SPECIAL: SynoPsis AssistEd Secure CollaboratIve AnaLytics

Chenghong Wang
Indiana University
cw166@iu.edu

Lina Qiu
Boston University
qlina@bu.edu

Johes Bater
Tufts University
johes.bater@tufts.edu

Yukui Luo
Umass Dartmouth
yluo2@umassd.edu

ABSTRACT

Secure collaborative analytics (SCA) enables the processing of analytical SQL queries across data from multiple owners, even when direct data sharing is not possible. While traditional SCA provides strong privacy through data-oblivious methods, the significant overhead has limited its practical use. Recent SCA variants that allow controlled leakages under differential privacy (DP) strike balance between privacy and efficiency but still face challenges like unbounded privacy loss, costly execution plan, and lossy processing.

To address these challenges, we introduce SPECIAL, the first SCA system that simultaneously ensures bounded privacy loss, advanced query planning, and lossless processing. SPECIAL employs a novel *synopsis-assisted secure processing model*, where a one-time privacy cost is used to generate private synopses from owner data. These synopses enable SPECIAL to estimate compaction sizes for secure operations (e.g., filter, join) and index encrypted data without additional privacy loss. These estimates and indexes can be prepared before runtime, enabling efficient query planning and accurate cost estimations. By leveraging one-sided noise mechanisms and private upper bound techniques, SPECIAL guarantees lossless processing for complex queries (e.g., multi-join). Our comprehensive benchmarks demonstrate that SPECIAL outperforms state-of-the-art SCAs, with up to 80× faster query times, 900× smaller memory usage for complex queries, and up to 89× reduced privacy loss in continual processing.

PVLDB Reference Format:
Chenghong Wang, Lina Qiu, Johes Bater, and Yukui Luo. SPECIAL: SynoPsis AssistEd Secure CollaboratIve AnaLytics. PVLDB, 18(4): 1035 - 1048, 2024.
doi:10.14778/3717755.3717764

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/lovingmage/caplan.

1 INTRODUCTION

Organizations, such as hospitals, frequently hold sensitive data in separate silos to comply with privacy laws, despite the valuable insights that could be gained from sharing this information. Recent advancement of Secure Collaborative Analytics (SCA) [7–9, 25, 33, 44, 54, 55, 57, 70, 71] provides an exciting solution to tackle this dilemma. These systems leverage advanced multi-party secure computation (MPC) [76] primitives to empower multiple data owners, who previously could not directly share data, to collaboratively process analytical queries over their combined data while ensuring the privacy of each individual's data.

While MPC can effectively conceal data values [76], its security guarantees do not immediately extend to the protection of execution transcripts. Consequently, data-dependent processing patterns such as memory traces and read/write volumes can still reveal critical information, risking privacy breaches [11, 15, 29, 38, 50, 60, 78] even when the core data remains encrypted. To ensure strong privacy, modern SCA systems utilize *data-oblivious* primitives that exhaustively pad query processing complexities to a worst-case and data-independent upper bound [7, 44, 54]. However, such stringent protections can largely reduce system efficiency and hinder the generalization of conventional optimization techniques to SCA, which are typically data-dependent [44]. To address this, recent efforts [8, 55, 70, 71] have introduced Differentially Private SCA (DPSCA). This approach allows controlled information leakage under DP [23] to mitigate constant worst-case overhead. For instance, systems under this model can dynamically compact an intermediate query size to a noisy estimate close to the actual size, avoiding exhaustive padding. As such, queries under DPSCA experience largely boosted efficiencies (e.g., up to 10^5× faster [71]) compared to their "no leakage" counterparts. Despite these substantial performance gains, existing DPSCAs still face critical limitations that impede their practical uses, as elaborated below:

• *L-1. Unbounded privacy loss.* Most DPSCA systems utilize a per-operator privacy expenditure model [8, 16, 19, 55, 70, 71], meaning each query operator (e.g., join, filter) independently consumes a portion of the privacy budget. This approach can lead to either unbounded privacy loss or the forced cessation of query responses upon budget exhaustion. To mitigate this, some studies [13, 56, 77] propose private, locality-sensitive grouping, incurring a one-time privacy cost to pre-group data based on specific attributes. Subsequent queries on those attributes can be directly applied to a smaller subset and need no additional privacy budget. However, this method only supports simple queries (e.g., point and range); complex queries like joins still suffer from unbounded privacy loss.

• *L-2. Unoptimized execution plan.* Conventional query planners can pre-estimate sizes for equivalent plans of a given query and select the most efficient plan with minimized intermediate sizes before execution [12, 61]. In contrast, SCA systems lack this capability, and even DPSCA designs [8, 9, 55, 69, 71, 77] can only reactively determine plan sizes during runtime. This inherent limitation often forces existing systems to settle for less efficient query plans, such as costly join orders, which lead to significantly inflated intermediate sizes (§ 7.2) and substantially hinder performance.

• *L-3. Lossy processing.* Noise from randomized mechanisms in DPSCA also introduces a unique accuracy issue (e.g., conventional DP mechanisms may generate negative noise, applying which to obfuscate the sizes of intermediate query results can cause losing

Proceedings of the VLDB Endowment, Vol. 18, No. 4 ISSN 2150-8097.
doi:10.14778/3717755.3717764

qualified real tuples), and unfortunately, no existing DPSCA can mitigate such loss for complex queries [28, 69–71, 77]. Furthermore, stronger DP settings can further increase noise variance, which amplifies errors, significantly impacting the utility of SCA systems.

1.1 Overview of SPECIAL

In this work, we introduce SPECIAL, an innovative SCA system that resolves the aforementioned limitations all at once through a new paradigm called *synopsis-assisted secure processing*. At its core, SPECIAL incurs a one-time privacy cost to gather DP synopses (statistics of base tables) from owners' data. These synopses are then used to accelerate complex query processing, and enhance SCA query planning. Notably, SPECIAL is the first system to provide all of the following benefits: (1) *Bounded privacy*—the privacy loss in SPECIAL is strictly limited to the one-time synopses release stage, with absotely no additional privacy cost during complex query processing and planning; (2) *Advanced query planning*—it builds an advanced SCA planner that can exploit plan sizes before runtime; and (3) *Lossless processing*—it ensures exact results with no data omissions. An overview of SPECIAL is shown in Figure 1.

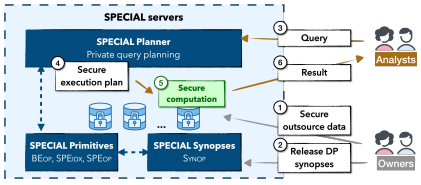

Figure 1: Overview of SPECIAL workflow.

SPECIAL operates under a standard server-aided MPC model [37] with three key participants: data owners, at least two SPECIAL servers, and a vetted analyst. The process begins with data owners securely outsourcing their data, typically through secret sharing [37, 70, 71, 77], and privately releasing corresponding DP synopses (§ 4) to the servers. SPECIAL introduces a set of novel primitives (§ 5) that can leverage these synopses to accelerate secure query operations. Once the data and synopses are in place, analysts can submit Select-Project-Join-Aggregation (SPJA) queries [61] for analytics. To process queries, a private planner (§ 6), running on SPECIAL servers, strategically orchestrates SPECIAL primitives to process the query and optimize performance. Finally, the results are securely returned to the analyst.

1.2 Unique challenges and key contributions

Leveraging DP synopses in SCA holds significant promise for achieving our desired objectives. However, this also introduces unique challenges. Below, we highlight the key challenges and summarize our non-trivial contributions to address them:

• *C-1. How to select proper synopses?* Even for a single relation, one can find numerous attribute combinations for generating synopses. Improper selection can lead to large errors (e.g. using too many synopses or high-dimensional attributes [79]), or reduced functionalities (e.g., using only simple attributes [77]). Hence, a key challenge is selecting a limited set of DP synopses to optimize the privacy budget for complex query processing. Our approach is informed by two observations: (i) secure joins are resource-intensive and need prioritized acceleration, and (ii) synopses for common filtering predicates are vital as they allow pre-built indexes on base relations for fast access. Consequently, we propose a focused strategy (§ 4) that targets low-dimensional (1D and 2D) attributes frequently involved in joins and filters within a representative workload.

• *C-2. How to enforce lossless processing?* Private synopses do not immediately implies lossless guarantees. Thus, a second challenge is designing practical approaches to achieve lossless results without violating privacy goals. To address this, we employ one-sided DP noise (either strictly positive or negative, § 4) in generating synopses, and design novel primitives (§ 5) based on them to pessimistically estimate filter cardinalities and intervals of index structures. To ensure lossless processing of complex joins, we extend upon cutting-edge join upper bound techniques [34] to privately estimate lossless join sizes using DP synopses. To our knowledge, this is the first study to support private join upper bound estimation.

• *C-3. How DP synopses can empower efficient query processing?* The use of DP synopses in SCA is largely underexplored, leaving a knowledge gap regarding their potential to enhance query efficiency. To navigate this potential, we explore various use of synopses in accelerating secure processing including private indexes SPEidx (§ 5.2), and compacted oblivious operations SPEop (§ 5.3). We also design a novel private query planner (§ 6) that efficiently orchestrate the execution of SPECIAL primitives (e.g., SPEidx, SPEop) to process SPJA queries. The planner uses available synopses to privately estimate intermediate result sizes and operation costs for a set of equivalent execution plans of a given query. It then executes the one with the lowest estimated cost.

• *C-4. How to systematically evaluate SPECIAL?* A major is the absence of open benchmarks. We address this by initiating an open-source evaluation set, accessible to the public. Specifically, we use public financial data [1] and design eight test queries, ranging from simple linear queries to complex 5-way joins. We also re-produce an open version of the HealthLNK benchmark. We evaluate our prototype, SPECIAL, against the state-of-the-art (SOTA) DPSCA system, Shrinkwrap [8], and the conventional SCA system, SMCQL [7]. Results indicate that SPECIAL outperforms Shrinkwrap, reducing query latency by up to 80.3×, and SMCQL, with at least a 114× reduction in query latency. Additionally, SPECIAL improves memory efficiency in complex join processing by more than 900× compared to both systems. Moreover, scaling experiments show that SPECIAL can effectively scale up to 8.8M rows dataset and up to 9-way complex joins. All benchmarks, including our prototype implementation, are open-sourced and available at [45].

2 BACKGROUND

General notations. We consider the logical database $\mathcal{D}$ to contain multiple private (base) relations $\{D_1, D_2, ...\}$, where each relation D_i is owned by a specific party P_i. A base relation D (we omit subscript for simplicity) has a set of attributes $attr(D)$. The domain of an attribute $A \in attr(D)$ is denoted by $dom(A)$, and the combined domain of a collection of attributes $\mathbf{A} = \{A_1, A_2, ...\} \subseteq attr(D)$ is denoted by $dom(\mathbf{A}) = \prod_{A \in \mathbf{A}} dom(A)$. For a tuple $t \in D$, and $\mathbf{A} \subseteq attr(D)$, we use $t.\mathbf{A}$ to denote the attribute value of $\mathbf{A}$ in t. A logical query, represented by $q(\mathcal{D})$, applies transformations and

computations on $\mathcal{D}$. In this work, we focus on SPJA [61] queries.

Frequency (count). Given D, $\mathbf{A} \subseteq attr(D)$, and a set of values $\mathbf{v} \in dom(\mathbf{A})$, the frequency (count) of $\mathbf{v}$ in D is the total number of tuples $t \in D$ with $t.\mathbf{A} = \mathbf{v}$. In addition, the max frequency moments (MF) of $\mathbf{A}$ is defined as $\mathtt{mf}(\mathbf{A}, D) = \max_{\mathbf{v} \in dom(\mathbf{A})} |\{t \in D \mid t.\mathbf{A} = \mathbf{v}\}|$.

