

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) enable the processing of analytical SQL queries across multiple owners' data, even when direct data sharing is not feasible. Although essential for strong privacy, the large overhead from data-oblivious primitives in traditional SCA has hindered its practical adoption. Recent SCA variants that permit controlled leakages under differential privacy (DP) show a better balance between privacy and efficiency. However, they still face significant challenges, such as potentially unbounded privacy loss, suboptimal query planning, 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 spent to acquire private synopses (table statistics) from owner data. These synopses then allow SPECIAL to estimate (compaction) sizes for secure operations (e.g., filter, join) and index encrypted data without extra privacy loss. Crucially, these estimates and indexes can be prepared before runtime, thereby facilitating efficient query planning and accurate cost estimations. Moreover, by using one-sided noise mechanisms and private upper bound techniques, SPECIAL ensures strict lossless processing for complex queries (e.g., multi-join). Through a comprehensive benchmark, we show that SPECIAL significantly outperforms cutting-edge SCAs, with up to 80× faster query times and over 900× smaller memory for complex queries. Moreover, it also achieves up to an 89× reduction in privacy loss under continual processing.

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) [8–10, 26, 35, 46, 57, 58, 61, 72, 73] provides an exciting solution to tackle this dilemma. These systems leverage advanced multi-party secure computation (MPC) [78] 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 [78], 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 [12, 16, 31, 40, 53, 64, 80] even when the core data remains encrypted. To ensure strong privacy, modern SCA systems also utilize *data-oblivious* primitives that exhaustively pad query processing complexities to a worst-case, data-independent upper bound [8, 46, 57]. However, such stringent protections can largely reduce system efficiency and hinder the generalization of conventional optimization techniques to SCA,

which are typically data-dependent [46]. To address this, recent efforts [9, 58, 72, 73] have introduced differentially private SCA (DP-SCA). This approach allows controlled information leakage under differential privacy [24] to mitigate constant worst-case overhead. For instance, systems under this model can dynamically compact intermediate query sizes to a noisy estimate close to actual sizes, avoiding exhaustive padding. As such, queries under DPSCA experience largely boosted efficiencies (e.g., up to $10^5\times$ faster [73]) 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 [9, 17, 20, 58, 72, 73], 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 [14, 59, 79] 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. Suboptimal 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 [13, 65]. In contrast, SCA systems lack this capability, and even DPSCA designs [9, 10, 58, 71, 73, 79] can only reactively determine plan sizes during runtime. This inherent limitation often forces existing systems to settle for less efficient query plans, such as suboptimal 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., losing tuples after compaction), and unfortunately, no existing design can provide deterministic accuracy guarantees for complex queries [30, 71–73, 79]. Furthermore, stronger privacy 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 designed to address all these challenges simultaneously 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 compact query intermediate results, index encrypted data, and enhance SCA query planning. Notably, SPECIAL is the first system to provide all of the following benefits: (1) *Bounded privacy*—it manages complex queries, such as multi joins, within strict

privacy limits; (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’s architecture is shown in Figure 1.

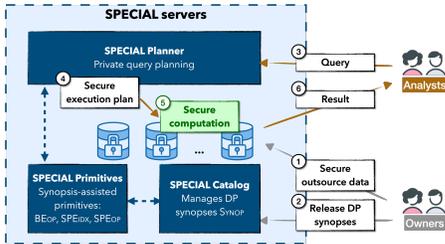


Figure 1: Overview of SPECIAL workflow.

SPECIAL operates under a standard server-aided MPC model [39] with three key participants: data owners, at least two SPECIAL servers, and a vetted analyst. The process starts with data owners securely outsourcing their data (e.g. using secret sharing [39, 72, 73, 79]), and privately release a set of DP synopses (table statistics) to the servers. Once the data and synopses are ready, analysts can then submit queries on the outsourced data. To process queries, SPECIAL uses synopses to privately estimate tight result compaction bounds or build private indexes for faster access, thus avoiding exhaustive padding. Crucially, this information is exploit before runtime, so that SPECIAL can pre-identify an optimal secure execution plan that minimizes intermediate sizes and overall costs. Finally, the optimized plan is executed via MPC across the secret-shared data, and the results are securely returned to the analyst.

In the SPECIAL prototype, we adopt the same privacy model used in existing DPSCA designs, assuming a computationally bounded, “honest-but-curious” adversary [27] who can compromise up to $n-1$ out of n owners and $m-1$ out of m servers. SPECIAL ensures that such an adversary can only learn a limited leakage profile L_{kg} about honest parties’ data, which strictly adheres to DP constraints. Note that, SPECIAL’s core components, including synopses management and query planning do not leverage raw data at all. Thus, any of the servers, alone or jointly, can manage these tasks. Also, once analysts are authorized by data owners, they are considered trusted, and additional randomization of query results is not required. It is worth mentioning that SPECIAL’s corruption model (e.g. all except one) is entirely determined by the underlying MPC, and SPECIAL’s design does not modify them at all. Hence, one may also opt to a weaker corruption model, such as a supermajority of owners and servers must remain uncorrupted [46, 67], to enhance efficiency.

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 a single relation can have numerous options for generating synopses, such as using different attributes or their combinations. However, improper selection can lead to large errors (e.g. using too many synopses or leverage high-dimensional attributes [81]), or reduced functionalities (e.g., using only simple attributes [79]). 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) for generating synopses and design novel primitives (§ 5) based on them to pessimistically estimate filter cardinalities and intervals of index structures. This enable us to efficiently discard unnecessary records during processing without breaking privacy and lossless guarantees. To ensure lossless processing of complex joins, we extend upon cutting-edge join upper bound techniques [36] to privately estimate join compaction sizes using available 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 then develop a Selinger-style query planner (§ 6) that supports both standard secure operators and SPEOP. Employing a customized cost model and tailored heuristics, this planner efficiently generates optimal execution plans for queries using a mix of standard and SPEOP operators. This in essence, also address the limitations of sub-optimal execution plans inherent in standard DPSCA.

- *C-4. How to systematically evaluate SPECIAL?* A major challenge in SCA design evaluation is the absence of open-source benchmarks. We address this by initiating an open-source evaluation platform accessible to the public, utilizing public Financial data [1] (1.1M rows and 59 columns), and designing eight test queries ranging from simple linear to complex 5-way joins. We evaluated our prototype SPECIAL alongside the state-of-the-art DPSCA system, Shrinkwrap [9], and conventional SCA, SMCQL [8]. 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 linear and binary joins up to 8× (8.8M rows) and 5-way joins up to 4× (4.4M rows) workloads. All benchmarks, including our prototype implementation, are open-sourced and available at [47].

The remainder of the paper is structured as follows: We begin with essential background in § 2 and a formal definition of our privacy model in § 3 before delving into design details. § 4 discusses the SPECIAL catalog design, including synopses selection and generation. § 5 details novel synopses-assisted secure database operators, and § 6 describes our planner design. Benchmark evaluations are presented in § 7, followed related works and conclusions.

2 BACKGROUND

General notations. We consider the logical database \mathcal{D} to contain multiple private relations $\{D_1, D_2, \dots\}$, where each relation D_i is belongs owned by a specific party P_i , where $D \in \mathcal{D}$ is owned by a specific owner and has a set of attributes as $\text{attr}(D)$. The domain for attribute $A \in \text{attr}(D)$ is given by $\text{dom}(A)$, and the combined domain for a collection of attributes $\mathbf{A} = \{A_1, A_2, \dots\} \subseteq \text{attr}(D)$ is denoted as $\text{dom}(\mathbf{A}) = \prod_{A \in \mathbf{A}} \text{dom}(A)$. For a tuple $t \in D$, and $\mathbf{A} \subseteq \text{attr}(D)$, we denote $t.\mathbf{A}$ as the attribute value of \mathbf{A} in t .

Frequency (count). Given D , and $\mathbf{A} \subseteq \text{attr}(D)$, and $\mathbf{v} \in \text{dom}(\mathbf{A})$, 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 $\text{mf}(\mathbf{A}, D) = \max_{\mathbf{v} \in \text{dom}(\mathbf{A})} |\{t \in D \mid t.\mathbf{A} = \mathbf{v}\}|$.

Histograms. Given D , and $\mathbf{A} \subseteq \text{attr}(D)$, the (equal-width) histogram $\mathbf{h}(\mathbf{A}, D) = (c_1, c_2, \dots, c_m)$ is a list of counts for the attribute values in \mathbf{A} . Specifically, \mathbf{h} partitions¹ $\text{dom}(\mathbf{A})$ into m “equal-sized” domain intervals (B_1, \dots, B_m) , and each count $c_i \in \mathbf{h}$ corresponds to the total number of $t \in D$ such that $t.\mathbf{A}$ fall within B_i .

