Computers & Security 140 (2024) 103803

Contents lists available at ScienceDirect

Computers & Security

journal homepage: www.elsevier.com/locate/cose

BopSkyline: Boosting privacy-preserving skyline query service in the cloud

Weibo Wang [a], Yifeng Zheng [a,*], Songlei Wang [a], Zhongyun Hua [a], Lei Xu [b], Yansong Gao [c]

[a] School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Guangdong 518055, China
[b] School of Mathematics and Statistics, Nanjing University of Science and Technology, Nanjing, China
[c] Data61, CSIRO, Sydney, Australia

ARTICLE INFO

Keywords:
Service outsourcing
Cloud computing
Skyline query
Privacy protection

ABSTRACT

With the widespread adoption of cloud computing, there has been great popularity of storing and querying databases in the cloud. However, such service outsourcing also entails critical data privacy concerns, as the cloud providers are generally not in the same trust domain as the data owners/users and could even suffer from data breaches. In this paper, different from most existing works that propose security designs for keyword search, we focus on secure realizations of advanced skyline query processing, which plays an important role in multi-criteria decision support applications. We propose BopSkyline, a new system framework for privacy-preserving skyline query service in cloud computing. BopSkyline is designed to not only ensure the confidentiality of outsourced databases, skyline queries, and query results, but also conceal data patterns (like the dominance relationships among database tuples) and search access patterns that may indirectly lead to data leakages. Notably, through a delicate synergy of key ideas on secure database shuffling and differentially private database padding, BopSkyline achieves a significant performance boost over the state-of-the-art. Extensive experiments demonstrate that compared with the state-of-the-art prior work, BopSkyline is up to 4.7× better in query latency and achieves up to 99.38% cost savings in communication.

1. Introduction

Storing and querying databases in the cloud has gained great popularity along with widespread adoption of cloud computing that provides well-understood benefits [34,19]. On another hand, such service outsourcing also raises critical data privacy concerns, as the cloud service providers are not in the same trust domain as the data owners/users and could even suffer from data breaches. For example, in 2022, FlexBooker suffered from a data breach that exposed personal information of up to 19 million users due to that their account on Amazon's AWS servers got compromised [3]. Consequently, it is essential to integrate security measures into such outsourced database query services. Numerous efforts have been made to support queries over encrypted outsourced databases, where most existing works have been concentrated on encrypted keyword search.

In contrast to the majority of prior research, this paper's focus lies on secure realizations of advanced skyline query processing over outsourced databases. Skyline query [20] is an advanced type of query for analytics of multi-dimensional databases, which aims to find data points/tuples from a database which are not dominated by any other point in the database. Consider for example two data points **a** and **b** from a database. Given a query point **q**, **a** is said to dominate **b** with respect to **q**, if **a** is closer to **q** than **b** in at least one dimension and not farther in any other dimension. Skyline query can benefit a wide range of multi-criteria decision support applications (especially for those where it is hard to define a single distance metric for all dimensions [27,28]), such as web information systems [2], wireless mobile ad-hoc networks [18], and geographical information systems [11].

To illustrate how skyline query works and our target problem with more clarity, we describe an exemplary use case of skyline query. Let us consider that a medical center outsources its database of patient records to the cloud, which can then offer skyline query services to doctors from other medical centers. For simplicity, assume that there are 5 patient records in the original database **D**, each of which is associated with two attributes "Age" and "Systolic Blood Pressure (SBP)". The doctors from other medical centers can issue skyline queries to the cloud to retrieve records similar to their patients, for the purpose of treatment enhancement and personalization. Specifically, given a query $\mathbf{q} = (46, 130)$, the skyline query service expects the cloud to retrieve all

* Corresponding author.
E-mail addresses: weibo.wang.hitsz@outlook.com (W. Wang), yifeng.zheng@hit.edu.cn (Y. Zheng), songlei.wang@outlook.com (S. Wang), huazhongyun@hit.edu.cn (Z. Hua), xuleicrypto@gmail.com (L. Xu), gao.yansong@hotmail.com (Y. Gao).

https://doi.org/10.1016/j.cose.2024.103803
Received 4 September 2023; Received in revised form 11 December 2023; Accepted 4 March 2024
Available online 8 March 2024

Table 1
An Example of Original Database **D** and Mapped Database **B** of Patient Records.

Original Database	Age	SBP
d_1	50	150
d_2	42	135
d_3	44	120
d_4	52	125
d_5	40	145
Mapped Database	**Age**	**SBP**
b_1	4	20
b_2	4	5
b_3	2	10
b_4	6	5
b_5	6	15

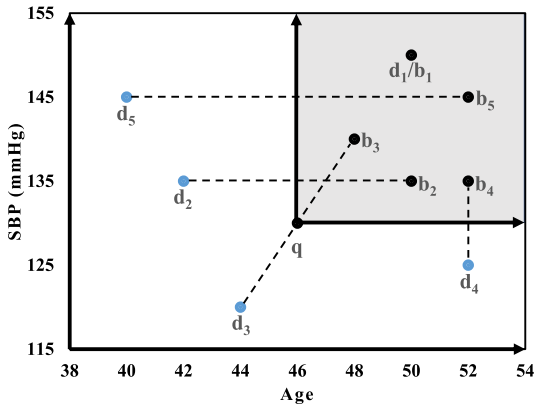

Fig. 1. An example of skyline query under a query point **q**.

records from **D** that are not dominated by any other record with respect to **q**. To achieve this, the cloud first maps the original database **D** to a new database (called mapped database **B**) by utilizing the mapping function [27,28]: $\mathbf{b}_i[j] = |\mathbf{d}_i[j] - \mathbf{q}[j]|, i \in [1,5], j \in [1,2]$. The original database and mapped database are shown in Table 1, and the mapping relationships between them are illustrated in Fig. 1. The resulting patient records returned as the target skyline points are $\mathbf{d}_2$ and $\mathbf{d}_3$. This is because $\mathbf{b}_2$ dominates $\mathbf{b}_1$, $\mathbf{b}_4$ and $\mathbf{b}_5$; and $\mathbf{b}_3$ dominates $\mathbf{b}_1$ and $\mathbf{b}_5$. The formal definition of dominance is provided in Section 3.1.

In this paper, we aim to develop techniques that allow the cloud to process skyline queries over outsourced encrypted databases in an efficient and privacy-preserving manner. Taking the aforementioned application scenario as a concrete example, our goal is to enable the cloud, which hosts an encrypted version of the database **D** to produce the encrypted version of the skyline query result $\{\mathbf{d}_2, \mathbf{d}_3\}$ with respect to the query point **q**. It is noted that in such outsourced skyline query service, not only the data content confidentiality demands protection, but also the data patterns and search access patterns which could be sources of indirect data leakages [27,28,12]. Here, the data patterns refer to the relationships of dominance among the database tuples and the number of database tuples that each skyline tuple dominates; the search pattern indicates whether a skyline query is a repeated one; and the access pattern indicates which database tuples are identified as the skyline tuples with respect to the query.

In the literature, a few works [5,27,28,12,44,43,38,40] have been presented regarding privacy-preserving skyline queries over encrypted databases. The state-of-the-art design that is most related to our work is SecSkyline [44], which builds on lightweight cryptographic techniques to support privacy-preserving skyline queries. However, the solution of SecSkyline is not yet fully satisfactory due to the high theoretical performance complexities in the query processing phase. Specifically, in each round of secure skyline finding, to prevent the cloud from learning the dominance relationships, SecSkyline has to let the cloud obliviously mark skyline tuples and their dominated tuples, without actually eliminating them from the encrypted mapped/original database. This undesirably poses a requirement that in each round the cloud has to always iterate over all database tuples in the ciphertext domain, even though most tuples should have stayed out of the processing.

In light of the above, in this paper, we design, implement, and evaluate BopSkyline, a new system framework that achieves a remarkable performance boost over the state-of-the-art prior work SecSkyline [44]. At a very high level, BopSkyline is built from a delicate synergy of lightweight secret sharing techniques [10], differential privacy (DP) [14], and secure shuffle [15]. The key insight of BopSkyline in achieving remarkably boosted performance over SecSkyline is to devise mechanisms that allow the cloud to *delete* the skyline tuple and its dominated tuples in each round of secure skyline finding while being oblivious to the true data patterns and access pattern.

To this end, our main idea is to first have the cloud obliviously shuffle the tuples in the encrypted original database by a secret random permutation before securely processing a skyline query. Subsequently, in each round of secure skyline finding, the cloud can safely reveal which tuples in the encrypted *permuted* database are the skyline and dominated tuples and delete them. As a result, the performance complexity can gradually decrease over the rounds. It is worth noting that since the random permutation for (oblivious) database shuffling is unknown to the cloud, it cannot determine which encrypted tuples in the original database are the skyline or dominated tuples, and thus cannot learn the true dominance relationships and access pattern.

Within such secure database shuffling-based framework, however, there is a subtle issue to be considered. In particular, directly revealing which tuples are the skyline and dominated tuples in the encrypted permuted database will leak the exact number of database tuples that each skyline tuple dominates, which also demands protection [27,28,12]. To mitigate such leakage, our key idea is to let the data owner add some dummy tuples into the original database in the database preparation phase, so that the number of database tuples that each skyline tuple dominates is obfuscated. Based on the notion of differential privacy (DP), we devise a DP-based mechanism for theoretically determining the number of dummy tuples to be added. Furthermore, we show how to delicately construct the dummy tuples so that their existence in the outsourced database will not affect the result accuracy of skyline query.

We implement BopSkyline's protocols and conduct extensive empirical evaluations on multiple datasets. The results demonstrate that BopSkyline outperforms SecSkyline [44] with a significant improvement in performance. Specifically, BopSkyline is up to 4.7× better than SecSkyline in query latency and achieves up to 99.38% savings in communication cost. Moreover, the performance advantage of BopSkyline over SecSkyline grows sharply as the number of dimensions increases. Our key contributions are outlined below:

- We present BopSkyline, a new system framework enabling privacy-preserving skyline query services outsourced to the cloud, with a significant performance boost over the state-of-the-art.
- We propose the idea of secure database shuffling to allow oblivious tuple deletion in the secure skyline query processing stage. It greatly reduces the performance complexity compared to the state-of-the-art SecSkyline, while concealing the true dominance relationships and access pattern.
- We devise delicate mechanisms for adding dummy tuples to the outsourced database, which allow protection for the private information regarding the number of database tuples each skyline tuple dominates, while preserving the result accuracy.
- We conduct a formal security analysis of BopSkyline and conduct extensive empirical evaluations on multiple datasets. The results show that BopSkyline greatly outperforms the state-of-the-art prior work SecSkyline [44], with up to 4.7× better performance in query latency and up to 99.38% cost savings in communication.

The rest of this paper is structured as follows. Section 2 reviews the related work, while Section 3 introduces the necessary preliminaries. In Section 4, we present our system architecture and threat model. Section 5 presents the design of BopSkyline, followed by the security analysis and experiments in Sections 6 and 7, respectively. Lastly, Section 8 concludes this paper.

2. Related work

2.1. Skyline query without privacy protection

Börzsönyi et al. [4] are the first that propose the skyline operator for database systems. Since then, a lot of efforts have been presented for improving the algorithms for skyline query processing. An approach for processing skyline queries leveraging the nearest neighbor method is designed by Kossmann et al. [21]. Papadias et al. [31] present the branch-and-bound skyline algorithm, which achieves superior efficiency and storage over prior works. Another line of work has investigated how to process skyline queries in various scenarios, such as uncertain skyline [26,33], skyline queries on data streams [36], subspace skyline [9,32], and group-based skyline [25,42]. Despite the usefulness, these works deal with skyline queries in the plaintext domain and do not provide privacy protection.