Histograms. Given D, and $\mathbf{A} \subseteq attr(D)$, the (equal-width) histogram $\mathbf{h}(\mathbf{A}, D) = (c_1, c_2, ..., c_m)$ is a list of counts for the attribute values in $\mathbf{A}$. Specifically, $\mathbf{h}$ partitions $dom(\mathbf{A})$ into m "equal-sized" domain intervals $(B_1, ..., B_m)$, and a count $c_i \in \mathbf{h}$ is the number of tuples $t \in D$ with $t.\mathbf{A}$ in the interval of B_i.

Query planning. Modern databases parse queries into physical plans [61] that can be executed by the underlying query engine. These plans specify the operations like scans, joins, and sorts, and the order in which they're performed. The same query can have multiple equivalent execution plans, but their performance can vary greatly depending on resource usage and data access patterns. Query planning [12], done before runtime, involves selecting cost-efficient plans from these options. A key part of this process is accurately estimating intermediate result sizes, known as cardinality estimation (CE) [31], which relies heavily on table statistics. Two crucial statistics in modern cardinality estimation methods are: (i) *histograms,* which are vital for estimating selectivities in filters, and (ii) *max frequency,* which is crucial for estimating join sizes.

Multi-party secure computation (MPC). MPC [10, 26, 46, 76] is a cryptographic technique that allows multiple parties $P_1, P_2, ...$ to jointly compute a function $f(x_1, x_2, ...)$ over their own private input x_i. MPC ensures no unauthorized information is revealed to any party, except the desired output of f, emulating a computation as if performed by a trusted third party. Traditional MPC required all parties to actively participate in intensive computations. However, recent server-aided MPC [37, 48, 59] schemes allow offloading computations to powerful servers, without sacrificing security. In this model, parties secretly share their inputs with servers, which jointly evaluate an MPC protocol to reconstruct the secrets and compute the function.

Differential privacy [23]. DP ensures that modifying a single input tuple to a mechanism produces only a negligible change in its output. To elaborate, consider D and D' as two relations differing by just one tuple, then DP defines the following.

Definition 2.1 ((ϵ, δ)-DP). Given $\epsilon > 0$, and $\delta \in (0, 1)$. A randomized mechanism $\mathcal{M}$ is said to be (ϵ, δ)-DP if for all $D \sim D'$ pairs, and any possible output $o \subset Range(\mathcal{M})$, the following holds:

$$\Pr[\mathcal{M}(D) \in o] \leq e^{\epsilon} \Pr\left[\mathcal{M}(D') \in o\right] + \delta$$

Secret sharing and secure array. SPECIAL uses the 2-out-of-2 boolean secret share [4] over ring $\mathbb{Z}_{2^{32}}$ for securely outsourcing owners' data and storing query execution results. Specifically, each data, x, is divided into two shares: $x1, x2$ that are uniformly distributed over the ring $\mathbb{Z}_{2^{32}}$ such that $x = x1 \oplus x2$. Each server S_i receives one secret shares, s_i, where $i \in \{0, 1\}$. By retrieving shares from any two servers, an authorized party can successfully reconstruct the value of x. However, a single server alone learns nothing about x. For clarity and to abstract out the lower-level details, we leverage a logically unified data structure, namely the secure array [8, 71], denoted as $\langle \mathbf{x} \rangle = (\langle x_1 \rangle, \langle x_2 \rangle, ...)$, which is a collection of secret-shared relational tuples.

Oblivious (relational) operators. Oblivious operators are data-independent MPC protocols that implement the same functionalities as their plaintext database counterparts (e.g., filter and join). Data-independent execution requires that the control flow and memory access patterns of a function are indistinguishable given different inputs of the same size, and typically requires costly computation. For example, a linear scan is required to fulfill oblivious filtering [80], and join requires nested-loop over the two inputs [25]. The output sizes of such operators are usually padded with dummy tuples to the worst case: N rows for filters and N^2 for joins, given size N inputs. The dummy tuples will not affect the query result but can significantly impact the performance [25, 80]. To enhance efficiency while maintaining strong privacy, DPSCAs introduce a new type of oblivious operators [8, 55, 56, 71]. These operators typically involve two steps: *Compute* and *Compact*. The *Compute* step is fully oblivious, while the *Compact* resizes the output, often by obliviously sorting valid tuples to the front and trimming the output to a noisy DP size (the true size plus DP noise). While this approach can significantly reduce the computation cost and query sizes, it may lead to lossy query processing if the DP size is smaller than the true size (e.g., negative DP noise), as *valid tuples could be excluded during the compaction* [70, 71, 77]. We emphasize that when DP sizes exceed true sizes, there is no accuracy loss as it only includes extra dummy data that do not impact accuracy [25, 80].

Private indexes. In conventional databases, indexes are powerful data structures that map attribute values to positions in a sorted array, allowing a predicate selection to quickly access the desired data via index lookup without the need for full table scan. However, traditional indexes are unsuitable in SCAs due to their data-dependent nature, which can easily lead to privacy breaches. To address this, recent research has proposed DP indexes [57], where the mapping of attribute values to their positions is intentionally distorted with DP noise. To process queries, the system first pre-fetch a small set of data using DP indexes, followed by oblivious selection. This effectively avoids full table oblivious scan and sorting-based result compaction. However, the uncertainty inherent in DP indexes can lead to the loss of valid tuples. For example, in the DP index of [57], a true index range of positions $[10, 20]$ might be distorted into positions $[12, 17]$, causing data at position $10, 11, 18, 19$ and 20 to be missed. Nevertheless, *if DP indexes overestimate the range, subsequent oblivious selection can losslessly identify all valid tuples.*

3 SYSTEM AND PRIVACY MODEL

In general, SPECIAL follows a standard server-aided MPC [37] model, involving (i) a set of mutually distrustful data owners $P_1, ..., P_n$, (ii) two non-colluding servers S_0 and S_1, and (iii) a trusted analyst. We assume an *admissible adversary* [48] $\mathcal{A}$, capable of corrupting $n - 1$ out of n clients and at most one of the two servers. An instance of such adversary can be a malicious server that creates Sybil owners to form a malicious collation, attempting to steal sensitive information from an honest owner. Additionally, $\mathcal{A}$ is considered honest-but-curious, meaning it follows the protocol without deviation but may try to infer information from observed protocol transcripts, such as randomness, memory access patterns, and communication messages. The combination of these information is referred to as the view of $\mathcal{A}$. We also assume $\mathcal{A}$ is computationally

bounded as a probabilistic polynomial time (p.p.t.) adversary, which is a standard requirement in MPC protocols to ensure that adversaries cannot break cryptographic primitives. This threat model is consistent with prior SCA designs [7, 8, 48, 70, 71]. Given this setup, we design SPECIAL to satisfy the following:

Definition 3.1 (MPC protocol with DP leakage). Given a set of parties (owners) P_i with private data D_i and a secure query protocol Π that applies over $\mathcal{D} = \{D_1, D_2, \cdots\}$. We define a randomized mechanism $\mathsf{Lkg}(\mathcal{D}) = \{\mathsf{Lkg}(D_1), \mathsf{Lkg}(D_2), \cdots\}$ as the leakage profile, consisting of the control flow and access patterns of running Π over $\mathcal{D}$. The protocol Π is said to be secure with DP leakage if, for the subset of uncorrupted parties with data $\mathbf{D} \subseteq \mathcal{D}$, leakage profile $\mathsf{Lkg}(\mathbf{D}) \subseteq \mathsf{Lkg}(\mathcal{D})$, and any p.p.t. adversary $\mathcal{A}$:

- $\mathsf{Lkg}(\mathbf{D})$ *satisfies* (ϵ, δ)*-DP (definition 2.1).*
- *There exists a p.p.t. simulator* $\mathcal{S}$ *with only access to public parameters* pp *and* $\mathsf{Lkg}(\mathbf{D})$ *that satisfies:*

$$\begin{aligned} &\Pr\left[\mathcal{A}\left(\mathsf{VIEW}^{\Pi}(\mathcal{D}, \mathsf{pp}) = 1\right)\right] \\ &\leq \Pr\left[\mathcal{A}\left(\mathsf{VIEW}^{\mathcal{S}}(\mathsf{Lkg}(\mathbf{D}), \mathsf{pp})\right) = 1\right] + \mathsf{negl}(\kappa) \end{aligned} \tag{1}$$

where VIEW^{Π} *is* $\mathcal{A}$*'s view in* Π*'s execution and* $\mathsf{VIEW}^{\mathcal{S}}$ *is a simulated view produced by* $\mathcal{S}$ *using* Lkg*;* pp *denotes all public parameters, and* $\mathsf{negl}(\kappa)$ *is a negligible function related to a security parameter* κ*.*

Simply put, Definition 3.1 requires that the knowledge any p.p.t. adversary adversary can gain about each individual tuple of an honest owner, by observing the protocol execution, is bounded to what can be inferred from the outputs of the (ϵ, δ)-DP mechanism Lkg. We stress that this notion focuses on DP at the event (tuple) level without loss of generality. Due to the group-privacy properties of DP [22, 40, 66, 75], event-level DP can be extended to user-level DP. For instance, in a logical database $\mathcal{D}$ where any single user owns at most l tuples, if a protocol satisfies (ϵ, δ) event-level DP, it also satisfies $(l\epsilon, le^{(l-1)\epsilon}\delta)$ user-level DP. Moreover, we say that SPECIAL can be relaxed to employ a weaker corruption model, such as requiring a supermajority of owners and servers to remain uncorrupted, to enhance efficiency [44, 64]. This adjustment does not change the privacy guarantee outlined in Definition 3.1, but it does affect the security assumptions. Under the relaxed corruption model, Definition 3.1 is only satisfied when at least two-thirds of the parties remain uncorrupted. Due to space concerns, we defer the complete privacy proof of SPECIAL to our full version [18].

4 SPECIAL SYNOPSES

We now discuss the details of private synopses used in SPECIAL, while in later sections we will show how they accelerate query processing (§ 5) and aid in query planning (§ 6).

Challenges. We reiterate the main challenges in generating private synopses for a relation include: (C-1) selecting a set of attribute combinations that enable functional and efficient query processing; (C-2) ensuring that the subsequent query processing based on the private synopses is lossless.

Key ideas. Given a SPJA query, join operations typically need prioritized acceleration, as they are more resource-intensive than other operations. A k-way join can have $O(n^k)$ complexity without optimizations [7, 8, 25, 71, 80]. Additionally, synopses for frequently queried filter attributes play an important role in efficient query processing, as they enable fast indexing (§ 5.2) and effective filtering of unnecessary data before heavy joins. As such, our first key idea is to focus on histograms-based synopses that cover frequently queried join and filter attributes. To minimize noise, we focus on low-dimensional synopses: only 1D or 2D histograms.

To address the second challenge, our approach combines two strategies. First, to support accurate indexing and filtering, we use one-sided DP noise to generate DP histograms that consistently overestimate or underestimate attribute distributions. We will show later that such special histograms allow lossless filtering and indexing that reliably overestimate true filter sizes and indexing ranges (§ 5.2). Second, for lossless join output compaction, we incorporate noisy max frequency moments (MF) into the synopses. MF allows us to build on advanced join upper bound techniques [34] to privately estimate join sizes without data loss (§ 5.2).

4.1 Synopses generation

We now elaborate on the details of synopsis generation, which mainly contains two phases: (i) Attributes selection, where the SPECIAL servers select appropriate attributes for the generation, which are then distributed to owners; (ii) Local synopses release, where the owners create corresponding synopses using a DP mechanism and upload them to SPECIAL servers.

Attributes selection (servers). The first step is to identify a set of attributes for deriving synopses. In general, we consider the existence of a representative workload, Q_{R} [42], which can be sourced from a warm-up run or annotated by the administrator. Note that the representative workload does not involve any private data and thus is leakage-free. The servers first identify representative attribute pairs, $\mathsf{pair} = \{\mathsf{pair}_k\}_{k\geq 1}$, for each private relation $D \in \mathcal{D}$ via Q_{R}. The designated pairs include: (i) 2-way attribute pairs, which correspond to frequently queried *filter-join key* combinations; (ii) frequently queried individual attributes not covered by these pairs. By default, each $\mathsf{pair}_k = (A_{\mathsf{ft}}, A_{\mathsf{j}})$ contains two valid attributes (case i), but either A_{ft} or A_{j} may be empty (case ii).