Multi-party secure computation (MPC). MPC [11, 28, 48, 78] is a cryptographic techniques to allow a group of participants P_1, P_2, \dots to jointly compute a function $f(x_1, x_2, \dots)$ over their respective private input x_i . MPC ensures that no unauthorized information is revealed to any involved party, excepted the desired output of function f , emulating a computation as if performed by a trusted third party. Earlier MPC typically require all parties to participate in intensive computations. Recent schemes [39, 51, 63] allow these computations to be offloaded to multiple powerful servers without sacrificing original security guarantees, known as the server-aided MPC. In this approach, each party, say P_i , secretly shares their inputs, x_i to servers. These servers then jointly evaluate an MPC that functionally replicates the logic of first reconstructing the secrets x_1, x_2, \dots and then evaluating $f(x_1, x_2, \dots)$. Depending on designs, the result may be directly returned to the parties or re-shared among the servers for later retrieval. By default, SPECIAL considers server-aided MPC with two non-colluding servers.

Secret sharing and secure array. SPECIAL uses the 2-out-of-2 boolean secret share [5] 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: x_1, x_2 that are uniformly distributed over the ring $\mathbb{Z}_{2^{32}}$ such that $x = x_1 \oplus x_2$. 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 [9, 73], denoted as $\langle \mathbf{x} \rangle = (\langle x_1 \rangle, \langle x_2 \rangle, \dots)$, which is a collection of secret-shared relational tuples.

Oblivious (relational) operators. Oblivious operators are specific MPC protocols, applied over secure arrays, that implement database operator functionalities such as filter, join, and aggregation. Specifically, an oblivious operator $\langle \mathbf{o} \rangle \leftarrow \text{op}(\langle \mathbf{a} \rangle, \langle \mathbf{b} \rangle, \dots)$ accepts one or more secure arrays as inputs, performs database operations over the inputs, and then writes the result to an output secure array. Moreover, the oblivious property stipulates that absolutely no

¹We use a condensed notation that omits arguments when \mathbf{A} and D are obvious.

memory access pattern information is disclosed [18, 46, 60] (or in other words, the memory access is fully independent from input data) during the protocol’s execution. Since revealing output size can also enable numerous reconstruction attacks, recent designs of oblivious operators also consider hiding output volumes, i.e., by padding the outputs to their maximum sizes. Thus, unless otherwise stated, we assume that all the subsequent oblivious operators mentioned have considered volume-hiding.

Differential privacy [24]. DP ensures that modifying a single input data tuple to a mechanism produces only a negligible change in its output. To elaborate, consider D and D' as two databases differing by just one tuple—termed as neighboring databases. Then:

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

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

3 PRIVACY MODEL

In this section, we formally define the privacy definitions of SPECIAL and explain the corresponding semantics. Similar to previous work on DPSCA [73, 79], we follow computational DP (SIM-CDP) [49] framework for privacy model formulation.

Definition 3.1 (Secure protocol with DP leakage). Given a party (owner) P with private data D and a randomized mechanism $\text{Lkg}(D)$, referred to as the leakage profile of P ’s data. Consider Π a secure query processing protocol applied to D . The protocol Π is said to be secure with DP leakage if, for any honest party P with data D and any probabilistic polynomial time (p.p.t.) adversary \mathcal{A} :

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

$$\Pr \left[\mathcal{A} \left(\text{VIEW}^\Pi(\mathcal{D}, \text{pp}) = 1 \right) \right] \leq \Pr \left[\mathcal{A} \left(\text{VIEW}^\mathcal{S}(F(\mathcal{D}), \text{pp}) = 1 \right) \right] + \text{negl}(\kappa) \quad (1)$$

where VIEW^Π , and $\text{VIEW}^\mathcal{S}$ denotes the adversary’s view against the protocol execution and the simulator’s outputs, respectively. And $\text{negl}(\kappa)$ is a negligible function related to a security parameter κ .

Simply put, Definition 3.1 specifies that as long as there is at least one honest owner, the knowledge any p.p.t. adversary can obtain about this owner’s data from observing Π execution transcripts is bounded to what can be inferred from the outputs of an (ϵ, δ)-DP mechanism Lkg . By default, we define Lkg within (ϵ, δ)-tuple level DP [23], that’s said primary privacy objective is to protect individual tuples. However, due to group-privacy properties [42, 77] of DP, privacy can extend across multiple tuples and to user-level, as long as a user owns a limited number of tuples. Throughout this paper, our focus remains on developing algorithms that guarantee tuple-level privacy, while upholding all stated privacy guarantees, albeit with potentially different privacy parameters.

THEOREM 3.2. *The privacy of SPECIAL adheres to Definition 3.1*

PROOF. Due to space concerns, we defer the complete privacy proof of SPECIAL to Appendix A. \square

4 SPECIAL SYNOPSES

SPECIAL maintains independent DP synopses for all outsourced data. Each synopsis holds noisy statistics that reflect the corresponding data’s distributional traits. As per our core design idea, these statistics will be used to dynamically optimize query processing—for example, by estimating tight compaction bounds or enabling fast access to eliminate unnecessary tuples, which significantly compacts intermediate query sizes, all without any extra privacy cost.

Challenges. A major design challenge is selecting proper synopses to achieve our design goals. This involves decisions such as attribute selection, combinations thereof, and the types of statistics used (e.g., histograms, min/max/avg, frequency counts, etc.). Using too many attributes or leveraging complex (high-dimensional) combinations can introduce significant noise [81] and increase storage costs. Another challenge involves privately releasing synopses to aid in lossless processing, such as result compaction and private indexing. Ensuring lossless compaction requires consistently overestimating the output sizes of subplans. Supporting lossless indexes is even more complex: for a given indexed range $D[\text{lo}, \text{hi}]$, we need to overestimate the upper index hi and underestimate the lower index lo . Unfortunately, traditional DP methods, which typically use symmetric additive noises, struggle to ensure these conditions.

Key ideas. We observe that joins are more resource-intensive than any other operations. For instance a k -way oblivious join can incur a $O(n^k)$ complexity if without optimisations [8, 9, 26, 73, 82]. Hence, joins require prioritized acceleration. Moreover, synopses available for frequently queried filter attributes on base relations are vital [73] as well. These statistics not only help to compact input data at an early stage before heavy joins, but also enable fast indexing (§ 5.2) for quick data access. Hence, to address the first challenge, our key idea is to focus on commonly queried join and base relation filter attributes, as well as their low-dimensional combinations.

To tackle the second challenge, our key idea is twofold. First, to ensure lossless result compaction of filters and indexing, we employ one-sided DP noise to generate private histograms that either overestimate or underestimate attribute distributions consistently. Second, to achieve lossless compaction of join outputs, we integrate noisy max frequency moments (MF) into the synopses. This integration enables us to extend upon advanced join upper bound techniques [36] to privately estimate join compaction sizes without losing data (§ 5.2).

4.1 Synopses generation

Steered by our key ideas, we now elaborate on our synopsis generation details. The general process is outlined as follows: Initially, the SPECIAL servers selects appropriate attributes to generate private synopses, which are then distributed to owners. The owners create the synopses using a DP mechanism and upload them to SPECIAL servers. Finally, the server applies post-processing to these synopses to aid in query processing.

Attributes selection (servers). The first step is to identify a smaller set of attributes for deriving synopses. In general, we consider the existence of a representative workload, Q_R [44], which can be sourced from a warm-up run of SPECIAL or annotated by

the administrator. Note that this process do not incur any private data thus is leakage-free. The servers first identify representative attribute pairs, $\text{pair} = \{\text{pair}_k\}_{k \geq 1}$, 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 $\text{pair}_k = (A_{\text{ft}}, A_j)$ 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 each owner, and subsequently, the owner independently dispatches a private synopsis and returns it to servers. In what follows, we focus on the DP synopses generation mechanism evaluated by each owner, Algorithm 1 illustrates the overall flow.

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

Input: $\text{pair} = \{\text{pair}_k\}_{k \geq 1}$ from servers; private data D .

- 1: P self-determines privacy parameters ϵ, δ , and init $\text{synop} \leftarrow \emptyset$
- 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** $\text{MF}_k = 0$
- 7: **else if** $A_{\text{ft}} \in \text{pair}_k \neq \emptyset$ **then**
 ▷ assuming \mathbf{h} partitions $\text{dom}(A_{\text{ft}})$ into $\{B_1, \dots, B_m\}$
- 8: $D^\ell \leftarrow \sigma_{A_{\text{ft}} \in B_\ell}(D)$ **for** $\ell = 1, 2, \dots, m$
- 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: $\text{synop} \leftarrow \text{synop} \cup (\text{pair}_k, \{\mathbf{h}^+, \mathbf{h}^-\}, \text{MF}_k)$
- 12: **release** $\text{synop}, \epsilon, \delta$ to servers

Generally, we expect owners to set a desired privacy budget for their data (using parameters ϵ and δ) before initiating the synopsis generation. For each pair_k , the owner first creates a histogram $\mathbf{h}(\text{pair}_k, D)$. By default, we assume there exist global parameters, i.e. bin sizes of each attribute, which guide all owners to partition attributes consistently. 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 noises to every bin of $\mathbf{h}(\text{pair}_k, D)$.

