Toward Privacy-Preserving Aggregate Reverse Skyline Query With Strong Security

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

***Abstract*—It has been witnessed that Aggregate Reverse Skyline (ARS) query has recently received a wide range of practical applications due to its marvelous property of identifying the influence of query requests. Nevertheless, the query users may hesitate to participate in such query services as the query requests and query results may leak sensitive personal data or valuable business data assets to the service providers. To tackle the concerns, a promising solution is to encrypt the query requests, conduct the ARS queries over encrypted query requests without decrypting, and return the encrypted query results. Unfortunately, many existing solutions are either deployed over a two-server model or unable to fully preserve query privacy. In this paper, we propose a novel privacy-preserving aggregate reverse skyline query (PPARS) scheme on a single server model while ensuring full query privacy. Specifically, we first transform the problem of ARS query into a combination of set membership test and logical expressions. Then, by employing the prefix encoding technique, bloom filter technique, and fully homomorphic encryption, we run the transformed logical expressions to obtain the encrypted aggregate values without leaking query requests, query results, and access patterns. Furthermore, we propose an interpolation-based packing technique to improve the communication efficiency of PPARS. Detailed and formal security analysis demonstrates that our proposed schemes can guarantee strong security. In addition, extensive experiments are conducted, and the results validate the efficiency of our proposed schemes.**

***Index Terms*—Aggregate reverse skyline, privacy preservation, single server model, bloom filter, Lagrange interpolation.**

I. Introduction

As a powerful tool for multicriteria data analysis, data mining, and decision making, the skyline query and its variants can return a set of interesting points that are the best trade-offs among the different dimensions of a huge data space [1], [2]. Among various skyline queries, the reverse skyline query has been widely applied to real-world applications, such as business location planning [3], environmental monitoring [4], and marketing insights [5], due to its unique practical significance. Specifically, the reverse skyline can identify the influence of a query request on a multi-dimensional dataset, i.e., how many data points in the dataset are interesting in the query request [6]. Two typical examples of the reverse skyline query are shown as follows.

Manuscript received 24 January 2022; revised 2 May 2022; accepted 15 June 2022. Date of publication 4 July 2022; date of current version 15 July 2022. This work was supported in part by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery under Grant 04009, Grant 03787, and Grant RGPIN-2022-03244. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Edgar Weippl. *(Corresponding author: Rongxing Lu.)*

Songnian Zhang, Suprio Ray, Rongxing Lu, Yunguo Guan, and Yandong Zheng are with the Faculty of Computer Science, University of New Brunswick, Fredericton, NB E3B 5A3, Canada (e-mail: szhang17@unb.ca; sray@unb.ca; rlu1@unb.ca; yguan4@unb.ca; yzheng8@unb.ca).

Jun Shao is with the School of Computer and Information Engineering, Zhejiang Gongshang University, Hangzhou 310018, China (e-mail: chn.junshao@gmail.com).

Digital Object Identifier 10.1109/TIFS.2022.3188147

TABLE I
Three Possible Choices for the Manufacturer

Choices	Price	Size	Storage
Choice 1	\$1200	14-inch	256 GB
Choice 2	\$1000	15-inch	128 GB
Choice 3	\$1500	13-inch	512 GB

Example 1. A computer manufacturer plans to design a new type of computer, and he/she has several possible choices in some configuration parameters for the new computer. In Table I, we give an example with three possible choices in the parameters of price, size, and storage. Assume that a dealer has a database of customers' preferences about computers. By providing these choices (query requests) to the dealer, the computer manufacturer wants to know a set of parameters that attracts the maximum number of customers (largest influence) as the forthcoming computer's parameters.

Example 2. A more commonplace example is that a tourist plans to visit a city and has several potential hotels to choose from. The tourist can provide these hotels' locations (query requests) to a POI (points of interest) database owner and choose a hotel that can maximize the number of POI, such as, tourist attractions and restaurants.

In the above examples, the most influential query requests can be obtained by launching the reverse skyline query that can find the data points (in a database) "attracted" by a query request [3]–[6]. After aggregating the number of attracted data points for different query requests, it is easy to determine the most influential query request. We formally define the (aggregate) reverse skyline query in Section III-A and refer readers to that section for the details. To illustrate the aggregate characteristic of the reverse skyline query, we use the concept of the aggregate reverse skyline (ARS) query hereinafter.

In practical applications, the databases in the above examples are usually owned by some big companies, e.g., the POI database owned by Yelp, and these companies (service providers) can provide the query services (ARS query services) to the query users (computer manufacturers or tourists). However, the query users may hesitate to participate in such query services due to the privacy concerns regarding the query requests and query results. In the aforementioned examples, the computer parameters are valuable business data assets to the computer manufacturer, while the hotel locations may leak the query users' final locations. A promising solution is to encrypt query requests, make the service providers perform ARS queries without decrypting the query requests, and obtain the encrypted query result at the end. But that is not enough, since the service providers have databases in plaintexts, we must hide access patterns when performing the queries. Otherwise, the service providers can infer the query requests and query results by comparing the relations between the query results and the owned data points (see detailed analysis in Section IV-A). Here, we define the access pattern as the information about which data points are selected as reverse skyline points. However, when conducting the ARS search, we have to run a *compute-then-compare* operation, which is challenging to be achieved over encrypted data. In order to address this challenge, existing schemes either adopt a two-server model [7]–[9] or lower the security level, e.g., leaking order relations [10] or access patterns [11]. However, in our scenario: i) the proposed scheme needs to be run in a single server model due to the practical considerations; ii) the proposed scheme must ensure that no information, which is relevant to the query requests, is leaked. Thus, it is quite challenging to achieve the ARS query in a single server model while providing strong security, i.e., protecting the privacy of query requests, query results, and access patterns.

Aiming at the above challenge, in this paper, we propose a privacy-preserving ARS query scheme that can ensure full query privacy with a single service provider. The key point to address this challenge is that we transform the problem of the ARS query into a combination of set membership test and logical expressions. See detailed transformation in Section IV-A. With this knowledge, instead of encrypting the query request directly, we first convert it into bloom filters by using a prefix encoding technique [11] and then encrypt the elements in the bloom filters with a fully homomorphic encryption (FHE) scheme [12], [13]. Consequently, we can determine whether a value is in a set or not by performing multiplications over FHE encrypted elements. If the underlying plaintext of the result is 1, it means that the value is in the set. Otherwise, the result is 0, indicating the value is not in the set. Afterward, by running the logical expressions over FHE encrypted data, we can obtain an encrypted flag to identify whether a data point is a reverse skyline point. If yes, the underlying plaintext is 1. Otherwise it is 0. Finally, by using the addition homomorphic property of FHE, it is easy to obtain the number of reverse skyline points. Specifically, the main contributions of this paper are three folds as follows.

- First, we propose a novel privacy-preserving ARS query scheme (PPRAS) in a single server model, which can preserve the privacy of query requests, query results, and access patterns. To achieve it, we carefully design an approach that can convert the ARS query into set membership test and logical expressions. By employing the prefix encoding technique, bloom filters, and homomorphic properties of FHE, we can obtain the encrypted query results while hiding access patterns. Note that it is a general approach and can be used to support other privacy-preserving dynamic skyline queries [7]–[9] in such a scenario. To the best of our knowledge, we are the first to consider the privacy-preserving ARS queries in a single server model while providing strong security.
- Second, we propose an interpolation-based packing technique to improve communication efficiency. To reduce the number of encrypted data items delivered from the query user to the service provider, we elaborately design a Lagrange interpolation-based approach to pack several ciphertexts into one, thus improving communication efficiency. We denote the communication-efficient scheme as CE-PPARS.
- Third, we formally prove the security of our schemes in the real/ideal model and demonstrate that our PPARS and CE-PPARS schemes can indeed achieve our privacy requirements. Furthermore, we conduct extensive experiments to evaluate the performance of our proposed schemes, and the results validate the efficiency of our proposed schemes.

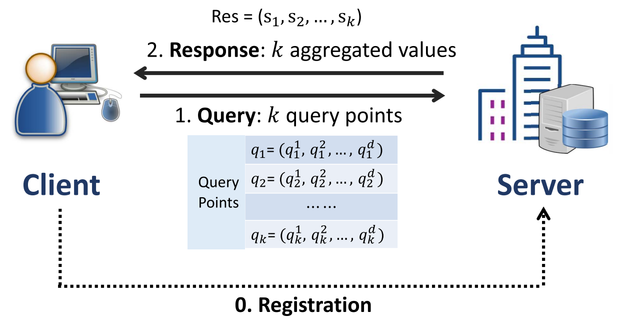

$\text{Res} = (s_1, s_2, \ldots, s_k)$
2. **Response**: k aggregated values
1. **Query**: k query points
Client
Server
Query Points
$q_1 = (q_1^1, q_1^2, \ldots, q_1^d)$
$q_2 = (q_2^1, q_2^2, \ldots, q_2^d)$
……
$q_k = (q_k^1, q_k^2, \ldots, q_k^d)$
0. Registration

Fig. 1. System model under consideration.

The remainder of this paper is organized as follows. In Section II, we introduce our system model, security model, and design goal. Then, we review the preliminaries in Section III. After that, we present our privacy-preserving aggregate reverse skyline query scheme in Section IV, followed by security analysis and performance evaluation in Section V and Section VI, respectively. Finally, we discuss some related works in Section VII and draw our conclusion in Section VIII.

II. Models and Design Goal

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

A. System Model

In our system model, we consider a practical and typical client-server model, which is comprised of two entities: a powerful server $\mathcal{S}$ providing query services and a client $\mathcal{C}$ enjoying services, as shown in Fig. 1.

1) Server $\mathcal{S}$: In our system model, the server $\mathcal{S}$ is a practical service provider, who holds enriched databases and can provide the ARS query services to users. We assume that $\mathcal{S}$ is equipped with powerful storage and computing resources.

It is reasonable since: i) some big companies (are also the data owners) may have their own cloud server platforms, e.g., Amazon; ii) some data mining companies prefer to manage their valuable databases by themselves and build server platforms for providing the query services; and iii) some datasets are publicly available, e.g., the location keywords data can be searched on Google map. For ease of description, we use $\mathcal{X}$ to denote a database and say each data record in $\mathcal{X}$ has d dimensions, i.e., $\mathcal{X} = \{\mathrm{x}_i = (\mathrm{x}_i^1, \mathrm{x}_i^2, \cdots, \mathrm{x}_i^d) \mid 1 \le i \le n\}$.

2) Client $\mathcal{C}$: In our system, the client $\mathcal{C}$ is a query user that can enjoy the ARS query services from the server $\mathcal{S}$. Specifically, the client $\mathcal{C}$ sends k d-dimensional query requests, i.e., $\mathcal{Q} = \{\mathrm{q}_i = (\mathrm{q}_i^1, \mathrm{q}_i^2, \cdots, \mathrm{q}_i^d) \mid 1 \le i \le k\}$, to $\mathcal{S}$. After conducting the aggregate reverse skyline query algorithm in the server side, $\mathcal{S}$ returns k aggregated values $\{\mathsf{s}_1, \mathsf{s}_2, \cdots, \mathsf{s}_k\}$ to $\mathcal{C}$, each of which represents the number of reverse skyline points of a query request. Before enjoying the query services, we assume that the client $\mathcal{C}$ needs to register to $\mathcal{S}$ for obtaining query qualification. As this work focuses on secure computing techniques, we do not consider the authentication and verification issues here.

Note that we assume both data records and query requests are integers. It is reasonable since we can easily convert non-integer data into non-negative integers [14].

B. Security Model

In our security model, since the client $\mathcal{C}$ wants to obtain the correct query results, it has no motivation to deviate from the proposed scheme. As a result, we consider $\mathcal{C}$ to be *honest*, i.e., it will honestly offer the query requests to the server $\mathcal{S}$. However, in our model, $\mathcal{S}$ is considered to be *honest-but-curious* [15], [16]. That is, $\mathcal{S}$ will faithfully follow the designed scheme but may be curious to learn the private information of $\mathcal{C}$. For example, $\mathcal{S}$ may be interested in the query requests uploaded by $\mathcal{C}$. As shown in Example 1, the business plans are valuable business secrets for computer manufacturer. To ensure privacy, the client $\mathcal{C}$ can report the encrypted query requests $\mathsf{E}(\mathcal{Q}) = \{\mathsf{E}(\mathrm{q}_t) = (\mathsf{E}(\mathrm{q}_t^1), \mathsf{E}(\mathrm{q}_t^2), \cdots, \mathsf{E}(\mathrm{q}_t^d)) \mid 1 \le t \le k\}$ to the server $\mathcal{S}$. Nevertheless, $\mathcal{S}$ may still attempt to infer the query requests in the process of performing the aggregate reverse skyline queries, for example, by analyzing the relationship between query requests and the owned data records. Note that, since this work mainly focuses on the privacy computation, other active attacks, e.g., Denial of Service (DoS) attacks, are beyond the scope of this paper, and will be explored in our future work.

C. Design Goal

In this work, we aim to present privacy-preserving and efficient ARS query schemes. In particular, the following objectives should be attained.

1) Privacy Preservation: The fundamental requirement of the proposed schemes is to protect the privacy of query requests and query results against the server $\mathcal{S}$. In addition, it is necessary to hide access patterns, i.e., $\mathcal{S}$ has no idea on which data points are selected as the reverse skyline.

2) Efficiency: It is inevitable that the privacy requirements will incur additional costs. Therefore, we also aim to minimize the performance costs when conducting the privacy-preserving aggregate reverse skyline query schemes.

III. Preliminaries

In this section, we first formally define the aggregate reverse skyline queries. Then, we introduce fully homomorphic encryption and bloom filter techniques, which will be used in our proposed schemes.

A. Aggregate Reverse Skyline Queries

The reverse skyline query has been extensively studied in the data mining community due to its wide applications [3]–[5], [17]. For the skyline computation, the core operation is to determine the dominance relation between two data records. Consequently, we will start with the definition of dynamic reverse dominance.

Definition 1 Dynamic Reverse Dominance: Given two d-dimensional data records $\mathrm{x}_1, \mathrm{x}_2 \in \mathcal{X}$ and a query request q in the workplace, x_2 dynamically dominates q with regard to x_1, denoted as $\mathrm{x}_2 \prec_{\mathrm{x}_1} \mathrm{q}$, if:

i) $\forall i \in [1, d]$: $|\mathrm{x}_2^i - \mathrm{x}_1^i| \le |\mathrm{q}^i - \mathrm{x}_1^i|$;

ii) $\exists j \in [1, d]$: $|\mathrm{x}_2^j - \mathrm{x}_1^j| < |\mathrm{q}^j - \mathrm{x}_1^j|$.