Synopses release (owners). Next, servers pushes the identified pairs to owners, and subsequently, the owners independently dispatche private synopses and return them to servers. We now focus on the DP synopses generation mechanism run by each owner, Algorithm 1 illustrates the workflow.

In general, we expect owners to set a desired privacy budget for their data (using parameters ϵ and δ). Algorithm 1 produces synopses formalized as :

Definition 4.1 (SPECIAL synopses). Given Q_{R}*, we consider for each relation D, its corresponding synopsis* synop *is the collection of* $\{(\mathsf{pair}_k, \mathbf{H}(\mathsf{pair}_k, D), \mathtt{MF}_k)\}_{k\geq 1}$*, such that*

- $\mathsf{pair}_k = (A_{\mathsf{ft}}, A_{\mathsf{j}}) \in Q_{\mathsf{R}}$ *is a frequently queried attribute pair.*
- $\mathbf{H}(\mathsf{pair}_k, D) = \{\mathbf{h}^+, \mathbf{h}^-\}$ *is the DP bounding histogram for* pair_k*, where* $\mathbf{h}^+$ *(resp.* $\mathbf{h}^-$*) is a DP histogram that overestimate (resp. underestimate) the true histogram of* pair_k*.*
- $\mathtt{MF}_k$ *represents a collection of privately overestimated join key MFs categorized by* $\mathsf{pair}_k.A_{\mathsf{ft}}$*.*

We now present a detailed explanation of generating these synopsis structures, starting with the private bounding histograms (Alg 1,

Algorithm 1 DP synopsis gen $\mathcal{M}_{\text{synop}}$ (in the view of P)

Input: pair = $\{\text{pair}_k\}_{k\geq 1}$ from servers; private data D.
1: P self-determines privacy parameters ϵ, δ, and init synop $\leftarrow \emptyset$
2: **for** each pair_k **do**
3: $\mathbf{h}(\text{pair}_k, D) \leftarrow \text{HistGen}(\text{pair}_k, D)$
DP histograms:
4: $\mathbf{h}^+(\text{pair}_k, D) \leftarrow \mathbf{h} + \text{Lap}^+(\epsilon, \delta, \mathbf{h}.\text{shape})$
5: $\mathbf{h}^-(\text{pair}_k, D) \leftarrow \mathbf{h} + \text{Lap}^-(\epsilon, \delta, \mathbf{h}.\text{shape})$
▷ *adding independently sampled noise to every bin of* $\mathbf{h}^+$, $\mathbf{h}^-$
DP max frequencies:
6: **if** $A_j \in \text{pair}_k = \emptyset$ **or** A_j is unique valued **then** $\text{MF}_k = \emptyset$
7: **else if** $A_{ft} \in \text{pair}_k \neq \emptyset$ **then**
▷ *assuming* $\mathbf{h}$ *partitions* $dom(A_{ft})$ *into* $\{B_1, ..., B_m\}$
8: $D^\ell \leftarrow \sigma_{A_{ft} \in B_\ell}(D)$ **for** $\ell = 1, 2, ..., m$
9: compute noisy MF table, $\text{MF}_k = \{\widehat{\text{mf}}(A_j, D^\ell)\}_{1\leq \ell \leq m}$
10: **else** $\text{MF}_k = \widehat{\text{mf}}(A_j, D)$
11: synop $\leftarrow$ synop $\cup$ $(\text{pair}_k, \{\mathbf{h}^+, \mathbf{h}^-\}, \text{MF}_k)$
12: **release** synop, ϵ, δ to servers

lines 2:5). Specifically, for each pair_k, each owner first constructs a histogram $\mathbf{h}(\text{pair}_k, D)$. By default, we assume there exists global parameters (e.g., bin sizes) for each attribute, that ensure consistent data partitioning among all owners. Next, the owner derives two noisy histograms, $\mathbf{h}^+(\text{pair}_k, D)$ and $\mathbf{h}^-(\text{pair}_k, D)$, by adding independently sampled one-sided Laplace noise (Definition 4.2) to every bin of $\mathbf{h}(\text{pair}_k, D)$. This guarantees that $\mathbf{h}^+$ always overestimates the true histogram, while $\mathbf{h}^-$ consistently underestimates it.

Definition 4.2 (One-sided Laplace variable). $\text{Lap}^+(\epsilon, \delta) = \max(0, z)$ *(resp.* $\text{Lap}^-(\epsilon, \delta) = \min(0, z)$*) is a one-sided Laplace random variable in the range of* $[0, \infty)$ *(resp.* $(-\infty, 0]$*) if* z *is drawn from a distribution with the following density function*

$$\Pr[z = x] = \frac{e^\epsilon - 1}{e^\epsilon + 1} e^{-\epsilon|x-\mu|} \tag{2}$$

where $\mu = 1 - \frac{1}{\epsilon}\ln(\delta(e^\epsilon + 1))$ *(resp.* $\mu = \frac{1}{\epsilon}\ln(\delta(e^\epsilon + 1)) - 1$*).*

Next, we detail the generation of (noisy) join key MFs (Alg 1 lines 6:10). We assume that both A_{ft} and A_j are non-empty and that $\mathbf{h}(\text{pair}_k, D)$ partitions $dom(A_{ft})$ into bins $\{B_1, \ldots, B_m\}$. Each owner then generates a table of noisy MFs, $\{\widehat{\text{mf}}(A_j, D^\ell)\}_{1\leq \ell \leq m}$, where each entry $\widehat{\text{mf}}(A_j, D^\ell)$ represents an independently generated MF statistic for the join key attribute A_j, calculated over a specific subset of data filtered by the attribute A_{ft}, such that

$$\widehat{\text{mf}}(A_j, D^\ell) \leftarrow \widehat{\max}_\epsilon \left(\mathcal{G}_{\textbf{count}(A_j)}\left(\sigma_{A_{ft} \in B_\ell}(D)\right)\right) \tag{3}$$

here $\mathcal{G}_{\textbf{count}(A_j)}$ is a group-by-count operation over A_j, and $\widehat{\max}_\epsilon$ is a *report noisy max* mechanism [23]. It first adds i.i.d. noise from the exponential distribution $\text{Exp}(\frac{2}{\epsilon})$ to each grouped count, then outputs the largest noisy count. We stress that $\mathcal{M}_{\text{synop}}$ will not generate noisy MFs for non-join key attributes, and when A_{ft} is empty, a global MF will be generated instead of MF tables (Alg 1:10). Moreover, since SPECIAL enables owners to label attributes as unique-valued, if A_j is known to be unique-valued, then $\widehat{\text{mf}}(A_j, \cdot)$ is always 1. Nevertheless, as exponential noises are non-negative, thus $\widehat{\text{mf}} \geq \text{mf}$ holds for all cases.

Theorem 4.3. *Given* $|\text{pair}| = c$, $\epsilon, \delta > 0$, *the synopsis generation (Algorithm 1) is* $(\hat{\epsilon}, \hat{\delta})$*-DP where* $\hat{\epsilon} \leq 6\epsilon\sqrt{c\ln(1/\delta)}$*, and* $\hat{\delta} = (c+1)\delta$

For space concern, we move complete proofs to the full version [18]. In a sketch, adding Lap^+ (or Lap^-) to a single bin is (ϵ, δ)-DP. By parallel and sequential composition, generating $\mathbf{H}$ is $(2\epsilon, 2\delta)$-DP. Moreover, each noisy max is $(\epsilon, 0)$-DP, and by parallel composition, the generation of the entire MF table is also $(\epsilon, 0)$-DP. In this way, we know that the generation of each $(\text{pair}_k, \mathbf{H}, \text{MF}_k)$ is at most $(3\epsilon, 2\delta)$-DP. Given there are in total c such pairs, and thus the total privacy loss is subject to c-fold advanced composition [23].

Synopsis transformations. We say that one can perform transformations on released synopses without incurring extra privacy loss, per the post-processing theorem of DP [23]. Now, we outline the key synopsis transformation relevant to SPECIAL's design. First, given any (2d) bounding histogram $\mathbf{H}(\text{pair}, D)$ with both $A_{ft}, A_j \in \text{pair}$ are non-empty, one can derive the (1d) bounding histograms, i.e. $\mathbf{H}(A_j, D)$ and $\mathbf{H}(A_{ft}, D)$, for any single attribute A_{ft} or A_j by marginal sums $\mathbf{h}^+, \mathbf{h}^- \in \mathbf{H}(\text{pair}, D)$ over A_j or A_{ft}, respectively. This enables the creation of statistics on individual attributes, even when A_{ft} and A_j are not included as a standalone synopsis attribute. Moreover, it's possible to derive relevant join key statistics following a selection on the base relation. For example, given $A_{ft}, A_j \in \text{pair} \neq \emptyset$, and let $D' \leftarrow \sigma_{A_{ft} \in vals}(D)$, one can obtain the (1d) bounding histogram $\mathbf{H}(A_j, D')$ by conducting a selective marginal sum of $\mathbf{h}^+, \mathbf{h}^- \in \mathbf{H}(\text{pair}, D)$ over bins of A_{ft} that intersect with *vals*. Beyond bounding histograms, join key MFs over pre-filtered data can also be computed by

$$\widehat{\text{mf}}(A_j, D') = \min\left(\textstyle\sum_{B_\ell \cap vals \neq \emptyset} \widehat{\text{mf}}(A_j, D^\ell), \widehat{\text{mf}}(A_j, D)\right) \tag{4}$$

Note that $\widehat{\text{mf}}(A_j, D)$ exists if A_j is also included as a standalone synopsis attribute; otherwise, Eq 4 yields only the first term.

5 SPECIAL PRIMITIVES

We next introduce the secure primitives in SPECIAL. One major challenge in designing these primitives is the knowledge gap on how private synopses can accelerate oblivious query processing, i.e., C-3. To address this, we explore various usage of synopses, including creating private indexes (SPEIdx § 5.2) and designing compacted oblivious operations (SPEOp § 5.3). Given that joins are the most resource-intensive operations, we optimize join algorithms by combining private indexing and compaction techniques to develop a novel, parallel-friendly oblivious join (§ 5.3). Another challenge is ensuring lossless processing, which we tackle by integrating mechanisms that pessimistically estimate selection cardinalities, indexing ranges, and join sizes using synopses (§ 4) and advanced upper bound techniques [34]. For simplicity, we assume all input relations are of size n and all 1D histograms contain m bins.

5.1 Basic Operations

SPECIAL supports conventional fully-oblivious operators [7], which logically the same as non-private ones but with data-independent execution and worst-case padding for results. We briefly introduce these operations: (i) **Default data access (SeqACC).** By default, query execution begins with loading all data into a secure array via sequential scan. Each loaded tuple gets a secret bit ret (initially '0'),

marking its validity; (ii) **SELECT.** This secure filter $\sigma_p(R)$ performs a linear scan over the secure array $\langle R \rangle$, updating the ret bit to '1' for tuples satisfying predicate p and '0' for others; (iii) **PROJECT.** Removes irrelevant attributes from relation $\langle R \rangle$, but retains the ret bit; (iv) **JOIN.** Implements a secure θ-join $R_0 \bowtie_\theta R_1$ by computing the cartesian product $\langle R_0 \times R_1 \rangle$ and marking joined tuples with SELECT. The output is padded to the worst-case maximum size; (v) **COUNT, SUM, MIN/MAX.** These aggregation operators scan the secure array and update a secret-shared aggregation value for each tuple; (vi) **ORDER-BY, DISTINCT, GROUP-BY(AGG).** Built on the oblivious sort primitive [6]. ORDER-BY sorts the array by a given attribute. DISTINCT sorts and then identifies unique tuples, marking only the last in a sequence of identical tuples with ret = '1'. GROUP-BY(AGG) first uses DISTINCT to find unique tuples. For each distinct tuple (ret set to '1'), it appends an aggregation value derived from the tuple and a dummy attribute (e.g., '-1') for non-distinct tuples.