Definition 4.1 (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

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

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

The one-sided Laplace noise can be strictly positive (Lap^+) or negative (Lap^-). This ensures that \mathbf{h}^+ always overestimates the actual histogram, while \mathbf{h}^- consistently underestimates it. For the ease of description, we refer $\mathbf{H}(\text{pair}_k, D) = \{\mathbf{h}^+, \mathbf{h}^-\}$ as the *bounding histogram* of attribute pair_k , with \mathbf{h}^+ , and \mathbf{h}^- to be the upper and lower histograms, respectively.

We continue with the details of the (noisy) MFs generation. Assume both $A_{\text{ft}}, A_j \neq \emptyset$, and the $\mathbf{h}(\text{pair}_k, D)$ partitions $\text{dom}(A_{\text{ft}})$ into

$\{B_1, \dots, B_m\}$. The mechanism $\mathcal{M}_{\text{synop}}$ generates a table of noisy MFs, $\{\widehat{\text{mf}}(A_j, D^\ell)\}_{\ell \geq 1}$, where each entry $\widehat{\text{mf}}(A_j, D^\ell)$ is an independent MF statistic for A_j over specific filtered data on A_{ft} , such that

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

here $\mathcal{G}_{\text{count}(A_j)}$ is a group-by-count operation over A_j , and $\widehat{\text{max}}_\epsilon$ is a *report noisy max* mechanism [24]. 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. Given the discussion so far, we summarize SPECIAL synopses as:

Definition 4.2 (SPECIAL synopses). Given representative workload Q_R , we consider for each relation D , its corresponding synopsis synop to be the collection of $\{(\text{pair}_k, \mathbf{H}(\text{pair}_k, D), \text{MF}_k)\}_{k \geq 1}$, such that

- pair_k is a frequently queried attribute (pair) over D found in Q_R .
- \mathbf{H} is the private bounding histogram for pair_k over D .
- MF_k represents a collection of privately overestimated join key MFs that corresponds to a group of tuples in \mathbf{h} , categorized by $\text{pair}_k \cdot A_{\text{ft}}$.

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 the complete proof to Appendix A. 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 report 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 $(3\epsilon, \delta)$ -DP. Given there are in total c such pairs, and thus the total privacy loss is subject to c -fold advanced composition [24].

Synopsis transformations. We say that one can perform transformations on released synopses without incurring extra privacy loss, per the post-processing theorem of DP [24]. Now, we briefly outline key synopsis transformations relevant to the SPECIAL design. First, given any (2d) bounding histogram $\mathbf{H}(\text{pair}, D)$ with both $A_{\text{ft}}, A_j \in \text{pair}$ are non-empty, one can derive the (1d) bounding histograms, i.e. $\mathbf{H}(A_j, D)$ and $\mathbf{H}(A_{\text{ft}}, D)$, 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 even following a selection on the base relation. For example, given $A_{\text{ft}}, A_j \in \text{pair} \neq \emptyset$, and let $D' \leftarrow \sigma_{A_{\text{ft}} \in \text{vals}}(D)$, 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

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

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 now outline key secure database operations offered by SPECIAL. A major challenge in designing synopsis-assisted secure processing is the evident knowledge gap on how private synopses could potentially accelerate oblivious query processing. To bridge this gap, we explore various uses of synopses, such as creating private indexes (SPEIDX 5.2) and designing compacted oblivious operations (SPEOP 5.3). Moreover, as discussed earlier, joins are the most intensive operations. Hence, we extensively optimize join algorithms, synergizing both private indexing and compaction techniques to develop a novel, parallel-friendly oblivious join (§ 5.3). Another challenge in primitive design arises from ensuring lossless guarantees. To address this, we integrate mechanisms into our designs that pessimistically estimate selection cardinalities, indexing ranges, and complex join sizes, using a combination of available synopses and advanced upper bound techniques [36]. For clarity, we'll assume, WLOG., that all input relations below are of size n and all 1d histograms contain m bins.

5.1 Basic Operations

SPECIAL supports conventional fully-oblivious operators [8]. In general, these operators function logically the same as non-private ones, but their execution transcripts are completely data independent, with results consistently padded to the worst-case maximum.

Default data access SeqACC. By default, all query executions proceed with a standard data access, loading all data into a secure array using a sequential scan². Additionally, each loaded tuple gets a secret bit (initially '0'), *ret*, used to mark tuple validity.

SELECT. is the secure version of filter $\sigma_p(R)$. It conducts a linear scan over secure array $\langle R \rangle$ for relation R and updates the *ret* bit: '1' for tuples satisfying predicate p and '0' for others. Every tuple's *ret* bit is updated, regardless of whether it matches the predicate.

PROJECT. discards irrelevant attributes for relation stored in $\langle R \rangle$. Note that the secret bit *ret* cannot be discarded by a PROJECT.

JOIN. implements the secure version of θ -join, $R_0 \bowtie_\theta R_1$. It processes two secure arrays $\langle R_0 \rangle$ and $\langle R_1 \rangle$, computing their cartesian product $\langle R_0 \times R_1 \rangle$ and then using SELECT to mark the joined tuples. It's important to note that JOIN expands output, always padding it to the worst-case max (product of input sizes).

COUNT, SUM, MIN/MAX are the aggregation primitives. The operators linearly scan a secure array and continuously update a secret-shared aggregation value after each tuple access.

ORDER-BY, DISTINCT, GROUP-BY (AGG). are built on the oblivious sort primitive [7]. Specifically, ORDER-BY obliviously sorts a secure array according to a given attribute. DISTINCT sorts a secure array first, then performs a linear scan to identify unique tuples. For instance, among consecutively sorted identical tuples, it sets the *ret* bit of the last tuple to '1' and the rest to '0'. GROUP-BY (AGG) starts by using DISTINCT to identify 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.

²In the context of a secret-shared array $\langle D \rangle$, this step can be each server sequentially load the respective secret shares of D into memory, readying them for MPC.

5.2 SPECIAL Index SPEIDX

In conventional databases, indexes are powerful data structures that map attribute values to positions within a sorted array, enabling fast data access. However, generalizing such feature to SCA remains largely underexplored with existing solutions facing significant limitations such as high data loss [61], dependence on complex data structures filled with dummy tuples [14, 79], and limited query capabilities (i.e., linear queries across fixed attributes). Moreover, these methods only index base relations. SPEIDX innovatively overcomes these issues by enabling the creation of lossless indexes directly on outsourced data or query intermediate results, without the need for additional structures or dummy injections.

In general, SPEIDX builds upon the typical indexing model that utilizes cumulative frequencies (CF) [45]. 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 (e.g. 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 desired records. For instance, as illustrated in Figure 2, the SPECIAL index might estimate the index range for attribute value 24 as $[g^-(23), g^+(24)] = [198, 267]$.

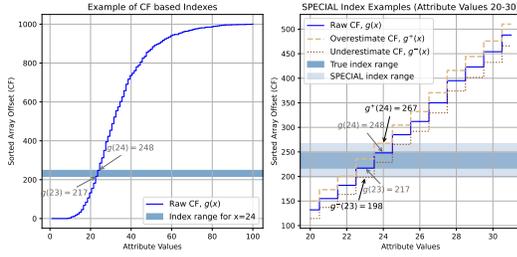


Figure 2: True index (left) vs. SPECIAL index (right) for $x = 24$

In what follows, we provide the formal explanations on how SPEIDX derives indexes from DP synopses. Specifically, SPEIDX first determines the bounding histograms $H(A, D)$, which may be either transformed from an available 2D histogram $H(pair, D)$ with $A \in pair$, or sourced directly if $H(A, D)$ 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 $H(A, D) = \{h^+, h^-\}$, and assume $h^+ = (c_1^+, \dots, c_m^+)$, $h^- = (c_1^-, \dots, c_m^-)$ partitions $dom(A)$ into $\{B_1, \dots, B_m\}$. We say $SPEIDX(A, D) = \{idx_i = [lo_i, hi_i]\}_{1 \leq i \leq m}$ is the SPECIAL index of D over A with:

- $\forall i \geq 1, hi_i = \min(|D|, \sum_{k=1}^i c_k^+)$.
- $lo_1 = 0$, and $\forall i \geq 2, lo_i = \sum_{k=1}^{i-1} \max(0, c_k^-)$.

By this construction, all tuples $t \in D$ such that $t.A \in B_i$ will be organized into the subset $D[idx_i] \subseteq D$. 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 values. The bounding histogram’s pessimistic estimation ensures that all tuples where $t.A \in B_i$ are accurately contained within $D[idx_i]$, thereby achieving lossless indexing. In contrast to existing methods that are limited to indexing base relations [14, 61, 79], 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 $H(A, D')$ from $H(A^*, D)$ and subsequently build indexes on D' . Importantly, since index creation is a post-processing activity using available DP synopses, it incurs no additional privacy loss.