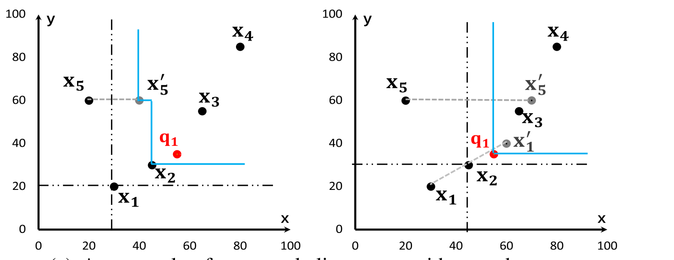
The definition of *Dynamic **Reverse** Dominance* is relative to that of *Dynamic Dominance*, which is defined as "x_2 dynamically dominates x_1 with regard to q, i.e., $\mathrm{x}_2 \prec_{\mathrm{q}} \mathrm{x}_1$". See the detailed definition of *Dynamic Dominance* in [9].

Definition 2 Reverse Skyline Query: Given a dataset $\mathcal{X}$ and a query request q, the reverse skyline query returns a reverse skyline set $S_{\mathrm{q}} \subseteq \mathcal{X}$, where $S_{\mathrm{q}} = \{\mathrm{x}_i \in \mathcal{X} | \nexists \mathrm{x}_j \in \mathcal{X}$ such that $\mathrm{x}_j \prec_{\mathrm{x}_i} \mathrm{q}\}$. We call S_{q} as the reverse skyline of q.

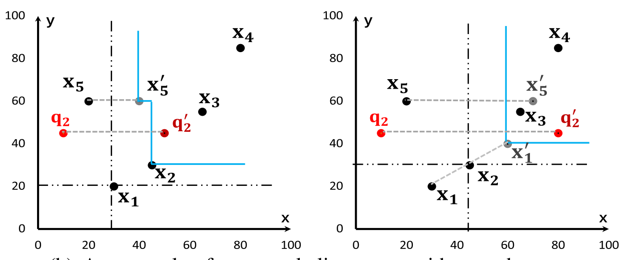
In real-world applications, the reverse skyline query is used to identify the influence of a query request on a multi-dimensional database. The larger cardinality of S_{q} means the larger influence of the corresponding query request. Although almost all reverse skyline query works [3]–[6] mentioned using the cardinality of the reverse skyline set to measure the influence, there is no explicit definition of it. In this paper, considering formalization, we define it as the aggregate reverse skyline query.

Definition 3 Aggregate Reverse Skyline Query: Given a dataset $\mathcal{X}$ and a set of query requests $\mathcal{Q} = \{\mathrm{q}_i = (\mathrm{q}_i^1, \mathrm{q}_i^2, \cdots, \mathrm{q}_i^d) \mid 1 \le i \le k\}$, the aggregate reverse skyline query returns an aggregated set $\mathsf{AS} = \{\mathsf{s}_i \mid 1 \le i \le k\}$, where s_i is the number of reverse skyline points of q_i, i.e., $\mathsf{s}_i = |S_{\mathrm{q}_i}|$.

A simple example of the aggregate reverse skyline (ARS) query is shown in Fig. 2, in which $\mathcal{Q} = \{\mathrm{q}_1, \mathrm{q}_2\}$ and $\mathcal{X} = \{\mathrm{x}_i \mid 1 \le i \le 5\}$. Fig. 2(a) plots the reverse skyline query of q_1 and shows the dynamic reverse dominance about x_1 and x_2. We can see that x_1 is not a reverse skyline point of q_1, while x_2 is. In Fig. 2, to visually illustrate the absolute values, we map all points into a new space where the point x_i is the origin. For instance, when checking whether x_1 is a reverse skyline point or not, we map other points into the space with the origin of x_1. As shown in the first figure of Fig. 2(a),



(a) An example of reverse skyline query with regard to q_1



(b) An example of reverse skyline query with regard to q_2

Fig. 2. Aggregate reverse skyline query example, in which we check whether x_1 and x_2 are reverse skyline points of q_1 and q_2, respectively: (a) x_1 is not a reverse skyline point of q_1 while x_2 is; (b) both x_1 and x_2 are not the reverse skyline point of q_2.

x_5 is mapped to x_5'. After checking each point in $\mathcal{X}$, we can obtain $S_{q_1} = \{x_2, x_3, x_5\}$. Similarly, we have $S_{q_2} = \{x_5\}$ from Fig. 2(b). As a result, the aggregated set AS is $\{3, 1\}$.

B. Fully Homomorphic Encryption

Fully homomorphic encryption (FHE) is a form of encryption that provides a way to encrypt data while supporting computations through the encryption envelope [18]. Due to its nice homomorphic properties, it is widely employed to design searchable encryption schemes [19]–[21]. In general, an FHE scheme satisfies the following two homomorphic properties: i) Homomorphic addition: $\mathsf{E}(m_1) + \mathsf{E}(m_2) \rightarrow \mathsf{E}(m_1 + m_2)$; ii) Homomorphic multiplication: $\mathsf{E}(m_1) \cdot \mathsf{E}(m_2) \rightarrow \mathsf{E}(m_1 \cdot m_2)$, where m_1 and m_2 are two plaintexts (usually, they are integers), and $\mathsf{E}(m_1)$ and $\mathsf{E}(m_2)$ are the corresponding ciphertexts encrypted by the FHE encryption algorithm. Since our proposed scheme does not depend on a specific FHE scheme, here, we do not elaborate the FHE algorithms. We refer readers to [12], [13], [22] for detailed FHE algorithms.

C. Bloom Filter

Bloom filter (BF) is a space- and time-efficient probabilistic data structure introduced by Burton Bloom [23], which allows membership queries on a set S. A bloom filter represents a set of m elements using an array of η bits, denoted as $\mathsf{BF}[1], \cdots, \mathsf{BF}[\eta]$. Initially, all bits $\{\mathsf{BF}[p] \mid 0 \le p \le \eta\}$ in the array are set to 0. Then, with l independent hash functions $\{h_1, \cdots, h_l\}$, each element $x \in \mathsf{S}$ is mapped to the array, i.e., $\mathsf{BF}[h_i(x)] = 1$. Given an element x', to answer a query about whether $x' \in \mathsf{S}$ or not, one can check whether all $\mathsf{BF}[h_i(x')]$ are set to 1. If not, x' is not a member of S. If yes, x' is in S with a high probability. Obviously, a bloom filter may yield *false positive*, and the *false positive* probability is:

$$f_p = \left(1 - (1 - 1/\eta)^{lm}\right)^l \approx (1 - e^{-l\frac{m}{\eta}})^l. \quad (1)$$

Given m and η, the value of l that minimizes the *false positive* probability is: $l = \ln 2 \cdot (\eta/m)$. In this case, $f_p \approx (1/2)^l \approx (0.6185)^{\eta/m}$. In our proposed schemes, f_p and m will be given. After computing $l = -\log_2 f_p$, we can determine the size of the bloom filter array as $\eta = \lceil l \cdot m / \ln 2 \rceil$, which is the minimum size to ensure the *false positive* that is less than f_p.

IV. Our Proposed Scheme

In this section, we first analyze the problem of ARS query over plaintexts. Then, we design a novel secure scheme to check the dynamic reverse dominance relation, denoted as SDRD, which is the core component in the ARS query. After that, we propose our privacy-preserving ARS query scheme, PPARS. Finally, we present a communication-efficient PPARS scheme, denoted as CE-PPARS.

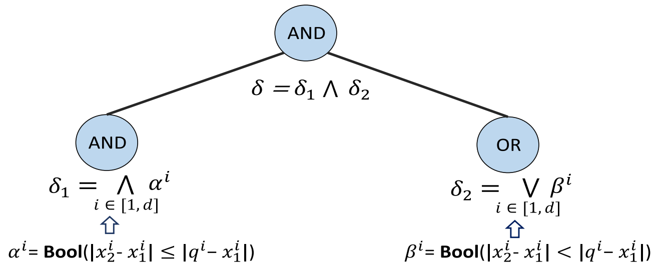
Algorithm 1: ARS Query Over Plaintexts

```
Input: A $d$-dimensional dataset $\mathcal{X} = \{x_i \mid 1 \le i \le n\}$; A set of query
       points $\mathcal{Q} = \{q_t \mid 1 \le t \le k\}$.
Output: A set containing $k$ aggregated values, $\mathsf{AS}$.
1:  $\mathsf{AS} \leftarrow \varnothing$;
2:  for each query point $q_t \in \mathcal{Q}$ do
3:      $S_{q_t} \leftarrow \varnothing$;
4:      for each data point $x_i \in \mathcal{X}$ do
5:          $\delta \leftarrow$ false;
6:          for $x_j \in \mathcal{X}$ and $x_j \neq x_i$ do
7:              if $x_j \prec_{x_i} q_t$ then                ▷Definition 1
8:                  $\delta \leftarrow$ true;
9:                  break;
10:         if $\delta$ = false then
11:             $S_{q_t}$.add($x_i$);
12:     $\mathsf{AS}$.add($|S_{q_t}|$);
    return $\mathsf{AS}$.
```

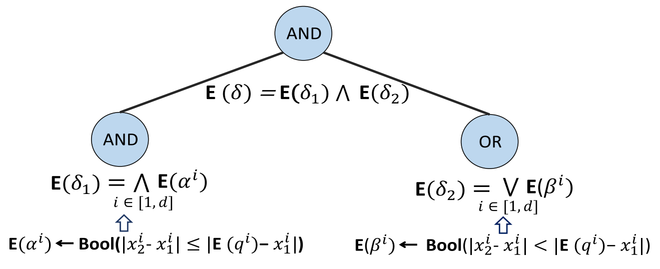
A. Analyzing ARS Queries

Based on the definitions in Section III-A, we can formally present the ARS query over plaintexts in Algorithm 1, in which the inputs are a dataset $\mathcal{X}$ and a query set $\mathcal{Q}$, and the output is an aggregated set AS. By analyzing this algorithm, it is easy to identify that the block of lines 7-9 is the core part of the algorithm. This block is to check the dynamic reverse dominance relation among x_j, x_i, and q_t, i.e, whether $x_j \prec_{x_i} q_t$ holds. If yes, it makes δ be true, otherwise false.

Fig. 3(a) depicts the logical expression of determining a dynamic reverse dominance relation. We can see that determining dynamic reverse dominance is a typical *compute-then-compare* operation, which is hard to be achieved: i) over encrypted data; ii) on the single server model; and iii) without leaking any predicate or access pattern information to the server. Existing approaches either deploy a two-server model [7]–[9] or leak predicate or access pattern information [10], [11]. In our model, our goal is to protect the query


AND
$\delta = \delta_1 \wedge \delta_2$
AND
OR
$\delta_1 = \bigwedge_{i \in [1,d]} \alpha^i$
$\delta_2 = \bigvee_{i \in [1,d]} \beta^i$
$\alpha^i = \mathbf{Bool}(|x_2^i - x_1^i| \le |q^i - x_1^i|)$
$\beta^i = \mathbf{Bool}(|x_2^i - x_1^i| < |q^i - x_1^i|)$

(a) Logical expression of dynamic reverse dominance over plaintexts


AND
$\mathsf{E}(\delta) = \mathsf{E}(\delta_1) \wedge \mathsf{E}(\delta_2)$
AND
OR
$\mathsf{E}(\delta_1) = \bigwedge_{i \in [1,d]} \mathsf{E}(\alpha^i)$
$\mathsf{E}(\delta_2) = \bigvee_{i \in [1,d]} \mathsf{E}(\beta^i)$
$\mathsf{E}(\alpha^i) \leftarrow \mathbf{Bool}(|x_2^i - x_1^i| \le |\mathsf{E}(q^i) - x_1^i|)$
$\mathsf{E}(\beta^i) \leftarrow \mathbf{Bool}(|x_2^i - x_1^i| < |\mathsf{E}(q^i) - x_1^i|)$

(b) Logical expression of secure dynamic reverse dominance

Fig. 3. Logical expressions of dynamic reverse dominance, where **Bool**(*predicate*) returns 1 (true) if *predicate* holds, otherwise returns 0 (false).

set $\mathcal{Q}$ and query result AS against the single server $\mathcal{S}$. As a result, the query set $\mathcal{Q}$ will be encrypted before sending it to the server $\mathcal{S}$, and the ARS query will be conducted over encrypted $\mathcal{Q}$. Finally, $\mathcal{S}$ obtains and returns the encrypted query result AS to the client $\mathcal{C}$. In addition, in the process of performing ARS queries, we cannot leak any predicate results or access patterns to the server $\mathcal{S}$, e.g., the information about whether the variable δ in Algorithm 1 is true or not. That is because if $\mathcal{S}$ knows such information, he/she can infer the query requests $\mathrm{q}_t \in \mathcal{Q}$ according to the dataset $\mathcal{X}$ and the leaked information. We show a more concrete example as follows:

Example 3. Assume $\mathcal{S}$ knows that "δ= false", which is from the predicate information about "$|\mathrm{x}_2^i - \mathrm{x}_1^i| \le |\mathsf{E}(\mathrm{q}_t^i) - \mathrm{x}_1^i|$ does not hold". In this way, $\mathcal{S}$ can easily infer that q_t^i must fall inside the range of $[\mathrm{x}_1^i - \mathrm{x}_2^i, \mathrm{x}_1^i + \mathrm{x}_2^i]$. Therefore, although q_t^i is encrypted, $\mathcal{S}$ can still infer its approximate value according to x_1^i, x_2^i and the predicate information.

Thus, the main challenge in the privacy-preserving ARS query is to design a secure dynamic reverse dominance scheme over the single server model while preserving the privacy of query requests, including the privacy of predicate results (order comparison results) and access patterns (whether a data points is added into S_{q_t}, i.e., whether it is a reverse skyline point).