5.2 SPECIAL Index SPEIDX

Existing index techniques in SCA have several drawbacks: loss of qualified data [57], reliance on intricate data structures with large overhead [13, 56, 77], and restricted query support [13, 77]. Furthermore, all these techniques only support indexing the base relations. SPEIDX offers a breakthrough by enabling the creation of lossless indexes directly on outsourced data and the query of intermediate results, eliminating the need for extra structures or storing dummy data.

In general, SPEIDX builds upon the typical indexing model that utilizes cumulative frequencies (CF) [43]. Specifically, given D sorted by $A \in attr(D)$, all records $t \in D$ where $t.A = x$ can be indexed by the interval $[g(x-1), g(x)]$, where $g(x) = |\{t \mid t.A \leq x\}|$ is the CF function. For better illustration, we show an example index lookup in Figure 2: to get all records with an attribute value of 24, one may compute $[g(23), g(24)] = [217, 248]$ and access the relevant data from the subset $D[217:248]$. To make this indexing method private and lossless, the key idea of SPEIDX is to derive two noisy CF curves from synopses (i.e., bounding histograms). One curve, $g^+(x)$, consistently overestimates $g(x)$, while the other, $g^-(x)$, consistently underestimates it. Then for any attribute value x, we can now derive a private interval $[g^-(x-1), g^+(x)]$ that losslessly indexes all the desired records. As illustrated in Figure 2, the SPECIAL index might estimate the index range for attribute value 24 as $[g^-(23), g^+(24)] = [198, 267]$.

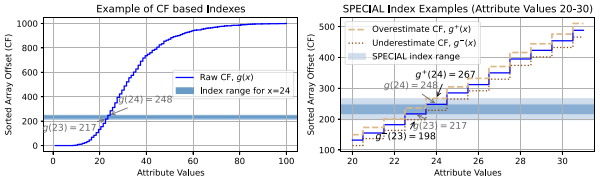

Figure 2: True (left) vs. SPECIAL (right) index for $x = 24$

In what follows, we provide the formal explanations on how SPEIDX derives indexes from DP synopses. Specifically, SPEIDX first determines the bounding histograms $\mathbf{H}(A, D)$, which may be either transformed from an available 2D histogram $\mathbf{H}(\text{pair}, D)$ with $A \in$ pair, or sourced directly if $\mathbf{H}(A, D)$ is already included in the synopses. It then constructs the noisy mapping as follows:

Definition 5.1 (SPECIAL index). Given D sorted by A, the bounding histogram $\mathbf{H}(A, D) = \{\mathbf{h}^+, \mathbf{h}^-\}$, and assume $\mathbf{h}^+ = (c_1^+, ..., c_m^+)$, $\mathbf{h}^- = (c_1^-, ..., c_m^-)$ partitions $dom(A)$ into $\{B_1, ..., B_m\}$. We say SPEIDX$(A, D) = \{\text{idx}_i = [\text{lo}_i, \text{hi}_i]\}_{1 \leq i \leq m}$ is the SPECIAL index of D over A with:

- $\forall\, i \geq 1$, $\text{hi}_i = \min(|D|, \sum_{k=1}^{i} c_k^+)$.
- $\text{lo}_1 = 0$, *and* $\forall\, i \geq 2$, $\text{lo}_i = \sum_{k=1}^{i-1} \max(0, c_k^-)$.

By this construction, all tuples $t \in D$ such that $t.A \in B_i$ will be organized into the subset $D[\text{idx}_i] \subseteq D$. This subset can be quickly accessed if D is already sorted, without the need for special data structures or inclusion of dummy tuples. Depending on how bounding histograms are constructed, SPEIDX(A, D) can support indexing lookups with varying granularity. This can range from indexing individual attribute values (where each B_i corresponds to a single domain value) to indexing a range of of values. The bounding histogram's pessimistic estimation ensures that all tuples where $t.A \in B_i$ are accurately contained within $D[\text{idx}_i]$, thereby achieving lossless indexing. In contrast to existing methods that are limited to indexing base relations [13, 57, 77], SPEIDX extends its capabilities to create private indexes on query intermediate results. For instance, consider $D' \leftarrow \sigma_{A^* \in vals}(D)$ where the attribute pair (A^*, A) is included in synop. Here, SPEIDX can derive $\mathbf{H}(A, D')$ from $\mathbf{H}(A^*, D)$ and subsequently build indexes on D'. Importantly, since index creation is a post-processing procedure using available DP synopses, it incurs no additional privacy loss.

Indexed store and fast data access IdxAcc. SPEIDX enables a new storage layout for outsourced data, namely indexed datastore. Specifically, by analyzing a representative workload Q_R, one may identify the "hottest" attribute per base relation, sort them according to the "hottest" attribute, and then build indexes over the sorted data. This storage layout enables fast indexed access (IdxAcc) to retrieve a compact subset of data from the outsourced relations, thereby eliminating the need for a full table sequential scan (SeqAcc) and can directly produce a compact input. We emphasize that the SPECIAL design does not require replicating the outsourced datastore to accommodate multiple query types [13, 77]. However, creating compact replicas (e.g., column replicas [35] over frequently queried attributes) can be optionally employed to enhance query processing speed. Moreover, the generation of all aforementioned objects (indexed store and column replicas) requires only three primitives: projection, oblivious sorting, and SPEIDX. In other words, this implies that one can selectively adjust these objects to align with dynamic query workloads, without incurring extra privacy loss.

5.3 SPECIAL Operators SPEOP

We introduce SPEOP, a set of novel synopsis-assisted operators that maintain full obliviousness, while enabling lossless compaction. To our knowledge, SPEOP is the first primitive of its kind in any SCA.

Oblivious compaction: OPAC. is a fundamental operation critical to other SPEOP primitives. Given input $\langle R \rangle$, OPAC sorts it based on the secret bit ret, moving tuples with ret = '1' to the front. Then, OPAC retains only the first k tuples from the sorted array. The compaction is *lossless* if k is greater than or equal to the number of tuples with ret = '1'; otherwise, it is *lossy*.

SPECIAL selections: (OP)SELECT, (SP)SELECT, (DC)SELECT. Let R to be a relation and $A \in attr(R)$, we now introduce three advanced selections that implements $\sigma_{A \in vals}(R)$.

(OP)SELECT. is mainly implemented based on the oblivious compaction (OPAC) operation. Specifically, the operation first conducts a standard SELECT on the input secure array $\langle R \rangle$ to label selected tuples, followed by an OPAC to to eliminate a large portion of non-matching tuples. To determine the compaction size cs, (OP)SELECT examines the synopsis of R and pessimistically estimates the cardinality of $\sigma_{A \in vals}(R)$ as shown in Algorithm 2. Since cs never underestimates the actual cardinality, and thus, the compaction is lossless with no missing tuples. Moreover, as OPAC is fully oblivious and cs is determined completely from post-processing over DP synopsis, thus, (OP)SELECT causes no privacy loss.

Algorithm 2 CardEst($\sigma_{A \in vals}(R)$, synop)

1: $\mathbf{h} = \emptyset, c = 0$
2: **if** $\mathbf{H}(A, R) \in$ synop **then** $\mathbf{h} \leftarrow \mathbf{h}^+ \in \mathbf{H}(A, R)$
3: **else if** $\exists$ pair $\in$ synop, s.t. $A \in$ pair **then**
4: $\quad \mathbf{h} \leftarrow$ marginal sum $\mathbf{h}^+ \in \mathbf{H}(A, R)$ over (pair $\setminus A$).
5: **else return** $cs = |R|$
6: **return** $cs = \min(|R|, \sum_{i=1}^{m} c_i \in \mathbf{h} : (B_m \cap vals \neq \emptyset))$

(SP)SELECT. The running complexity of (OP)SELECT depends on OPAC, which is typically linearithmic (see § 6.1 or [58]). However, when CardEst($\sigma_{A \in vals}(R)$, synop) is relatively small, oblivious selection can be achieved without necessarily incurring linearithmic cost. Specifically, we consider (SP)SELECT, which first creates an empty output array $\langle R_o \rangle$ with size equals to cs before any computations. Next, it evaluates two linear scans over $\langle R \rangle$, where the first scan obliviously marks all selected tuples, and in the second scan, it privately writes all marked tuples into $\langle R_o \rangle$. Specifically, in the second scan, (SP)SELECT internally maintains the last *actual write* position idx in $\langle R_o \rangle$. Then for every newly accessed tuple $\langle t \rangle$ in $\langle R \rangle$, a write action occurs on all tuples in $\langle R_o \rangle$. If $\langle t \rangle$ is selected, then an *actual write* is made that writes $\langle t \rangle$ to $\langle R_o[\text{idx} + 1] \rangle$ and a *dummy write* is made to elsewhere. If not, dummy writes are made throughout $\langle R_o \rangle$. We say that, in the context of the secret-shared secure array $\langle \mathbf{a} \rangle$, a dummy write to $\langle \mathbf{a}[i] \rangle$ is simply a re-sharing of $\mathbf{a}[i]$ through secure protocols without changing its value.

(DC)SELECT. Finally, if the underlying data is already indexable on A, a direct pre-fetch can be applied to avoid full table scan and compaction. The operator simply looks up SPEIDX(A, R), and accesses $R[a, b]$, where $a = \min_i(\text{idx}_i.\text{lo}), b = \max_i(\text{idx}_i.\text{hi})$, and idx_i dentoes the index in SPEIDX(A, R) with bin $B_i \cap [a, b] \neq \emptyset$. A standard SELECT is then applied to $R[a, b]$.

SPECIAL join: (MX)JOIN. We now introduce a novel MF-Index based oblivious join operation. The advancements of (MX)JOIN stand out in three aspects. First, compared to the standard JOIN, (MX)JOIN stands out for its ability to significantly compact the output size, coupled with a highly parallelizable fast processing mode. Second, existing DP oblivious joins typically require spending privacy budget [21] to learn join sensitivity [21] or necessitate truncation on joined tuples [8, 71]. (MX)JOIN eliminates this need. Moreover, (MX)JOIN is unique as the first oblivious join that enables lossless output compaction without extra privacy loss. We illustrate the construction details in Algorithm 3.

Algorithm 3 (MX)JOIN (base and pre-filtered relations)

Input: relations R_0, R_1; join attribute A_j; we consider synopses (histograms) of A_j are partitioned into bins $B_1, ...B_m$.
1: **if** MXReady(R_0, R_1) == True **then** BucketJoin(R_0, R_1, A_j)
2: **else if** R_0, R_1 are either base or pre-filtered relation **then**
3: $\quad$ **for** $b \in \{0, 1\}$ **do**
4: $\quad\quad$ derive $\widehat{\text{mf}}(A_\text{j}, R_b)$ from synop_b (§ 4)
5: $\quad\quad$ build index SPEIDX(A_j, R_b) = $\{\text{idx}_i\}_{i=1,..,m}$ (§ 5.2)
6: $\quad$ **if** $\forall b$, $\widehat{\text{mf}}(A_\text{j}, R_b)$, and SPEIDX($A_\text{j}, R_b$) $\neq$ null **then**
7: $\quad\quad$ oblivious sort R_0, R_1 on A_j, BucketJoin(R_0, R_1, A_j)
8: $\quad$ **else assert** "not applicable for (MX)JOIN"

BucketJoin(R_0, R_1, A_j):
9: **for** $i = 1, 2, ..., m$ **do**
10: $\quad R_{0,1}^{(i)} \leftarrow \sigma_{A_j \in B_i}(R_{0,1})$ using (DC)SELECT/ SPEIDX($A_\text{j}, R_{0,1}$)
11: $\quad$ compute $O_i \leftarrow (R_0^{(i)} \bowtie_{A_\text{j}} R_1^{(i)})$ via standard JOIN
12: $\quad cs_i \leftarrow \min\left(\frac{|R_0^{(i)}|}{\widehat{\text{mf}}(A_\text{j}, R_0)}, \frac{|R_1^{(i)}|}{\widehat{\text{mf}}(A_\text{j}, R_1)}\right) \times \widehat{\text{mf}}(A_\text{j}, R_0) \cdot \widehat{\text{mf}}(A_\text{j}, R_1)$
13: $\quad R_\text{out} \leftarrow R_\text{out} \cup \text{OPAC}(O_i, cs_i)$
14: **return** R_out

In general, (MX)JOIN can be applied to two types of data: the base and pre-filtered relations where the join key attribute is included in synop. Specifically, (MX)JOIN starts with computing the join key MFs (Alg 3:4) and constructing private indexes (Alg 3:5) for both inputs. All these operations are conducted through "privacy cost-free" transformations using available DP synopses. Once these objects are obtained, the algorithm employs oblivious sort to rearrange both inputs (Alg 3:6,7), rendering them indexable with tuples logically distributed into independent buckets by join key values. Next, (MX)JOIN simply adopts standard JOIN to join tuples exclusively within the same buckets (Alg 3:10). Finally, (MX)JOIN performs per-bucket output compaction, where it first determines the *MF join bound* [34] for each bucket join and invokes OPAC to compact the output according to the learned size (Alg 3:11,12). As bucket-wise operations are independent, the aforementioned steps lend themselves well to parallelized processing. As (MX)JOIN derives join compaction sizes completely from post-processing of DP synopses, it thus incurs no extra privacy loss. Additionally, the noisy MF bounds guarantee that compaction sizes are consistently overestimated, ensuring lossless compaction of join results.