Indexed store and fast data access IdxAcc. SPEIDX enables the exploration of a new storage layout for organizing outsourced data, namely indexed datastore, that facilitates private data access. 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) potentially bypassing selection operations (§ 5.3). We emphasize that the SPECIAL design does not require replicating the outsourced datastore to accommodate multiple query types [14, 79]. However, creating compact replicas (e.g., column replicas [37] 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, all achieved without extra privacy loss. In other words, this implies that one can selectively adjust these objects to align with dynamic query workloads, without incurring any privacy costs.

5.3 SPECIAL Operators SPEOP

We introduce SPEOP, a set of novel synopsis-assisted operators that maintain full obliviousness (e.g., operators’ execution cause no privacy loss) while enabling lossless compaction. To our knowledge, SPEOP is the first primitive of its kind in any SCA system.

Oblivious compaction: OPAC. We first introduce the fundamental operation of oblivious compaction, a key component for many 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 eliminate a large portion of non-matching tuples. To determine the compaction size cs , (OP)SELECT

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

```

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

```

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.

(SP)SELECT. The running complexity of (OP)SELECT depends on OPAC, which is typically linearithmic (see § 6.1 or [62]). 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 $\langle D \rangle$, 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 $\langle R_o[idx + 1] \rangle$ and a *dummy write*³ is made to elsewhere. If not, dummy writes are made throughout $\langle R_o \rangle$. Similarly, (SP)SELECT does not yield additional privacy loss, and the complexity is linear in its input.

(DC)SELECT. Finally, if the underlying data is already indexable on A , a direct selection over the data can be applied that bypasses secure computations. The operator simply looks up SPEIDX(A, R), and accesses $R[a, b]$, where $a = \min_j(idx_j.lo), b = \max_j(idx_j.hi)$, where idx_j denotes the index in SPEIDX(A, R) where $B_j \cap vals \neq \emptyset$.

SPECIAL join: (MX)JOIN. We now introduce a novel MF-Index based oblivious join operation. The advancements of (MX)JOIN stand out in two key aspects compared to existing oblivious joins. 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 oblivious joins with DP leakages typically require additional privacy budget to learn join sensitivity [22] or necessitate truncations on joined tuples [9, 73]. 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.

In general, (MX)JOIN can be applied to two types of data: the base and pre-filtered relations where the join key attribute is contained by the synopses. 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

³In 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.

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

```

Input: relations  $R_0, R_1$ ; join attribute  $A_j$ .
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:   for  $b \in \{0, 1\}$  do
4:     derive  $\widehat{mf}(A_j, R_b)$  from  $\text{synop}_b$  (§ 4)
5:     build index SPEIDX( $A_j, R_b$ ) =  $\{idx_i\}_{i=1, \dots, m}$  (§ 5.2)
6:   if  $\forall b, \widehat{mf}(A_j, R_b)$ , and SPEIDX( $A_j, R_b$ )  $\neq$  null then
7:     oblivious sort  $R_0, R_1$  on  $A_j$ , BucketJoin( $R_0, R_1, A_j$ )
8:   else assert “not applicable for (MX)JOIN”

BucketJoin( $R_0, R_1, A_j$ ):
9: for  $i = 1, 2, \dots, m$  do
10:  compute  $O_i \leftarrow (R_0[idx_i] \bowtie_{A_j} R_1[idx_i])$  via basic JOIN
11:   $cs_i \leftarrow \min\left(\frac{|R_0[idx_i]|}{\widehat{mf}(A_j, R_0)}, \frac{|R_1[idx_i]|}{\widehat{mf}(A_j, R_1)}\right) \times \widehat{mf}(A_j, R_0) \cdot \widehat{mf}(A_j, R_1)$ 
12:   $R_{out} \leftarrow R_{out} \cup \text{OPAC}(O_i, cs_i)$ 
13: return  $R_{out}$ 

```

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* [36] 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. For better illustration, we visualize (MX)JOIN’s processing flow in Figure 3.

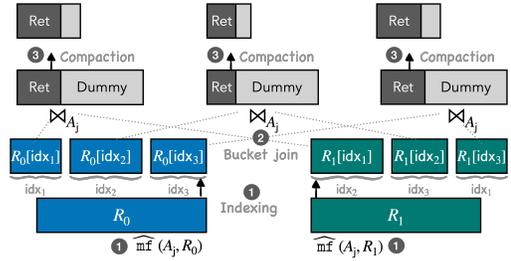


Figure 3: Example processing flow of (MX)JOIN.

Since (MX)JOIN derives join compaction sizes completely from post-processing of DP synopses, it thus incurs no extra privacy loss. Moreover this design choice also eliminates the need for computing join sensitivities — a major hurdle in existing — as accurate join sensitivity estimation is hard and typically requires a significant privacy budget [22]. Additionally, the noisy MF bounds guarantee that compaction sizes are consistently overestimated, ensuring lossless compaction of join results.

6 SPECIAL PLANNER

A major limitation of existing DPSCA designs is sub-optimal execution planning. Without the ability to pre-estimate intermediate result sizes, traditional methods cannot identify the most efficient execution strategies. To address this, SPECIAL employs an innovative query planner that leverages available synopses for plan size estimation, enabling optimal execution planning.

At a high level, our planner is modeled on the Selinger-style (cost-based) optimizer [13]. It uses a bottom-up, dynamic programming approach to enumerate all equivalent secure execution plans for a given query and estimate their costs to select the optimal one. However, challenges in the design space still remain. First, the introduction of SPECIAL primitives can significantly impact the cost modeling for oblivious operations, rendering existing models [9, 46] ineffective. Additionally, the equivalent plan search space is extensive, requiring further strategies to simplify the search space. To address these challenges, we first conduct a comprehensive analysis of the complexities of SPECIAL primitives and develop a new cost model (§ 6.1) for SPECIAL planner. Next, we design protocol-specific heuristics (§ 6.2) tailored for the SPECIAL planner to simplify the plan search space. Note that existing features of the Selinger-style optimizer, such as plan enumerations, are not within our contributions. For detailed understanding of these features, we encourage readers to explore the original *IBM System R* paper [13].

6.1 Cost Model

Similar to the cost model proposed by Shrinkwrap [9], we view the cost of a secure execution plan cost as a combination of each operator’s I/O and secure evaluation costs. In addition, we formulate these costs as functions over their processed data sizes. Specifically, given a plan with ℓ operators, op_1, \dots, op_ℓ , and let $\mathbf{I} = \{I_1, \dots, I_\ell\}$, $\mathbf{O} = \{O_1, \dots, O_\ell\}$, to be the input and output sizes of each operators. The plan cost is formulated as:

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

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. If the accessed data has already loaded into memory, C_{in} becomes negligible. 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 expenses for evaluating each operators, typically constituting the dominant cost. Note that, in practice, the exact formulas for C_{in} , C_{out} , and C_{eval} can vary depending on the specific secure protocol employed (garbled circuits [78], secret sharing [48], etc.) as well as the particular hardware configurations in use. Nonetheless, the understanding of the asymptotic costs is adequate for comprehending the principles of our query planning and optimization strategies [9, 46]. 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. Oblivious sorting and compaction (OPAC) are crucial for many SPEOP operators, so we will begin our analysis with these two primitives. While oblivious sorting algorithms with optimal $O(n \log n)$ complexity exist, they often necessitate either impractically large constants [3, 29] or client-side memory [6], 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 [62]

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

Projection, grouping and aggregation. The PROJECT accesses private relations and discards unnecessary columns independently on each server, without secure computations. 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 primarily due to 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 [62]. 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 $\text{hist_bound} = O(\sum_{B_i \cap vals \neq \emptyset} |B_i|)$, 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. It simply loads indexed data into secure arrays without secure computations or result write-backs. As such, both C_{eval} s and C_{out} are negligible, with C_{in} within $\text{idx_bound} = O(\max_i(\text{idx}_i.\text{hi}) - \min_i(\text{idx}_i.\text{lo}))$. 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, using a Cartesian product approach, doesn’t reduce output size, so both C_{eval} and C_{out} are within $O(n^2)$. 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 relatively low.

6.2 Heuristics

H-1. Filter push down. Filter pushdown is a widely used optimization technique in conventional databases, where it involves moving the selection operation to the earliest stage in the data processing pipeline, such as directly to the base relation. This reduces the amount of data that needs to be loaded, transferred, and processed, by following operators especially complex joins. However, in SCA designs, data obliviousness typically necessitates padding selection sizes to the worst-case scenario, rendering filter pushdown ineffective [26, 46, 57, 82]. With SPEOP selections, the selection outputs can be significantly compacted, making filter pushdown effective again. Therefore, we include filter push down as one of the optimization heuristic for SPEPLAN.