2.2. Secure skyline query processing

The problem of secure skyline query processing is first studied by Bothe et al. [5]. They formulate the secure skyline problem and propose an approach relying on secret matrices for data protection. However, their protocol falls short of providing rigorous security guarantees. The recent works in [28,12] propose cryptography-based schemes which can provide rigorous security guarantees on data content confidentiality, data patterns, and search access patterns. Yet they have limited practicability since they build on heavy cryptosystems. Very recently, Zheng et al. [44] present SecSkyline, which fully builds on lightweight cryptography and achieves performance substantially better than the schemes in [28,12]. However, as mentioned above, in each round of secure skyline finding, SecSkyline just lets the cloud obliviously flag the skyline and dominated tuples instead of actually deleting them. In this way, the cloud has to always go through all the database tuples in each round, leading to high performance complexity.

There exist other studies [43,40] concentrating on securely processing skyline queries under different scenarios than ours. Zhang et al. [43] study privacy-preserving user-defined skyline queries, where the client can select some target dimensions, tailor the preference on each target dimension, and define a constrained region for the values on the target dimensions. The type of skyline query considered in [43] is not a general one and the design in [43] does not protect the data patterns and search access patterns. In another independent work [40], Wang et al. concentrate on providing support for result verification for location-based skyline queries that only involve two spatial attributes. Additionally, Wang et al. [38] leverage trusted hardware for secure skyline query computation. While in general such kind of approach achieves better performance than cryptographic approaches, it comes at the price of a much weaker threat model that demands trust on the vendors of trusted hardware and the enclave implementation. Meanwhile, a series of several attacks [16,37,23,22] targeting trusted hardware have emerged, threatening systems that rely on trusted hardware. Yet the attacks are overlooked by [38]. Hence, the state-of-the-art prior work that is most related to ours is SecSkyline [44].

3. Preliminaries

3.1. Skyline query

Definition 1. (Skyline [4]). Consider a database $\mathbf{D} = \{\mathbf{d}_1, \cdots, \mathbf{d}_n\}$ in an m-dimensional space (each dimension corresponds to an attribute) and two tuples $\mathbf{d}_a$ and $\mathbf{d}_b$ in $\mathbf{D}$. $\mathbf{d}_a$ is said to dominate $\mathbf{d}_b$ iff $\forall j \in [1, m]$, $\mathbf{d}_a[j] \leq \mathbf{d}_b[j]$ and $\exists j \in [1, m]$, $\mathbf{d}_a[j] < \mathbf{d}_b[j]$. The skyline tuples are those which cannot be dominated by any other tuple in the database $\mathbf{D}$.

Algorithm 1 Plaintext-Domain Skyline Query Processing.

Input: An m-dimensional database $\mathbf{D}$ of n tuples and a query tuple $\mathbf{q}$.
Output: The set of skyline tuples $\mathcal{R}_\mathbf{q}$ with respect to $\mathbf{q}$.

1: **for** $i = 1$ to n **do**
2: **for** $j = 1$ to m **do**
3: $\mathbf{b}_i[j] = |\mathbf{d}_i[j] - \mathbf{q}[j]|$.
4: **end for**
5: **end for**
6: Set $\{\mathbf{b}_1, \cdots, \mathbf{b}_n\}$ as the initial mapped database $\mathbf{B}^{(0)}$.
7: **for** $i = 1$ to n **do**
8: $\mathbf{s}[i] = \sum_{j=1}^{m} \mathbf{b}_i[j]$.
9: **end for**
10: $k = 0$.
11: **while** $\mathbf{B}^{(k)} \neq \emptyset$ **do**
12: Select from the current mapped database $\mathbf{B}^{(k)}$ the tuple $\mathbf{b}_{i^*}$ with the minimum $\mathbf{s}[i^*]$, denoted by $\mathbf{b}_\star$.
13: Add the tuple in $\mathbf{D}$ corresponding to $\mathbf{b}_\star$ to the skyline pool $\mathcal{R}_\mathbf{q}$.
14: Delete $\mathbf{b}_\star$ and tuples dominated by $\mathbf{b}_\star$ from $\mathbf{B}^{(k)}$.
15: $\mathbf{B}^{(k+1)} = \mathbf{B}^{(k)}$.
16: $k++$.
17: **end while**
18: **return** The set of skyline tuples $\mathcal{R}_\mathbf{q}$ with respect to $\mathbf{q}$.

Definition 2. (Skyline Query [28,12,44]). Suppose there are a database $\mathbf{D} = \{\mathbf{d}_1, \cdots, \mathbf{d}_n\}$ and a query tuple $\mathbf{q}$ in the same m-dimensional space. For any two tuples $\mathbf{d}_a$ and $\mathbf{d}_b$ in $\mathbf{D}$, we say $\mathbf{d}_a$ dominates $\mathbf{d}_b$ with respect to $\mathbf{q}$ iff $\forall j \in [1, m]$, $|\mathbf{d}_a[j] - \mathbf{q}[j]| \leq |\mathbf{d}_b[j] - \mathbf{q}[j]|$ and $\exists j \in [1, m]$, $|\mathbf{d}_a[j] - \mathbf{q}[j]| < |\mathbf{d}_b[j] - \mathbf{q}[j]|$. The skyline tuples with respect to $\mathbf{q}$, denoted by $\mathcal{R}_\mathbf{q}$, are the ones that cannot be dominated by any other tuple in the database $\mathbf{D}$.

It is noted that the conventional skyline computation (i.e., Definition 1) can be seen as a special case of skyline query when the query tuple is a vector of zeros. Algorithm 1 presents the steps involved in processing a skyline query in the plaintext domain [44,28]. Specifically, given a database $\mathbf{D}$ and a query $\mathbf{q}$, the skyline query result is produced through the following steps. The first step involves mapping the database $\mathbf{D}$ to a database $\mathbf{B}^{(0)}$, called mapped database, with respect to query $\mathbf{q}$. After that, the sum across all attributes for each tuple in $\mathbf{B}^{(0)}$ is computed (i.e., lines 7-9). Skyling query processing then goes through multiple rounds. During each round k, the tuple $\mathbf{b}_\star$ in the current mapped database $\mathbf{B}^{(k)}$ with the smallest attribute sum is selected as the (intermediate) skyline tuple. The tuple in $\mathbf{D}$ corresponding to $\mathbf{b}_\star$ is then inserted into the result set of skyline tuples $\mathcal{R}_\mathbf{q}$. Then, $\mathbf{b}_\star$ and the tuples it dominates are removed from $\mathbf{B}^{(k)}$, which produces $\mathbf{B}^{(k+1)}$ for the subsequent round. The process described above is repeated until there are no tuples remaining in the mapped database.

3.2. Additive secret sharing

Additive secret sharing [10] is an increasingly popular secure computation technique. It protects a secret value $x \in \mathbb{Z}_{2^l}$ by additively dividing it into two secret shares $\langle x \rangle_1, \langle x \rangle_2 \in \mathbb{Z}_{2^l}$, such that $x = \langle x \rangle_1 + \langle x \rangle_2$. Each individual share does not reveal any information about the underlying secret value x. The shares can then be distributed among two parties, who subsequently can perform some secure computation over the shares. Given the secret sharings of two values x and y, which are denoted as $[\![x]\!]$ and $[\![y]\!]$ respectively, the operations that can be directly supported in the secret sharing domain are introduced as follows. Addition/subtraction between $[\![x]\!]$ and $[\![y]\!]$ only needs local computation at each party, i.e., $\langle z \rangle_i = \langle x \rangle_i \pm \langle y \rangle_i, i \in \{1, 2\}$. The same applies to multiplication by a public constant, i.e., $\langle z \rangle_i = \eta \times \langle x \rangle_i$, where η is a public constant. As for multiplication between two secret-shared values

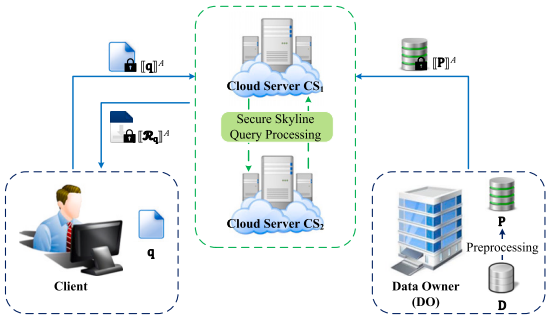

Fig. 2. The architecture of BopSkyline.

$[\![x]\!]$ and $[\![y]\!]$, denoted by $[\![z]\!] = [\![x]\!] \times [\![y]\!]$ for simplicity, it requires one round of communication among the two parties, with an extra secret-shared input called Beaver's triple that is data independent and can be generated offline by a third-party [35]. So in our design it is assumed that the secret shares of the triples have been properly distributed in advance for use in online secure computation.

When the secret values are binary values, i.e., they are in the ring $\mathbb{Z}_2$, the addition/subtraction and multiplication in the arithmetic domain are simply replaced by binary operations XOR ($\oplus$) and AND ($\otimes$), respectively. For clarity, we write $[\![\cdot]\!]^A$ for arithmetic secret sharing and $[\![\cdot]\!]^B$ for binary secret sharing. Accordingly, arithmetic shares are represented by $\langle\cdot\rangle_1^A$ and $\langle\cdot\rangle_2^A$, while binary shares are represented by $\langle\cdot\rangle_1^B$ and $\langle\cdot\rangle_2^B$. In the binary secret sharing domain, the NOT operation (denoted by $\neg$) can be easily supported. That is, one of the two parties can locally flip the secret share it holds, e.g., let $\langle\neg x\rangle_1^B = \neg\langle x\rangle_1^B$. Then, the other share is $\langle\neg x\rangle_2^B = \langle x\rangle_2^B$.

3.3. Differential privacy

Differential privacy [14] is a rigorous mathematical framework for ensuring the privacy of individual elements in a dataset. Intuitively, a differentially private mechanism ensures that the presence or absence of any individual element in an input dataset should not significantly alter the outcome. The formal definition of (ε, δ)-DP is as follows.

Definition 3. A randomized mechanism $\mathcal{M}$ is said to satisfy (ε, δ)-DP if and only if for any two neighboring datasets $\mathbf{D}$ and $\mathbf{D}'$ that differ by adding/removing a tuple, we have $\forall \mathbf{T} \in \mathbf{R}_{\mathcal{M}}$,

$$Pr[\mathcal{M}(\mathbf{D}) = \mathbf{T}] \leq e^{\varepsilon} \cdot Pr[\mathcal{M}(\mathbf{D}') = \mathbf{T}] + \delta,$$

where $\mathbf{R}_{\mathcal{M}}$ denotes the set of all possible outputs of mechanism $\mathcal{M}$, ε is the privacy budget, and δ is a privacy parameter.

In order to achieve differential privacy for a mechanism, one typically needs to add calibrated noises to the mechanism's output. We will use a truncated and discretized Laplace distribution [17] for drawing Laplace noises in our design. Its formal definition is as follows [17].

Definition 4. A discrete random variable x follows $Lap(\varepsilon, \delta, \Delta)$ distribution if its probability density function is

$$Pr[x] = \frac{e^{\frac{\varepsilon}{\Delta}} - 1}{e^{\frac{\varepsilon}{\Delta}} + 1} \cdot e^{\frac{-\varepsilon \cdot |x-\mu|}{\Delta}}, \forall x \in \mathbb{Z},$$

where μ is the mean of the Laplace distribution:

$$\mu = -\frac{\Delta \cdot \ln[(e^{\frac{\varepsilon}{\Delta}} + 1) \cdot (1 - (1-\delta)^{\frac{1}{\Delta}})]}{\varepsilon}, \tag{1}$$

and Δ is the sensitivity of a function f, defined as $\Delta = max|f(x) - f(x')|$, which captures the magnitude by which a single user's data can change the output of f in the worst case [14].