B. Secure Dynamic Reverse Dominance Scheme

Given two data records $\{\mathrm{x}_1, \mathrm{x}_2\}$ and a query request q, our secure dynamic reverse dominance scheme, SDRD, can determine whether $\mathrm{x}_2 \prec_{\mathrm{x}_1} \mathrm{q}$ holds or not. If yes, it outputs $\mathsf{E}(\delta) = \mathsf{E}(1)$, otherwise $\mathsf{E}(\delta) = \mathsf{E}(0)$. The basic idea of our SDRD scheme is to achieve the logical expression over encrypted data, as shown in Fig. 3(b). Assume we have obtained $\mathsf{E}(\alpha^i) = \mathsf{E}(1)$ when $|\mathrm{x}_2^i - \mathrm{x}_1^i| \le |\mathsf{E}(\mathrm{q}^i) - \mathrm{x}_1^i|$ holds, otherwise $\mathsf{E}(\alpha^i) = \mathsf{E}(0)$. Since $\mathsf{E}(\alpha^i) \wedge \mathsf{E}(\alpha^j) = \mathsf{E}(\alpha^i) \cdot \mathsf{E}(\alpha^j), i, j \in [1, d]$, we have $\mathsf{E}(\delta_1) = \prod_{i=1}^{d} \mathsf{E}(\alpha^i)$. Besides, if we define $\mathsf{E}(\beta^i) = \mathsf{E}(1)$ when $|\mathrm{x}_2^i - \mathrm{x}_1^i| < |\mathsf{E}(\mathrm{q}^i) - \mathrm{x}_1^i|$ holds, otherwise $\mathsf{E}(\beta^i) = \mathsf{E}(0)$, we can get $\mathsf{E}(\delta_2)$ by repeatedly computing $\mathsf{E}(\beta^i) \vee \mathsf{E}(\beta^j) = \mathsf{E}(\beta^i) + \mathsf{E}(\beta^j) - \mathsf{E}(\beta^i) \cdot \mathsf{E}(\beta^j), i, j \in [1, d]$. Obviously, computing $\mathsf{E}(\delta_2)$ is not efficient. As a result, different from the above definition of $\mathsf{E}(\beta^i)$, we assume $\mathsf{E}(\beta^i) = \mathsf{E}(0)$ when $|\mathrm{x}_2^i - \mathrm{x}_1^i| < |\mathsf{E}(\mathrm{q}^i) - \mathrm{x}_1^i|$ holds, otherwise $\mathsf{E}(\beta^i) = \mathsf{E}(1)$. In this way, we can easily calculate $\mathsf{E}(\delta_2) = \mathsf{E}(1) + \prod_{i=1}^{d} \mathsf{E}(\beta^i) \cdot \mathsf{E}(-1) = \mathsf{E}(1 - \prod_{i=1}^{d} \beta^i)$ with the homomorphic properties of FHE. Thus, we have the following equations:

$$\begin{cases} \mathsf{E}(\delta) = \mathsf{E}(\delta_1) \wedge \mathsf{E}(\delta_2) = \mathsf{E}(\delta_1) \cdot \mathsf{E}(\delta_2) = \mathsf{E}(\delta_1 \cdot \delta_2); \\ \mathsf{E}(\delta_1) = \mathsf{E}(\alpha^1) \wedge \mathsf{E}(\alpha^2) \wedge \cdots \wedge \mathsf{E}(\alpha^d) = \mathsf{E}(\prod_{i=1}^{d} \alpha^i); \\ \mathsf{E}(\delta_2) = \mathsf{E}(\beta^1) \vee \mathsf{E}(\beta^2) \vee \cdots \vee \mathsf{E}(\beta^d) = \mathsf{E}(1 - \prod_{i=1}^{d} \beta^i). \end{cases} \tag{2}$$

Next, the problem is how to make $\mathsf{E}(\alpha^i) = \mathsf{E}(1)$ (resp. $\mathsf{E}(\beta^i) = \mathsf{E}(0)$) when $|\mathrm{x}_2^i - \mathrm{x}_1^i| \le |\mathsf{E}(\mathrm{q}^i) - \mathrm{x}_1^i|$ (resp. $|\mathrm{x}_2^i - \mathrm{x}_1^i| < |\mathsf{E}(\mathrm{q}^i) - \mathrm{x}_1^i|$) holds, otherwise $\mathsf{E}(\alpha^i) = \mathsf{E}(0)$ (resp. $\mathsf{E}(\beta^i) = \mathsf{E}(1)$). Our key idea is to transform the *compute-then-compare* operation into *set membership test*, allowing us to make it possible in a single cloud server while ensuring full query privacy.

1) Prefix Encoding Technique: Before delving into the transformation, we first introduce a prefix encoding technique [11] to check whether a value v falls inside a range R, i.e., $v \in R$. Given a range $R = [r_{min}, r_{max}]$, where r_{min} and r_{max} are two integers with ϕ bits, we encode it into $\mathcal{G}(R)$ by extracting the minimum set of prefix elements that cover the range $[r_{min}, r_{max}]$. For example, if $R = [0, 6]$ and $\phi = 3$, we have $\mathcal{G}([0, 6]) = \{0{*}{*}, 10{*}, 110\}$, as illustrated in Fig. 4. As demonstrated in [24], the number of prefix elements is at most $2\phi - 2$. Given a ϕ-bit integer v and its binary format $v_1 v_2 \cdots v_\phi$, we encode it into a set $\mathcal{F}(v)$ with $\phi + 1$ elements, where $\mathcal{F}(v) = \{v_1 v_2 \cdots v_\phi, v_1 v_2 \cdots v_{\phi-1}{*}, \cdots, v_1 {*} \cdots {*}, {*}{*} \cdots {*}\}$, namely, the i-th prefix element is $v_1 v_2 \cdots v_{\phi-i+1} {*} \cdots {*}$. For example, the integer 5 can be encoded into $\mathcal{F}(5) = \{101, 10{*}, 1{*}{*}, {*}{*}{*}\}$. After encoding the value v and the range R, we have

$$v \in R \Leftrightarrow \mathcal{F}(v) \cap \mathcal{G}(R) \neq \emptyset. \tag{3}$$

In the above example, since $\mathcal{F}(5) \cap \mathcal{G}([0, 6]) = \{10{*}\} \neq \emptyset$, we have $5 \in [0, 6]$. The essential reason is that, if $v \in R$, the prefix family of $v_1 v_2 \cdots v_\phi$ must be one of the elements covering R. It also indicates that, if $v \in R$, there is only one common element for $\mathcal{F}(v)$ and $\mathcal{G}(R)$, i.e., $|\mathcal{F}(v) \cap \mathcal{G}(R)| = 1$.

2) Transformation From Compute-Then-Compare to Set Membership Test: Now, we discuss the transformation from *compute-then-compare* operation ($|\mathrm{x}_2^i - \mathrm{x}_1^i| \le |\mathrm{q}^i - \mathrm{x}_1^i|$ and $|\mathrm{x}_2^j - \mathrm{x}_1^j| < |\mathrm{q}^j - \mathrm{x}_1^j|$) into *set membership test*. First, we have

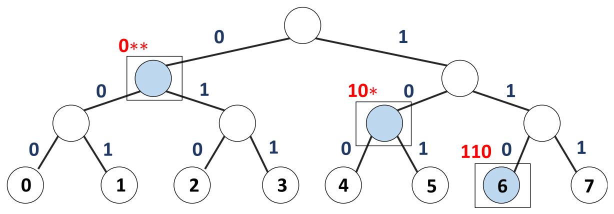


Fig. 4. Prefix encoding technique.

the following observation:

$$\begin{aligned}|x_2^i - x_1^i| \le |q^i - x_1^i| &\Leftrightarrow (x_2^i - x_1^i)^2 \le (q^i - x_1^i)^2\\ &\Leftrightarrow (x_2^i)^2 - 2x_2^i x_1^i \le (q^i)^2 - 2q^i x_1^i\\ &\Leftrightarrow (x_2^i)^2 - (q^i)^2 - 2x_1^i(x_2^i - q^i) \le 0\\ &\Leftrightarrow (x_2^i - q^i)(x_2^i - 2x_1^i + q^i) \le 0\end{aligned} \tag{4}$$

Therefore, checking $|x_2^i - x_1^i| \le |q^i - x_1^i|$ is equivalent to determining whether a value falls inside a range:

$$\begin{cases}\text{if } x_2^i \ge 2x_1^i \Rightarrow x_2^i \le q^i \Leftrightarrow x_2^i \in [0, q^i];\\ \text{if } x_2^i < 2x_1^i \Rightarrow \begin{cases}(x_2^i \le q^i) \wedge \big((2x_1^i - x_2^i) \le q^i\big)\\ \quad \Leftrightarrow \max(x_2^i, 2x_1^i - x_2^i) \in [0, q^i]\\ (x_2^i \ge q^i) \wedge \big((2x_1^i - x_2^i) \ge q^i\big)\\ \quad \Leftrightarrow \min(x_2^i, 2x_1^i - x_2^i) \in [q^i, T^i]\end{cases}\end{cases} \tag{5}$$

where T^i is the upper bound (domain) of the i-th dimension. If we encode $\{[0, q^i], [q^i, T^i]\}$ into $\{\mathcal{G}([0, q^i]), \mathcal{G}([q^i, T^i])\}$ and $\{x_2^i, \max(x_2^i, 2x_1^i - x_2^i), \min(x_2^i, 2x_1^i - x_2^i)\}$ into $\{\mathcal{F}(x_2^i), \mathcal{F}(\max(x_2^i, 2x_1^i - x_2^i)), \mathcal{F}(\min(x_2^i, 2x_1^i - x_2^i))\}$, respectively, we can convert *compute-then-compare* operation into *set membership test*. i.e.,

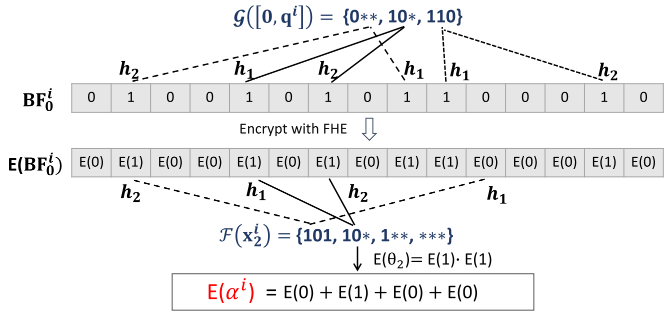
$$\begin{cases}\text{if } x_2^i \ge 2x_1^i : \alpha^i = \textbf{Bool}(x_2^i \in [0, q^i])\\ \text{if } x_2^i < 2x_1^i : \alpha^i = \textbf{Bool}(\max(x_2^i, 2x_1^i - x_2^i) \in [0, q^i])\\ \vee \textbf{Bool}(\min(x_2^i, 2x_1^i - x_2^i) \in [q^i, T^i]).\end{cases} \tag{6}$$

Next, we can adopt the bloom filter technique to check the set membership and integrate the FHE scheme to preserve the query privacy. Specifically, we map each element in $\mathcal{G}([0, q^i])$ into a bloom filter, denoted as BF_0^i, and $\mathcal{G}([q^i, T^i])$ into another bloom filter, denoted as BF_T^i. Then, we encrypt each element in BF_0^i and BF_T^i with fully homomorphic encryption. We denote the encrypted bloom filters as $\mathsf{E}(\mathsf{BF}_0^i)$ and $\mathsf{E}(\mathsf{BF}_T^i)$. After that, we can calculate $\mathsf{E}(\alpha^i)$ with the encrypted bloom filters and encoded set $\mathcal{F}(v)$, here $v \in \{x_2^i, \max(x_2^i, 2x_1^i - x_2^i), \min(x_2^i, 2x_1^i - x_2^i)\}$. We denote this calculation as $\mathsf{E}(\alpha^i) = \textbf{Calc}\Big(\mathsf{E}(\mathsf{BF}_\pi^i), \mathcal{F}(v)\Big)$, where $\pi = \{0, T\}$, and formally depict the calculation of **Calc**() in Algorithm 2.

Taking computing $\mathsf{E}(\alpha^i)$ under the case of $x_2^i \ge 2x_1^i$ as an example, we have $\mathsf{E}(\alpha^i) = \textbf{Calc}\Big(\mathsf{E}(\mathsf{BF}_0^i), \mathcal{F}(x_2^i)\Big)$. In particular, for the u-th element e_u in $\mathcal{F}(x_2^i)$ $(1 \le u \le |\mathcal{F}(x_2^i)|)$, we calculate $\mathsf{E}(\theta_u) = \prod_{j=1}^{l} \mathsf{E}(\mathsf{BF}_0^i[h_j(e_u)])$, as shown in line 3. Then, we have $\mathsf{E}(\alpha^i) = \mathsf{E}(\theta_1) \vee \mathsf{E}(\theta_2) \vee \cdots \vee \mathsf{E}(\theta_u) \vee \cdots \vee \mathsf{E}(\theta_{|\mathcal{F}(x_2^i)|})$. When one of elements in $\mathcal{F}(x_2^i)$ hits the

Algorithm 2: Calc$\big(\mathsf{E}(\mathsf{BF}), \mathcal{F}(v)\big)$

```
Input: An encrypted bloom filter, $\mathsf{E}(\mathsf{BF})$, in which $\mathsf{BF}$ is mapped
        by a set $\mathcal{G}(R)$, and $R$ is a range; A set $\mathcal{F}(v)$, where $v$ is an
        integer.
Output: An encrypted binary flag, $\mathsf{E}(\alpha)$.
1: $\mathsf{E}(\alpha) = \mathsf{E}(0)$
2: for each element $e_u \in \mathcal{F}(v)$ do
3:     $\mathsf{E}(\theta_u) = \prod_{j=1}^{l} \mathsf{E}(\mathsf{BF}[h_j(e_u)])$;
4:     $\mathsf{E}(\alpha) = \mathsf{E}(\alpha) + \mathsf{E}(\theta_u)$;
5: return $\mathsf{E}(\alpha)$;
```



𝒢([0, q^i]) = {0**, 10*, 110}
h2
h1
h2
h1
h1
h2
BF_0^i
0 1 0 0 1 0 1 0 1 1 0 0 0 1 0
Encrypt with FHE
E(BF_0^i)
E(0) E(1) E(0) E(0) E(1) E(0) E(1) E(0) E(1) E(1) E(0) E(0) E(0) E(1) E(0)
h2
h1
h2
h1
ℱ(x_2^i) = {101, 10*, 1**, ***}
E(θ2)= E(1)· E(1)
E(α^i) = E(0) + E(1) + E(0) + E(0)

Fig. 5. An example of computing $\mathsf{E}(\alpha^i)$ with assumptions: i) $q^i = 6$, $x_2^i = 5$, and $x_2^i \ge 2x_1^i$; ii) two independent hash functions $\{h_1, h_2\}$ for BF_0.

bloom filter BF_0^i (mapped by $\mathcal{G}([0, q^i])$), $\alpha^i = 1$, otherwise $\alpha^i = 0$. Since $\mathcal{F}(x_2^i)$ and $\mathcal{G}([0, q^i])$ only have one common element when $x_2^i \in [0, q^i]$, we can optimize the calculation of $\mathsf{E}(\alpha^i)$ by directly adding $\mathsf{E}(\theta_u)$ together, i.e., $\mathsf{E}(\alpha^i) = \sum_{u=1}^{|\mathcal{F}(x_2^i)|} \mathsf{E}(\theta_u)$ showing in line 4. Fig. 5 shows an example of computing $\mathsf{E}(\alpha^i)$ with $q^i = 6$ and $x_2^i = 5$. According to the property of bloom filter, only $\mathcal{F}(x_2^i) \cap \mathcal{G}([0, q^i]) \neq \emptyset$ (with acceptable *false positive*), we have $\mathsf{E}(\alpha^i) = \mathsf{E}(1)$. Since $\mathcal{F}(x_2^i) \cap \mathcal{G}([0, q^i]) \neq \emptyset \Leftrightarrow x_2^i \in [0, q^i]$, we have $\mathsf{E}(\alpha^i) = \mathsf{E}(1) \Leftrightarrow |x_2^i - x_1^i| \le |q^i - x_1^i|$ (when $x_2^i \ge 2x_1^i$) according to Eq. (6).