H-2. Predicates fusion. Let R to be any relation, $A_1, A_2, \dots, A_k \subseteq \text{attr}(D)$, and $\mathbf{v} = \{v_1, v_2, \dots, v_k\}$. We say that for multiple selection over R such that $\sigma_{A_1 \in v_1} (\dots \sigma_{A_k \in v_k} (R))$, one can always fuse them into one selection $\sigma_{A_C \in v} (R)$. This can reduce the number of secure computation invocations from k rounds to just one. Additionally, the selection size can be estimated as $\min_{A_i} (\text{CardEst}(\sigma_{A_i \in v_i} (R)))$.

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 [36], one can also update the MF for R by computing $\widehat{\text{mf}}(A_j, R) = \widehat{\text{mf}}(A_j, R_0) \times \widehat{\text{mf}}(A_j, R_1)$. 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 advantages over current SCA approaches? To what extent can SPECIAL enhance efficiency compared to existing state-of-the-arts (SOTAs)?
- **Question-2:** For the DP-based SPECIAL design, is there a trade-off between privacy and efficiency? Can we adjust privacy levels to achieve various efficiency objectives?
- **Question-3:** Can SPECIAL scale complex analytical (e.g. multi-way join) queries to large-scale (multi-million rows) datasets?

7.1 Experimental setups

We briefly outline our experimental setup, covering the baseline systems, prototype SPECIAL implementation, workloads, datasets, and default configurations.

Baseline systems and SPECIAL prototype. We compare SPECIAL with two baseline systems: Shrinkwrap [9], the SOTA DPSCA, and SMCQL [8], a typical SCA system using exhaustive padding for leakage hiding, 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 [46, 57] similar to SMCQL use exhaustive padding but improve efficiency through protocol-level optimizations, we exclude them from our benchmarks for fair comparison concerns. However, it's important to note that these optimizations are orthogonal to

Table 2: Query workloads

Query No.	Type	Description
Q1	Linear	Point query with single predicate.
Q2	Linear	Range query with conjunctive predicates.
Q3	Binary Join	Non-expanding binary join.
Q4	Binary Join	Expanding binary join.
Q5	Multi Join	3-way mixed non-expanding and expanding joins.
Q6	Multi Join	3-way all expanding joins.
Q7	Multi Join	4-way mixed non-expanding and expanding joins.
Q8	Multi Join	5-way mixed non-expanding and expanding joins.

both the baseline systems and our designs. We re-implement important query features for the two baseline systems, and built SPECIAL using the same MPC package, specifically the `sh2pc-0.2.2` protocol within `EMPToolkit-0.2.5`. Since managing DP synopses and generating indexes do not involve secure computations, thus we implement the related features using Python (3.7) scripts. All implementations are open-sourced [47].

Datasets and workloads. Although SMCQL and Shrinkwrap use the HealthLNK dataset, it is not public data and thus not accessible to the broader research community. Hence, we developed a new benchmark based on a publicly available dataset, and mimic Shrinkwrap's workload to create a set of testing queries. Specifically, we use the *Czech Financial Dataset* [1], an anonymized Czech bank transaction dataset for loan applications. This dataset comprises 8 relational tables with a total of 55 columns and 1.1 million rows. For scaling experiments, we use the raw data and schema to generate synthetic data with up to 10 million rows. Shrinkwrap's benchmark evaluates four queries: a range query, two 2 binary joins, and a 3-way join. Mirroring this approach, we designed eight query workloads based on the Financial data. All workloads are selection-projection-join (SPJ) queries followed by statistical aggregations such as count, count distinct, and group by. The workloads range from simple linear queries to complex multi-way joins. Due to space limitations, a brief summary of the workloads is provided in Table 2, with the full specification in Appendix B.

Default configurations. For SPECIAL, each relational table is assigned a fixed privacy budget with $\epsilon = 1.5$ and $\delta = 0.00005$ to release synopses. This setup ensures that the total privacy loss per table does not exceed $\epsilon = 1.5$. For all equal-width histograms generated in SPECIAL, we configure them to have at most 8 bins. For Shrinkwrap, we use a privacy budget of $\epsilon = 1.5$ and $\delta = 0.00005$ for each query processing, as specified in [9]. However, Shrinkwrap does not provide bounded privacy loss guarantees, so the privacy guarantee of Shrinkwrap will be no better than SPECIAL. We conduct all experiments on bare-metal Mac machines with M2 Max CPUs and 96GB unified memory. In addition, we consider Q2, Q4, Q8 are default testing queries in each query type, and are used as the representative workload to guide synopses. Moreover, for baseline systems, as they do not have join ordering optimizations, thus we will assume a random join order for them.

7.2 End-to-end performance comparisons

To address **Question-1**, we first conduct an end-to-end performance comparison of SPECIAL, Shrinkwrap, and SMCQL across the full benchmark workloads (Q1-Q8). We use two typical performance metrics, query execution time and memory usage in our evaluations. The results are summarized in Figure 5 and Figure 6.

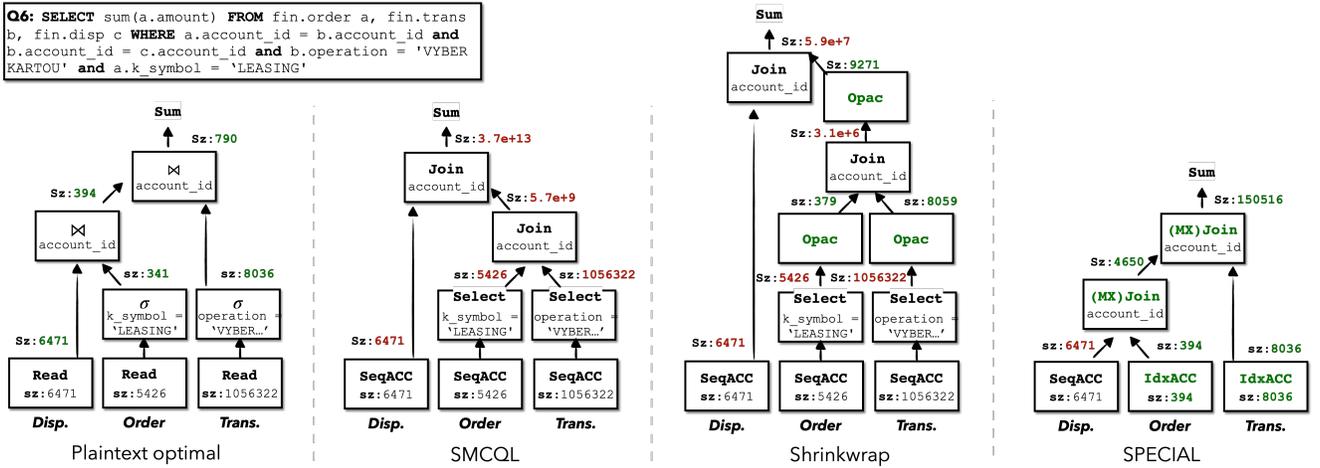


Figure 4: In-depth comparisons in execution plans: (i) The exhaustive padding in SMCQL can lead to significant memory blowup; (ii) Both SMCQL and Shrinkwrap suffer from suboptimal join ordering; (iii) Although Shrinkwrap reduces intermediate sizes, it still requires substantial memory to materialize the Cartesian-style standard oblivious join; (iv) SPECIAL effectively identifies optimal join orders and significantly reduces intermediate materialization sizes; (v) The IdxAcc method in SPECIAL not only bypasses secure selections but also directly reduces input I/O costs.

Due to significant memory consumption, SMCQL cannot complete the full benchmark and stops at Q4. Therefore, we only include full statistics up to Q3, and for Q4, we provide a projected performance evaluation using 10% of the input data.

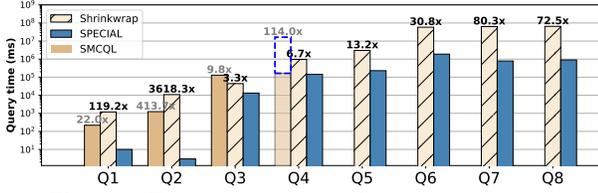


Figure 5: End-to-end comparison: query latency

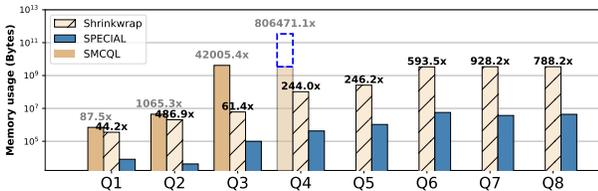


Figure 6: End-to-end comparison: memory usage

Observation 1. SPECIAL significantly outperforms all baseline systems in query latency, achieving performance gains of up to 3618.3× for linear queries, 114× for binary joins, and 80.3× for multi-joins. Figure 5 shows significant performance gains of SPECIAL in contrast to Shrinkwrap and SMCQL. For linear queries, SPECIAL exhibits substantial performance gains, achieving up to 3618.3× improvement (Q2), primarily due to its index-based fast data access. This approach not only reduces large I/O costs but also eliminates the need for secure computation by enabling direct data retrieval via DP indexes. We observe that in binary joins, SPECIAL has a less pronounced advantage over Shrinkwrap. This is because Shrinkwrap does not have sub-optimal planning issue when dealing with binary join—as there is only one join order. Hence, the

performance gains for SPECIAL in Q3 and Q4 stem solely from fast data access and the efficient (MX)JOIN method. However, for more complex multi-way joins, SPECIAL’s advantage becomes more pronounced again due to its ability to select the optimal join order. For instance, Shrinkwrap shows more than 80× slow down in query latency than SPECIAL in group Q7.