4. Problem statement

4.1. System architecture

The system architecture of BopSkyline is shown in Fig. 2. At the core, there are three types of entities: the client, the cloud, and the data owner. The data owner, such as a medical center, is a trusted entity that possesses a database and intends to provide the client (e.g., a doctor) with skyline query services. In order to harness the benefits of cloud computing while preserving data privacy, the data owner produces an encrypted version of its database before outsourcing the service. For high efficiency, BopSkyline resorts to the lightweight additive secret sharing technique to protect its database and to efficiently facilitate the subsequent secure skyline query service deployed at the cloud. Accordingly, BopSkyline adopts a distributed trust model, where the power of the cloud entity is supplied by two independent and non-colluding cloud service providers, leading to two cloud servers (referred to CS_1 and CS_2 respectively) holding the shares of the outsourced database and collaboratively providing the secure skyline query service. The adoption of such distributed trust model has also appeared in prior works [44,27,28,12] on secure skyline query processing as well as in other secure systems and applications [29,6,13,8,41]. In this paper, for simplicity of presentation, we will simply write $CS_{\{1,2\}}$ to represent the two cloud servers CS_1 and CS_2. Once the encrypted database is deployed at the cloud side, the client can issue an encrypted version of its query $\mathbf{q}$, denoted by $[\![\mathbf{q}]\!]^A$, to $CS_{\{1,2\}}$, which will then process the query as per BopSkyline's protocol and return the encrypted skyline query result $[\![\mathcal{R}_{\mathbf{q}}]\!]^A$ to the client for reconstruction.

4.2. Trust assumptions and security goals

BopSkyline assumes a non-colluding and semi-honest adversary model regarding the cloud servers, following prior work on secure skyline query processing [27,28,12,44] and other secure systems [13,6,39,1] under the two-server model. In particular, each cloud server will honestly execute the designated protocol for secure skyline query processing, yet is interested in inferring private information about the database and query along the service flow. In addition, the data owner and the client are considered as trusted entities [28,12]. With the above trust assumptions and consistent with prior work [44,28,12], BopSkyline aims to provide the following security guarantees:

- **Privacy of data content.** The cloud servers should be prevented from learning the data content of the original database $\mathbf{D}$, the query tuple $\mathbf{q}$, as well as the tuples in the skyline result $\mathcal{R}_{\mathbf{q}}$.
- **Privacy of data patterns.** The cloud servers should be prevented from learning private data pattern information including the dominance relationships among database tuples, and the number of database tuples that each skyline tuple dominates.
- **Privacy of search access patterns.** The cloud servers should be prevented from learning the search access patterns, which are defined as follows.
  - **Search pattern**. Given two skyline queries $\mathbf{q}$ and $\mathbf{q}'$, $\mathbf{q}$ and $\mathbf{q}'$ are identical, if and only if all corresponding attributes values of $\mathbf{q}$ and $\mathbf{q}'$ are identical. Let $\mathrm{I}(\mathbf{q}, \mathbf{q}') \in \{0, 1\}$ represent such identical relationship, i.e., $\mathrm{I}(\mathbf{q}, \mathbf{q}') = 1$ means that $\mathbf{q}$ and $\mathbf{q}'$ are identical. Given a non-empty sequence of skyline queries $\mathbf{Q} = \{\mathbf{q}_1, \cdots, \mathbf{q}_l\}$, the search pattern reveals an $l \times l$ matrix of the resulting of $\mathrm{I}(\mathbf{q}_i, \mathbf{q}_j)$.
  - **Access pattern**. Given a database $\mathbf{D}$ and a skyline query $\mathbf{q}$, the access pattern reveals the indices of skyline tuples with respect to $\mathbf{q}$ in $\mathbf{D}$.

5. The design of BopSkyline

5.1. Overview

We start by describing the starting point SecSkyline [44], which is the state-of-the-art protocol for privacy-preserving skyline query processing. SecSkyline has the same system architecture as ours (as shown in Fig. 2) and allows the cloud servers $CS_{\{1,2\}}$ to obliviously perform the functionality given in Algorithm 1 in the secret sharing domain. Specifically, SecSkyline consists of three secure components: 1) secure database mapping, which allows $CS_{\{1,2\}}$ to securely map the outsourced encrypted database to a new encrypted mapped database with respect to the encrypted query; 2) secure skyline fetching, which allows $CS_{\{1,2\}}$ to obliviously fetch an encrypted skyline tuple from the encrypted mapped database and the corresponding encrypted tuple from the outsourced encrypted database; 3) secure skyline and dominated tuples filtering, which allows $CS_{\{1,2\}}$ to obliviously filter out the skyline tuples and the tuple dominated by it from the encrypted mapped database as well as the corresponding encrypted tuples from the outsourced encrypted database.

Along the processing pipeline in SecSkyline, we observe that a crucial performance limitation exists in the way they deal with the (obliviously) identified skyline tuples and their dominated tuples. In particular, for protecting the data patterns and access pattern, the skyline tuple and its dominated tuples are not actually deleted in the component of secure skyline and dominated tuples filtering. Instead, they are just obliviously tagged via a specialized mechanism. Subsequently, in each round of secure skyline fetching, $CS_{\{1,2\}}$ have to always iterate over all tuples in the encrypted (mapped) database, leading to high performance complexity.

To achieve performance boost over SecSkyline [44], our main insight is to allow $CS_{\{1,2\}}$ to actually delete the skyline and dominated tuples found in each round while being oblivious to the true data patterns and access pattern. The key idea is to first have $CS_{\{1,2\}}$ permute the tuples in the encrypted original database by a random permutation unknown to them before securely processing a skyline query. After that, in each round $CS_{\{1,2\}}$ can safely reveal which tuples in the encrypted *permuted* database are the skyline and dominated tuples and delete them. In this way, the performance complexity can gradually decrease over the rounds. Since the encrypted tuples are permuted by a random permutation unknown to $CS_{\{1,2\}}$, they cannot determine which encrypted tuples in the original database are the skyline or dominated tuples, and thus cannot learn the true dominance relationships and access pattern.

One subtle issue here, however, is that directly revealing which tuples are the skyline and dominated tuples in the encrypted permuted database will leak the number of database tuples that each skyline tuple dominates. To tackle this issue, our solution is to let the data owner add some dummy tuples through tailored mechanisms into the original database in the database preparation phase. In this way, the number of database tuples that each skyline tuple dominates is obfuscated.

With the synergy of the above insights, we develop BopSkyline achieving a significant performance boost over SecSkyline. The high-level protocol workflow in BopSkyline is as follows. Firstly, in the secure database preparation phase (Section 5.2), the data owner blends some dummy tuples into its database, and then adequately encrypts each original tuple and dummy tuple in the padded database via arithmetic secret sharing, followed by sending the secret shares to $CS_{\{1,2\}}$. When the client issues its skyline query, it also encrypts the query via arithmetic secret sharing, and then sends the corresponding secret shares to $CS_{\{1,2\}}$. In the secure skyline query processing phase (Section 5.3), $CS_{\{1,2\}}$ obliviously process the encrypted skyline query on the encrypted outsourced (padded) database as per the tailored design of BopSkyline. Specifically, this phase is supported by the following secure components: secure database shuffling, secure database mapping, secure skyline fetching, and secure skyline and dominated tuples elimination. After $CS_{\{1,2\}}$ obliviously find all encrypted skylines, they return the corresponding secret shares to the client for reconstruction and producing the skyline query result.

5.2. Secure database preparation

In this phase, the data owner takes as input its original database and then produces an encrypted database to be outsourced. At a high level, the data owner needs to perform the following. Firstly, as mentioned above, in order to protect the data patterns and access pattern in the subsequent online secure query process, the data owner first adds some dummy tuples to the original database $\mathbf{D}$, producing a padded database $\mathbf{P}$. Then, the data owner can encrypt $\mathbf{P}$ via arithmetic secret sharing, producing the shares $\langle \mathbf{P} \rangle_1^A$ and $\langle \mathbf{P} \rangle_2^A$ which are sent to CS_1 and CS_2 respectively.

Along this workflow, the process of adding dummy tuples requires a delicate treatment. At a first glance, it seems that one could simply sample values from the same distribution as the original tuples to form the dummy tuples and then encrypt them by arithmetic secret sharing. Since encrypting the same value at different times under arithmetic secret sharing will always produce different shares, the resulting ciphertexts of dummy tuples are indistinguishable from that of original database tuples. However, the problem here is that the dummy tuples may affect the query accuracy as the dummy tuples may dominate real skyline tuples and get chosen to be returned in the query process.

To tackle this problem, our insight is to add an extra dimension for each database tuple with a delicate value. Specifically, holding an m-dimensional database $\mathbf{D}$, the data owner first appends an extra dimension to each original database tuple $\mathbf{d}_i$ ($i \in [1, n]$) and sets the value in the $(m+1)$-th dimension to zero, i.e., $\mathbf{d}_i[m+1] = 0$. Then the data owner generates some $(m+1)$-dimensional dummy tuples. Let $\hat{\mathbf{d}}$ denote a dummy tuple. The values in the original m dimensions, i.e., $\hat{\mathbf{d}}[1], \cdots, \hat{\mathbf{d}}[m]$, are constructed through sampling values from the same distribution as original tuples. But for the last dimension, $\hat{\mathbf{d}}[m+1]$ is set to a random positive number. After encryption, the encrypted dummy tuple and the encrypted original tuple are indistinguishable to $CS_{\{1,2\}}$. Like the original tuples, a dummy tuple may be chosen as a skyline tuple or a dominated tuple, which prevents $CS_{\{1,2\}}$ from learning which tuple is dummy. Meanwhile, such tailored design ensures that any dummy tuples will not dominate the real skyline tuples because the values in their extra dimensions are definitely greater than that of the real skyline tuples.

With the above design, the query accuracy will not be affected, for which we analyze as follows. First of all, the extra dimension added to each original database tuple has no impact on the dominance relationships among the original database tuples themselves. Meanwhile, the condition that $\mathbf{d}_i[m+1] < \hat{\mathbf{d}}[m+1]$ rules out the possibility that a true skyline tuple is dominated by any dummy tuple and is missing in the query result. Lastly, same as the original database tuple, the dummy tuple may be chosen as the skyline tuple which only dominates some other dummy tuples, but it can be easily filtered out by the client checking the value on the extra dimension of the tuples returned. Therefore, while some dummy tuples are added, all the actual skyline tuples with respect to the query can be correctly obtained and thus the skyline query accuracy is still guaranteed.

To facilitate clearer understanding of the design intuition, we give in Table 2 a padded version of the example database shown in Table 1. As shown in Table 2, we add an extra dimension with value 0 to each original database tuple, and then add two dummy tuples $\hat{\mathbf{d}}_1 = (50, 120, 5)$ and $\hat{\mathbf{d}}_2 = (52, 145, 10)$ to produce the padded database. The padded database is mapped to the mapped database with respect to query $\mathbf{q} = (46, 130, 0)$. Clearly, the dummy tuples cannot dominate any true tuple since the values at extra dimensions (i.e., 5 and 10) of dummy tuples are definitely greater than the zero value at the extra dimension of each true tuple. In addition, since the true tuples may dominate the dummy tuples (e.g., $\mathbf{b}_2$ dominates $\mathbf{b}_6$ and $\mathbf{b}_7$), the number of tuples dominated by each skyline

Table 2
An Example of Skyline Query on Padded Database.

Padded Database	Age	SBP	Extra Dimension
d_1	50	150	0
d_2	42	135	0
d_3	44	120	0
d_4	52	125	0
d_5	40	145	0
$\hat{d}_1$	50	120	5
$\hat{d}_2$	52	145	10
Mapped Database	**Age**	**SBP**	**Extra Dimension**
b_1	4	20	0
b_2	4	5	0
b_3	2	10	0
b_4	6	5	0
b_5	6	15	0
b_6	4	10	5
b_7	6	15	10

tuple is obfuscated. The final skyline query result is $\{\mathbf{d}_2, \mathbf{d}_3\}$ corresponding to $\{\mathbf{b}_2, \mathbf{b}_3\}$, same as the result of the example shown in Table 1.

The remaining challenge here is how to appropriately set the number of dummy tuples to delicately balance the trade-off between *efficiency* and *privacy*. Specifically, adding more dummy tuples will incur more system overhead, while adding less dummy tuples will result in weaker privacy guarantees on the data patterns. Therefore, a tailored design is required to provide a theoretically sound method by which the data owner can appropriately set the number of dummy tuples to balance efficiency and privacy. Our main insight is to resort to the DP [14] to make the leakage about the size of database differentially private, which in turn leads to obfuscation of the number of database tuples that each skyline tuple dominates.

