963.

[27] Ximeng Liu, Rongxing Lu, Jianfeng Ma, Le Chen, and Haiyong Bao. 2016. Efficient and privacy-preserving skyline computation framework across domains. *Future Generation Computer Systems* 62 (2016), 161–174.

[28] Hans Lo and Gabriel Ghinita. 2013. Authenticating spatial skyline queries with low communication overhead. In *Proceedings of the ACM Conference on Data and Application Security and Privacy*. 177–180.

[29] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 223–238.

[30] Dimitris Papadias, Yufei Tao, Greg Fu, and Bernhard Seeger. 2005. Progressive skyline computation in database systems. *ACM Transactions on Database Systems (TODS)* 30, 1 (2005), 41–82.

[31] Stavros Papadopoulos, Spiridon Bakiras, and Dimitris Papadias. 2010. Nearest neighbor search with strong location privacy. In *Proceedings of the VLDB Endowment*, Vol. 3. 619–629.

[32] Raluca Ada Popa, Frank H Li, and Nickolai Zeldovich. 2013. An ideal-security protocol for order-preserving encoding. In *IEEE Symposium on Security and Privacy*. IEEE, 463–477.

[33] Mehdi Sharifzadeh and Cyrus Shahabi. 2006. The spatial skyline queries. In *Proceedings of the VLDB Endowment*. 751–762.

[34] Emil Stefanov, Marten Van Dijk, Elaine Shi, T-H Hubert Chan, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2018. Path oram: An extremely simple oblivious ram protocol. *JACM* 65, 4 (2018), 1–26.

[35] Marc Van Kreveld, Otfried Schwarzkopf, Mark de Berg, and Mark Overmars. 2000. *Computational geometry: algorithms and applications*. Springer.

[36] Weiguo Wang, Hui Li, Yanguo Peng, Sourav S Bhowmick, Peng Chen, Xiaofeng Chen, and Jiangtao Cui. 2020. SCALE: An efficient framework for secure dynamic skyline query processing in the cloud. In *Proceedings of the DASFAA*. Springer.

[37] Wai Kit Wong, David Wai-lok Cheung, Ben Kao, and Nikos Mamoulis. 2009. Secure kNN computation on encrypted databases. In *Proceedings of the ACM SIGMOD*. 139–152.

[38] Songrui Wu, Qi Li, Guoliang Li, Dong Yuan, Xingliang Yuan, and Cong Wang. 2019. Servedb: Secure, verifiable, and efficient range queries on outsourced database. In *Proceedings of the ICDE*. IEEE, 626–637.

[39] Cheng Xu, Ce Zhang, and Jianliang Xu. 2019. vChain: Enabling verifiable boolean range queries over blockchain databases. In *Proceedings of the ACM SIGMOD*. 141–158.

[40] Yin Yang, Stavros Papadopoulos, Dimitris Papadias, and George Kollios. 2009. Authenticated indexing for outsourced spatial databases. *The VLDB Journal* 18, 3 (2009), 631–648.

[41] Man Lung Yiu, Eric Lo, and Duncan Yung. 2011. Authentication of moving knn queries. In *Proceedings of the ICDE*. IEEE, 565–576.

[42] Sepanta Zeighami, Gabriel Ghinita, and Cyrus Shahabi. 2021. Secure dynamic skyline queries using result materialization. In *Proceedings of the ICDE*. IEEE, 157–168.

[43] Baihua Zheng, Ken CK Lee, and Wang-Chien Lee. 2008. Location-dependent skyline query. In *International Conference on Mobile Data Management*. IEEE, 148–155.