Observation 2. SPECIAL shows profound improvement in memory usage (Figure 6) 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 10⁵× over SMCQL—we will zoom into a specific query, Q6, and compare the detailed execution plans of the three systems. The choice of Q6 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 derived directly from the logical plan using actual cardinalities, illustrating the ground truth optimal execution; a hypothetical SMCQL plan (since Q6 cannot be completed by SMCQL so we project the cardinalities); and two actual execution plans from our experiments with Shrinkwrap and SPECIAL. The detailed comparisons and observations are summarized in Figure 4.

7.3 Privacy comparisons

Continue to address **Question-1**, we now compare the privacy guarantees between Shrinkwrap and SPECIAL, with the results detailed in Figure 7. As privacy loss is independent of query types, our analysis will focus on comparing the privacy composition curves under

continual query answering for both Shrinkwrap and SPECIAL. We consider two privacy composition models: advanced composition (Adv.)[24] and concentrated DP composition (CDP)[15].

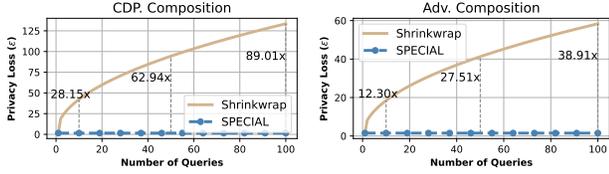


Figure 7: End-to-end comparison privacy loss.

Observation 3. Under continual query answering, SPECIAL demonstrates significantly lower privacy loss compared to Shrinkwrap, achieving up to 89.01x and 38.91x 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 7. 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., respectively, 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.

7.4 Privacy efficiency tradeoffs

We address **Question-2** by evaluating SPECIAL at various privacy levels and observing the performance impacts. Specifically, we maintain δ constant while varying ϵ from 0.1 to 10 and assess the performance across default testing queries (Q2, Q4, Q8). The results are shown in Figure 8 and Figure 9.

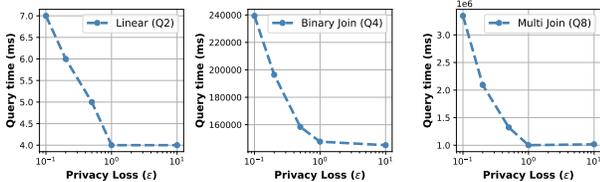


Figure 8: Privacy vs. Performance (query time)

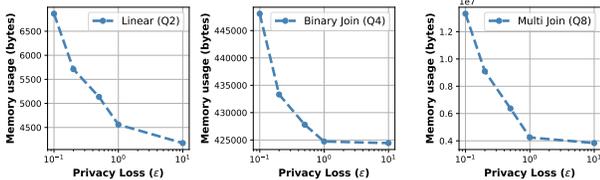


Figure 9: Privacy vs. Performance (memory)

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 already 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.

Observation 5. Different queries exhibit varying performance sensitivities to changes in privacy levels. Multi-join queries are significantly affected when privacy levels change. For instance, the performance overhead of Q8 at $\epsilon = 0.1$ is more than 3x higher than at $\epsilon = 1$. In contrast, linear query (Q2) and binary join (Q4) overhead increases to only about 1.5x under the same privacy settings. This difference stems from the error amplification inherent in multi-join operations, which frequently reuse synopses and require multiple cardinality estimations. Therefore, practitioners dealing with complex multi-join workloads may need careful fine-tune privacy parameters, as performance impacts can be substantial.

7.5 Scaling experiments

To address **Question-3**, we stress SPECIAL with scaled workloads by duplicating the raw financial dataset to sizes of 2x, 4x, and 8x. We then evaluate the default testing queries on these scaled datasets to assess SPECIAL’s efficiency. The results are shown in Figure 10.

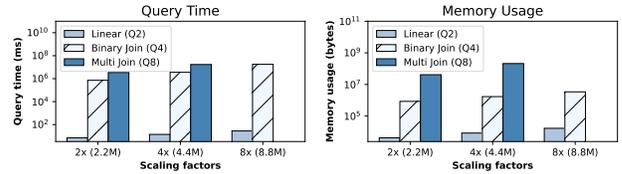


Figure 10: End-to-end comparison privacy loss.

Observation 6. SPECIAL demonstrates significant potential to scale up to multi-million records, even when handling complex 5-way joins. Figure 10 shows SPECIAL’s effective scaling: up to 8x data for linear queries and binary joins, and up to 4x data for complex 5-way joins like Q8. For instance, Q2 can be completed within 290ms under 8x, 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 8x, while the more complex 5-way join Q8 takes less than 280 minutes for 4x data. As a reference, Shrinkwrap would require over 1035 minutes to complete Q8 even with unscaled data (Figure 5). This result can, from another angle, showcase the significant performance improvements of SPECIAL over current SOTA DPSCA designs.

8 DISCUSSION

In this section, we discuss potential extensions to accommodate broader data models and possible adaptations of SPECIAL designs to other secure primitives.

Supporting growing data. In general SPECIAL considers read-intensive analytical workloads and a static data model, which

is the same as current SOTA SCA efforts [9, 46, 57, 69]. In this section, we will briefly discuss how SPECIAL could be extended to support the growing data model where new records are continuously added to the logical database [72, 73, 79]. One possible strategy involves segmenting the large logical database into discrete, non-overlapping time intervals, each represented by its own database with independent synopses. Specifically, for new incoming data, SPECIAL will initially cache it until a designated time interval concludes. After this period, SPECIAL will batch all the cached data into a new database corresponding to the past time interval and release synopses for this database. As the data within these intervals is disjoint, queries can be easily partitioned and independently optimized and executed across corresponding databases in parallel, before being aggregated. Note that prior to releasing new synopses, all cached data will be processed using standard SCA processing (e.g. SMCQL). Since each synopsis release covers only newly inserted data, the total privacy loss adheres to parallel composition [24], and thus the bounded privacy loss still hold.

Adaptability to other secure primitives. Beyond MPC-based SCA, the key design insights from SPECIAL are transferable to other secure outsourced databases such as the TEE-based analytical systems [26, 58, 59, 82] or searchable encryption based private key-value store [4, 19, 30]. This is because SPECIAL’s core components, including synopses management, indexing, query planning, and compaction estimation, can all be executed without relying on secure computation primitives. For example, in TEE databases, one may conduct a similar query planning outside the TEE, with the optimal plan then executed within the TEE for fast oblivious processing. Similarly, SPECIAL’s design principles could be applied to searchable encryption to construct private indexes in advance on sorted data, enable fast and private information retrieval.

9 RELATED WORK

SCA systems. Two main approaches exist for designing MPC-based SCA systems. The first is *peer-to-peer paradigm* [2, 52, 57, 69, 76], 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 smaller group of parties. Unfortunately, this approach burdens data owners with heavy computational overheads, particularly for complex operations like joins. Moreover, given real-world data owners often lack robust computing resources and reliable service capabilities, these methods are difficult to scale and promise consistent SCA services to external analysts. On the other hand, the *server-aided-MPC model* [9, 10, 39, 43, 46, 51, 61, 67, 72, 73, 79] offers a more practical solution. It allows data owners to outsource both expensive MPC computations and secure data storage to a set of capable cloud service providers (CSPs), who then jointly evaluate MPC to provide reliable SCA services. For the SPECIAL prototype, we focus on the server-aided-MPC model under a strong corruption assumption, allowing up to $n - 1$ of n servers to be compromised and collate with others. While, as we discussed before, SPECIAL’s core design principles are protocol-agnostic. This enables interoperability with various MPC models, including the peer-to-peer settings or a weaker corruption setting (for performance purpose) where a supermajority of servers need to be honest [43, 46, 67].

DP leakages. Leakage-abuse attacks [12, 16, 31, 40, 53, 64, 80], exploit data-dependent processing patterns, are persistent threats to SCA systems. To mitigate these risks, oblivious computation techniques [6, 18, 21, 38, 41, 50, 55, 56, 62, 66, 68, 74, 75] have become the *de facto* solution—processing data such that the entire execution transcript (e.g. memory traces and read/write volumes) is padded to a worst-case constant. While these techniques ensure the strongest privacy guarantees by eliminating data-dependent leakages, they also introduce a fundamental contention with modern database optimizations, which often rely heavily on data-dependent operations [9, 71–73]. To this end, many recent efforts seek a practical balance in the privacy-performance trade-off by allowing controlled leakage under DP [9, 17, 20, 30, 34, 54, 58, 59, 70–73, 79]. However, a common issue of these approaches is unbounded privacy loss. While some works propose to address this [14, 59, 79], their approaches are restricted to only simple linear queries. Furthermore, existing DP leakage designs also lack lossless guarantees for complex query processing. The inherent randomness of DP noises [30, 72, 73] and uncertainty in complex join sensitivities [9, 20, 73] often lead to improper output compaction and lose tuples. SPECIAL addresses all these limitations together, and to our knowledge, is the first system to ensure both bounded privacy and lossless results for securely processing complex queries.