Specifically, the data owner first draws a noise ξ from the discrete Laplace distribution $Lap(\varepsilon, \delta/2, 1)$, which is set as the number of dummy tuples to be added in the database. Here, the sensitivity Δ is set to 1 as the addition or removal of a single tuple alters the database size by 1. Then the total number of tuples in the padded database is $n+\xi$, denoted as $\hat{n}$. However, the drawn noise ξ could be negative, which means that in such case the data owner needs to delete some true tuple from its database. Obliviously, this will seriously degrade the effectiveness of the subsequent secure skyline query process. To deal with this issue, BopSkyline lets the data owner truncate ξ to 0 (i.e., $\xi = max(\xi, 0)$), inspired by [17]. In Section 6.1, we will formally prove that the leakage about the size of database is still differentially private although ξ may be truncated to 0.

5.3. Secure skyline query processing

5.3.1. Overview

After the encrypted padded database is deployed at the cloud, the client can send an encrypted skyline query tuple $[\![\mathbf{q}]\!]^A$ to the cloud for processing. Note that before encryption the true query tuple should also be augmented with an extra dimension with value 0. Algorithm 2 gives the overall workflow of secure skyline query processing in BopSkyline, which comprises the following core components: secure database shuffling (denoted as secShuffle), secure database mapping (denoted as secMap), secure skyline fetching (denoted as secFetch), and secure skyline and dominated tuples elimination (denoted as secEli).

At a high level, secure skyline query processing in BopSkyline proceeds as follows. Firstly, secShuffle is invoked to allow $CS_{\{1,2\}}$ to obliviously shuffle the encrypted padded database $[\![\mathbf{P}]\!]^A$, producing an encrypted shuffled database $[\![\widetilde{\mathbf{P}}^{(0)}]\!]^A$. Then, secMap is invoked to allow $CS_{\{1,2\}}$ to obliviously map $[\![\widetilde{\mathbf{P}}^{(0)}]\!]^A$ to the encrypted mapped database $[\![\mathbf{B}^{(0)}]\!]^A$ with respect to the encrypted query $[\![\mathbf{q}]\!]^A$. After that, $CS_{\{1,2\}}$ leverage secFetch and secEli to obliviously find the skyline tuples through multiple rounds until the encrypted mapped database becomes empty.

Algorithm 2 Secure Skyline Query Processing in BopSkyline.

Input: The encrypted padded database $[\![\mathbf{P}]\!]^A$ and the encrypted skyline query $[\![\mathbf{q}]\!]^A$.
Output: The encrypted skyline query result set $[\![\mathcal{R}_\mathbf{q}]\!]^A$.
1: **Initialization:** $[\![\mathcal{R}_\mathbf{q}]\!]^A = \emptyset, [\![\mathbf{s}]\!]^A = [\![\mathbf{0}]\!]^A$.
2: $[\![\widetilde{\mathbf{P}}^{(0)}]\!]^A = \text{secShuffle}([\![\mathbf{P}]\!]^A)$.
3: $[\![\mathbf{B}^{(0)}]\!]^A = \text{secMap}([\![\widetilde{\mathbf{P}}^{(0)}]\!]^A, [\![\mathbf{q}]\!]^A)$.
4: $[\![\mathbf{s}^{(0)}[i]]\!]^A = \sum_{j=1}^{m+1} [\![\mathbf{b}_i^{(0)}[j]]\!]^A, i \in [1, \hat{n}]$; # $\hat{n}$ is the number of tuples in $[\![\mathbf{B}^{(0)}]\!]^A$; $m+1$ is the number of dimensions of tuples.
5: $k = 0$.
6: **while** $[\![\mathbf{B}^{(k)}]\!]^A \neq \emptyset$ **do**
7: ind = secFetch($[\![\mathbf{s}^{(k)}]\!]^A$).
8: $CS_{\{1,2\}}$ retrieve $[\![\widetilde{\mathbf{p}}_\star]\!]^A$ with the index ind from $[\![\widetilde{\mathbf{P}}^{(k)}]\!]^A$.
9: $[\![\mathcal{R}_\mathbf{q}]\!]^A.append([\![\widetilde{\mathbf{p}}_\star]\!]^A)$.
10: $([\![\widetilde{\mathbf{P}}^{(k+1)}]\!]^A, [\![\mathbf{B}^{(k+1)}]\!]^A, [\![\mathbf{s}^{(k+1)}]\!]^A) \leftarrow$ secEli($[\![\mathbf{P}^{(k)}]\!]^A, [\![\mathbf{B}^{(k)}]\!]^A, [\![\mathbf{s}^{(k)}]\!]^A$, ind).
11: $k++$.
12: **end while**
13: **return** The encrypted skyline query result set $[\![\mathcal{R}_\mathbf{q}]\!]^A$.

Specifically, in the k-th round ($k \geq 0$), $CS_{\{1,2\}}$ leverage secFetch to securely fetch the index ind of the skyline tuple which has the minimum attribute sum in the current sum vector $[\![\mathbf{s}^{(k)}]\!]^A$. $CS_{\{1,2\}}$ can then retrieve the encrypted skyline tuple $[\![\widetilde{\mathbf{p}}_\star]\!]^A$ from the current shuffled database $[\![\widetilde{\mathbf{P}}^{(k)}]\!]^A$ via the index ind, followed by adding it to the encrypted result set $[\![\mathcal{R}_\mathbf{q}]\!]^A$. Afterwards, with secEli, $CS_{\{1,2\}}$ eliminate the skyline tuple and tuples dominated by it from $[\![\widetilde{\mathbf{P}}^{(k)}]\!]^A$, producing $[\![\widetilde{\mathbf{P}}^{(k+1)}]\!]^A$. Meanwhile, $CS_{\{1,2\}}$ eliminate from $[\![\mathbf{B}^{(k)}]\!]^A$ and $[\![\mathbf{s}^{(k)}]\!]^A$ the encrypted tuples/elements corresponding to the skyline and dominated tuples, producing $[\![\mathbf{B}^{(k+1)}]\!]^A$ and $[\![\mathbf{s}^{(k+1)}]\!]^A$. Finally, $CS_{\{1,2\}}$ return the encrypted skyline query result set $[\![\mathcal{R}_\mathbf{q}]\!]^A$ to the client for decryption. Note that $\mathcal{R}_\mathbf{q}$ may contain some dummy tuples, but the client can easily filter out them by checking the extra dimension. Recall that the value in the extra dimension of true skyline tuples is 0 but that of dummy skyline tuples is random positive number. In what follows, we present the detailed construction of each secure component.

5.3.2. Secure database shuffling

We first introduce how $CS_{\{1,2\}}$ obliviously shuffle the encrypted database $[\![\mathbf{P}]\!]^A$ via the secure component secShuffle. Recall that the purpose of secShuffle is to securely shuffle the tuples in $[\![\mathbf{P}]\!]^A$ (treated as a matrix in which each row corresponds to a tuple) to allow $CS_{\{1,2\}}$ to later safely reveal the skyline and dominated tuples in the permuted space in each round of skyline finding, while being oblivious to true data patterns and access pattern. We identify that the state-of-the-art secure shuffling protocol in [15] is well suited for our purpose, as it operates on secret-shared data under a two-party setting. We adapt this protocol in BopSkyline to achieve secure database shuffling as follows. Suppose that CS_1 holds $\langle \mathbf{P} \rangle_1^A$, π_1, $\mathbf{A}_1$ and Δ, and CS_2 holds $\langle \mathbf{P} \rangle_2^A$, π_2, $\mathbf{A}_2$ and $\mathbf{C}$, where $\mathbf{A}_1, \mathbf{A}_2, \mathbf{C}$ are random matrices with the same size as $\mathbf{P}$; π_1 and π_2 are random permutations over $\{1, 2, \cdots, \hat{n}\}$ ($\hat{n}$ is the number of tuples in $\mathbf{P}$); Δ is a matrix satisfying $\Delta = \pi_1(\pi_2(\mathbf{A}_1) + \mathbf{A}_2) - \mathbf{C}$, where the permutations are applied rowwise. These input-independent auxiliary values can be prepared and distributed by a third party offline (e.g., the data owner). Then $CS_{\{1,2\}}$ can obliviously shuffle the encrypted database $[\![\mathbf{P}]\!]^A$ to produce the encrypted shuffled database $[\![\widetilde{\mathbf{P}}^{(0)}]\!]^A$ as follows: (1) CS_1 sends $\mathbf{Z}_1 = \langle \mathbf{P} \rangle_1^A - \mathbf{A}_1$ to CS_2; (2) CS_2 sets $\langle \widetilde{\mathbf{P}}^{(0)} \rangle_2^A = \mathbf{C}$ and sends $\mathbf{Z}_2 = \pi_2(\mathbf{Z}_1 + \langle \mathbf{P} \rangle_2^A) - \mathbf{A}_2$ to CS_1; and (3) CS_1 sets $\langle \widetilde{\mathbf{P}}^{(0)} \rangle_1^A = \pi_1(\mathbf{Z}_2) + \Delta$. Finally, CS_1 and CS_2 hold the secret-shared shuffled database $[\![\widetilde{\mathbf{P}}]\!]^A = [\![\pi_1(\pi_2(\mathbf{P}))]\!]^A$, while neither of them knows the complete permutation $\pi_1(\pi_2(\cdot))$. The correctness holds since:

$$\begin{aligned}\langle \widetilde{\mathbf{P}}^{(0)} \rangle_1^A + \langle \widetilde{\mathbf{P}}^{(0)} \rangle_2^A &= \pi_1(\mathbf{Z}_2) + \Delta + \mathbf{C} \\ &= \pi_1(\mathbf{Z}_2) + \pi_1(\pi_2(\mathbf{A}_1) + \mathbf{A}_2)\end{aligned}$$

Algorithm 3 Secure Database Mapping (following SecSkyline [44]).

Input: The encrypted shuffled database $[\![\widetilde{\mathbf{P}}^{(0)}]\!]^A$ and the encrypted skyline query $[\![\mathbf{q}]\!]^A$.
Output: The encrypted mapped database $[\![\mathbf{B}^{(0)}]\!]^A$.
1: **for** $i = 1$ to $\hat{n}$ **do**
2: **for** $j = 1$ to $m+1$ **do**
3: $[\![\rho]\!]^B = \mathsf{SecCmp}([\![\widetilde{\mathbf{p}}_i^{(0)}[j]]\!]^A, [\![\mathbf{q}[j]]\!]^A)$.
4: $[\![\rho']\!]^B = [\![\neg\rho]\!]^B$.
5: $[\![\mathbf{b}_i^{(0)}[j]]\!]^A = \mathsf{SecMulBA}([\![\rho]\!]^B, [\![\mathbf{q}[j]]\!]^A - [\![\widetilde{\mathbf{p}}_i^{(0)}[j]]\!]^A)$ $+\mathsf{SecMulBA}([\![\rho']\!]^B, [\![\widetilde{\mathbf{p}}_i^{(0)}[j]]\!]^A - [\![\mathbf{q}[j]]\!]^A)$.
6: **end for**
7: **end for**
8: **return** The encrypted mapped database $[\![\mathbf{B}^{(0)}]\!]^A$.

Algorithm 4 Secure Skyline Fetching.