6 SPECIAL PLANNER

Current DPSCA designs struggle with costly execution plans because they cannot pre-estimate query intermediate sizes, and thus unable to identify effective execution plans with minimized cost. SPECIAL overcomes this challenge by introducing a novel query planner that uses synopses for size estimation, and thus enabling both private and efficient SCA query planning.

At a high level, our planner is modeled after the Selinger-style optimizer [12]. It uses a bottom-up, dynamic programming approach to enumerate all equivalent plans for a given query, estimates their costs (heavily influenced by intermediate sizes) using available

Table 1: Asymptotic costs for secure operators

Operator	Input I/O (C_{in})	Eval. (C_{eval})	Output I/O (C_{out})
PROJECT	$O(n)$	N/A	$O(n)$
Agg.	$O(n)$	$O(n)$	$O(1)$
Group & Order	$O(n)$	$O(n \log^2 n)$	$O(n)$
SELECT	$O(n)$	$O(n)$	$O(n)$
(OP)SELECT	$O(n)$	$O(n \log n)$	hist_bound
(SP)SELECT	$O(n)$	$O(n)$	$O(1)$
(DC)SELECT	idx_bound	N/A	N/A
JOIN	$O(n)$	$O(n^2)$	$O(n^2)$
(MX)JOIN	$O(n)$	$O(n^2)$*	mf_bound

*Assuming the max size of the indexed buckets is bounded by $O(\frac{n}{\log n})$.

synopses, and selects the plan with minimal cost. The introduction of SPECIAL primitives significantly impacts cost modeling for oblivious operations, rendering existing models [8, 44] inadequate. Furthermore, the design-space challenge of query planning persists, and the extensive equivalent plan search space necessitates strategies to simplify the search process. To address these, we first systematically analyze the complexities of SPECIAL primitives and develop a new cost model (§ 6.1). We then design protocol-specific heuristics (§ 6.2) tailored to our planner to narrow the search space.

6.1 Cost Model

We adopt the standard SCA cost framework [8] to develop SPECIAL's cost model, viewing the cost of a secure execution plan as the sum of each operator's I/O and secure evaluation costs. Specifically, given a plan with ℓ operators, $\text{op}_1, ..., \text{op}_\ell$, and let $\mathbf{I} = \{I_1, ..., I_\ell\}$, $\mathbf{O} = \{O_1, ..., O_\ell\}$, to be the input and output sizes of each operators. The plan cost is:

$$\text{Cost} = \sum_{i=1}^{\ell} C_{\text{in}}^{\text{op}_i}(I_i) + C_{\text{eval}}^{\text{op}_i}(I_i) + C_{\text{out}}^{\text{op}_i}(O_i) \tag{5}$$

Here, C_{in} represents the data access cost (input I/O), primarily capturing the expenses when moving data from persistent storage to an in-memory secure array. C_{out} denotes the output I/O cost, modeling the expenses when writing operator results into output arrays. C_{eval} accounts for the secure computing cost for evaluating an operator, typically constituting the dominant cost. Note that, in practice, the exact formulas for $C_{\text{in}}, C_{\text{out}}$, and C_{eval} can vary depending on the specific secure protocol employed (garbled circuits [76], secret sharing [46], etc.) as well as the particular hardware configurations in use. Nonetheless, the understanding of the asymptotic costs is adequate for comprehending the principles of SCA query planning and optimization strategies [8, 44]. In what follows, we provide detailed analysis on the asymptotic costs for each SPECIAL operator. Similarly, we assume that all input data sizes mentioned henceforth in this section are of size n, and all 1D histograms have m bins. Table 1 summarizes the operator costs.

Oblivious sorting and compaction. While oblivious sorting algorithms with optimal $O(n \log n)$ complexity exist, they often necessitate either impractically large constants [3, 27] or client-side memory [5], both do not fit with SCA scenario. Consequently, we will consider the well-established bitonic sorting based implementation for oblivious sort, which come with $O(n \log^2 n)$ complexity. Nonetheless, efficient OPAC implementations with $O(n \log n)$ complexity remain achievable [58].

Projection, grouping and aggregation. The PROJECT accesses private relations and discards unnecessary columns independently on each server, which is naturally oblivious. Thus, I/O costs dominate this operation, with both input and output costs bounded by $O(n)$. The costs of ORDER-BY, DISTINCT, and GROUP-BY are dominated by oblivious sorting, resulting in a complexity of $O(n \log^2 n)$. Additionally, as these operators do not reduce output sizes, both C_{in} and C_{out} are bounded by $O(n)$. Finally, the cost of aggregations, i.e. COUNT, SUM, and MIN/MAX subjects to a oblivious linear scan, typically outputting a single secret-shared value. Hence, its C_{eval} is bounded by $O(n)$, with C_{in} at $O(n)$ and C_{out} at $O(1)$.

Selections. The primary cost of SELECT stems from an oblivious linear scan, making C_{eval} within $O(n)$. Since SELECT does not shrink the output size, both C_{in} and C_{out} are within $O(n)$. (OP)SELECT requires an oblivious compaction (OPAC) before writing outputs, where OPAC usually yields an $O(n \log n)$ complexity [58]. Consequently, its C_{eval} is bounded by $O(n \log n)$ with input I/O cost same as SELECT. However, as (OP)SELECT compacts output size, C_{out} is reduced to hist_bound $= O\left(\sum_{B_i \cap vals \neq \emptyset} |B_i|\right)$, where $B_i \cap vals \neq \emptyset$ are bins in the synopsis histogram intersecting with selection conditions. If $\sum_{B_i \cap vals \neq \emptyset} |B_i| \sim O(1)$, (SP)SELECT becomes preferable, with its running cost dominated by a two-phase linear scan, and thus C_{eval} is now $O(n)$, and the output cost is $O(1)$. (DC)SELECT is the most efficient selection, though it requires indexable input data. All costs are directly related to the size of the indexed data, so C_{in}, C_{evals} and C_{out} are all bounded by idx_bound $= O\left(\max_i(\text{idx}_i.\text{hi}) - \min_i(\text{idx}_i.\text{lo})\right)$. Here, idx_i are indexed regions that intersect with selection conditions.

Joins. Both JOIN and (MX)JOIN have $O(n)$ data access costs, but differ in C_{eval} and C_{out}. JOIN, conducting a Cartesian product for two input tables, has C_{eval} and C_{out} both bounded by $O(n^2)$. Compared to JOIN, in the worst-case scenario where the join keys follow a highly biased distribution, i.e. max bucket size reaches $O(n)$, (MX)JOIN's asymptotic cost is at most $O(n^2 \log n)$. However, when join keys are distributed more uniformly, the cost can be asymptotically better. For instance, with $m = \log n$ and assuming a max bucket size of $O(\frac{n}{\log n})$, each bucket join costs $O(\frac{n^2}{\log n})$, leading to a total cost of $O(n^2)$, equivalent to JOIN. Recall that bucket joins in (MX)JOIN can be executed concurrently, hence, the processing latency is indeed dominated by the bucket-wise cost, i.e. $O(\frac{n^2}{\log n})$. Additionally, the output cost is lowered from $O(n^2)$ to the sum of per-bucket MF upper bounds (Alg 3:11), which can be substantially less if the join key MFs are low.

6.2 Heuristics

H-1. Filter push down. is a common query planning optimization, moves selection operations to the earliest possible stage to reduce data processed by subsequent operations. In conventional SCAs, data obliviousness often requires padding selection sizes, making filter pushdown ineffective [25, 44, 54, 80]. However, SPECIAL's innovative selection methods enable compacting selections to approximate true cardinalities without compromising privacy, restoring the effectiveness of filter pushdown. Hence, we include filter push down as one of the optimization heuristic for SPEPLAN.

H-2. Predicates fusion. Let R to be any relation, $A_1, A_2, ..., A_k \subseteq attr(R)$, and $\mathbf{v} = \{v_1, v_2, ..., v_k\}$. We say that for multiple selection over R such that $\sigma_{A_1 \in v_1}(...\sigma_{A_k \in v_k}(R))$, one can always fuse them into one selection $\sigma_{\mathbf{A} \subset \mathbf{v}}(R)$. This can reduce the number of secure

computation invocations from k rounds to just one. Additionally, the selection size can be estimated as $\min_{A_i}\left(\texttt{CardEst}(\sigma_{A_i \in v_i}(R))\right)$.

H-3. Join statistics propagation. A key property of SPECIAL join, `(MX)JOIN`, is that the output is already indexed and bucketized by the join key. Therefore, for any output R of `(MX)JOIN` computing $R_0 \bowtie_{A_j} R_1$, a new index $\text{SPEIDX}(A_j, R)$ across R can be easily derived. Moreover, as per [34], one can also update the MF for R by computing $\widehat{\texttt{mf}}(A_j, R) = \widehat{\texttt{mf}}(A_j, R_0) \times \widehat{\texttt{mf}}(A_j, R_1)$. As a result, we say that the output of `(MX)JOIN` as *MF-and-index-ready*, enabling direct application of another `(MX)JOIN` on the same join attribute.

7 EVALUATION

In this section, we present evaluation results of our proposed framework. Specifically, we address the following questions: **Question-1**: Does SPECIAL offer efficiency and privacy advantages over existing SCAs? **Question-2**: For SPECIAL design, is there a trade-off between privacy guarantees and system efficiency? How different synopses scenarios affect system performance? **Question-3**: Can SPECIAL scale complex analytical (e.g. multi-way join) queries to large-scale (multi-million rows) datasets?

7.1 Experimental setups

Baseline systems and SPECIAL prototype. We compare SPECIAL with two baseline systems: Shrinkwrap [8], the SOTA DPSCA, and SMCQL [7] (also used as a baseline for Shrinkwrap). For consistency, we consider the same circuit-model implementations for both baseline systems and the SPECIAL prototype. While some works [44, 54] similar to SMCQL use exhaustive padding but improve efficiency through protocol-level optimizations, we exclude them from our benchmarks for fair comparison concerns. We re-implemented key query features for the two baseline systems and built SPECIAL using the same MPC package (`EMPtoolkit-0.2.5`). All implementations are open-sourced [45].

Datasets and workloads. We developed two open benchmarks. The first reproduces the HealthLNK benchmark used by Shrinkwrap and SMCQL which simulates a real-world scenario where medical researchers want to perform secure analytics across multiple cohorts' sensitive data. We use an open schema [62] to generate a scalable synthetic dataset with 4 tables, 222K rows and 24 columns. The benchmark involves four multi-cohort medical study queries, identical to those used in Shrinkwrap and SMCQL. Our second benchmark simulates secure collaborative analytics within the financial sector. Imagine multiple banks needing to analyze their combined, private data to study loan and financial statistics—all without compromising sensitive customer information. We use the anonymized *Czech Financial Dataset* [1] for this, assuming each entry represents data owned by a different bank or financial organization unable to directly share information. This dataset comprises 8 relational tables with a total of 55 columns and 1.1 million rows. For testing workloads, we design eight query workloads, ranging from simple linear queries to complex multi-way join-aggregation queries. A brief summary of the workloads is provided in Table 2.