SCA query planning. Query planning [65] is crucial in conventional databases. A few studies [8, 46, 57, 69] explored query planning within SCA frameworks. Unlike conventional planners, which exploit size disparities across different execution plans to choose the most efficient one with minimized plan sizes [25, 32, 33], these methods typically prohibit the use of such data-dependent information. Consequently, they primarily rely on data-independent metrics for planning. For example, [8, 46, 69] optimize based on the MPC execution costs, while [57] focuses on decomposing large MPC tasks into smaller 2/3PC ones. Since these methods do not optimize or compact intermediate sizes, significant query execution overhead from these large sizes remains evident, especially when processing complex multi-joins [9, 46]. Shrinkwrap [9] uses DP leakage models and allows planner to optimize query intermediate sizes in a privacy-preserving manner. However, it cannot estimate cardinalities before runtime, mandating a random selection of plan structures (e.g. join order) before execution. Consequently, planning focuses solely on optimally allocating privacy budgets across the chosen plan, which often result in suboptimal query executions. In contrast, SPECIAL offers an advanced query planner capable of pre-estimating plan sizes and comparing costs among different plan structures before execution.

10 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. The benchmark results demonstrate that SPECIAL significantly outperforms SOTA SCA systems and scales complex multi-join queries to multi-million datasets

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, Krishnamurthy Korthikar, 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 $O(n \log n)$ sorting network. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, 1–9.
- [4] Ghouse Amjad, Sarvar Patel, Giuseppe Persiano, Kevin Yeo, and Moti Yung. 2021. Dynamic volume-hiding encrypted multi-maps with applications to searchable encryption. *Cryptology ePrint Archive* (2021).
- [5] 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.
- [6] 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.
- [7] Kenneth E. Batcher. 1968. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. 307–314.
- [8] 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.
- [9] Johes Bater, Xi He, William Ehrlich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: efficient sql query processing in differentially private data federations. *Proceedings of the VLDB Endowment* 12, 3 (2018).
- [10] Johes Bater, Yongjoo Park, Xi He, Xiao Wang, and Jennie Rogers. 2020. Saq: practical privacy-preserving approximate query processing for data federations. *Proceedings of the VLDB Endowment* 13, 12 (2020), 2691–2705.
- [11] 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.
- [12] Laura Blackstone, Seny Kamara, and Tarik Moataz. 2019. Revisiting leakage abuse attacks. *Cryptology ePrint Archive* (2019).
- [13] Mike W. Blasgen, Morton M. Astrahan, Donald D. Chamberlin, J.N. Gray, W.F. 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.
- [14] Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O’Neill. 2021. Epsolute: Efficiently Erying Databases While Providing Differential Privacy. (2021).
- [15] Mark Bun and Thomas Steinke. 2016. Concentrated differential privacy: Simplifications, extensions, and lower bounds. In *Theory of Cryptography Conference*. Springer, 635–658.
- [16] 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.
- [17] 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.
- [18] 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.
- [19] Guoxing Chen, Ten-Hwang Lai, Michael K. Reiter, and Yinqian Zhang. 2018. Differentially private access patterns for searchable symmetric encryption. In *IEEE INFOCOM 2018-IEEE conference on computer communications*. IEEE, 810–818.
- [20] 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).
- [21] 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.
- [22] 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.
- [23] 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.
- [24] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.* 9, 3-4 (2014), 211–407.
- [25] R. Elmasri, SB Navathe, R. Elmasri, and SB Navathe. 2020. Fundamentals of Database Systems. In *Advances in Databases and Information Systems: 24th European Conference, ADBIS 2020, Lyon, France, August 25–27, 2020, Proceedings*, Vol. 12245. Springer Nature, 139.
- [26] Saba Eskandarian and Matei Zaharia. 2017. Oblidb: Oblivious query processing for secure databases. *arXiv preprint arXiv:1710.00458* (2017).
- [27] Oded Goldreich. 2004. *Foundations of Cryptography, Volume 2*. Cambridge university press Cambridge.
- [28] Oded Goldreich. 2009. *Foundations of cryptography: volume 2, basic applications*. Cambridge university press.
- [29] 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.
- [30] Adam Groce, Peter Rindal, and Mike Rosulek. 2019. Cheaper private set intersection via differentially private leakage. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019).
- [31] 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.
- [32] 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).
- [33] Hazar Harmouch and Felix Naumann. 2017. Cardinality estimation: An experimental survey. *Proceedings of the VLDB Endowment* 11, 4 (2017), 499–512.
- [34] 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.
- [35] 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.
- [36] Axel Hertzschuch, Claudio Hartmann, Dirk Habich, and Wolfgang Lehner. 2021. Simplicity Done Right for Join Ordering. In *CIDR*.
- [37] 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.
- [38] Kristján Valur Jónsson, Gunnar Kreitz, and Misbah Uddin. 2011. Secure multi-party sorting and applications. *Cryptology ePrint Archive* (2011).
- [39] Seny Kamara, Payman Mohassel, and Mariana Raykova. 2011. Outsourcing multi-party computation. *Cryptology ePrint Archive* (2011).
- [40] 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.
- [41] 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, Kaohsiung, Taiwan, ROC, December 7–11, 2014, Proceedings, Part II 20*. Springer, 506–525.
- [42] 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.
- [43] 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.
- [44] 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.
- [45] 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.
- [46] John Liagouris, Vasiliki Kalavri, Muhammad Faisal, and Mayank Varia. 2023. {SECURITY}: Secure collaborative analytics in untrusted clouds. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*. 1031–1056.
- [47] Lovingmage. 2024. Synopsis Assisted Secure Collaborative Analytics. <https://github.com/lovingmage/SPECIAL/>.
- [48] 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.
- [49] Ilya Mironov, Omkant Pandey, Omkar Reingold, and Salil Vadhan. 2009. Computational differential privacy. In *Annual International Cryptology Conference*. Springer, 126–142.
- [50] 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.

- [51] 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.
- [52] Dimitris Mouris, Daniel Masny, Ni Trieu, Shubho Sengupta, Prasad Budhavarapu, and Benjamin Case. 2024. Delegated Private Matching for Compute. *Proceedings on Privacy Enhancing Technologies* (2024).
- [53] 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.
- [54] 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.
- [55] 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.
- [56] 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.
- [57] 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.
- [58] 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).
- [59] Lina Qiu, Georgios Kellaris, Nikos Mamoulis, Kobbi Nissim, and George Kollios. [n.d.]. Doquet: Differentially Oblivious Range and Join Queries with Private Data Structures. ([n. d.]).
- [60] Vijaya Ramachandran and Elaine Shi. 2021. Data oblivious algorithms for multi-cores. In *Proceedings of the 33rd ACM Symposium on Parallelism in Algorithms and Architectures*. 373–384.
- [61] 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.
- [62] 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.
- [63] 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.
- [64] Zhiwei Shang, Simon Oya, Andreas Peter, and Florian Kerschbaum. 2021. Obfuscated access and search patterns in searchable encryption. *arXiv preprint arXiv:2102.09651* (2021).
- [65] Abraham Silberschatz, Henry F Korth, and Shashank Sudarshan. 2011. Database system concepts. (2011).
- [66] Emil Stefanov, Elaine Shi, and Dawn Song. 2011. Towards practical oblivious RAM. *arXiv preprint arXiv:1106.3652* (2011).
- [67] 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.
- [68] 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.
- [69] 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.
- [70] Sameer Wagh, Paul Cuff, and Prateek Mittal. 2016. Differentially private oblivious ram. *arXiv preprint arXiv:1601.03378* (2016).
- [71] 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.
- [72] 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.
- [73] 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).
- [74] Chenghong Wang, David Pujó, Kartik Nayak, and Ashwin Machanavajjhala. 2023. Private Proof-of-Stake Blockchains using Differentially-private Stake Distortion. *Cryptology ePrint Archive* (2023).
- [75] 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.
- [76] 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.
- [77] 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.
- [78] 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.
- [79] 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.
- [80] 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.
- [81] 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.
- [82] 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.

A PRIVACY PROOF

THEOREM A.1. *The one-sided Laplace mechanism as described in Definition 4.1 satisfies (ϵ, δ) -DP.*

PROOF. WLOG. we illustrate proof details by assuming positive mechanism, Lap^+ , and by symmetric, the properties proved here also holds for Lap^- . In general, we write the mechanism as $\mathcal{M}(D) \leftarrow f(D) + \text{Lap}^+(\epsilon, \delta)$, where f is a counting query. Note that the sensitive for counting query is 1, and thus we will implicitly omit the sensitivity notations.