Input: The encrypted sum vector $[\![\mathbf{s}^{(k)}]\!]^A$ in the k-th round.
Output: The (disguised) index ind of the skyline tuple in the k-th round.
1: Initialization: ind = 1.
2: Let α denote the length of $\mathbf{s}^{(k)}$.
3: **for** $i = 2$ to α **do**
4: $[\![\rho]\!]^B = \mathsf{SecCmp}([\![\mathbf{s}^{(k)}[i]]\!]^A, [\![\mathbf{s}^{(k)}[\mathsf{ind}]]\!]^A)$.
5: $CS_{\{1,2\}}$ open $[\![\rho]\!]^B$ and obtain ρ.
6: **if** $\rho = 1$ **then**
7: ind = i.
8: **end if**
9: **end for**
10: **return** The (disguised) index ind of the skyline tuple in the permuted space.

$$= \pi_1(\pi_2(\mathbf{Z}_1 + \langle\mathbf{P}\rangle_2^A) + \pi_2(\mathbf{A}_1))$$
$$= \pi_1(\pi_2(\langle\mathbf{P}\rangle_1^A + \langle\mathbf{P}\rangle_2^A))$$
$$= \pi_1(\pi_2(\mathbf{P})).$$

5.3.3. Secure database mapping

We now introduce how $CS_{\{1,2\}}$ securely map the encrypted shuffled database $[\![\widetilde{\mathbf{P}}^{(0)}]\!]^A$ to the encrypted mapped database $[\![\mathbf{B}^{(0)}]\!]^A$ with respect to the encrypted query $[\![\mathbf{q}]\!]^A$ via the secure component secMap, which follows [44]. According to Algorithm 1, what needs to be securely computed are $\mathbf{b}_i^{(0)}[j] = |\widetilde{\mathbf{p}}_i^{(0)}[j] - \mathbf{q}[j]|, i \in [1, \hat{n}], j \in [1, m+1]$, where $m+1$ is the length of $\mathbf{b}_i^{(0)}$, $\widetilde{\mathbf{p}}_i^{(0)} \in \widetilde{\mathbf{P}}^{(0)}$, and $\mathbf{b}_i^{(0)} \in \mathbf{B}^{(0)}$. Therefore, the main operation $CS_{\{1,2\}}$ need to securely perform is the encrypted absolute value of the difference between $\widetilde{\mathbf{p}}_i^{(0)}[j]$ and $\mathbf{q}[j]$, which is introduced as follows.

Firstly, given two values a and b, we have $|a-b| = (a<b)\cdot(b-a) + \neg(a<b)\cdot(a-b)$, where $\neg$ represents the NOT operation; and $(a<b) = 1 \in \mathbb{Z}_2$ if $a<b$ and otherwise $(a<b) = 0 \in \mathbb{Z}_2$. With such transformation, the only operation that is not naturally supported in the secret sharing domain is the comparison operation. We note that SecSkyline uses a secure comparison protocol in the secret sharing domain, denoted by SecCmp. Given two secret sharings $[\![x]\!]^A$ and $[\![y]\!]^A$, the encrypted result of $(x<y)$ can be securely computed by $\mathsf{SecCmp}([\![x]\!]^A, [\![y]\!]^A)$. Namely, if $x<y$, $\mathsf{SecCmp}([\![x]\!]^A, [\![y]\!]^A) = [\![1]\!]^B$, and otherwise $\mathsf{SecCmp}([\![x]\!]^A, [\![y]\!]^A) = [\![0]\!]^B$. As the secret-shared comparison result is produced in the binary secret sharing domain, another protocol, denoted by SecMulBA, is used for secure multiplication between $[\![x]\!]^B$ and $[\![y]\!]^A$. We refer the readers to [44] for more details regarding SecCmp and SecMulBA. With the use of these two protocols, the encrypted absolute value of the difference $\widetilde{\mathbf{p}}_i^{(0)}[j] - \mathbf{q}[j]$ can be computed as per the above transformation. For completeness, we present the component of secure database mapping in Algorithm 3.

5.3.4. Secure skyline fetching

We now introduce how secFetch works to allow $CS_{\{1,2\}}$ to securely fetch the plaintext index ind (in the permuted space) of the skyline tuple in the k-th round. Specifically, $CS_{\{1,2\}}$ need to fetch the index ind of the skyline tuple who has the minimum attribute sum $sMin$ in the current sum vector $[\![\mathbf{s}^{(k)}]\!]^A$. This can be achieved leveraging the aforementioned secure comparison gadget SecCmp to securely compare the elements in $[\![\mathbf{s}^{(k)}]\!]^A$. Note that for the secure comparison of two elements, the comparison result can be *safely* opened so that $CS_{\{1,2\}}$ can know the index of the smaller element (yet without knowing its content). In this way, $CS_{\{1,2\}}$ can finally obtain the index ind of the minimum element in $[\![\mathbf{s}^{(k)}]\!]^A$, and thus the index of the skyline tuple (in the permuted space). We emphasize that since the database has been shuffled by a random permutation unknown to $CS_{\{1,2\}}$, the index ind obtained by $CS_{\{1,2\}}$ is in the permuted space (i.e., an disguised one) and so leaks no information about the actual position of the corresponding skyline tuple in the original database.

Algorithm 4 gives the detailed construction of secure skyline fetching. It is noted that for practical implementation, the divide-and-conquer approach can be leveraged for performance improvement. For instance, the secure calculation of the minimum in an encrypted 4-dimensional vector $[\![\mathbf{u}]\!]^A$ can be accomplished by: $\mathsf{Min}(\mathsf{Min}([\![\mathbf{u}[1]]\!]^A, [\![\mathbf{u}[2]]\!]^A), \mathsf{Min}([\![\mathbf{u}[3]]\!]^A, [\![\mathbf{u}[4]]\!]^A))$. As a result, the computation of $\mathsf{Min}([\![\mathbf{u}[1]]\!]^A, [\![\mathbf{u}[2]]\!]^A)$ and $\mathsf{Min}([\![\mathbf{u}[3]]\!]^A, [\![\mathbf{u}[4]]\!]^A)$ can be performed in parallel and the communication can be batched, saving the rounds of interactions.

5.3.5. Secure skyline and dominated tuples elimination

Given the encrypted (shuffled) database $[\![\widetilde{\mathbf{P}}^{(k)}]\!]^A$, mapped database $[\![\mathbf{B}^{(k)}]\!]^A$, attribute sum vector $[\![\mathbf{s}^{(k)}]\!]^A$, and index ind of the skyline tuple (in the permuted space), $CS_{\{1,2\}}$ need to obliviously eliminate the skyline tuple $[\![\mathbf{b}_\star]\!]^A$ and tuples dominated by it from $[\![\mathbf{B}^{(k)}]\!]^A$ without learning the dominance relationships among the original database tuples. Meanwhile, the tuples and values in $[\![\widetilde{\mathbf{P}}^{(k)}]\!]^A$ and $[\![\mathbf{s}^{(k)}]\!]^A$ that correspond to the eliminated tuples in $[\![\mathbf{B}^{(k)}]\!]^A$ are required to be eliminated as well. We present secEli to achieve this, as shown in Algorithm 5.

The main idea is to securely compute an elimination flag vector $[\![\mathbf{e}]\!]^B$ which indicates the positions of the tuples/values to be eliminated. The computation of this elimination flag vector is performed in the secret sharing domain, and at the end it can be safely opened to reveal the positions of tuples/values to be eliminated, as the positions are also in the permuted space. Specifically, $[\![\mathbf{e}]\!]^B = [\![\mathbf{0}]\!]^B$ at the beginning. If the i-th tuple is to be eliminated after secure evaluation, then $[\![\mathbf{e}[i]]\!]^B = [\![1]\!]^B$. Note that $CS_{\{1,2\}}$ directly set $[\![\mathbf{e}[\mathsf{ind}]]\!]^B = [\![1]\!]^B$, as ind is the position of the currently identified skyline tuple. Next we should consider how to allow $CS_{\{1,2\}}$ to securely set the flag corresponding to each tuple dominated by $[\![\mathbf{b}_\star]\!]^A$ to 1.

Recall that according to Definition 1, $\mathbf{b}_\star$ is said to dominate $\mathbf{b}_i$ if and only if $\forall j \in [1, m+1], \mathbf{b}_\star[j] \le \mathbf{b}_i[j]$, and $\exists j \in [1, m+1], \mathbf{b}_\star[j] < \mathbf{b}_i[j]$. We first evaluate the first condition (i.e., $\mathbf{b}_\star[j] \le \mathbf{b}_i[j], j \in [1, m+1]$), and then evaluate whether $\mathbf{b}_\star \neq \mathbf{b}_i$. With the SecCmp function, $CS_{\{1,2\}}$ can obliviously evaluate whether $\mathbf{b}_\star[j] \le \mathbf{b}_i[j], j \in [1, m+1]$ by

$$[\![\delta_j]\!]^B = \neg\mathsf{SecCmp}([\![\mathbf{b}_i[j]]\!]^A, [\![\mathbf{b}_\star[j]]\!]^A), \tag{2}$$

where $\neg$ represents the NOT operation and $\delta_j = 1$ indicates $\mathbf{b}_\star[j] \le \mathbf{b}_i[j]$. Then $CS_{\{1,2\}}$ aggregate the comparison results to $[\![\widehat{\delta}_i]\!]^B$ by

$$[\![\widehat{\delta}_i]\!]^B = [\![\delta_1]\!]^B \otimes \cdots \otimes [\![\delta_{m+1}]\!]^B, \tag{3}$$

where $\widehat{\delta}_i = 1$ indicates that $\mathbf{b}_i$ satisfies the first condition. More specifically, $\widehat{\delta}_i = 1$ means that $\forall j \in [1, m+1], \delta_j = 1$, i.e., $\mathbf{b}_\star[j] \le \mathbf{b}_i[j], j \in [1, m+1]$. Then $CS_{\{1,2\}}$ need to evaluate whether $\mathbf{b}_\star \neq \mathbf{b}_i$, which can be achieved by

$$[\![\sigma_i]\!]^B = \mathsf{SecCmp}([\![sMin]\!]^A, [\![\mathbf{s}[i]]\!]^A), \tag{4}$$

where $\sigma_i = 1$ indicates that the attribute sums of $\mathbf{b}_\star$ and $\mathbf{b}_i$ are different, i.e., $\mathbf{b}_\star \neq \mathbf{b}_i$. After that, $CS_{\{1,2\}}$ set the corresponding elimination flag $\mathbf{e}[i]$ by

$$[\![\mathbf{e}[i]]\!]^B = [\![\widehat{\delta}_i]\!]^B \otimes [\![\sigma_i]\!]^B, \tag{5}$$

where $\mathbf{e}[i] = 1$ only if both $\widehat{\delta}_i$ and σ_i are equal to 1, i.e., $\mathbf{b}_\star[j] \leq \mathbf{b}_i[j], j \in [1, m+1]$ and $\mathbf{b}_\star \neq \mathbf{b}_i$. That is, when $\mathbf{e}[i] = 1$, $\mathbf{b}_\star$ is smaller than $\mathbf{b}_i$ at least in one dimension and not larger in other dimensions. Therefore, $\mathbf{e}[i] = 1$ indicates that $\mathbf{b}_i$ is dominated by the current skyline tuple $\mathbf{b}_\star$ and needs to be eliminated.

Finally, $CS_{\{1,2\}}$ can safely open the elimination flag vector $[\![\mathbf{e}]\!]^B$ and eliminate the corresponding tuples or elements from $[\![\mathbf{B}^{(k)}]\!]^A$, $[\![\widetilde{\mathbf{P}}^{(k)}]\!]^A$, and $[\![\mathbf{s}^{(k)}]\!]^A$. Note that since secShuffle permutes the original database by a random permutation unknown to $CS_{\{1,2\}}$, they cannot learn which tuples in the original database are the dominated tuples. In addition, the existence of dummy tuples prevents $CS_{\{1,2\}}$ from knowing the exact number of the tuples dominated by the current skyline tuple.

Algorithm 5 Secure Skyline and Dominated Tuples Elimination.

Input: The encrypted padded database $[\![\widetilde{\mathbf{P}}^{(k)}]\!]^A$, mapped database $[\![\mathbf{B}^{(k)}]\!]^A$, attribute sum vector $[\![\mathbf{s}^{(k)}]\!]^A$ and (disguised) index ind of the current skyline tuple.

Output: The updated encrypted padded database $[\![\widetilde{\mathbf{P}}^{(k+1)}]\!]^A$, mapped database $[\![\mathbf{B}^{(k+1)}]\!]^A$, and attribute sum vector $[\![\mathbf{s}^{(k+1)}]\!]^A$.