Regarding $|x_2^j - x_1^j| < |q^j - x_1^j|$, it is equivalent to checking $(x_2^i - q^i)(x_2^i - 2x_1^i + q^i) < 0$. Since we want to output 0 if $(x_2^i - q^i)(x_2^i - 2x_1^i + q^i) < 0$, we can transform it into "$(x_2^i - q^i)(x_2^i - 2x_1^i + q^i) \ge 0 \Leftrightarrow \mathsf{E}(\beta^i) = \mathsf{E}(1)$". Similarly, we have the following discussions Eq. (7), as shown at the bottom of the next page, and

$$\begin{cases}\text{if } x_2^i \ge 2x_1^i : \beta^i = \textbf{Bool}(x_2^i \in [q^i, T^i])\\ \text{if } x_2^i < 2x_1^i :\\ \beta^i = \Big(\textbf{Bool}(x_2^i \in [q^i, T^i]) \wedge \textbf{Bool}((2x_1^i - x_2^i) \in [0, q^i])\Big)\\ \vee \Big(\textbf{Bool}(x_2^i \in [0, q^i]) \wedge \textbf{Bool}((2x_1^i - x_2^i) \in [q^i, T^i])\Big).\end{cases} \tag{8}$$

Since $\textbf{Bool}(x_2^i \in [q^i, T^i]) \vee \textbf{Bool}(x_2^i \in [0, q^i]) = 1$, we can simplify Eq. (8) into the following equation according to the distributive law of logical expressions, Eq. (9), as shown at the bottom of the next page. Finally, with the same approach to compute $\mathsf{E}(\alpha^i)$, we can also obtain $\mathsf{E}(\beta^i)$ based on Eq. (9).

3) Formal Description of Our SDRD Scheme: Given two data records $\{x_1, x_2\}$ and a query request q, our SDRD

Algorithm 3: SDRD Check Algorithm

```
Input: Two $d$-dimensional data records, $\{\mathtt{x}_1, \mathtt{x}_2\}$; A query token of the query
       request $\mathtt{q}$, $\mathrm{QT}_{\mathtt{q}} = \{\mathsf{E}(\mathsf{BF}_0^i), \mathsf{E}(\mathsf{BF}_{\mathrm{T}}^i) \mid i \in [1, d]\}$;
Output: An encrypted dynamic reverse dominance result, $\mathsf{E}(\delta)$.
1:  for $i$-th dimension $i \in [1, d]$ do
2:      if $\mathtt{x}_2^i \geq 2\mathtt{x}_1^i$ then
3:          build $\mathcal{F}(\mathtt{x}_2^i)$;
4:          $\mathsf{E}(\alpha^i) \leftarrow$ Calc$\left(\mathsf{E}(\mathsf{BF}_0^i), \mathcal{F}(\mathtt{x}_2^i)\right)$
5:          $\mathsf{E}(\beta^i) \leftarrow$ Calc$\left(\mathsf{E}(\mathsf{BF}_{\mathrm{T}}^i), \mathcal{F}(\mathtt{x}_2^i)\right)$
6:
        else
7:          build $\mathcal{F}(\mathtt{x}_2^i)$, $\mathcal{F}(2\mathtt{x}_1^i - \mathtt{x}_2^i)$;
8:          if $\mathtt{x}_2^i \geq 2\mathtt{x}_1^i - \mathtt{x}_2^i$ then
9:              $\mathsf{E}(\alpha^i) \leftarrow$ Calc$\left(\mathsf{E}(\mathsf{BF}_0^i), \mathcal{F}(\mathtt{x}_2^i)\right)$
10:                 $\vee$ Calc$\left(\mathsf{E}(\mathsf{BF}_{\mathrm{T}}^i), \mathcal{F}(2\mathtt{x}_1^i - \mathtt{x}_2^i)\right)$
11:             $\mathsf{E}(\beta^i) \leftarrow$ Calc$\left(\mathsf{E}(\mathsf{BF}_{\mathrm{T}}^i), \mathcal{F}(\mathtt{x}_2^i)\right)$
12:                 $\wedge$ Calc$\left(\mathsf{E}(\mathsf{BF}_0^i), \mathcal{F}(2\mathtt{x}_1^i - \mathtt{x}_2^i)\right)$
13:
            else
14:             $\mathsf{E}(\alpha^i) \leftarrow$ Calc$\left(\mathsf{E}(\mathsf{BF}_{\mathrm{T}}^i), \mathcal{F}(\mathtt{x}_2^i)\right)$
15:                 $\vee$ Calc$\left(\mathsf{E}(\mathsf{BF}_0^i), \mathcal{F}(2\mathtt{x}_1^i - \mathtt{x}_2^i)\right)$
16:             $\mathsf{E}(\beta^i) \leftarrow$ Calc$\left(\mathsf{E}(\mathsf{BF}_0^i), \mathcal{F}(\mathtt{x}_2^i)\right)$
17:                 $\wedge$ Calc$\left(\mathsf{E}(\mathsf{BF}_{\mathrm{T}}^i), \mathcal{F}(2\mathtt{x}_1^i - \mathtt{x}_2^i)\right)$
18: $\mathsf{E}(\delta) \leftarrow \mathsf{E}(\prod_{i=1}^d \alpha^i) \cdot \mathsf{E}(1 - \prod_{i=1}^d \beta^i)$;
    return $\mathsf{E}(\delta)$;
```

scheme can be formally defined as $\prod_{\mathsf{SDRD}} = \{\mathsf{SDRD.Setup}, \mathsf{SDRD.TokenGen}, \mathsf{SDRD.Check}\}$.

a) $\mathsf{SDRD.Setup}(\lambda)$*:* Given a security parameter λ, the setup algorithm outputs a pair of key (pk, sk) for an FHE scheme, where pk is the public key, and sk is the secret key. Then, it chooses l independent hash functions $\mathcal{H} = \{h_1, h_2, \cdots, h_l\}$.

b) $\mathsf{SDRD.TokenGen}(\mathtt{q}, pk, \mathcal{H})$*:* On input of a d-dimensional query request $\mathtt{q}$, the public key pk, and a set of hash functions $\mathcal{H}$, the token generation algorithm outputs $\mathsf{E}(\mathsf{BF}_0^i)$ and $\mathsf{E}(\mathsf{BF}_{\mathrm{T}}^i)$ for the i-th dimension and takes encrypted bloom filters as the query token of $\mathtt{q}$, denoted as $\mathrm{QT}_{\mathtt{q}}$, i.e., $\mathrm{QT}_{\mathtt{q}} = \{\mathsf{E}(\mathsf{BF}_0^i), \mathsf{E}(\mathsf{BF}_{\mathrm{T}}^i) \mid i \in [1, d]\} = \mathsf{SDRD.TokenGen}(\mathtt{q}, pk, \mathcal{H})$.

c) $\mathsf{SDRD.Check}(\mathtt{x}_1, \mathtt{x}_2, QT_{\mathtt{q}}, \mathcal{H})$*:* On input of two d-dimensional data records $\{\mathtt{x}_1, \mathtt{x}_2\}$, the query token $\mathrm{QT}_{\mathtt{q}}$, and a set of hash functions $\mathcal{H}$, the SDRD check algorithm determines whether $\mathtt{x}_2 \prec_{\mathtt{x}_1} \mathtt{q}$ or not. If yes, it outputs $\mathsf{E}(\delta) = \mathsf{E}(1)$, otherwise $\mathsf{E}(\delta) = \mathsf{E}(0)$. Briefly, there are two main steps for the algorithm: i) calculating $\mathsf{E}(\alpha^i)$ and $\mathsf{E}(\beta^i)$ with **Calc**(); ii) calculating $\mathsf{E}(\delta)$ according to Eq. (2) after obtaining $\mathsf{E}(\alpha^i)$ and $\mathsf{E}(\beta^i)$. The detailed process of the SDRD check algorithm is shown in Algorithm 3. Note that, in the algorithm, **Calc()** is performed following Algorithm 2, the "AND" operation ($\wedge$) is achieved by $\mathsf{E}(a) \wedge \mathsf{E}(b) = \mathsf{E}(a \cdot b)$, and the "OR" operation ($\vee$) is calculated by $\mathsf{E}(a) \vee \mathsf{E}(b) = \mathsf{E}(a + b - a \cdot b)$, where $\mathsf{E}(a)$ and $\mathsf{E}(b)$ are the returned data of **Calc()**.

C. Our PPARS Scheme

Based on the above SDRD scheme, we present our privacy-preserving aggregate reverse skyline query (PPARS) scheme that allows a client $\mathcal{C}$ to obtain the aggregated values from the server $\mathcal{S}$ without leaking query requests, query results, and access patterns. Specifically, our PPARS scheme mainly includes three phases: i) Query Token Report; ii) Privacy-Preserving ARS Search; iii) Result Recovery.

1) Query Token Report: Assume a client $\mathcal{C}$ has a query set $\mathcal{Q} = \{\mathtt{q}_t \mid 1 \leq t \leq k\}$ that contains k query requests. In this phase, $\mathcal{C}$ has the following three steps:

Step-1. The client $\mathcal{C}$ employs the $\mathsf{SDRD.Setup}(\lambda)$ algorithm to generate a set of hash functions $\mathcal{H}$ and (pk, sk) of the FHE scheme.

Step-2. For each query request $\mathtt{q}_t \in \mathcal{Q}$, the client $\mathcal{C}$ calls the $\mathsf{SDRD.TokenGen}(\mathtt{q}_t, pk, \mathcal{H})$ algorithm to generate the corresponding token: $\mathrm{QT}_{\mathtt{q}_t}$. We define the query token for the query set $\mathcal{Q}$ as $\mathrm{QT} = \{\mathrm{QT}_{\mathtt{q}_t} \mid 1 \leq t \leq k\}$. To prevent the server from inferring the query request $\mathtt{q}_t$ by the size of bloom filters, $\mathcal{C}$ sets all bloom filters to the same size. First,

$$\begin{cases} \text{if } \mathtt{x}_2^i \geq 2\mathtt{x}_1^i \Rightarrow \mathtt{x}_2^i \geq \mathtt{q}^i \Leftrightarrow \mathtt{x}_2^i \in [\mathtt{q}^i, \mathtt{T}^i]; \\ \\ \text{if } \mathtt{x}_2^i < 2\mathtt{x}_1^i \Rightarrow \begin{cases} (\mathtt{x}_2^i \geq \mathtt{q}^i) \wedge \left((2\mathtt{x}_1^i - \mathtt{x}_2^i) \leq \mathtt{q}^i\right) \\ \quad \Leftrightarrow (\mathtt{x}_2^i \in [\mathtt{q}^i, \mathtt{T}^i]) \wedge \left((2\mathtt{x}_1^i - \mathtt{x}_2^i) \in [0, \mathtt{q}^i]\right) \\ (\mathtt{x}_2^i \leq \mathtt{q}^i) \wedge \left((2\mathtt{x}_1^i - \mathtt{x}_2^i) \geq \mathtt{q}^i\right) \\ \quad \Leftrightarrow (\mathtt{x}_2^i \in [0, \mathtt{q}^i]) \wedge \left((2\mathtt{x}_1^i - \mathtt{x}_2^i) \in [\mathtt{q}^i, \mathtt{T}^i]\right). \end{cases} \end{cases} \tag{7}$$

$$\begin{cases} \text{if } \mathtt{x}_2^i \geq 2\mathtt{x}_1^i : \beta^i = \mathbf{Bool}(\mathtt{x}_2^i \in [\mathtt{q}^i, \mathtt{T}^i]) \\ \text{if } \mathtt{x}_2^i < 2\mathtt{x}_1^i : \begin{cases} \text{if } \mathtt{x}_2^i \geq 2\mathtt{x}_1^i - \mathtt{x}_2^i : \\ \quad \Rightarrow \beta^i = \mathbf{Bool}(\mathtt{x}_2^i \in [\mathtt{q}^i, \mathtt{T}^i]) \\ \qquad\qquad \wedge \mathbf{Bool}((2\mathtt{x}_1^i - \mathtt{x}_2^i) \in [0, \mathtt{q}^i]) \\ \text{if } \mathtt{x}_2^i < 2\mathtt{x}_1^i - \mathtt{x}_2^i \\ \quad \Rightarrow \beta^i = \mathbf{Bool}(\mathtt{x}_2^i \in [0, \mathtt{q}^i]) \\ \qquad\qquad \wedge \mathbf{Bool}((2\mathtt{x}_1^i - \mathtt{x}_2^i) \in [\mathtt{q}^i, \mathtt{T}^i]). \end{cases} \end{cases} \tag{9}$$

Algorithm 4: Privacy-Preserving ARS Search

```
Input: A $d$-dimensional dataset $\mathcal{X} = \{x_i \mid 1 \le i \le n\}$; A query token
       $QT = \{QT_{q_t} \mid 1 \le t \le k\}$; A set of hash functions
       $\mathcal{H}$; The public key of FHE, $pk$.
Output: A set containing $k$ encrypted aggregated values,
       $E(AS) = \{E(s_t) \mid 1 \le t \le k\}$.
1:  $E(AS) \leftarrow \varnothing$;
2:  for each token $QT_{q_t} \in QT$ do
3:      $E(s_t) \leftarrow E(0)$;                                  ▷Initialization
4:      for each data point $x_i \in \mathcal{X}$ do
5:          $E(f_i) \leftarrow E(1)$;                              ▷Initialization
6:          for $x_j \in \mathcal{X}$ and $x_j \neq x_i$ do
7:              $E(\delta) \leftarrow SDRD.Check(x_i, x_j, QT_{q_t}, \mathcal{H})$;
8:              $E(f_i) \leftarrow E(f_i) \cdot (E(1) + E(-1) \cdot E(\delta)) = E(f_i \cdot (1-\delta))$;
9:          $E(s_t) \leftarrow E(s_t) + E(f_i) = E(s_t + f_i)$;
10:     $E(AS).add(E(s_t))$;
11: return $E(AS)$.
```

the client $\mathcal{C}$ determines an acceptable false positive probability f_p. Then, $\mathcal{C}$ calculates the maximum bit length of the query request q_t, denoted as $\phi_{\max}$, and sets the number of elements mapped to the bloom filters as $m = 2\phi_{\max} - 2$. Finally, the size of bloom filters can be set as $\eta = \lceil l \cdot m / \ln 2 \rceil$, where $l = -\log_2 f_p$.