Given any two neighboring databases D and D' , and let $c \leftarrow f(D)$ and $c' \leftarrow f(D')$, WLOG. we will assume $c \leq c'$. Now we will evaluate three disjoint output domains for f , namely $O_1 = (-\infty, c)$, $O_2 = [c, c']$, and $O_3 = (c', +\infty)$. Since the noise is non-negative, thus it's clear that $\Pr[\mathcal{M}(D) \in O_1] = 0$ and $\Pr[\mathcal{M}(D') \in O_1] = 0$. Next, we evaluate the domain O_3 and consider an auxiliary domain $O^* = \{x - c' : x \in O_3\}$. We compute the following ratio:

$$\begin{aligned} \frac{\Pr[\mathcal{M}(D) \in O_3]}{\Pr[\mathcal{M}(D') \in O_3]} &= \frac{\Pr[c + \text{Lap}^+(\epsilon, \delta) \in O_3]}{\Pr[c' + \text{Lap}^+(\epsilon, \delta) \in O_3]} \\ &= \frac{\Pr[\text{Lap}^+(\epsilon, \delta) \in O^*]}{\Pr[\text{Lap}^+(\epsilon, \delta) \in O^* + (c' - c)]} \\ &= \frac{e^{-\epsilon|x-\mu|}}{e^{-\epsilon|x-(c'-c)-\mu|}} \quad (6) \\ &= e^{\epsilon(|x-(c'-c)-\mu|-|x-\mu|)} \\ &\leq e^{\epsilon(|c'-c|)} \quad (\text{triangle inequality}) \\ &= e^\epsilon \end{aligned}$$

Then we evaluate the last output domain O_2 , and we can see that $\Pr[\mathcal{M}(D') \in O_2] = 0$ and $\Pr[\mathcal{M}(D) \in O_2] \geq 0$, and thus there will be unbounded privacy loss within the domain of $O_2 = [c', c]$. However, we can bound this probability by δ , which is equivalent to prove that with density function

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

where $\mu = 1 - \frac{1}{\epsilon} \ln(\delta(e^\epsilon + 1))$, the probability of sampling a value z within the range $(-\infty, 1]$ is less than or equal to δ . Since we consider a discrete distribution, thus we compute the following

$$\begin{aligned} \Pr[z < 1] &= \sum_{x=-\infty}^0 \frac{e^\epsilon - 1}{e^\epsilon + 1} e^{-\epsilon(\mu-x)} \\ &= \frac{e^\epsilon - 1}{e^\epsilon + 1} e^{-\epsilon\mu} \sum_{x=-\infty}^0 e^{\epsilon x} \\ &= \frac{e^\epsilon - 1}{e^\epsilon + 1} e^{-\epsilon\mu} \frac{1}{1 - e^{-\epsilon}} \quad (\text{geometric series sum of } e^{\epsilon x}) \\ &= \frac{e^\epsilon - 1}{e^\epsilon + 1} e^{-\epsilon(1 - \frac{1}{\epsilon} \ln(\delta(e^\epsilon + 1)))} \frac{1}{1 - e^{-\epsilon}} \quad (\text{substitute } \mu) \\ &= \frac{e^\epsilon}{e^\epsilon + 1} e^{-\epsilon + \ln(\delta(e^\epsilon + 1))} \\ &= \frac{e^\epsilon}{e^\epsilon + 1} e^{-\epsilon} \delta(e^\epsilon + 1) \\ &= \delta. \end{aligned} \quad (7)$$

Combining all these we obtain the following

$$\begin{aligned} \Pr[\mathcal{M}(D) \in O] &= \Pr[\mathcal{M}(D) \in (O_1 \cup O_2 \cup O_3)] \\ &= \Pr[\mathcal{M}(D) \in O_2] + \Pr[\mathcal{M}(D) \in O_3] \quad (8) \\ &\leq \delta + e^\epsilon \Pr[\mathcal{M}(D') \in (O_2)] \end{aligned}$$

The claim thus holds. \square

Next we prove Theorem 4.3.

PROOF. (Theorem 4.3) We prove this theorem by analyzing privacy loss using composition theorems. Generating noisy histograms with one-sided noise is equivalent to releasing multiple counting queries across disjoint datasets. Therefore, by the parallel composition theorem [24], each histogram release is (ϵ, δ) -DP. The release of the maximum frequency via a noisy max mechanism is $(\epsilon, 0)$ -DP. Given c pairs, the sequential composition theorem [24] applies, resulting in a total privacy loss under $2c$ -fold advanced composition. Consequently, the entire process adheres to $(\hat{\epsilon}, \hat{\delta})$ -DP, with $\hat{\epsilon} \leq 6\epsilon\sqrt{c \ln(1/\delta)}$. \square

THEOREM A.2. *The privacy of SPECIAL adheres to Definition 3.1*

PROOF. Since the entire DP synopses releasing satisfies DP. Thus, we prove this theorem by constructing a simulator, \mathcal{S} , which takes only the DP synopses as input and can simulate execution transcripts that are indistinguishable from those of the actual protocol. For ease of notation, we abstract SPECIAL as a composed secure protocol π .

We first examine a hybrid protocol, π_1 , differing from π in that the secure outsourcing procedure—where owners upload secret shares of their data to two servers—is replaced by a simulator, \mathcal{S}_1 . This simulator, which takes no input, simulates secret shares with randomly sampled data $(x_1, x_2) \leftarrow \mathbb{Z}_{2^{32}}$. We say that a \mathcal{S}_1 must exist that produces results indistinguishable from the secret shares in π , as the absence of such a simulator would imply the non-existence of secure secret sharing techniques. Next, we consider another hybrid protocol, π_2 , which differs from π_1 only in that the query planning function is replaced by a simulator \mathcal{S}_2 . This simulator takes as inputs all synopses and a set of random values (simulated secure outsourced data) generated by \mathcal{S}_1 , and produces a simulated execution plan for every requested query. We say that an \mathcal{S}_2 should also exist with outputs indistinguishable from π 's execution plans. Given that Selinger optimizers can pre-generate execution plans using only table statistics, the non-existence of such a simulator, \mathcal{S}_2 , would imply the non-existence of Selinger optimizers, which is contradictory. Finally, we construct $\mathcal{S} = (\mathcal{S}_1, \mathcal{S}_2)$, where \mathcal{S}_1 simulates secure data outsourcing and \mathcal{S}_2 simulates query planning. \mathcal{S} executes the query plan using randomly generated values from \mathcal{S}_1 through MPC computation. Since both the execution plans and input data are indistinguishable from those in π , the MPC execution must also be indistinguishable. Otherwise, it would violate the security guarantees of MPC. \square

B QUERY WORKLOADS

We illustrate the benchmark query workloads as follows:

Listing 1: Benchmark query workload

```

-- Q1 Range query
SELECT * FROM financial.loan WHERE financial.loan.duration = 36;

-- Q2 Range query 2
SELECT * FROM financial.order WHERE financial.order.amount > 10000 and
financial.order.amount < 20000 and financial.order.k_symbol = 'LEASING';

-- Q3 Binary join 1 - non-expanding join
SELECT count(distinct b.account_id) FROM financial.client a, financial.disp b
where a.client_id = b.client_id and
a.district_id = 18 and b.type = 'DISPONENT';

-- Q4 Binary join 2 - expanding join (group by)
SELECT a.date, count(a.date) FROM financial.account a, financial.trans b
where a.account_id = b.account_id and
b.operation = 'VYBER_KARTOU' and a.district_id = 18
group by a.date
order by count(a.date);

-- Q5 3 way join, mixed expanding and non-expanding
SELECT count(distinct a.account_id) FROM financial.account a, financial.trans b, financial.order c
where a.account_id = b.account_id and a.account_id = c.account_id and
b.operation = 'VYBER_KARTOU' and a.district_id = 18 and c.k_symbol = 'LEASING';

-- Q6 3 way join, all expanding
SELECT sum(a.amount) FROM financial.order a, financial.trans b, financial.disp c
where a.account_id = b.account_id and b.account_id = c.account_id and
b.operation = 'VYBER_KARTOU' and a.k_symbol = 'LEASING';

-- Q7 4 way join I
SELECT min(c.amount) FROM financial.account a, financial.trans b, financial.order c, financial.disp d
where a.account_id = b.account_id and a.account_id = d.account_id and c.account_id = d.account_id and
b.operation = 'VYBER_KARTOU' and a.district_id = 18 and c.k_symbol = 'LEASING';

-- Q8 5 way join II
SELECT max(c.amount) FROM financial.account a, financial.trans b, financial.order c, financial.disp d, financial.loan e
where a.account_id = b.account_id and b.account_id = c.account_id and c.account_id = d.account_id and
b.operation = 'VYBER_KARTOU' and c.k_symbol = 'LEASING' and a.district_id = 18 and e.duration = 36;

```