Initialize $[\![\mathbf{e}]\!]^B = [\![\mathbf{0}]\!]^B$.
$CS_{\{1,2\}}$ obtain $[\![\mathbf{b}_\star]\!]^A$ and $[\![sMin]\!]^A$ with the index ind from $[\![\mathbf{B}^{(k)}]\!]^A$ and $[\![\mathbf{s}^{(k)}]\!]^A$, respectively.
Let β denote the number of tuples in $\mathbf{B}^{(k)}$.
for $i = 1$ to β **do**
  if $i =$ ind **then**
    $[\![\mathbf{e}[i]]\!]^B = [\![1]\!]^B$.
  else
    for $j = 1$ to $m+1$ **do**
      $[\![\delta_j]\!]^B = \neg\mathsf{SecCmp}([\![\mathbf{b}_i^{(k)}[j]]\!]^A, [\![\mathbf{b}_\star]\!]^A)$.
    end for
    $[\![\widehat{\delta}_i]\!]^B = [\![\delta_1]\!]^B \otimes \cdots \otimes [\![\delta_{m+1}]\!]^B$.
    $[\![\sigma_i]\!]^B = \mathsf{SecCmp}([\![sMin]\!]^A, [\![\mathbf{s}^{(k)}[i]]\!]^A)$.
    $[\![\mathbf{e}[i]]\!]^B = [\![\widehat{\delta}_i]\!]^B \otimes [\![\sigma_i]\!]^B$.
  end if
end for
$CS_{\{1,2\}}$ open $[\![\mathbf{e}]\!]^B$ and eliminate the corresponding tuples or elements from $[\![\mathbf{B}^{(k)}]\!]^A$, $[\![\widetilde{\mathbf{P}}^{(k)}]\!]^A$, and $[\![\mathbf{s}^{(k)}]\!]^A$.
return The updated encrypted padded database $[\![\widetilde{\mathbf{P}}^{(k+1)}]\!]^A$, mapped database $[\![\mathbf{B}^{(k+1)}]\!]^A$, and attribute sum vector $[\![\mathbf{s}^{(k+1)}]\!]^A$.

Remark. As mentioned above, for protecting the data patterns and access pattern, in each round of secure skyline fetching, the state-of-the-art work SecSkyline [44] does not let $CS_{\{1,2\}}$ actually delete the skyline tuple and its dominated tuples, and thus $CS_{\{1,2\}}$ have to always iterate over all tuples in the encrypted (mapped) database in each round, leading to high performance complexity. In contrast, by delicately integrating the idea of dummy tuples and secure shuffle, BopSkyline allows $CS_{\{1,2\}}$ to actually delete the skyline and dominated tuples found (in the permuted space) in each round while being oblivious to the true data patterns and access pattern. By this way, the performance complexity can gradually decrease over the rounds. In addition, as demonstrated by the experimental results in Section 7, the performance gap between BopSkyline and SecSkyline grows sharply as the size and dimension of database increase.

6. Security analysis

In this section, we provide formal analysis for the security guarantees BopSkyline offers in outsourced skyline query service. Recall that BopSkyline blends some dummy tuples based on DP into the original database before database outsourcing. Therefore, we will first present the proof that BopSkyline can make the leakage about the size of the outsourced database differentially private (which in turns leads to obfuscation of the number of tuples dominated by each skyline tuple in data patterns in the query phase), and then prove data confidentiality against the cloud servers.

6.1. Differential privacy-related analysis

Theorem 1. *BopSkyline can achieve (ε, δ)-DP for the database size according to Definition 3.*

Proof. Given the sizes n_i, n_j of databases $\mathbf{P}_i$ and $\mathbf{P}_j$ where $|n_i - n_j| = 1$. If both the noises drawn for $\mathbf{P}_i$ and $\mathbf{P}_j$ from $Lap(\varepsilon, \delta/2, 1)$ are non-negative, the probability of them outputting the same noisy size $\hat{n}$ is bounded by

$$\begin{aligned}\frac{Pr[\hat{n} - n_i]}{Pr[\hat{n} - n_j]} &= \frac{e^{\frac{-\varepsilon\cdot|\hat{n}-n_i-\mu|}{1}}}{e^{\frac{-\varepsilon\cdot|\hat{n}-n_j-\mu|}{1}}} \\ &= e^{\varepsilon\cdot(|\hat{n}-n_j-\mu|-|\hat{n}-n_i-\mu|)} \\ &\leq e^{\varepsilon\cdot|n_j-n_i|} = e^{\varepsilon}.\end{aligned}$$

In addition, the probability of drawing a negative noise from $Lap(\varepsilon, \delta/2, 1)$ is $\frac{e^{-\mu\cdot\varepsilon}}{e^{\varepsilon}+1}$ [17]. Given Eq. (1) and $\Delta \geq 0$, we have $\frac{e^{-\mu\cdot\varepsilon}}{e^{\varepsilon}+1} = 1 - (1 - \delta/2) = \delta/2$, which means that the probability of both the noises drawn for $\mathbf{P}_i$ and $\mathbf{P}_j$ are non-negative is $(1-\delta/2)(1-\delta/2) = 1 + \delta^2/4 - \delta$, and thus the overall failing probability is $1 - (1 + \delta^2/4 - \delta) = \delta - \delta^2/4 < \delta$. Therefore, with $1 - \delta$, the probability to output the same noisy size $\hat{n}$ is bounded by e^{ε}, which satisfies (ε, δ)-DP as per Definition 3. □

6.2. Data confidentiality-related analysis

Our analysis for the data confidentiality protection follows the standard simulation-based paradigm [24]. For secure skyline query processing, the formal definition of the ideal functionality $\mathcal{F}$ is described as follows:

- **Input.** The data owner provides the database $\mathbf{D}$ to $\mathcal{F}$ and a client provides a query $\mathbf{q}$ to $\mathcal{F}$.
- **Computation.** Upon receiving the database $\mathbf{D}$ and the skyline query $\mathbf{q}$, $\mathcal{F}$ retrieves the skyline tuples $\mathcal{R}_\mathbf{q}$ with respect to $\mathbf{q}$ from $\mathbf{D}$.
- **Output.** $\mathcal{F}$ returns the skyline tuples $\mathcal{R}_\mathbf{q}$ to the client.

Let $\prod$ denote the protocol design in BopSkyline for secure skyline query processing realizing the ideal functionality $\mathcal{F}$ against the semi-honest adversary model. The security of $\prod$ can be formally defined as:

Definition 5. Let $\mathsf{View}^{\prod}_{CS_i}$ denote each CS_i's view during the execution of $\prod$. $\prod$ is secure under the non-colluding and semi-honest adversary model, if for each corrupted CS_i there exists a probabilistic polynomial time simulator which can generate a simulated view Sim_{CS_i} such that $\mathsf{Sim}_{CS_i} \approx \mathsf{View}^{\prod}_{CS_i}$, i.e., the simulated view and the real view are indistinguishable.

Theorem 2. *In the non-colluding and semi-honest adversary model, BopSkyline is secure based on Definition 5.*

Proof. Note that in the secure skyline query phase, BopSkyline is composed of four components: 1) secure database shuffling (denoted as secShuffle); 2) secure database mapping (denoted as secMap); 3) secure skyline fetching (denoted as secFetch); 4) secure skyline and dominated tuples elimination (denoted as secEli). Note that the inputs and outputs of these components are secret shares, and they are processed sequentially according to the pipeline. Therefore, if the simulator for each component exists, the simulator for the whole protocol exists, indicating that BopSkyline is secure. Let $\mathsf{Sim}^{\mathtt{X}}_{CS_i}$ denote the simulator simulating the view of CS_i in component X.

- $\mathsf{Sim}^{\mathsf{secShuffle}}_{CS_i}$. In this phase, CS_i only receives the secret shares during the execution of secure shuffle. The simulator $\mathsf{Sim}^{\mathsf{secShuffle}}_{CS_i}$ can

be trivially constructed by invoking the simulator of secure shuffle [15]. Therefore, the simulator $\mathsf{Sim}_{CS_i}^{\mathsf{secShuffle}}$ exists.

- $\mathsf{Sim}_{CS_i}^{\mathsf{secMap}}$. It is noted that secMap follows the secure database mapping component in SecSkyline [44]. We refer the readers to [44] for more details regarding the proof of the existence of simulator $\mathsf{Sim}_{CS_i}^{\mathsf{secMap}}$.
- $\mathsf{Sim}_{CS_i}^{\mathsf{secFetch}}$. It is noted that secFetch (i.e., Algorithm 4) consists of SecCmp and SecMulBA protocols except the basic secret-shared operations. Meanwhile, these operations are invoked in turn and their inputs and outputs are secret shares. Therefore, following the proof of SecSkyline [44], the simulators for the two protocols exist. In addition, for the opened value ρ (i.e., line 5 in Algorithm 4), the simulator can adjust the honest server's secret share such that the opened value is indeed what it receives from the ideal functionality [30]. Therefore, the simulator $\mathsf{Sim}_{CS_i}^{\mathsf{secFetch}}$ exists.
- $\mathsf{Sim}_{C_i}^{\mathsf{secEli}}$. secEli consists of SecCmp, secure bit flipping and basic secret-shared operations, which are invoked in turn and their inputs and outputs are secret shares. Since secure bit flipping is local operation and CS_i receives nothing during its execution, its simulator exists. Therefore, given the security of SecCmp [44] and basic secret-shared operations [10], the simulator $\mathsf{Sim}_{C_i}^{\mathsf{secEli}}$ exists.

The proof of Theorem 2 is completed. □

6.3. On protection for data patterns and search access patterns

We now explicitly analyze how BopSkyline protects the data patterns and search access patterns as follows:

- **Protecting the data patterns.** The data patterns in BopSkyline include the dominance relationships among database tuples and the number of database tuples that each skyline tuple dominates. It is noted that only the secure skyline fetching secFetch and secure skyline and dominated tuples elimination secEli may leak the dominance relationships. Since before the execution of secFetch and secEli, the secure database shuffling secShuffle permutes the tuples in the original database by a random permutation unknown to $CS_{\{1,2\}}$, they cannot infer the true dominance relationships between the tuples in the original database even if BopSkyline allows them to know which tuples in the permuted database are the skyline or dominated tuples. In addition, the dummy tuples blended into the original database in secure database preparation phase obfuscate the number of database tuples that each skyline tuple dominates. Therefore, BopSkyline can protect the data patterns.
- **Protecting the search pattern.** The search pattern reveals whether a skyline query is a repeated one. Given an encrypted skyline query $[\![\mathbf{q}]\!]^A$, $CS_i, i \in \{1,2\}$ only obtains the secret shares $\langle \mathbf{q} \rangle_i^A$ from the client. The security of secret sharing guarantees that the same secret value will be encrypted into different shares indistinguishable from uniformly random values [10] at different times of encryption. So $CS_{\{1,2\}}$ cannot determine whether a received encrypted skyline query is a repeated one.
- **Protecting the access pattern.** The access pattern reveals whether a tuple in the encrypted database $[\![\mathbf{P}]\!]^A$ is a skyline tuple, i.e., which tuples will appear in the result set $[\![\mathcal{R}_\mathbf{q}]\!]^A$. In BopSkyline, $CS_{\{1,2\}}$ fetch the encrypted skyline tuples from the shuffled database $[\![\widetilde{\mathbf{P}}^{(0)}]\!]^A$, which is produced by permuting the tuples in $[\![\mathbf{P}]\!]^A$ by a random permutation unknown to $CS_{\{1,2\}}$. Since the oblivious shuffle breaks the mappings between the tuples in $[\![\widetilde{\mathbf{P}}^{(0)}]\!]^A$ and $[\![\mathbf{P}]\!]^A$, even if $CS_{\{1,2\}}$ learn which encrypted tuples in the shuffled database $[\![\widetilde{\mathbf{P}}^{(0)}]\!]^A$ are skyline tuples, they cannot infer which encrypted tuples in the original database $[\![\mathbf{P}]\!]^A$ are skyline tuples. Therefore, BopSkyline can protect the access pattern.