Step-3. The client $\mathcal{C}$ puts the query token QT, the public key pk, and the hash function set $\mathcal{H}$ together, i.e., $QT||pk||\mathcal{H}$, and reports it to the server $\mathcal{S}$.

2) Privacy-Preserving ARS Search: Upon receiving $QT||pk||\mathcal{H}$, the server $\mathcal{S}$ conducts the ARS search over the dataset $\mathcal{X}$ to obtain k encrypted aggregated values for QT. The core part of this phase is to use the $\mathsf{SDRD.Check}(x_i, x_j, QT_{q_t}, \mathcal{H})$ algorithm to determine the dynamic reverse dominance relation for each query token $QT_{q_t} \in QT$. We depict the detailed process in Algorithm 4. For each data point $x_i \in \mathcal{X}$, the server $\mathcal{S}$ checks whether $\exists x_j \in \mathcal{X}$ such that x_j dynamically dominates q with regard to x_i, i.e., $x_j \prec_{x_i} q$. If yes, our $\mathsf{SDRD.Check}()$ algorithm outputs $E(\delta) = E(1)$, otherwise $E(\delta) = E(0)$ (see details in Algorithm 3). Recalling Definition 3, it indicates that if $\nexists x_j \in \mathcal{X}$ dynamically dominates q with regard to x_i, x_i would be a reverse skyline point. Therefore, in Algorithm 4, we initialize an encrypted flag $E(f_i)$ (line 5) to indicate: i) if $\exists x_j \in \mathcal{X}$ dominates x_i, i.e., x_i is not a reverse skyline point, $E(f_i) = E(0)$ due to existing $E(1-\delta) = E(0)$; ii) if $\nexists x_j \in \mathcal{X}$ dominates x_i, i.e., x_i is a reverse skyline point, $E(f_i) = E(1)$ as all $E(1-\delta)$ are $E(1)$. Therefore, the server $\mathcal{S}$ can obtain the aggregated reverse skyline query value $E(s_t)$ for QT_{q_t} by adding up all $E(f_i)$ (line 9). Finally, the server $\mathcal{S}$ sends $E(AS) = \{E(s_t) \mid 1 \le t \le k\}$ to the client $\mathcal{C}$.

3) Result Recovery: After receiving $E(AS) = \{E(s_t) \mid 1 \le t \le k\}$ from $\mathcal{S}$, the client $\mathcal{C}$ can easily recover $AS = \{s_t \mid 1 \le t \le k\}$ by using his/her secret key sk.

a) Support multiple clients: In the above process, we can see that the key pair (pk, sk) is generated by the client $\mathcal{C}$, and the server $\mathcal{S}$ only needs to know pk for the query request. If a new client would like to enjoy the privacy-preserving ARS queries, it can generate its own key pair and send the corresponding public key to the server $\mathcal{S}$. Since the key pair is generated by the client, and only the client itself knows the secret key, our proposed PPARS scheme can support multiple clients without compromising privacy or incurring key management issues.

b) Improve computational efficiency: In the privacy-preserving ARS search phase, the main computational costs are from the **Calc**() operation in the $\mathsf{SDRD.Check}()$ algorithm. Since the value of v ($v \in \{x_2^i, 2x_1^i - x_2^i\}$) may occur many times in a given dataset, the server $\mathcal{S}$ can build a hash table for a query token QT and store the result of $\mathbf{Calc}\Big(E(BF_\pi^i), \mathcal{F}(v)\Big)$ when performing the privacy-preserving ARS search, where the key is $v||i||\pi$, and the corresponding value is $\mathbf{Calc}\Big(E(BF_\pi^i), \mathcal{F}(v)\Big)$. Thus, before conducting the **Calc**() operation, $\mathcal{S}$ can first retrieve the hash table by constructing $v||i||\pi$. If there already exists the result of $\mathbf{Calc}\Big(E(BF_\pi^i), \mathcal{F}(v)\Big)$, $\mathcal{S}$ can directly use the result instead of calculating it again.

D. Our CE-PPARS Scheme

In our PPARS scheme, for a query request q, the client $\mathcal{C}$ needs to build $QT_q = \{E(BF_0^i), E(BF_T^i) \mid i \in [1, d]\}$ and sends it to the server $\mathcal{S}$. If we suppose each bloom filter has a length of η, and there are k query requests, the client $\mathcal{C}$ would send $2d \cdot \eta \cdot k$ ciphertexts of FHE. To reduce the number of delivered ciphertexts, we propose a communication-efficient PPARS scheme, denoted as CE-PPARS. The main idea is to pack several ciphertexts into one by leveraging the Lagrange polynomial interpolation [25]. Compared to the PPARS scheme, our CE-PPARS scheme has two differences.

1) Query Token Report Phase: Before encrypting a bloom filter into its encrypted version, the client $\mathcal{C}$ packs w elements (they are 0 or 1) into one, i.e., converting the bit sequence in the package into the corresponding integer, as shown in Fig. 6. After encrypting these packed integers, the client $\mathcal{C}$ constructs new encrypted bloom filters $E(\widehat{BF}_\pi^i)$ and further generates the query token QT with these new bloom filters. Finally, the client $\mathcal{C}$ sends $QT||pk||\mathcal{H}||w$ to $\mathcal{S}$.

2) Privacy-Preserving ARS Search Phase: Upon receiving $QT||pk||\mathcal{H}||w$, the server $\mathcal{S}$ has the following two steps:

Step-1. With w, $\mathcal{S}$ builds $2^w - 1$ polynomial functions from $f_{\underbrace{00\ldots01}_{w}}(x)$ to $f_{\underbrace{11\ldots1}_{w}}(x)$, each of which is built by the Lagrange interpolation approach [25] and has the degree with $2^w - 1$. To clearly show how to interpolate a polynomial function, we take building $f_{1010}(x)$ in Fig. 6 as an example, which is interpolated at nodes $\{(0,0), (1,0), (2,0), (3,0), (4,0), (5,0), (6,0), (7,0), (8,0), (9,0), (10,1), (11,1), (12,0), (13,0), (14,1), (15,1)\}$. For x dimension, the nodes' values are from 0 (resp. 0000) to 15 (resp. 1111). For y dimension, since the subscript of $f_{1010}(x)$ is 1010, if a node's x value satisfies $1{*}1{*}$, the corresponding y value is 1, otherwise it is 0, where $*$ is 0 or 1. In this example, only 10, 11, 14, 15 have their bit sequences satisfying $1{*}1{*}$.

Step-2. When conducting $\mathbf{Calc}\Big(E(\widehat{BF}_\pi^i), \mathcal{F}(v)\Big)$ in the $\mathsf{SDRD.Check}()$ algorithm, the server $\mathcal{S}$ first constructs a temporary bloom filter for each element e in $\mathcal{F}(v)$, denoted as TBF. All elements in TBF are initialized as $*$, which

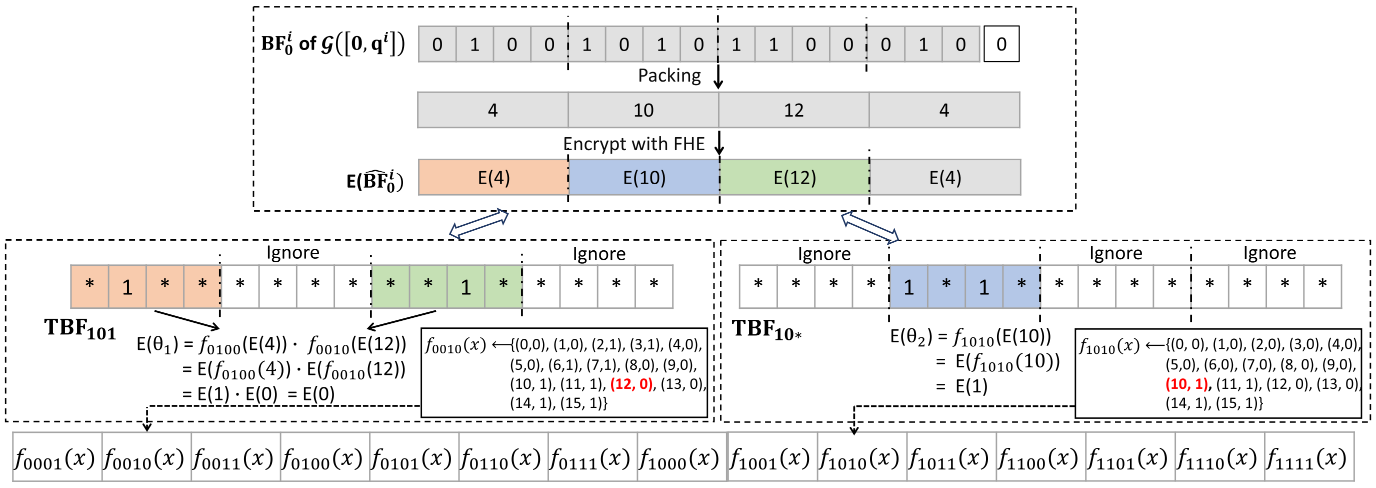

$\mathbf{BF}_0^i$ of $\mathcal{G}([0, q^i])$
0 1 0 0 1 0 1 0 1 1 0 0 0 1 0 0
Packing
4 10 12 4
Encrypt with FHE
$\mathbf{E}(\widehat{\mathbf{BF}}_0^i)$
E(4) E(10) E(12) E(4)
Ignore
Ignore
Ignore
Ignore
Ignore
$\mathbf{TBF}_{101}$
$\mathbf{TBF}_{10*}$
$E(\theta_1) = f_{0100}(E(4)) \cdot f_{0010}(E(12))$
$= E(f_{0100}(4)) \cdot E(f_{0010}(12))$
$= E(1) \cdot E(0) = E(0)$
$f_{0010}(x) \leftarrow$ {(0,0), (1,0), (2,1), (3,1), (4,0), (5,0), (6,1), (7,1), (8,0), (9,0), (10, 1), (11, 1), (12, 0), (13, 0), (14, 1), (15, 1)}
$E(\theta_2) = f_{1010}(E(10))$
$= E(f_{1010}(10))$
$= E(1)$
$f_{1010}(x) \leftarrow$ {(0, 0), (1,0), (2,0), (3,0), (4,0), (5,0), (6,0), (7,0), (8, 0), (9,0), (10, 1), (11, 1), (12, 0), (13, 0), (14, 1), (15, 1)}
$f_{0001}(x)$ $f_{0010}(x)$ $f_{0011}(x)$ $f_{0100}(x)$ $f_{0101}(x)$ $f_{0110}(x)$ $f_{0111}(x)$ $f_{1000}(x)$ $f_{1001}(x)$ $f_{1010}(x)$ $f_{1011}(x)$ $f_{1100}(x)$ $f_{1101}(x)$ $f_{1110}(x)$ $f_{1111}(x)$

Fig. 6. An example of our CE-PPARS scheme, where $w = 4$, TBF_{101} indicates the temporary bloom filter for the element 101, and TBF_{10*} is for the element $10*$. We have the same $\mathsf{E}(\theta_1)$ and $\mathsf{E}(\theta_2)$ as that in the example of our PPARS scheme.

means this position is ignored. With $\mathcal{H}$, the server $\mathcal{S}$ sets $\mathsf{TBF}[h_i(e)] = 1$, where $h_i \in \mathcal{H}$. After that, $\mathcal{S}$ treats every w elements in TBF as a group: i) if all elements in the group are $*$, $\mathcal{S}$ directly ignores the group; ii) if there exists 1 in the group, $\mathcal{S}$ needs to retrieve the corresponding polynomial for the group by matching the group's bit sequence (replacing $*$ with 0) to functions' subscripts. In Fig. 6, as the second group in TBF_{10*} is $1*1*$, we make $f_{1010}(x)$ be the group's polynomial function to calculate $\mathsf{E}(\theta_2)$. It is worth noting, since a polynomial function is composed of addition and multiplication, we can use the homomorphic properties of FHE to calculate the polynomial function over ciphertexts. In this example, we have $\mathsf{E}(\theta_2) = \mathsf{E}(1)$, as $\mathsf{E}(\theta_2) = f_{1010}(\mathsf{E}(10)) = \mathsf{E}(f_{1010}(10))$, in which $f_{1010}(x)$ must go through the node $(10, 1)$. Afterward, the server $\mathcal{S}$ can use the same approach of our PPARS scheme to calculate $\mathsf{E}(\mathsf{AS})$.

Compared with the PPARS scheme, our CE-PPARS scheme packs w elements into one for the encrypted bloom filters. In the privacy-preserving ARS search phase, the key difference between these two schemes is to calculate $\mathsf{E}(\theta)$ for each element e in $\mathcal{F}(v)$. As a result, the correctness of CE-PPARS is equivalent to proving $\mathsf{E}(\theta) = \mathsf{E}(1)$ if $e \in \mathcal{G}(R)$.

a) Correctness: We say our CE-PPARS scheme is correct if existing $\mathcal{F}(v)$'s element $e \in \mathcal{G}(R)$, we have $\mathsf{E}(\theta) = \mathsf{E}(1)$, otherwise $\mathsf{E}(\theta) = \mathsf{E}(0)$.