Default configurations. For SPECIAL, we employ a per-table privacy budget allocation strategy. Each table is allocated a one-time privacy budget of $\epsilon = 1.5$ and $\delta = 0.00005$ for synopses generation, and is evenly distributed across all DP synopses. For Shrinkwrap, we adopt their default per-query privacy allocation, assigning a privacy budget of $\epsilon = 1.5$ and $\delta = 0.00005$ to each query, as outlined in [8]. It is important to note that this configuration means Shrinkwrap will not offer guarantees on the bounded privacy loss across multiple queries. For the HealthLNK benchmark, we use *Dosage* and *Aspirin* as representative workloads, and for the Financial benchmark, we use *FQ2, FQ4, and FQ8*. Unless further elaborated, these workloads will also serve as default testing queries for our evaluation. For all equal-width histograms generated in SPECIAL, we configure them to have at most 8 bins. Moreover, for baseline systems, as they do not have join ordering optimizations, thus we will assume a random join order for them. We conduct all experiments on bare-metal Mac machines with M2 Max CPUs and 96GB unified memory.

Table 2: Query workloads

Bench.	Query	Type	Description
HealthLNK	**Dosage Study**	Binary Join	Expanding binary join.
	Comorbidity	Binary Join	Non-expanding binary join.
	Aspirin count	Multi Join	3 way mixed join
	3 Join Aspirin	Multi Join	4 way mixed join
Financial	**FQ1**	Linear	Point query.
	FQ2	Linear	Range query.
	FQ3	Binary Join	Non-expanding binary join.
	FQ4	Binary Join	Expanding binary join.
	FQ5	Multi Join	3-way mixed joins.
	FQ6	Multi Join	3-way all expanding joins.
	FQ7	Multi Join	4-way mixed joins.
	FQ8	Multi Join	5-way mixed joins.

7.2 End-to-end comparisons

To address **Question-1**, we first conduct an end-to-end performance comparison of SPECIAL, Shrinkwrap, and SMCQL across all benchmark workloads. The results are summarized in Figure 4, 5. We cannot complete full benchmark for SMCQL due to high memory cost, so we project evaluations for FQ4 (using 10% data) and omit results for other complex workloads.

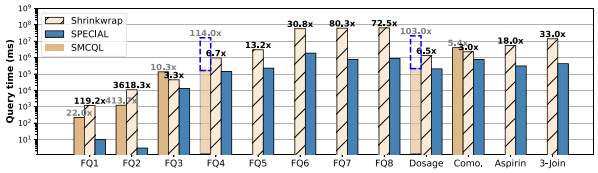

Figure 4: End-to-end comparison: query latency

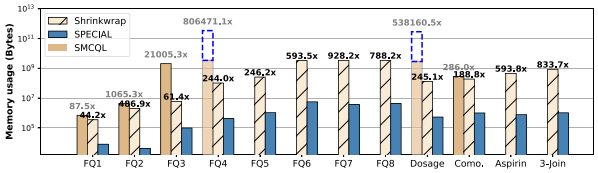

Figure 5: End-to-end comparison: memory usage

Observation 1. SPECIAL outperforms Shrinkwrap and SMCQL in query latency across all benchmarks, reaching up to 3618.3× **for linear queries,** 114× **for binary joins, and** 80.3× **for multi-joins.** Figure 4 shows the comparison results in query

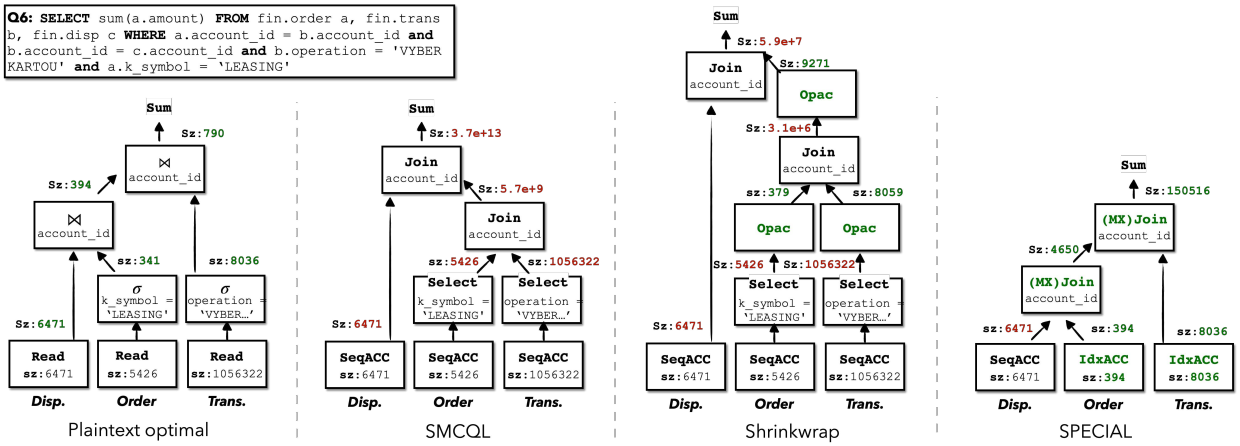

Figure 3: In-depth comparisons in execution plans: (i) The exhaustive padding in SMCQL can lead to significant memory blowup; (ii) Both SMCQL and Shrinkwrap suffer from unoptimized join ordering; (iii) Although Shrinkwrap reduces intermediate sizes, it still requires substantial memory to materialize join outputs; (iv) SPECIAL can identify efficient join execution orders to reduce intermediate sizes; (v) `IdxAcc` can significantly reduce input I/O costs.

time. First, SPECIAL shows significant speedups for linear queries, reaching up to 3618.3× (FQ2). This large performance gain is mainly attributed to its index-based fast data access. By directly fetching private data through DP indexes, SPECIAL eliminate substantial I/O costs (e.g., sequential reads) and bypass the need for secure computations (e.g., oblivious filter). Second, we observe that in binary joins, SPECIAL has a less pronounced advantage over Shrinkwrap. This is because binary joins have a single join order, eliminating the potential for join ordering problems (where different join orders lead to significantly different performance). Consequently, even though Shrinkwrap does not optimize join orders, it doesn't experience efficiency losses in this scenario. However, for more complex multi-way joins, SPECIAL's advantage becomes more pronounced again. For instance, more than 80× speedup in FQ7. This is because SPECIAL can pre-select efficient join orders before runtime.

Observation 2. SPECIAL shows profound improvement in memory usage (Figure 5) against baseline systems, especially in complex multi-way joins. This is primarily due to two factors: First, the `(MX)JOIN` used by SPECIAL is more memory-efficient compared to the joins implemented by Shrinkwrap and SMCQL. Second, SPECIAL's capability to identify optimal execution plans significantly reduces total intermediate sizes, which is particularly beneficial for complex joins that suffer from sub-optimal or exhaustive padding in other systems. To better understand the substantial improvements SPECIAL achieves—for instance, up to 928.2× over Shrinkwrap and more than 10^5× over SMCQL—we will zoom into a specific query, FQ6, and compare the detailed execution plans of the three systems. The choice of FQ6 is strategic because its complexity sufficiently highlights the differences in execution plans, yet it remains simple enough for clear visual representation. Our comparison features four execution plans: a plaintext optimal plan, illustrating the ground truth optimal execution; a hypothetical SMCQL plan (with projected cardinalities); and two actual execution plans from our experiments with Shrinkwrap and SPECIAL. The detailed comparisons and observations are summarized in Figure 3.

To continue address **Question-1**, we now compare the privacy guarantees of SPECIAL with the baseline systems (Figure 6). Specifically, we focus on comparing cumulative privacy loss in multiple query answering, w.r.t. two composition models: advanced composition (Adv.)[23] and concentrated composition (CDP.)[14].

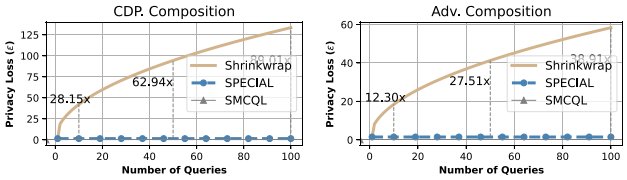

Figure 6: End-to-end comparison privacy loss.

Observation 3. Under continual query answering, SPECIAL demonstrates significantly lower privacy loss compared to Shrinkwrap, achieving up to 89.01× **and** 38.91× **improvements in the Adv. and CDP modes, respectively.** The privacy loss of SPECIAL is bounded to the initial synopsis release stage, so continual query answering does not incur additional privacy loss. In contrast, Shrinkwrap's privacy loss accumulates over time as each new query allocates a fresh privacy budget. Consequently, its privacy loss exhibits a logarithmic growth, as shown in Figure 6. This accumulation can result in significant privacy degradation when processing a large number of queries. For example, answering 100 queries in Shrinkwrap could result in a privacy loss of $\epsilon > 100$ in Adv. and $\epsilon \approx 60$ in CDP., respecitvely, even if each query only uses a small privacy budget of $\epsilon = 1.5$. As such, SPECIAL demonstrate significant improve in privacy guarantees towards SOTA DPSCA. Even when compared to standard SCA (e.g., SMCQL) with no privacy loss due to exhaustive padding, our system incurs only a small and fixed privacy cost (e.g. $\epsilon = 1.5$ per table) while delivering substantial performance gains.

7.3 Privacy efficiency tradeoffs

We address **Question-2** by evaluating SPECIAL at various privacy levels. Specifically, we maintain δ constant while varying ϵ from 0.1 to 10 and assess the performance across default testing queries. The results are shown in Figure 7.

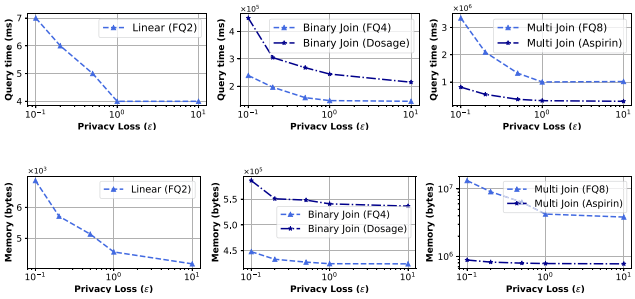

Figure 7: Privacy vs. Performance trade-offs

Observation 4. The privacy-efficiency tradeoff generally exists but exhibits varying trends at different privacy levels. For instance, SPECIAL shows a clear tradeoff at higher privacy levels ($\epsilon < 1$), while at lower privacy levels ($\epsilon > 1$), the tradeoff becomes less pronounced. When ϵ increases from 0.1 to 1, both memory usage and query latency for all test queries significantly decrease. However, increasing ϵ from 1 to 10 shows no significant performance gains. This may indicate that once ϵ exceeds 1, the impact of noises on cardinality estimation or index building is alredy minimal, and further reductions in ϵ do not lead to notable improvements. Therefore, if high privacy protection is required, practitioners should carefully fine-tune privacy parameters to optimize performance. Conversely, if performance is the priority, setting ϵ near 1 is typically sufficient.

7.4 Synopses impacts micro benchmarks

We continue to address **Question-2** to explore how synopses scenarios may affect SPECIAL's performance. Specifically, we study two key settings: (i) How different bin numbers (BinNum) in synopses can impact the efficiency of `(IDX)JOIN`, and (ii) how synopsis coverage levels for a single query can affect its overall execution. We will conduct micro-benchmarks for a thorough investigation. Note that simulate different synopses scenario on both HealthLNK and Financial benchmarks can be challenging (e.g., joins typically occur on the same key, so it is hard to simulate partial coverage), hence, to better control experimental variables and accurately assess impacts, we will now use synthetic data and workloads.