7. Experiments

7.1. Setup

We implement BopSkyline's protocols using C++ and conduct experiments on a 64-bit Windows 10 machine equipped with AMD Ryzen 7 5800H CPU cores and 16 GB RAM. In addition, we implement the state-of-the-art work SecSkyline [44] with C++ and test it on the same machine as a baseline. We run two threads to simulate two cloud serves on the same machine. We set the network delay to 1 ms, which is identical to SecSkyline [44]. The privacy parameter δ is set to 1×10^{-6} Following prior works [28,12,44], we generate and use three different kinds of synthetic datasets for evaluation: correlated (CORR), independent (INDE), and anti-correlated (ANTI). More specifically, for the CORR datasets, the tuples' values across all dimensions exhibit positive correlation; for the INDE datasets, all values are generated independently; for the ANTI datasets, tuples typically contain large values in one dimension while having small values in one or all of the other dimensions. All values in our experiments are in the ring $\mathbb{Z}_{2^{64}}$. Unless otherwise stated, the experiment results on a database are the average over 100 skyline queries.

7.2. Evaluation on accuracy

We start with examining the accuracy of BopSkyline to showcase its effectiveness. In particular, we generate 1000 skyline queries for different datasets and compare the results of skyline tuples obtained using BopSkyline to those obtained using the plaintext algorithm (i.e., Algorithm 1) with respect to these queries. Our observation is that the skyline tuples produced by BopSkyline over different datasets match exactly that produced by the plaintext Algorithm. In other words, the accuracy is measured to be 100%. So while providing strong security guarantees, BopSkyline does not sacrifice the accuracy.

7.3. Evaluation on performance

7.3.1. Evaluation on query latency

We now evaluate BopSkyline's query latency, and compare the results with that of the state-of-the-art work SecSkyline [44] to demonstrate the advantage of BopSkyline in query latency. The query latency is defined as the time it takes $CS_{\{1,2\}}$ to securely execute skyline search over the encrypted outsourced database to produce encrypted skyline tuples as the result, given an encrypted skyline query as input. We first evaluate and compare BopSkyline and SecSkyline with $m = 2, \varepsilon = 1$ (m is the number of dimensions and ε is the privacy budget of DP), for varying $n \in \{1 \times 10^5, 2 \times 10^5, 3 \times 10^5, 4 \times 10^5, 5 \times 10^5, 6 \times 10^5\}$ (n is the number of tuples). The results are provided in Fig. 3, Fig. 4 and Fig. 5. It can be observed that BopSkyline is $2.9\times - 4.6\times$ faster than SecSkyline. In particular, as n increases from 1×10^5 to 6×10^5, the query latency of BopSkyline only increases from 0.8s to 4.1s, but the query latency of SecSkyline increases from 2.4s to 18.2s. We then evaluate BopSkyline and SecSkyline with $n = 1 \times 10^5, \varepsilon = 1$, for varying $m \in \{2, 3, 4, 5, 6\}$. The results are presented in Fig. 6, Fig. 7, and Fig. 8. We can observe that BopSkyline is 2.2×-4.7× faster than SecSkyline. Moreover, we can observe that the gap between the query latency of BopSkyline and SecSkyline increases sharply as m and n increase. The results above demonstrate that BopSkyline achieves significant improvement in query latency compared with the SecSkyline.

7.3.2. Evaluation on communication performance

We now evaluate the online communication cost of BopSkyline and compare the results with SecSkyline [44]. The online communication cost is defined as the size of data that $CS_{\{1,2\}}$ communicate with each other in performing secure skyline query processing for an encrypted skyline query. We follow the identical parameter setting as in Section 7.3.1 and present a summary of the experiment results

Table 3
Communication cost (in GB) on Different Datasets, for Varying Number n of Tuples (with $m = 2, \varepsilon = 1$).

Number of tuples n	CORR			INDE			ANTI		
	SecSkyline	BopSkyline	Saving	SecSkyline	BopSkyline	Saving	SecSkyline	BopSkyline	Saving
1×10^5	1.05	**0.09**	91.43%	0.72	**0.09**	87.50%	1.40	**0.12**	91.43%
2×10^5	2.16	**0.19**	91.20%	1.52	**0.19**	87.50%	2.53	**0.25**	90.12%
3×10^5	3.39	**0.28**	91.74%	1.94	**0.28**	85.57%	5.19	**0.36**	93.06%
4×10^5	4.09	**0.37**	90.95%	2.30	**0.37**	83.91%	5.55	**0.48**	91.35%
5×10^5	5.62	**0.47**	91.64%	3.26	**0.46**	85.89%	6.69	**0.59**	91.18%
6×10^5	6.43	**0.56**	91.29%	3.74	**0.56**	85.03%	11.49	**0.76**	93.39%

Table 4
Communication cost (in GB) on Different Datasets, for Varying Number m of Dimensions (with $n = 1 \times 10^5, \varepsilon = 1$).

Dimension m	CORR			INDE			ANTI		
	SecSkyline	BopSkyline	Saving	SecSkyline	BopSkyline	Saving	SecSkyline	BopSkyline	Saving
2	1.05	**0.09**	91.43%	0.72	**0.09**	87.50%	1.40	**0.12**	91.43%
3	7.78	**0.16**	97.94%	6.42	**0.13**	97.98%	14.58	**0.21**	98.56%
4	42.13	**0.46**	98.91%	34.63	**0.24**	99.31%	43.30	**0.61**	98.59%
5	260.76	**2.25**	99.14%	128.42	**0.79**	99.38%	143.70	**2.11**	98.53%
6	661.70	**17.94**	97.29%	387.16	**3.40**	99.12%	318.18	**9.12**	97.13%

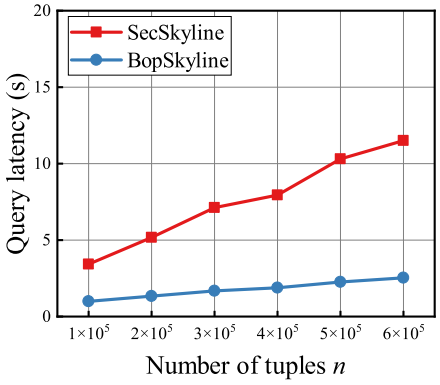

Fig. 3. Query latency on CORR, for varying number of tuples n (with the number of dimensions $m = 2$ and the privacy budget of DP $\varepsilon = 1$).

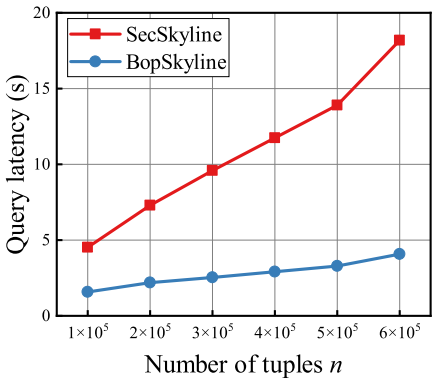

Fig. 5. Query latency on ANTI, for varying number of tuples n (with the number of dimensions $m = 2$ and the privacy budget of DP $\varepsilon = 1$).

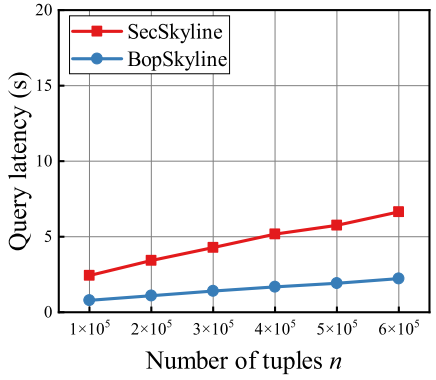

Fig. 4. Query latency on INDE, for varying number of tuples n (with the number of dimensions $m = 2$ and the privacy budget of DP $\varepsilon = 1$).

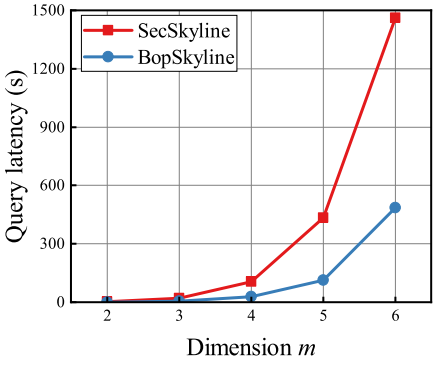

Fig. 6. Query latency on CORR, for varying the number of dimensions m (with the number of tuples $n = 1 \times 10^5$ and the privacy budget of DP $\varepsilon = 1$).

in Table 3 and Table 4. It can be observed that BopSkyline achieves $83.91\% - 99.38\%$ savings in communication cost against SecSkyline. The results demonstrate that BopSkyline achieves substantial improvement in the online communication performance over SecSkyline.

7.3.3. Query latency evaluation under varying privacy budget

We now report the query latency of BopSkyline with $n = 1 \times 10^5, m = 2$, for varying the privacy budget $\varepsilon \in \{0.2, 0.4, 0.6, 0.8, 1.0\}$. The results are shown in Fig. 9. It can be observed that the query latency increases gracefully as the privacy budget ε decreases. Table 5 summarizes the mean μ for the Laplace distribution at different privacy budgets ε, corresponding to the average number of dummy tuples blended into the original database. It can be observed that even at a limited privacy budget, the mean remains small, e.g., 65 with $\varepsilon = 0.2$. This results in a graceful increase in query latency due to the presence of dummy tuples.

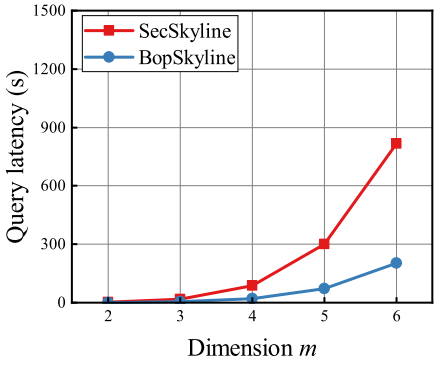

Fig. 7. Query latency on INDE, for varying the number of dimensions m (with the number of tuples $n = 1 \times 10^5$ and the privacy budget of DP $\varepsilon = 1$).

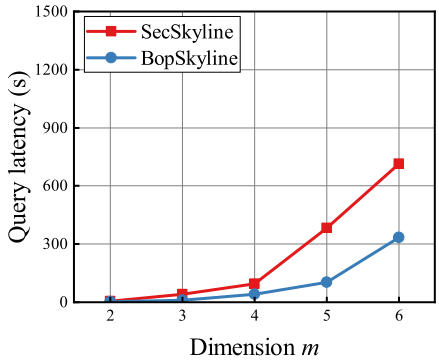

Fig. 8. Query latency on ANTI, for varying the number of dimensions m (with the number of tuples $n = 1 \times 10^5$ and the privacy budget of DP $\varepsilon = 1$).

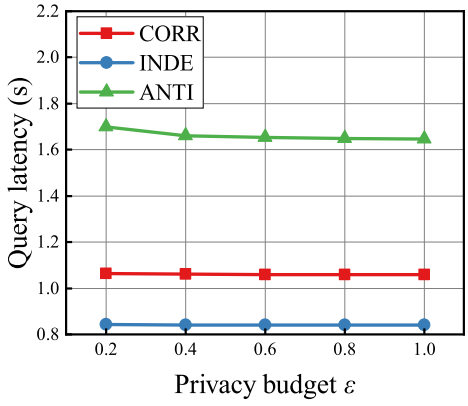

Fig. 9. Query Latency with $n = 1 \times 10^5, m = 2$, for varying privacy budget ε.

Table 5
The Mean μ for the Laplace Distribution at Different Privacy Budgets ε.

ε	0.2	0.4	0.6	0.8	1.0
μ	65	32	21	16	13

8. Conclusion and future work

In this paper, we present a new system framework BopSkyline for privacy-preserving skyline query processing in the cloud. BopSkyline is built from a delicate synergy of lightweight cryptography and differential privacy techniques to allow the cloud to obliviously process skyline queries, providing protections for not only data content confidentiality but also data patterns and search access patterns. We conduct evaluations over several datasets and the results demonstrate that compared with the state-of-the-art prior work SecSkyline, BopSkyline is up to 4.7× better in query latency and saves up to 99.38% communication cost.