Proof: In our CE-PPARS scheme, after mapping $\mathcal{G}(R)$ into BF, we will first encrypt it into $\mathsf{E}(\mathsf{BF})$ and then generate $\mathsf{E}(\widehat{\mathsf{BF}})$ by packing w elements into one. If we assume $\mathsf{E}(g_t)$, where $t \in [1, n]$ is the t-th group of $\mathsf{E}(\widehat{\mathsf{BF}})$ and n is the number of groups, we have $\mathsf{E}(\theta) = \prod_{t=1}^{n} f_{z_{w-1}\cdots z_1 z_0}(\mathsf{E}(g_t))$. Given an element e in $\mathcal{F}(v)$, if $e \in \mathcal{G}(R)$, we have $\mathsf{BF}[h_i(e)] = 1$ for each $h_i \in \mathcal{H}$. It means, for each group $\mathsf{E}(g_t)$ that contains the position $h_i(e)$, the underlying value g_t is mapped to 1, otherwise 0. In our CE-PPARS scheme, $f_{z_{w-1}\cdots z_1 z_0}(x)$ is interpolated at nodes $\{(a, b) \mid a \in [0, 2^w - 1]\}$, in which $b = 1$ if $a = \sum_{j=0}^{w-1}(2^j \cdot z_j)$, otherwise $b = 0$. Here, z_j is set to $*$ if $z_j = 0$, and $*$ could be 0 or 1. As a result, if $e \in \mathcal{G}(R)$, we have $f_{z_{w-1}\cdots z_1 z_0}(\mathsf{E}(g_t)) = \mathsf{E}(f_{z_{w-1}\cdots z_1 z_0}(g_t)) = \mathsf{E}(1)$ for the group $\mathsf{E}(g_t)$ that contains the position $h_i(e)$. If $\mathsf{BF}[h_i(e)] = 0$ for any $h_i \in \mathcal{H}$, it means $e \notin \mathcal{G}(R)$, and the corresponding group can be mapped to 0, i.e., $f_{z_{w-1}\cdots z_1 z_0}(\mathsf{E}(g_t)) = \mathsf{E}(f_{z_{w-1}\cdots z_1 z_0}(g_t)) = \mathsf{E}(0)$. Thus, $e \in \mathcal{G}(R) \Leftrightarrow \mathsf{E}(\theta) = \prod_{t=1}^{n} f_{z_{w-1}\cdots z_1 z_0}(\mathsf{E}(g_t)) = \mathsf{E}(1)$. □

b) Remark: Our CE-PPARS scheme can reduce the number of ciphertexts from $2d \cdot \eta \cdot k$ to $\lceil \frac{2d \cdot \eta \cdot k}{w} \rceil$. Although it may incur extra computational costs in calculating polynomial functions, the server $\mathcal{S}$ only needs to calculate at most l polynomial functions for each bloom filter, where l is the number of hash functions, i.e., $l = |\mathcal{H}|$, and most of groups will be ignored.

V. Security Analysis

In this section, we discuss the security properties of the proposed PPARS and CE-PPARS schemes. Specifically, following our design goals, we focus on how the proposed schemes can preserve the privacy of query requests, query results, and access patterns. Since both PPARS and CE-PPARS schemes are built on our SDRD scheme, we will first prove the security of the SDRD scheme and then discuss the privacy preservation of our PPARS and CE-PPARS schemes.

Before delving into the detailed proofs, we briefly review the security model for securely realizing an ideal functionality in the presence of the static semi-honest adversary [16]. In our models (Section II), since only the server $\mathcal{S}$ is semi-honest (i.e., honest-but-curious), we will prove that our schemes are secure against the server $\mathcal{S}$.

Real world model: The real world execution of a scheme Π takes place in $\mathcal{S}$ and an adversary $\mathcal{A}$, who corrupts $\mathcal{S}$. Assume that x indicates the input of Π and y is the auxiliary input, e.g., the length of ciphertexts. With the inputs of x and y, the execution of Π under $\mathcal{A}$ in the real world model is defined as:

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

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

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

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

Definition 4 Security against semi-honest adversary: Let $\mathcal{F}$ *be a deterministic functionality and* Π *be a scheme in* $\mathcal{S}$*. We say that* Π *securely realizes* $\mathcal{F}$ *if there exists* $\texttt{Sim}$ *of PPT (Probabilistic Polynomial Time) transformations (where* $\texttt{Sim} = \texttt{Sim}(\mathcal{A})$*) such that for semi-honest PPT adversary* $\mathcal{A}$*, for* x *and* y*, for* $\mathcal{S}$ *holds:*

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

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

A. SDRD Scheme Is Privacy-Preserving

First, we use Definition 4 to demonstrate that our SDRD scheme can guarantee the security of the query request $\mathtt{q}$ and the output δ of SDRD against $\mathcal{S}$.

Theorem 1: The SDRD scheme securely computes the dynamic reverse dominance relation in the presence of semi-honest adversary $\mathcal{A}$*.*

Proof: Here, we show how to construct the simulator. $\texttt{Sim}$ randomly chooses two data records $\{\mathtt{x}'_1, \mathtt{x}'_2\}$ and a query request $\mathtt{q}'$. Then, $\texttt{Sim}$ simulates $\mathcal{A}$ as follows: i) it encodes $\mathtt{q}'$ into $\{\mathcal{G}([0, \mathtt{q}'^i]), \mathcal{G}([\mathtt{q}'^i, \mathrm{T}^i]) \mid i \in [1, d]\}$ and builds bloom filters $\{\mathsf{BF}'^i_0, \mathsf{BF}'^i_\mathrm{T} \mid i \in [1, d]\}$; ii) it encrypts each element in bloom filters with the encryption algorithm of FHE, generating $\{\mathsf{E}(\mathsf{BF}'^i_0), \mathsf{E}(\mathsf{BF}'^i_\mathrm{T}) \mid i \in [1, d]\}$; iii) it generates α'^i and β'^i by determining the relations between $|\mathtt{x}'^i_2 - \mathtt{x}'^i_1|$ and $|\mathtt{q}'^i - \mathtt{x}'^i_1|$ and further obtains $\{\delta'_1, \delta'_2, \delta'\}$ with the logical expressions in Fig. 3(a); iv) it encrypts $\{\alpha'^i, \beta'^i, \delta'_1, \delta'_2, \delta' \mid i \in [1, d]\}$ with the encryption algorithm of FHE, generating $\{\mathsf{E}(\alpha'^i), \mathsf{E}(\beta'^i), \mathsf{E}(\delta'_1), \mathsf{E}(\delta'_2), \mathsf{E}(\delta') \mid i \in [1, d]\}$; v) it outputs $\{\mathsf{E}(\mathsf{BF}'^i_0), \mathsf{E}(\mathsf{BF}'^i_\mathrm{T}) \mid i \in [1, d]\}$ and $\{\mathsf{E}(\alpha'^i), \mathsf{E}(\beta'^i), \mathsf{E}(\delta'_1), \mathsf{E}(\delta'_2), \mathsf{E}(\delta') \mid i \in [1, d]\}$ as $\mathcal{A}$'s ideal view. In the real execution, $\mathcal{A}$ receives $\{\mathtt{x}_1, \mathtt{x}_2, \mathsf{E}(\mathsf{BF}^i_0), \mathsf{E}(\mathsf{BF}^i_\mathrm{T}) \mid i \in [1, d]\}$, as presented in Section IV-B.3, and obtains $\{\mathsf{E}(\alpha^i), \mathsf{E}(\beta^i), \mathsf{E}(\delta_1), \mathsf{E}(\delta_2), \mathsf{E}(\delta) \mid i \in [1, d]\}$. We can see that the semantic security of FHE scheme guarantees that the views of $\mathcal{A}$ in the real and the ideal worlds are indistinguishable. Specifically, i) $\mathcal{A}$ cannot distinguish $\{\mathsf{E}(\mathsf{BF}'^i_0), \mathsf{E}(\mathsf{BF}'^i_\mathrm{T}) \mid i \in [1, d]\}$ and $\{\mathsf{E}(\mathsf{BF}^i_0), \mathsf{E}(\mathsf{BF}^i_\mathrm{T}) \mid i \in [1, d]\}$; ii) $\mathcal{A}$ cannot distinguish $\{\mathsf{E}(\alpha'^i), \mathsf{E}(\beta'^i), \mathsf{E}(\delta'_1), \mathsf{E}(\delta'_2), \mathsf{E}(\delta') \mid i \in [1, d]\}$ and $\{\mathsf{E}(\alpha^i), \mathsf{E}(\beta^i), \mathsf{E}(\delta_1), \mathsf{E}(\delta_2), \mathsf{E}(\delta) \mid i \in [1, d]\}$ due to the security of FHE. Thus, our SDRD scheme can securely compute the dynamic reverse dominance relation without leaking the query request and the corresponding calculation result. □

B. PPARS and CE-PPARS Schemes Are Privacy-Preserving

In this section, we show that our PPARS and CE-PPARS schemes can preserve the privacy of query requests, query results, and access patterns against $\mathcal{S}$.

Theorem 2: The PPARS scheme securely computes the aggregate set $\mathsf{AS} = \{\mathsf{s}_t \mid 1 \le t \le k\}$ *without leaking query requests and query results to* $\mathcal{S}$*.*

Proof: For the work process of the simulator, $\texttt{Sim}$ simulates $\mathcal{A}$ as follows: i) it randomly chooses a set of encrypted bloom filters as query token $\mathrm{QT}' = \{\mathrm{QT}'_{\mathtt{q}_t} \mid 1 \le t \le k\}$, where $\mathrm{QT}'_{\mathtt{q}_t} = \{\mathsf{E}(\mathsf{BF}^{i\prime}_0), \mathsf{E}(\mathsf{BF}^{i\prime}_\mathrm{T}) \mid i \in [1, d]\}$; ii) it generates the dominating flag $\mathsf{E}(\delta')$ with the randomly selected datasets; iii) based on $\mathsf{E}(\delta')$, it computes $\mathsf{E}(f')$ and an aggregate set $\mathsf{E}(\mathsf{AS}') = \{\mathsf{E}(s'_1), \mathsf{E}(s'_2), \cdots, \mathsf{E}(s'_k)\}$; iv) it outputs $\{\mathrm{QT}'_\mathtt{q}, \mathsf{E}(\delta'), \mathsf{E}(f'), \mathsf{E}(\mathsf{AS}')\}$ as $\mathcal{A}$'s ideal view. In the real execution, according to Algorithm 4, $\mathcal{A}$ receives the query token $\mathrm{QT}_\mathtt{q}$ and obtains $\{\mathsf{E}(\delta), \mathsf{E}(f), \mathsf{E}(\mathsf{AS})\}$. We can see that the semantic security of FHE scheme guarantees that the views of $\mathcal{A}$ in the real and the ideal words are indistinguishable. Specifically, i) $\mathcal{A}$ cannot distinguish $\mathrm{QT}'_\mathtt{q}$ from $\mathrm{QT}_\mathtt{q}$ as it cannot distinguish FHE ciphertexts $\{\mathsf{E}(\mathsf{BF}^{i\prime}_0), \mathsf{E}(\mathsf{BF}^{i\prime}_\mathrm{T}) \mid i \in [1, d]\}$ from $\{\mathsf{E}(\mathsf{BF}^i_0), \mathsf{E}(\mathsf{BF}^i_\mathrm{T}) \mid i \in [1, d]\}$; ii) $\mathcal{A}$ cannot distinguish $\{\mathsf{E}(\delta'), \mathsf{E}(f'), \mathsf{E}(\mathsf{AS}')\}$ from $\{\mathsf{E}(\delta), \mathsf{E}(f), \mathsf{E}(\mathsf{AS})\}$ due to the semantic security of FHE ciphertexts. Thus, our PPARS scheme can securely compute the aggregate set of reverse skyline without leaking the query requests represented by $\mathrm{QT}_\mathtt{q}$ and query results $\mathsf{E}(\mathsf{AS})$. □

Following [11], [26], we define the access pattern as the information about which data points are selected as the reverse skyline. Recalling our PPARS scheme (Section IV-C), the encrypted flag $\mathsf{E}(f_i)$ is used to indicate whether a data point is a reverse skyline point, i.e., if $f_i = 1$, the corresponding data point $\mathtt{x}_i$ is a reverse skyline point, otherwise $\mathtt{x}_i$ is not. Next, we will prove that our PPARS scheme can hide access patterns by illustrating the adversary cannot distinguish the encrypted flags $\{\mathsf{E}(f_i) \mid i \in [1, n]\}$.