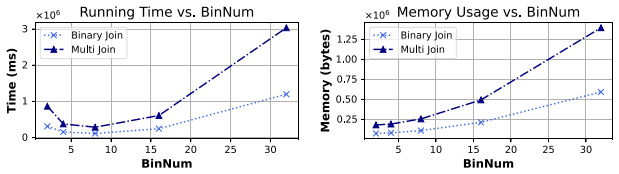

Figure 8: BinNum experiments.

Table 3: Synopses coverage experiments.

	Synopses Coverage	Time (ms)	Improv.	Mem. (bytes)	Improv.
	No coveragee	1831703	baseline	129048576	baseline
Join Key (JK)	**2 way (1 JK)**	276340	6.6×	28481424	4.5×
	3 way (2 JKs)	29200	62.7×	905808	142.5×
	All way (3 JKs)	6097	300.4×	65088	1982.7×
Filter*	**1 input**	444463	4.1×	40327680	3.2×
	2 inputs	108205	16.9×	12602400	10.2×
	3 inputs	24486	74.8×	3943200	32.72×
	All inputs	6847	267.5×	1303200	99×

* We synthesize a random selectivity between (0, 0.33) for each filter operation.

We first study how the BinNums impact the performance of `(IDX)JOIN`. To study this, we synthesize two join queries on fixed input data, generate join key synopses with varying bin numbers (2 to 64), and measure the performance of `(IDX)JOIN` in processing the queries. The results are shown in Figure 8.

Observation 6. The running time of `(IDX)JOIN` initially decreases but then increases as the BinNum grows. The memory usage consistently increases. `(IDX)JOIN` partitions larger joins into smaller sub-joins, but since we rely on DP indexes for partitioning, each sub-join inevitably includes additional dummy data. This increased plan size directly translates to higher memory usage and will grow when BinNum increases (more noises in DP indexes). On the other hand, partitioning large joins into smaller ones can enhance join efficiency, which is why we initially observe a decrease in execution time as BinNums increase (e.g., 2 to 8). However, the trade-off arises when the number of bins becomes excessive (e.g., > 8). The overhead of handling the increased dummy data starts to outweigh the benefits gained from partitioning. At this point, the performance improvement plateaus and starts to degrade as the system struggles with the inflated plan sizes.

We now study how synopses coverage impacts query performance. We synthesize and test a 4-way join query under two controlled scenarios: (i) *JK coverage:* We focus on varying the level of JK coverage, starting from 1 out of 3 JKs to full coverage, while ensuring no filter synopses are present. We then measure how this impacts query performance; and (ii) *Filter coverage*: We maintain full JK coverage and change the coverage of filter synopses on the query's input tables, ranging from 1 out of 4 inputs to full coverage. This allows us to examine the isolated effect of filter synopsis coverage on performance. Results are in Table 3.

Observation 7. For both groups, query efficiency significantly improves as synopses coverage grows. Nevertheless, even at the lowest coverage level, queries can still achieve notable speedups. Even with minimal synopsis coverage—like boosting only one join or applying synopses to just one input table—we observe significant speedups of 6.6× and 4.1×, respectively. This demonstrates the potential for substantial performance gains even with limited synopsis availability. Moreover, real-world workloads often involve joins on the same keys and similar filtered inputs (e.g., HealthLNK workloads), suggesting that high synopsis coverage is achievable even with a small set of representative workloads. As demonstrated by Table 3, adding even a single additional synopsis, whether for join keys or table filters, can yield a substantial performance boost (up to 10×) in query execution efficiency.

Table 4: Scaling query complexity experiments

Join scale	Query time (ms)	Improv.	Memory (bytes)	Improv.
Shrinkwrap Q6	58342782	baseline	3354165360	baseline
7 (FQ8 ⋈ FQ3)	1330602	43.8×	5334360	628×
8 (FQ8 ⋈ FQ5)	11789800	4.9×	7901016	424×
9 (FQ8 ⋈ FQ3 ⋈ FQ4)	13306150	4.4×	8999352	328×

7.5 Scaling experiments

To address **Question-3**, we stress SPECIAL with two types of scaled workloads: (i) *scaled data:* we duplicate the raw dataset to sizes of 2×, 4×, and 8× and evaluate default testing queries; *(ii) scaled query complexity:* We use standard inputs, but simulate complex multi joins (up to 9-way) by chaining together multiple join workloads. The results are shown in Figure 9 and Table 4.

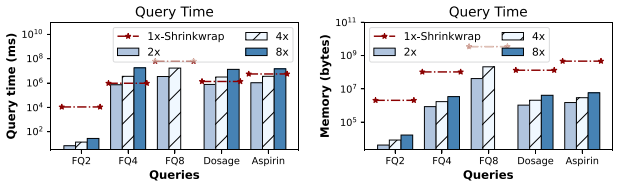

Figure 9: Scaling data experiments.

Observation 8. SPECIAL shows large potential to scale up to multi-million data, even for complex 5-way joins. Figure 9 shows SPECIAL's effective scaling: up to 8× data for linear queries and binary joins, and up to 4× data for complex 5-way joins like Q8. For instance, Q2 can be completed within 290ms under 8×, and in fact, since selection is bypassed (due to index access), thus the cost is mainly on I/O costs. Q4 finishes in 289 minutes at the same scale 8×, while the more complex 5-way join Q8 takes less than 280 minutes for 4× data. As a reference, Shrinkwrap would require over 1035 minutes to complete Q8 even with unscaled data.

Observation 9. SPECIAL can effectively process very large joins (e.g., 9-ways). Table 4 shows that the query processing time of SPECIAL at 9-way scale can still be 4.4× faster than Shrinkwrap at 3-way join scale (Q6), and the memory improvement is even more evident that is 328× smaller. We stress that these significant memory savings can become even more crucial when processing massive datasets. Techniques like Shrinkwrap or SMCQL, might be forced to rely on much slower persistent storage to handle intermediate results that exceed memory capacity. In contrast, SPECIAL can still maintain a fully in-memory query mode, potentially leading to even more pronounced efficiency gains in such scenarios.

8 RELATED WORK

SCA systems. Two main approaches exist for designing MPC-based SCA systems. The first is *peer-to-peer (P2P) paradigm* [2, 49, 54, 67, 74], where the goal is to improve efficiency by decomposing analytical queries and pushing them to data owners, so that they can either directly process in clear or running MPCs across a small group of parties. Unfortunately, this approach imposes large overhead on data owners, especially for complex operations like joins. Given that real-world data owners often lack robust computing resources and service capabilities, the P2P paradigm is hard to scale and support reliable SCA services to external analysts. The other paradigm is the *server-aided-MPC model* [8, 9, 37, 41, 44, 48, 57, 64, 70, 71, 77]. This model allows data owners to outsource both expensive MPC computations and secure data storage to a set of capable servers, which can then jointly evaluate MPC to provide reliable SCA services. SPECIAL is built upon the server-aided-MPC model and under a strong "all but one"corruption assumption. Moreover, SPECIAL's core design is protocol-agnostic, which allows interoperability with various MPC models, including the P2P or a weaker corruption where a supermajority of servers need to be honest [41, 44, 64].

DP leakages. Leakage-abuse attacks [11, 15, 29, 38, 50, 60, 78], exploit data-dependent processing patterns, are persistent threats to SCA systems. To mitigate these risks, oblivious computation [5, 17, 20, 36, 39, 47, 52, 53, 58, 63, 65, 72, 73] have become the *de facto* solution. While this technique ensures the strongest privacy guarantees by eliminating any data-dependent leakages, it also introduces a fundamental contention with modern database optimizations, which often rely heavily on data-dependent operations [8, 69–71]. To this end, many recent efforts seek a practical balance in the privacy-performance trade-off by allowing controlled leakage under DP [8, 16, 19, 28, 32, 51, 55, 56, 68–71, 77]. However, a common issue of these approaches is unbounded privacy loss. While some works propose to address this [13, 56, 77], their approaches are restricted to only simple linear queries. SPECIAL addresses all these limitations together, and to our knowledge, is the first SCA system that can simultaneously ensure both bounded privacy and lossless results for complex SPJA queries.

SCA query planning. Query planning [61] is crucial in conventional databases. Conventional planners can exploit size disparities across different query plans to choose efficient ones with smaller sizes [24, 30, 31]. However, such techniques use data-dependent information and are typically prohibited in SCA. A handful of studies [7, 44, 54, 67] that explore query planning within SCA frameworks primarily rely on data-independent metrics for planning, which usually lead to only moderate optimizations. Shrinkwrap [8] introduced a private planning method that optimally compacts intermediate sizes by efficiently allocating privacy budgets to minimize dummy data. However, it cannot pre-determine an optimal join order. SPECIAL offers an advanced query planner capable of pre-estimating intermediate sizes and comparing execution costs among different plan structures before runtime.

9 CONCLUSION

We introduce SPECIAL, the first SCA system that simultaneously supports: (i) handling complex queries with bounded privacy loss; (ii) advanced query planning that effectively exploit plan intermediate sizes before runtime; and (iii) delivering exact query results without missing tuples. This is achieved through a novel synopses-assisted SCA design, where a set of private table statistics are released with one-time privacy cost to guide subsequent secure SCA planning and processing.

ACKNOWLEDGMENTS

This work was supported by the NSF grant OAC-2419821, IUIAS Collaborative Award, and IU Faculty Startup Grant.

REFERENCES

[1] [n.d.]. Financial Dataset. https://relational-data.org/dataset/Financial. Accessed: 2024-03-30.

[2] Gagan Aggarwal, Mayank Bawa, Prasanna Ganesan, Hector Garcia-Molina, Krishnaram Kenthapadi, Rajeev Motwani, Utkarsh Srivastava, Dilys Thomas, and Ying Xu. 2005. Two Can Keep A Secret: A Distributed Architecture for Secure Database Services.. In *CIDR*, Vol. 2005. 186–199.

[3] Miklós Ajtai, János Komlós, and Endre Szemerédi. 1983. An 0 (n log n) sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*. 1–9.

[4] Toshinori Araki, Jun Furukawa, Yehuda Lindell, Ariel Nof, and Kazuma Ohara. 2016. High-throughput semi-honest secure three-party computation with an honest majority. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 805–817.

[5] Gilad Asharov, TH Hubert Chan, Kartik Nayak, Rafael Pass, Ling Ren, and Elaine Shi. 2020. Bucket oblivious sort: An extremely simple oblivious sort. In *Symposium on Simplicity in Algorithms*. SIAM, 8–14.

[6] Kenneth E Batcher. 1968. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. 307–314.

[7] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel N Kho, and Jennie Rogers. 2017. SMCQL: Secure Query Processing for Private Data Networks. *Proc. VLDB Endow.* 10, 6 (2017), 673–684.

[8] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: efficient sql query processing in differentially private data federations. *Proceedings of the VLDB Endowment* 12, 3 (2018).

[9] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. 2020. Saqe: practical privacy-preserving approximate query processing for data federations. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2691–2705.

[10] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 2019. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *Providing sound foundations for cryptography: on the work of Shafi Goldwasser and Silvio Micali*. 351–371.

[11] Laura Blackstone, Seny Kamara, and Tarik Moataz. 2019. Revisiting leakage abuse attacks. *Cryptology ePrint Archive* (2019).

[12] Mike W Blasgen, Morton M Astrahan, Donald D Chamberlin, JN Gray, WF King, Bruce G Lindsay, Raymond A Lorie, James W Mehl, Thomas G Price, Gianfranco R Putzolu, et al. 1981. System R: An architectural overview. *IBM systems journal* 20, 1 (1981), 41–62.

[13] Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2021. Epsolute: E iciently erying Databases While Providing Differential Privacy. (2021).

[14] Mark Bun and Thomas Steinke. 2016. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*. Springer, 635–658.

[15] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 668–679.

[16] T-H Hubert Chan, Kai-Min Chung, Bruce Maggs, and Elaine Shi. 2022. Foundations of differentially oblivious algorithms. *ACM Journal of the ACM (JACM)* 69, 4 (2022), 1–49.