Like the stat-of-the-art prior work [44], as well as earlier studies [27,28], the current design of BopSkyline does not consider the malicious adversary model. It is noted that the protocols providing semi-honest security are often considered as a stepping stone for the development of protocols capable of withstanding malicious adversaries. To harden our current protocol design with malicious security, one feasible technical direction is to leverage information-theoretic MAC [7]. Here we introduce the basic idea at a high level. The data owner additionally generates secret-shared MAC for each private value when outsourcing the encrypted database. Subsequently, when $CS_{\{1,2\}}$ perform secure skyline queries on the encrypted database, they not only undertake the specified secure operations on the private values but also extend these operations to the corresponding MACs. After the client receives the secret shares of the result, as well as the secret shares of the corresponding MACs, it can verify the integrity of the result based on their relationships. We leave a more detailed study as future work.

CRediT authorship contribution statement

Weibo Wang: Conceptualization, Formal analysis, Methodology, Software, Writing – original draft. **Yifeng Zheng:** Conceptualization, Funding acquisition, Methodology, Supervision, Writing – review & editing. **Songlei Wang:** Conceptualization, Methodology, Writing – review & editing. **Zhongyun Hua:** Conceptualization, Validation, Writing – review & editing. **Lei Xu:** Validation, Writing – review & editing. **Yansong Gao:** Visualization, Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgement

This work was supported in part by the Guangdong Basic and Applied Basic Research Foundation under Grants 2021A1515110027, 2023A1515010714, and 2021A1515011406, by the Shenzhen Science and Technology Program under Grants RCBS20210609103056041, JCYJ20220531095416037, and JCYJ20230807094411024, by the National Natural Science Foundation of China under Grant 62202228, by the Natural Science Foundation of Jiangsu Province under Grant BK20210330, and by the Fundamental Research Funds for the Central Universities under Grant 30923011023.

References

[1] N. Agrawal, A.S. Shamsabadi, M.J. Kusner, A. Gascón, QUOTIENT: two-party secure neural network training and prediction, in: Proc. of ACM CCS, 2019.
[2] W. Balke, U. Güntzer, J.X. Zheng, Efficient distributed skylining for web information systems, in: Proc. of EDBT, 2004.
[3] BleepingComputer, Flexbooker discloses data breach, over 3.7 million accounts impacted [Online], https://www.bleepingcomputer.com/news/security/flexbooker-discloses-data-breach-over-37-million-accounts-impacted/. (Accessed 6 March 2023).
[4] S. Börzsönyi, D. Kossmann, K. Stocker, The skyline operator, in: Proc. of IEEE ICDE, 2001.
[5] S. Bothe, A. Cuzzocrea, P. Karras, A. Vlachou, Skyline query processing over encrypted data: an attribute-order-preserving-free approach, in: Proc. of International Workshop on Privacy and Security of Big Data, 2014.
[6] W. Chen, R.A. Popa, Metal: a metadata-hiding file-sharing system, in: Proc. of NDSS, 2020.

[7] R. Cramer, I. Damgård, D. Escudero, P. Scholl, C. Xing, $SPDZ_{2^k}$: efficient MPC mod 2^k for dishonest majority, in: Proc. of CRYPTO, 2018.
[8] N. Cui, X. Yang, B. Wang, J. Li, G. Wang, Svknn: efficient secure and verifiable k-nearest neighbor query on the cloud platform, in: Proc. of IEEE ICDE, 2020.
[9] E. Dellis, A. Vlachou, I. Vladimirskiy, B. Seeger, Y. Theodoridis, Constrained subspace skyline computation, in: Proc. of ACM CIKM, 2006.
[10] D. Demmler, T. Schneider, M. Zohner, ABY - a framework for efficient mixed-protocol secure two-party computation, in: Proc. of NDSS, 2015.
[11] K. Deng, X. Zhou, H.T. Shen, Multi-source skyline query processing in road networks, in: Proc. of IEEE ICDE, 2007.
[12] X. Ding, Z. Wang, P. Zhou, K.-K.R. Choo, H. Jin, Efficient and privacy-preserving multi-party skyline queries over encrypted data, IEEE Trans. Inf. Forensics Secur. 16 (2021) 4589–4604.
[13] M. Du, S. Wu, Q. Wang, D. Chen, P. Jiang, A. Mohaisen, Graphshield: dynamic large graphs for secure queries with forward privacy, IEEE Trans. Knowl. Data Eng. 34 (7) (2020) 3295–3308.
[14] C. Dwork, Differential privacy, in: Proc. of ICALP, 2006.
[15] S. Eskandarian, D. Boneh, Clarion: anonymous communication from multiparty shuffling protocols, in: Proc. of NDSS, 2022.
[16] M. Hähnel, W. Cui, M. Peinado, High-resolution side channels for untrusted operating systems, in: Proc. of USENIX ATC, 2017.
[17] X. He, A. Machanavajjhala, C.J. Flynn, D. Srivastava, Composing differential privacy and secure computation: a case study on scaling private record linkage, in: Proc. of ACM CCS, 2017.
[18] Z. Huang, C.S. Jensen, H. Lu, B.C. Ooi, Skyline queries against mobile lightweight devices in MANETs, in: Proc. of IEEE ICDE, 2006.
[19] P. Jiang, Q. Wang, M. Huang, C. Wang, Q. Li, C. Shen, K. Ren, Building in-the-cloud network functions: security and privacy challenges, Proc. IEEE 109 (12) (2021) 1888–1919.
[20] M.E. Khalefa, M.F. Mokbel, J.J. Levandoski, Skyline query processing for incomplete data, in: Proc. of IEEE ICDE, 2008.
[21] D. Kossmann, F. Ramsak, S. Rost, Shooting stars in the sky: an online algorithm for skyline queries, in: Proc. of VLDB, 2002.
[22] D. Lee, D. Jung, I.T. Fang, C.-C. Tsai, R.A. Popa, An off-chip attack on hardware enclaves via the memory bus, in: Proc. of USENIX Security, 2020.
[23] S. Lee, M.-W. Shih, P. Gera, T. Kim, H. Kim, M. Peinado, Inferring fine-grained control flow inside sgx enclaves with branch shadowing, in: Proc. of USENIX Security, 2017.
[24] Y. Lindell, How to simulate it - a tutorial on the simulation proof technique, in: Tutorials on the Foundations of Cryptography, 2017, pp. 277–346.
[25] J. Liu, L. Xiong, J. Pei, J. Luo, H. Zhang, Finding Pareto optimal groups: group-based skyline, Proc. VLDB Endow. 8 (13) (2015) 2086–2097.
[26] J. Liu, H. Zhang, L. Xiong, H. Li, J. Luo, Finding probabilistic k-skyline sets on uncertain data, in: Proc. of ACM CIKM, 2015.
[27] J. Liu, J. Yang, L. Xiong, J. Pei, Secure skyline queries on cloud platform, in: Proc. of IEEE ICDE, 2017.
[28] J. Liu, J. Yang, L. Xiong, J. Pei, Secure and efficient skyline queries on encrypted data, IEEE Trans. Knowl. Data Eng. 31 (7) (2019) 1397–1411.
[29] X. Meng, H. Zhu, G. Kollios, Top-k query processing on encrypted databases with strong security guarantees, in: Proc. of IEEE ICDE, 2018.
[30] P. Mohassel, Y. Zhang, SecureML: a system for scalable privacy-preserving machine learning, in: Proc. of IEEE S&P, 2017.
[31] D. Papadias, Y. Tao, G. Fu, B. Seeger, Progressive skyline computation in database systems, ACM Trans. Database Syst. 30 (1) (2005) 41–82.
[32] J. Pei, W. Jin, M. Ester, Y. Tao, Catching the best views of skyline: a semantic approach based on decisive subspaces, in: Proc. of VLDB, 2005.
[33] J. Pei, B. Jiang, X. Lin, Y. Yuan, Probabilistic skylines on uncertain data, in: Proc. of VLDB, 2007.
[34] Z. Qin, J. Weng, Y. Cui, K. Ren, Privacy-preserving image processing in the cloud, IEEE Cloud Comput. 5 (2) (2018) 48–57.
[35] M.S. Riazi, C. Weinert, O. Tkachenko, E.M. Songhori, T. Schneider, F. Koushanfar, Chameleon: a hybrid secure computation framework for machine learning applications, in: Proc. of ACM AsiaCCS, 2018.
[36] Y. Tao, D. Papadias, Maintaining sliding window skylines on data streams, IEEE Trans. Knowl. Data Eng. 18 (2) (2006) 377–391.
[37] J. Van Bulck, N. Weichbrodt, R. Kapitza, F. Piessens, R. Strackx, Telling your secrets without page faults: stealthy page table-based attacks on enclaved execution, in: Proc. of USENIX Security, 2017.
[38] J. Wang, M. Du, S.S.M. Chow, Stargazing in the dark: secure skyline queries with SGX, in: Proc. of DASFAA, 2020.
[39] Q. Wang, J. Wang, S. Hu, Q. Zou, K. Ren, Sechog: privacy-preserving outsourcing computation of histogram of oriented gradients in the cloud, in: Proc. of ACM AsiaCCS, 2016.
[40] Z. Wang, X. Ding, H. Jin, P. Zhou, Efficient secure and verifiable location-based skyline queries over encrypted data, Proc. VLDB Endow. 15 (9) (2022) 1822–1834.
[41] L. Xu, J. Jiang, B. Choi, J. Xu, S.S. Bhowmick, Privacy preserving strong simulation queries on large graphs, in: Proc. of IEEE ICDE, 2021.
[42] W. Yu, Z. Qin, J. Liu, L. Xiong, X. Chen, H. Zhang, Fast algorithms for Pareto optimal group-based skyline, in: Proc. of ACM CIKM, 2017.
[43] S. Zhang, S. Ray, R. Lu, Y. Zheng, Y. Guan, J. Shao, Towards efficient and privacy-preserving user-defined skyline query over single cloud, in: IEEE Transactions on Dependable and Secure Computing, 2022.
[44] Y. Zheng, W. Wang, S. Wang, X. Jia, H. Huang, C. Wang, Secskyline: fast privacy-preserving skyline queries over encrypted cloud databases, IEEE Trans. Knowl. Data Eng. 35 (9) (2023) 8955–8967.

Weibo Wang received the BE degree in software engineering from Zhejiang University of Technology, China, in 2021. He is currently working toward the ME degree in the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His research interests include cloud computing security and secure muti-party computation.

Yifeng Zheng is an Assistant Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. He worked as a postdoc with the Commonwealth Scientific and Industrial Research Organization (CSIRO), Australia, and City University of Hong Kong. His current research interests are focused on security and privacy related to cloud computing, IoT, machine learning, and multimedia.

Songlei Wang received the BE degree in internet of things from China University of Petroleum (East China), Qingdao, China, in 2018, the ME degree in computer technology from Harbin Institute of Technology, Shenzhen, China, in 2021. He is currently working toward the PhD degree in the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, China. His research interests include cloud computing security and secure machine learning.

Zhongyun Hua received the B.S. degree from Chongqing University, Chongqing, China, in 2011, and the M.S. and Ph.D. degrees from University of Macau, Macau, China, in 2013 and 2016, respectively, all in software engineering. He is currently an Associate Professor with the School of Computer Science and Technology, Harbin Institute of Technology, Shenzhen, Shenzhen, China. His research interests include chaotic system and information security.

Lei Xu is currently an Associate Professor at the school of Mathematics and Statistics, Nanjing University of Science and Technology. He received his Ph.D. degree in Nanjing University of Science and Technology, 2019. He was also ever a visiting Ph.D. student at Faculty of Information Technology, Monash University during the period from April 2017 to April 2018. His main research interests are focused on applied cryptography and information security, including encrypted search, differential privacy and quantum-resistant cryptography.

Yansong Gao is a tenured Research Scientist at Data61, CSIRO. Prior to that, he was an Associate Professor at Nanjing University of Science and Technology. He received his M.Sc degree from the University of Electronic Science and Technology of China in 2013 and a Ph.D. degree from the University of Adelaide, Australia, in 2017. His current research interests are AI security and privacy, hardware security, and system security.