Theorem 3: The PPARS scheme preserves the access pattern privacy, i.e., $\mathcal{S}$ *has no idea about which data points are selected as the reverse skyline.*

Proof: Given a query point $\mathtt{q}_t$, $\texttt{Sim}$ runs $\mathcal{A}$ by: i) generating $\mathrm{QT}'_{\mathtt{q}_t} = \{\mathsf{E}(\mathsf{BF}^{i\prime}_0), \mathsf{E}(\mathsf{BF}^{i\prime}_\mathrm{T}) \mid i \in [1, d]\}$; ii) computing $\{\mathsf{E}(f'_i) \mid i \in [1, n]\}$ with Algorithm 4; iii) sending $\{\mathsf{E}(f'_i) \mid i \in [1, n]\}$ to the adversary $\mathcal{A}$. With the same query point $\mathtt{q}_t$, $\texttt{Sim}$ generates a new query token $\mathrm{QT}_{\mathtt{q}_t} = \{\mathsf{E}(\mathsf{BF}^i_0), \mathsf{E}(\mathsf{BF}^i_\mathrm{T}) \mid i \in [1, d]\}$. Using the generated $\mathrm{QT}_{\mathtt{q}_t}$ and the same approach, a set of encrypted flags $\{\mathsf{E}(f_i) \mid i \in [1, n]\}$ can be computed. Finally, $\{\mathsf{E}(f_i) \mid i \in [1, n]\}$ is sent to the adversary $\mathcal{A}$. Since $\{f'_i \mid 1 \in [1, n]\}$ and $\{f_i \mid 1 \in [1, n]\}$ are encrypted with FHE and respectively computed from two query tokens $\mathrm{QT}'_{\mathtt{q}_t}$ and $\mathrm{QT}_{\mathtt{q}_t}$ that have different encrypted bloom filters, the adversary $\mathcal{A}$ cannot distinguish $\{\mathsf{E}(f'_i) \mid i \in [1, n]\}$ from $\{\mathsf{E}(f_i) \mid i \in [1, n]\}$, although they are calculated from the same query point $\mathtt{q}_t$. Thus, our PPARS scheme can make the server $\mathcal{S}$ have no idea about which data points are selected as the reverse skyline. □

Compared to the PPARS scheme, our CE-PPARS scheme adopts the polynomial functions to calculate $\mathsf{E}(\theta_u)(1 \le u \le |\mathcal{F}(v)|)$. However, the elements in $\mathsf{E}(\widehat{\mathsf{BF}}^i_\pi)$ ($i \in [1, d]$ and $\pi = \{0, \mathrm{T}\}$) are still encrypted by FHE, and all operations are conducted over encrypted data. Therefore, similar to the proofs of the PPARS scheme, we can prove that our CE-PPARS

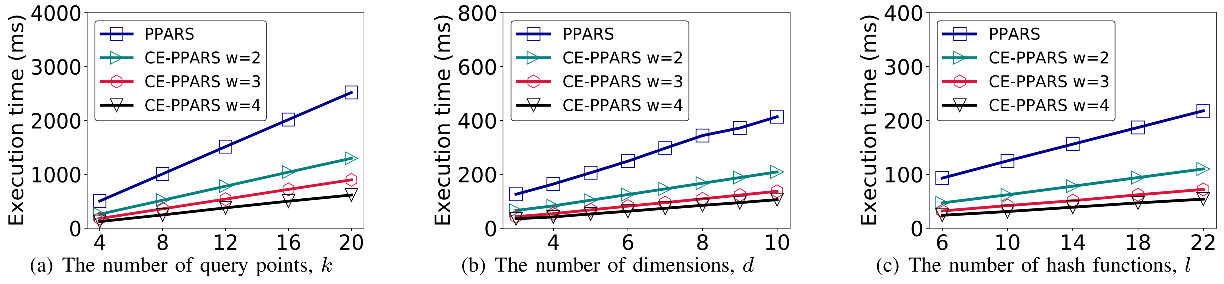


Fig. 7. Computational costs of query token report phase. (a) varying with the number of query points k, $d = 3$ and $l = 10$; (b) varying with the number of dimensions d, $k = 4$ and $l = 10$; (c) varying with the number of hash functions, l, $k = 4$ and $d = 3$.

scheme is also privacy-preserving. To save space, we omit the details here.

VI. Performance Evaluation

In this section, we will experimentally evaluate the performance of our proposed schemes and explore the impact of different parameters. Recalling Section IV-C, our proposed schemes mainly have three phases. Since the result recovery phase only involves data decryption, we will focus on the query token report phase and the privacy-preserving ARS search phase in this section.

Note that we are the first to propose the privacy-preserving ARS query schemes on the single server model, and the existing privacy-preserving skyline related solutions cannot be applied to address the problems in our PPARS scheme. See the detailed analysis in Section VII.

Experimental Setting: All of our proposed schemes were implemented with Java and executed on a machine with 16 GB memory, 3.4 GHz Intel(R) Core(TM) i7-3770 processors, and Ubuntu 16.04 OS. From Section IV, we know our proposed schemes only require that the employed encryption scheme can support addition and multiplication homomorphic properties. As a result, any fully homomorphic encryption scheme, even leveled homomorphic encryption schemes, can be employed in our proposed schemes. Since the symmetric homomorphic encryption (SHE) used in [27]–[29] is quite efficient, we adopted its public-key setting [29] as the cryptography primitive in our proposed schemes. For the security parameters of SHE, we let $k_0 = 4096$, $k_1 = 80$, and $k_2 = 160$ (see the details of these parameters in [29]). Note that, as SHE is a leveled homomorphic encryption scheme, and its maximum homomorphic multiplication depth is related to k_0, we can either adopt a bootstrapping protocol [28] between $\mathcal{S}$ and $\mathcal{C}$ to refresh ciphertexts or enlarge k_0 to support more homomorphic multiplication operations. For simplicity sake, here we set $k_0 = 4096$ and adopted the bootstrapping protocol to refresh ciphertexts. Regarding the dataset, in our experiments, we used a real-world dataset that contains the Airbnb data of Denver [30], denoted as **Denver**. In particular, we first extracted the fields of number type, such as *longitude*, *latitude*, and *acceptance rate*, and scaled them to integers. Then, we filtered out the missing-value items. Eventually, our **Denver** dataset has 2,547 items, and each item has ten attributes.

A. Performance of Query Token Report Phase

Recalling Section IV-C.1, the client $\mathcal{C}$ sends $\mathrm{QT}||pk||\mathcal{H}$ to the server $\mathcal{S}$, where $\mathrm{QT} = \{\mathrm{QT}_{\mathbb{q}_t} \mid 1 \leq t \leq k\}$ and $\mathrm{QT}_{\mathbb{q}_t} = \{\mathsf{E}(\mathsf{BF}_0^i), \mathsf{E}(\mathsf{BF}_\mathrm{T}^i) \mid i \in [1, d]\} = \mathsf{SDRD.TokenGen}(\mathbb{q}_t, pk, \mathcal{H})$. Consequently, the computational costs of the query token report phase are related to the number of query requests k, the number of dimensions d, and the number of hash functions l.

1) Impact of the Number of Query Requests k: Fig. 7(a) depicts the computational costs of query token report phase varying with the number of query requests k. Intuitively, the token generation time linearly increases with the number of query requests k. Through a simple calculation (of the slope of lines in Fig. 7(a)), we can obtain the average execution time of generating the query token for one query request. The results show that our PPARS scheme has the largest average execution time around 126 ms, while our CE-PPARS scheme takes 65 ms, 45 ms, and 31 ms when the packing window w is 2, 3, and 4, respectively (we will analyze the reason at the end of this subsection). As all of them are at the millisecond level, our proposed schemes are efficient in generating query token for one query request.

2) Impact of the Number of Dimensions d: Fig. 7(b) illustrates the computational costs of query token report phase varying with the number of dimensions d. Similarly, all of proposed schemes show a linearly increasing tread in this figure. That is because when we add one more dimension to the original query requests, our $\mathsf{SDRD.TokenGen}()$ algorithm needs to generate two more encrypted bloom filters for each query request. Therefore, as d increases, so does the execution time of generating a query token.

3) Impact of the Number of Hash Functions l: Fig. 8(c) plots the computational costs of query token report phase varying with the number of hash functions l. In our $\mathsf{SDRD.TokenGen}(\mathbb{q}, pk, \mathcal{H})$ algorithm, one of the inputs is the set of hash functions $\mathcal{H}$. The more hash functions in $\mathcal{H}$ indicates that we need to execute more hash functions to map an element into a bloom filter. As a result, the token generation time increases as the number of hash functions increases.

From the above three figures (Fig. 8(a), Fig. 8(b), Fig. 8(c)), we know that our PPARS scheme has the worst performance in the query token report phase, and when w increases, our CE-PPARS scheme has a better performance. The reason is that encrypting elements in the bloom filters accounts for most

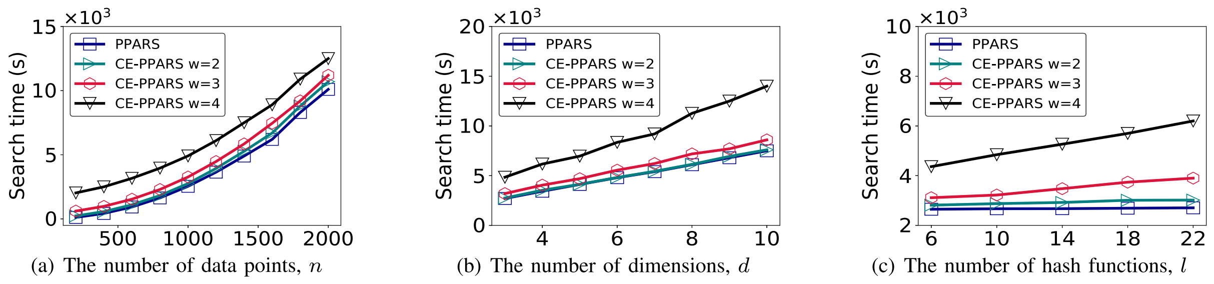


Fig. 8. Computational costs of privacy-preserving ARS search phase. (a) varying with the number of data points n, $d = 3$ and $l = 10$; (b) varying with the number of dimensions d, $n = 1000$ and $l = 10$; (c) varying with the number of hash functions l, $n = 1000$ and $d = 3$.

of the time consumption in this phase. For each query token QT, our PPARS scheme needs to encrypt $2d \cdot \eta \cdot k$ plaintexts into ciphertexts. Whereas, our CE-PPARS scheme only needs to encrypt $\lceil \frac{2d \cdot \eta \cdot k}{w} \rceil$ plaintexts. The larger w means that fewer plaintexts need to be encrypted. Thus, when w is larger, our CE-PPARS has a better performance in generating the query token. Meanwhile, comparing these three figures, we can see that the number of query requests k has the largest impact on the performance in this phase, followed by the number of dimensions d and the number of hash functions l. That is because we need to generate $2d$ bloom filters for a new query request, while it is two bloom filters for each dimension. Regarding l, when it increases, we do not need to increase the number of bloom filters and only need to execute more hash functions for generating bloom filters.

B. Performance of Privacy-Preserving ARS Search

From Algorithm 4, we know that the computational costs of this phase are related to the number of data records n, the number of dimensions d, and the number of hash functions.

1) Impact of the Number of Data Records n: Fig. 8(a) depicts the computational costs of privacy-preserving ARS search phase varying with the number of data records n. As Definition 2 shown, given a data record x_i and a query request q, we need to check whether there exists another data record x_j in the dataset that dominates q with regard to x_i. Therefore, the computational costs of this phase quadratically increase with the number of data records.

2) Impact of the Number of Dimensions d: Fig. 8(b) illustrates the computational costs of the privacy-preserving ARS search phase varying with the number of dimensions d. When n is fixed, we only need to check two more bloom filters in the $\mathsf{SDRD.Check}()$ algorithm with the growth of d. As a result, the computational costs of this phase linearly increase with the number of dimensions.

3) Impact of the Number of Hash Functions l: Fig. 8(c) plots the computational costs of the privacy-preserving ARS search phase varying with the number of hash functions l. Similar to the parameter d, the computational costs also show a linear increasing trend when varying with l. This is because we need to run l multiplications to calculate $\mathsf{E}(\theta_u)$ in our $\mathsf{SDRD.Check}()$ algorithm, i.e., $\mathsf{E}(\theta_u) = \prod_{j=1}^{l} \mathsf{E}(\mathsf{BF}_0^i[h_j(e_u)])$. It is worth noting, in our setting, the number of hash functions l is related to the bloom filter's *false positive*. When $l = 10$, $f_p \approx (1/2)^l = 0.1\%$, while $f_p \approx 0.0001\%$ when $l = 20$. Therefore, we can set a suitable l to guarantee an acceptable *false positive* for different applications.

From the above three figures (Fig. 8(a), Fig. 8(b), Fig. 8(c)), we know that, the CE-PPARS scheme has the worst performance when $w = 4$, while our PPARS scheme outperforms other schemes. The reason is that we need to run the interpolated polynomials over ciphertexts to calculate $\mathsf{E}(\theta_u)$ in our CE-PPARS scheme, and the largest degree of the interpolated polynomial is $2^w - 1$. However, our PPARS scheme only needs to run l multiplications to obtain $\mathsf{E}(\theta_u)$. Thus, our PPARS scheme has better performance than the CE-PPARS scheme in this phase, and the larger w indicates the larger computational costs. Notably, the skyline computation is computationally intensive, especially for the fully secure skyline query schemes. For example, the state-of-the-art secure dynamic skyline query schemes [7]–[9] have computational costs from 10^2 to 10^4 seconds for finding skyline points with a similar parameter setting. However, these schemes were designed for the two-server model, which naturally has less computational costs (but requires a lot of communication overhead) than the single server model under the same security level. Therefore, as a kind of fully secure skyline query scheme, our proposed schemes have acceptable computational costs in the single server model, which is more practical and does not incur additional communication overhead between servers.

C. Communication Overhead

In this section, we explore the communication overhead impact of different parameters. Here the communication overhead indicates the size of query tokens delivered from the client $\mathcal{C}$ to the server $\mathcal{S}$. Since *delieved bytes* $=$ *the number of ciphertexts* $*$ *the size of a ciphertext*, we will separately discuss *the number of ciphertexts* and *the size of a ciphertext* as follows:

- *the number of ciphertexts*. As discussed in Section IV, $\mathcal{C}$ sends $2d \cdot \eta \cdot k$ ciphertexts to $\mathcal{S}$ in our PPARS scheme, while it is $\lceil \frac{2d \cdot \eta \cdot k}{w} \rceil$ in our CE-PPARS scheme. It is clear that the parameters d and k have a linear impact on the communication overhead. To save space, here we only explore the impact of the number of hash functions l, which is related to η, and the packing parameter w on the communication overhead.

TABLE II
COMPARISON WITH EXISTING SECURE SKYLINE SCHEMES

Schemes	**Bothe** [31]	**Zaman** [32]	**Hua** [33]	**Zheng** [34]	**Wang** [35]	**Wang** [10]	**Liu** [7]	**Zhang** [9]	**Ding** [8]	**Ours**
Skyline query type	Basic	Basic	Basic	Basic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic	Dynamic/Reverse
Encryption	Matrix	SKE	ElGamal	Benaloh PKE	SKE	ORE	HE	HE	HE	HE
Access pattern	✘	✘	✘	✘	✘	✘	✔	✔	✔	✔
Single-server model	✔	✘	✔	✔	With SGX	✔	✘	✘	✘	✔

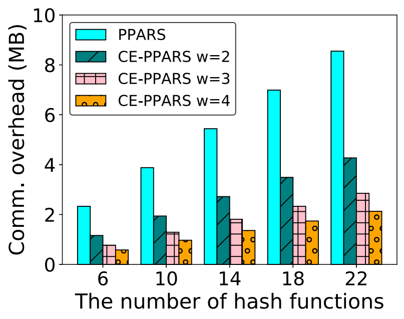


Fig. 9. Communication overhead of encrypted query tokens varying with the number of hash functions l.