[17] Zhao Chang, Dong Xie, Sheng Wang, and Feifei Li. 2022. Towards Practical Oblivious Join. In *Proceedings of the 2022 International Conference on Management of Data*. 803–817.

[18] Johes Bater Yukui Luo Chenghong Wang, Lina Qiu. 2024. SPECIAL: Synopsis Assisted Secure Collaborative Analytics. https://arxiv.org/abs/2404.18388.

[19] Shumo Chu, Danyang Zhuo, Elaine Shi, and TH Hubert Chan. 2021. Differentially oblivious database joins: Overcoming the worst-case curse of fully oblivious algorithms. *Cryptology ePrint Archive* (2021).

[20] Natacha Crooks, Matthew Burke, Ethan Cecchetti, Sitar Harel, Rachit Agarwal, and Lorenzo Alvisi. 2018. Obladi: Oblivious serializable transactions in the cloud. In *13th USENIX Symposium on Operating Systems Design and Implementation (OSDI 18)*. 727–743.

[21] Wei Dong, Juanru Fang, Ke Yi, Yuchao Tao, and Ashwin Machanavajjhala. 2022. R2t: Instance-optimal truncation for differentially private query evaluation with foreign keys. In *Proceedings of the 2022 International Conference on Management of Data*. 759–772.

[22] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N Rothblum. 2010. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*. 715–724.

[23] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407.

[24] R Elmasri, SB Navathe, R Elmasri, and SB Navathe. 2020. Fundamentals of Database Systems</Title. In *Advances in Databases and Information Systems: 24th European Conference, ADBIS 2020, Lyon, France, August 25–27, 2020, Proceedings*, Vol. 12245. Springer Nature, 139.

[25] Saba Eskandarian and Matei Zaharia. 2017. Oblidb: Oblivious query processing for secure databases. *arXiv preprint arXiv:1710.00458* (2017).

[26] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.

[27] Michael T Goodrich. 2014. Zig-zag sort: A simple deterministic data-oblivious sorting algorithm running in o (n log n) time. In *Proceedings of the forty-sixth annual ACM symposium on Theory of computing*. 684–693.

[28] Adam Groce, Peter Rindal, and Mike Rosulek. 2019. Cheaper private set intersection via differentially private leakage. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019).

[29] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2018. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 315–331.

[30] Yuxing Han, Ziniu Wu, Peizhi Wu, Rong Zhu, Jingyi Yang, Liang Wei Tan, Kai Zeng, Gao Cong, Yanzhao Qin, Andreas Pfadler, et al. 2021. Cardinality estimation in DBMS: A comprehensive benchmark evaluation. *arXiv preprint arXiv:2109.05877* (2021).

[31] Hazar Harmouch and Felix Naumann. 2017. Cardinality estimation: An experimental survey. *Proceedings of the VLDB Endowment* 11, 4 (2017), 499–512.

[32] Xi He, Ashwin Machanavajjhala, Cheryl Flynn, and Divesh Srivastava. 2017. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1389–1406.

[33] Zhian He, Wai Kit Wong, Ben Kao, David Wai Lok Cheung, Rongbin Li, Siu Ming Yiu, and Eric Lo. 2015. Sdb: A secure query processing system with data interoperability. *Proceedings of the VLDB Endowment* 8, 12 (2015), 1876–1879.

[34] Axel Hertzschuch, Claudio Hartmann, Dirk Habich, and Wolfgang Lehner. 2021. Simplicity Done Right for Join Ordering.. In *CIDR*.

[35] Dongxu Huang, Qi Liu, Qiu Cui, Zhuhe Fang, Xiaoyu Ma, Fei Xu, Li Shen, Liu Tang, Yuxing Zhou, Menglong Huang, et al. 2020. TiDB: a Raft-based HTAP database. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3072–3084.

[36] Kristjän Valur Jönsson, Gunnar Kreitz, and Misbah Uddin. 2011. Secure multiparty sorting and applications. *Cryptology ePrint Archive* (2011).

[37] Seny Kamara, Payman Mohassel, and Mariana Raykova. 2011. Outsourcing multi-party computation. *Cryptology ePrint Archive* (2011).

[38] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'neill. 2016. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1329–1340.

[39] Marcel Keller and Peter Scholl. 2014. Efficient, oblivious data structures for MPC. In *Advances in Cryptology–ASIACRYPT 2014: 20th International Conference on the Theory and Application of Cryptology and Information Security, Kaoshiung, Taiwan, ROC, December 7-11, 2014, Proceedings, Part II 20*. Springer, 506–525.

[40] Daniel Kifer and Ashwin Machanavajjhala. 2011. No free lunch in data privacy. In *Proceedings of the 2011 ACM SIGMOD International Conference on Management of data*. 193–204.

[41] Brian Knott, Shobha Venkataraman, Awni Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. Crypten: Secure multi-party computation meets machine learning. *Advances in Neural Information Processing Systems* 34 (2021), 4961–4973.

[42] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. Privatesql: a differentially private sql query engine. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1371–1384.

[43] Tim Kraska, Alex Beutel, Ed H Chi, Jeffrey Dean, and Neoklis Polyzotis. 2018. The case for learned index structures. In *Proceedings of the 2018 international conference on management of data*. 489–504.

[44] John Liagouris, Vasiliki Kalavri, Muhammad Faisal, and Mayank Varia. 2023. {SECRECY}: Secure collaborative analytics in untrusted clouds. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1031–1056.

[45] Lovingmage. 2024. Synopsis Assisted Secure Collaborative Analytics. https://github.com/lovingmage/SPECIAL/.

[46] Silvio Micali, Oded Goldreich, and Avi Wigderson. 1987. How to play any mental game. In *Proceedings of the Nineteenth ACM Symp. on Theory of Computing, STOC*. ACM New York, NY, USA, 218–229.

[47] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. 2018. Oblix: An efficient oblivious search index. In *2018 IEEE symposium on security and privacy (SP)*. IEEE, 279–296.

[48] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE symposium on security and privacy (SP)*. IEEE, 19–38.

[49] Dimitris Mouris, Daniel Masny, Ni Trieu, Shubho Sengupta, Prasad Buddhavarapu, and Benjamin Case. 2024. Delegated Private Matching for Compute. *Proceedings on Privacy Enhancing Technologies* (2024).

[50] Simon Oya and Florian Kerschbaum. 2021. Hiding the access pattern is not enough: Exploiting search pattern leakage in searchable encryption. In *30th USENIX security symposium (USENIX Security 21)*. 127–142.

[51] Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. 2019. Mitigating leakage in secure cloud-hosted data structures: Volume-hiding for multi-maps via hashing. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*. 79–93.

[52] Martin Pettai and Peeter Laud. 2015. Combining differential privacy and secure multiparty computation. In *Proceedings of the 31st annual computer security applications conference*. 421–430.

[53] Benny Pinkas and Tzachy Reinman. 2010. Oblivious RAM revisited. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*. Springer, 502–519.

[54] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M Hellerstein. 2021. Senate: a {Maliciously-Secure}{MPC} platform for collaborative analytics. In *30th USENIX Security Symposium (USENIX Security 21)*. 2129–2146.

[55] Lianke Qin, Rajesh Jayaram, Elaine Shi, Zhao Song, Danyang Zhuo, and Shumo Chu. 2022. Adore: Differentially oblivious relational database operators. *arXiv preprint arXiv:2212.05176* (2022).

[56] Lina Qiu, Georgios Kellaris, Nikos Mamoulis, Kobbi Nissim, and George Kollios. 2023. Doquet: Differentially Oblivious Range and Join Queries with Private Data Structures. *Proceedings of the VLDB Endowment* 16, 13 (2023), 4160–4173.

[57] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. 2020. Crypte: Crypto-assisted differential privacy on untrusted servers. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 603–619.

[58] Sajin Sasy, Aaron Johnson, and Ian Goldberg. 2022. Fast Fully Oblivious Compaction and Shuffling. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2565–2579.

[59] Peter Scholl, Nigel P Smart, and Tim Wood. 2017. When it's all just too much: outsourcing MPC-preprocessing. In *Cryptography and Coding: 16th IMA International Conference, IMACC 2017, Oxford, UK, December 12-14, 2017, Proceedings 16*. Springer, 77–99.

[60] Zhiwei Shang, Simon Oya, Andreas Peter, and Florian Kerschbaum. 2021. Obfuscated access and search patterns in searchable encryption. *arXiv preprint arXiv:2102.09651* (2021).

[61] Abraham Silberschatz, Henry F Korth, and Shashank Sudarshan. 2011. Database system concepts. (2011).

[62] SMCQL. [n.d.]. SMCQL: Secure Multi-party Computation Query Language. https://github.com/smcql/smcql/tree/master/conf/workload. Accessed: 2024-08-12.

[63] Emil Stefanov, Elaine Shi, and Dawn Song. 2011. Towards practical oblivious RAM. *arXiv preprint arXiv:1106.3652* (2011).

[64] Sijun Tan, Brian Knott, Yuan Tian, and David J Wu. 2021. CryptGPU: Fast privacy-preserving machine learning on the GPU. In *2021 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1021–1038.

[65] Afonso Tinoco, Sixiang Gao, and Elaine Shi. 2023. {EnigMap}:{External-Memory} Oblivious Map for Secure Enclaves. In *32nd USENIX Security Symposium (USENIX Security 23)*. 4033–4050.

[66] Salil Vadhan. 2017. The complexity of differential privacy. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich* (2017), 347–450.

[67] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Mayank Varia, Andrei Lapets, and Azer Bestavros. 2019. Conclave: secure multi-party computation on big data. In *Proceedings of the Fourteenth EuroSys Conference 2019*. 1–18.

[68] Sameer Wagh, Paul Cuff, and Prateek Mittal. 2016. Differentially private oblivious ram. *arXiv preprint arXiv:1601.03378* (2016).

[69] Sameer Wagh, Xi He, Ashwin Machanavajjhala, and Prateek Mittal. 2021. DP-cryptography: marrying differential privacy and cryptography in emerging applications. *Commun. ACM* 64, 2 (2021), 84–93.

[70] Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2021. DP-Sync: Hiding update patterns in secure outsourced databases with differential privacy. In *Proceedings of the 2021 International Conference on Management of Data*. 1892–1905.

[71] Chenghong Wang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2022. IncShrink: Architecting Efficient Outsourced Databases using Incremental MPC and Differential Privacy. *arXiv preprint arXiv:2203.05084* (2022).

[72] Chenghong Wang, David Pujo, Kartik Nayak, and Ashwin Machanavajjhala. 2023. Private Proof-of-Stake Blockchains using Differentially-private Stake Distortion. *Cryptology ePrint Archive* (2023).

[73] Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. 2014. Oblivious data structures. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*. 215–226.

[74] Yilei Wang and Ke Yi. 2021. Secure yannakakis: Join-aggregate queries over private data. In *Proceedings of the 2021 International Conference on Management of Data*. 1969–1981.

[75] Yonghui Xiao and Li Xiong. 2015. Protecting locations with differential privacy under temporal correlations. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1298–1309.

[76] Andrew Chi-Chih Yao. 1986. How to generate and exchange secrets. In *27th annual symposium on foundations of computer science (Sfcs 1986)*. IEEE, 162–167.

[77] Yanping Zhang, Johes Bater, Kartik Nayak, and Ashwin Machanavajjhala. 2023. Longshot: Indexing Growing Databases Using MPC and Differential Privacy. *Proceedings of the VLDB Endowment* 16, 8 (2023), 2005–2018.

[78] Yupeng Zhang, Jonathan Katz, and Charalampos Papamanthou. 2016. All your queries are belong to us: the power of {File-Injection} attacks on searchable encryption. In *25th USENIX Security Symposium (USENIX Security 16)*. 707–720.

[79] Zhikun Zhang, Tianhao Wang, Ninghui Li, Jean Honorio, Michael Backes, Shibo He, Jiming Chen, and Yang Zhang. 2021. {PrivSyn}: Differentially Private Data Synthesis. In *30th USENIX Security Symposium (USENIX Security 21)*. 929–946.

[80] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An oblivious and encrypted distributed analytics platform. In *14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17)*. 283–298.