• *the size of a ciphertext*. In our experiments, we use the SHE technique, and each SHE ciphertext has $2 \cdot k_0$ bits. Since we set $k_0 = 4096$, each SHE ciphertext has the size of 1KB.

Fig. 9 depicts the communication overhead of query tokens varying with the number of hash functions l. From this figure, we know that: i) the PPARS scheme has the largest communication overhead, and the communication overhead of our CE-PPARS scheme decreases with the growth of w. That is exactly the design goal of our CE-PPARS scheme as we reduce the number of ciphertexts from $2d \cdot \eta \cdot k$ to $\lceil \frac{2d \cdot \eta \cdot k}{w} \rceil$; ii) with the increase of l, the communication overhead increases. That is because we can calculate η with l, i.e., $\eta \leftarrow \lceil l * m / \ln 2 \rceil$, where m is set as $2\phi - 2$, and ϕ is the maximum bit length of the given query requests. Here, we set $\phi = 24$ since it can cover all query requests in our experiments.

VII. RELATED WORK

Skyline queries have a wide range of applications in many domains. Due to the privacy concerns in the cloud computing, privacy-preserving skyline queries have been extensively studied [7]–[10], [31]–[35].

A. Privacy-Preserving Basic Skyline Queries

The concept of the basic skyline query refers to the skyline queries that do not involve a query point, which is relative to the dynamic skyline query. In 2014, Bothe *et al.* [31] proposed the first privacy-preserving basic skyline query scheme, in which the skyline computation was transformed into the non-descending series, and the matrix encryption was adopted to encrypt them. However, this scheme is not secure since the adversary can infer the secret key by launching the known plaintext attack. Zaman *et al.* [32] designed a secure skyline computation scheme in MapReduce. In this scheme, the symmetric-key encryption (SKE) was used to encrypt sensitive data, and a trusted party needs to be deployed to obtain the order relation of data. In the e-Healthcare scenario, Hua *et al.* [33] proposed a privacy-preserving skyline query scheme to make remote diagnosis possible while protecting data privacy. However, this scheme focused on basic skyline computation and cannot protect the access patterns. In [34], Zheng *et al.* presented a privacy-preserving skyline computation protocol over encrypted data. By using the Benaloh public key encryption (PKE) [36], this scheme compares encrypted data in a non-interactive manner. Similar to [33], the work in [34] also dealt with the basic skyline computation and revealed access pattern privacy.

In summary, all of the above privacy-preserving skyline query schemes are aimed at the basic skyline queries. Since our proposed schemes involve the query points and are a kind of dynamic skyline query, the above schemes cannot be applied to design the privacy-preserving reverse skyline query schemes. Besides, the above schemes leak the access pattern privacy, which must be preserved in our scenario.

B. Privacy-Preserving Dynamic Skyline Queries

We consider all the skyline queries involving query points as dynamic skyline queries. In [10], Wang *et al.* proposed a secure dynamic skyline query scheme by using the order revealing encryption (ORE). Since the ORE scheme naturally reveals the order relations of underlying plaintexts, this scheme cannot fully protect the order relations for the outsourced data and may incur inference attacks [37]. In [35], the authors employed a secure hardware (SGX) to securely compute skylines. However, this work leaks access patterns and needs additional hardware. Regarding the works in [7]–[9], the authors respectively presented a secure dynamic skyline query scheme by designing a collection of secure protocols. Although these schemes can preserve the privacy of data records, query requests, query results, and access patterns, they were achieved in a two-server setting, in which two semi-honest cloud servers interact with each other to compute skylines. However, in our scenario, it is unreasonable to deploy an additional server for secure skyline computation. Different from the above privacy-preserving dynamic skyline query schemes, our proposed schemes are designed for the single server model and can ensure the privacy of query requests, query results, and access patterns.

In Table II, we compare our proposed schemes with the related works in terms of skyline query type, encryption technique, access pattern privacy, and the single-server model, in which Matrix indicates the matrix encryption, and HE means the homomorphic encryption.

VIII. Conclusion

In this paper, we have proposed a privacy-preserving aggregate reverse skyline query scheme on a single server model, denoted as PPARS. To our best knowledge, we are the first to achieve the privacy-preserving ARS queries in a single server model while ensuring full query privacy. In particular, we first transformed the problem of aggregate reverse skyline query into the set membership test and logical expressions. Then, we introduced a prefix encoding technique to encode query points and map the encoded set into a bloom filter. After that, we encrypted the elements in the bloom filter with the FHE scheme. By running the logical expressions over ciphertexts, we obtained the encrypted aggregate values as query results. To reduce the number of ciphertexts delivered between the client and cloud server, we proposed our CE-PPARS scheme with an interpolation-based packing technique. Finally, we formally analyzed the security of our proposed schemes and conducted extensive experiments to validate their efficiency.

References

[1] X. Han, J. Li, D. Yang, and J. Wang, “Efficient skyline computation on big data,” *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 11, pp. 2521–2535, Nov. 2013.

[2] J. Chomicki, P. Ciaccia, and N. Meneghetti, “Skyline queries, front and back,” *ACM SIGMOD Rec.*, vol. 42, no. 3, pp. 6–18, Oct. 2013.

[3] E. Dellis and B. Seeger, “Efficient computation of reverse skyline queries,” in *Proc. VLDB*, vol. 7, 2007, pp. 291–302.

[4] G. Wang, J. Xin, L. Chen, and Y. Liu, “Energy-efficient reverse skyline query processing over wireless sensor networks,” *IEEE Trans. Knowl. Data Eng.*, vol. 24, no. 7, pp. 1259–1275, Jul. 2012.

[5] B. Yin, K. Gu, X. Wei, S. Zhou, and Y. Liu, “A cost-efficient framework for finding prospective customers based on reverse skyline queries,” *Knowl.-Based Syst.*, vol. 152, pp. 117–135, Jul. 2018.

[6] X. Wu, Y. Tao, R. C.-W. Wong, L. Ding, and J. X. Yu, “Finding the influence set through skylines,” in *Proc. 12th Int. Conf. Extending Database Technol. Adv. Database Technol. (EDBT)*, Mar. 2009, pp. 1030–1041.

[7] J. Liu, J. Yang, L. Xiong, and J. Pei, “Secure skyline queries on cloud platform,” in *Proc. IEEE ICDE*, Apr. 2017, pp. 633–644.

[8] X. Ding, Z. Wang, P. Zhou, K.-K.-R. Choo, and H. Jin, “Efficient and privacy-preserving multi-party skyline queries over encrypted data,” *IEEE Trans. Inf. Forensics Security*, vol. 16, pp. 4589–4604, 2021.

[9] S. Zhang, S. Ray, R. Lu, Y. Zheng, Y. Guan, and J. Shao, “Achieving efficient and privacy-preserving dynamic skyline query in online medical diagnosis,” *IEEE Internet Things J.*, vol. 9, no. 12, pp. 9973–9986, Jun. 2022.

[10] W. Wang *et al.*, “Scale: An efficient framework for secure dynamic skyline query processing in the cloud,” in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, 2020, pp. 288–305.

[11] R. Li, A. X. Liu, H. Xu, Y. Liu, and H. Yuan, “Adaptive secure nearest neighbor query processing over encrypted data,” *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 1, pp. 91–106, Jan./Feb. 2022.

[12] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(Leveled) fully homomorphic encryption without bootstrapping,” *ACM Trans. Comput. Theory*, vol. 6, no. 3, pp. 1–36, Jul. 2014.

[13] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Springer, Dec. 2017, pp. 409–437.

[14] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser, “Machine learning classification over encrypted data,” in *Proc. NDSS*, 2015, p. 4325.

[15] K. Bonawitz *et al.*, “Practical secure aggregation for privacy-preserving machine learning,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2017, pp. 1175–1191.

[16] O. Goldreich, *Foundations of Cryptography: Basic Applications*, vol. 2. Cambridge, U.K.: Cambridge Univ. Press, 2009.

[17] Y. Gao, Q. Liu, B. Zheng, and G. Chen, “On efficient reverse skyline query processing,” *Expert Syst. Appl.*, vol. 41, no. 7, pp. 3237–3249, Jun. 2014.

[18] F. Bourse, M. Minelli, M. Minihold, and P. Paillier, “Fast homomorphic evaluation of deep discretized neural networks,” in *Proc. Annu. Int. Cryptol. Conf.* Cham, Switzerland: Springer, Aug. 2018, pp. 483–512.

[19] J. H. Cheon, M. Kim, and M. Kim, “Optimized search-and-compute circuits and their application to query evaluation on encrypted data,” *IEEE Trans. Inf. Forensics Security*, vol. 11, no. 1, pp. 188–199, Jan. 2016.

[20] M. Kim, H. T. Lee, S. Ling, B. H. M. Tan, and H. Wang, “Private compound wildcard queries using fully homomorphic encryption,” *IEEE Trans. Dependable Secure Comput.*, vol. 16, no. 5, pp. 743–756, Sep. 2019.

[21] X. Liu, R. H. Deng, K.-K.-R. Choo, Y. Yang, and H. Pang, “Privacy-preserving outsourced calculation toolkit in the cloud,” *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 5, pp. 898–911, Sep. 2020.

[22] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption,” *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, Mar. 2012.

[23] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors,” *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970.

[24] P. Gupta and N. McKeown, “Algorithms for packet classification,” *IEEE Netw.*, vol. 15, no. 2, pp. 24–32, Mar./Apr. 2001.

[25] (2001). *Lagrange Interpolation Formula*. [Online]. Available: https://encyclopediaofmath.org/index.php?

[26] G. Kellaris, G. Kollios, K. Nissim, and A. O’Neill, “Generic attacks on secure outsourced databases,” in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2016, pp. 1329–1340.

[27] H. Mahdikhani, R. Lu, Y. Zheng, J. Shao, and A. A. Ghorbani, “Achieving o $(\log^3 n)$ communication-efficient privacy-preserving range query in fog-based IoT,” *IEEE Internet Things J.*, vol. 7, no. 6, pp. 5220–5232, Jun. 2020.

[28] Y. Zheng, R. Lu, Y. Guan, J. Shao, and H. Zhu, “Efficient and privacy-preserving similarity range query over encrypted time series data,” *IEEE Trans. Dependable Secure Comput.*, early access, Feb. 23, 2021, doi: 10.1109/TDSC.2021.3061611.

[29] Y. Guan, R. Lu, Y. Zheng, S. Zhang, J. Shao, and G. Wei, “Toward privacy-preserving cybertwin-based spatiotemporal keyword query for its in 6G era,” *IEEE Internet Things J.*, vol. 8, no. 22, p. 16243–16255, Jul. 2021.

[30] (2021). *Inside Airbnb*. [Online]. Available: http://insideairbnb.com/get-the-data.html

[31] S. Bothe, A. Cuzzocrea, P. Karras, and A. Vlachou, “Skyline query processing over encrypted data: An attribute-order-preserving-free approach,” in *Proc. 1st Int. Workshop Privacy Secuirty Big Data (PSBD)*, Nov. 2014, pp. 37–43.

[32] A. Zaman, M. A. Siddique, and Y. Morimoto, “Secure computation of skyline query in *MapReduce*,” in *Proc. Int. Conf. Adv. Data Mining Appl.* Cham, Switzerland: Springer, Dec. 2016, pp. 345–360.

[33] J. Hua *et al.*, “CINEMA: Efficient and privacy-preserving online medical primary diagnosis with skyline query,” *IEEE Internet Things J.*, vol. 6, no. 2, pp. 1450–1461, Apr. 2019.

[34] Y. Zheng, R. Lu, B. Li, J. Shao, H. Yang, and K.-K. R. Choo, “Efficient privacy-preserving data merging and skyline computation over multi-source encrypted data,” *Inf. Sci.*, vol. 498, pp. 91–105, Sep. 2019.

[35] J. Wang, M. Du, and S. S. Chow, “Stargazing in the dark: Secure skyline queries with SGX,” in *Proc. Int. Conf. Database Syst. Adv. Appl.* Cham, Switzerland: Springer, Sep. 2020, pp. 322–338.

[36] J. Benaloh, “Dense probabilistic encryption,” in *Proc. Workshop Sel. Areas Cryptogr.*, May 1994, pp. 120–128.

[37] M. Naveed, S. Kamara, and C. V. Wright, “Inference attacks on property-preserving encrypted databases,” in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 644–655.

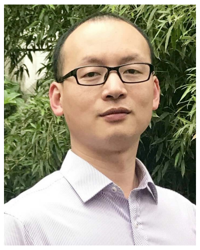
Songnian Zhang received the M.S. degree from Xidian University, China, in 2016. He is currently pursuing the Ph.D. degree with the Faculty of Computer Science, University of New Brunswick, Canada. His research interests include cloud computing security, big data query, and query privacy.

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

Rongxing Lu (Fellow, IEEE) received the Ph.D. degree from the Department of Electrical and Computer Engineering, University of Waterloo, Canada, in 2012. He worked as a Post-Doctoral Fellow with the University of Waterloo from May 2012 to April 2013. He is currently a Mastercard IoT Research Chair, a University Research Scholar, and an Associate Professor with the Faculty of Computer Science (FCS), University of New Brunswick (UNB), Canada. Before that, he worked as an Assistant Professor with the School of Electrical and Electronic Engineering, Nanyang Technological University (NTU), Singapore, from April 2013 to August 2016. He has published extensively in his areas of expertise. His research interests include applied cryptography, privacy enhancing technologies, and IoT-big data security, and privacy. He was a recipient of nine best (student) paper awards from some reputable journals and conferences. He was awarded the most prestigious "Governor General's Gold Medal," when he received his Ph.D. degree. He has won the Eighth IEEE Communications Society (ComSoc) Asia Pacific (AP) Outstanding Young Researcher Award in 2013. He also serves as the Chair of IEEE Communications and Information Security Technical Committee (ComSoc CIS-TC), and the Founding Co-Chair of IEEE TEMS Blockchain and Distributed Ledgers Technologies Technical Committee (BDLT-TC). He is the Winner of 2016–2017 Excellence in Teaching Award, FCS, UNB.

Yunguo Guan is currently pursuing the Ph.D. degree with the Faculty of Computer Science, University of New Brunswick, Canada. His research interests include applied cryptography and game theory.

Yandong Zheng received the M.S. degree from the Department of Computer Science, Beihang University, China, in 2017. She is currently pursuing the Ph.D. degree with the Faculty of Computer Science, University of New Brunswick, Canada. Her research interests include cloud computing security, big data privacy, and applied privacy.



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