Differentially Oblivious Multi-way Join

ZHIANG WU, University of Waterloo, Canada
WEI DONG, Nanyang Technological University, Singapore
XIAO HU*, University of Waterloo, Canada

Processing joins in encrypted database systems presents a fundamental trade-off between privacy and efficiency. Fully oblivious algorithms can offer perfect access-pattern privacy but incur prohibitive costs by padding the execution to the worst-case output size. This limitation can be further enlarged on multi-way joins, where the worst-case output size can be exponentially large in terms of the database size. To overcome this barrier, differentially oblivious (DO) algorithms are introduced to enable instance-specific efficiency by sacrificing the perfect access-pattern privacy. While promising for two-way joins, extending this paradigm to multi-way joins has remained a significant open challenge.

In this paper, we establish that designing efficient DO multi-way join algorithms is fundamentally equivalent to the problem of releasing join size differentially privately, but under new constraints imposed by an oblivious execution model. To solve this, we introduce relaxed-residual sensitivity, a novel sensitivity measure for counting the join size that is both differentially private and efficiently computable within an oblivious context. Based on this measure, we develop a principled DO padding mechanism that minimizes overhead while rigorously satisfying privacy. Our DO multi-way join algorithms achieve a polynomial speedup over their fully oblivious counterparts and come with a theoretical optimality guarantee for a large class of join queries. We have implemented our algorithms, and empirical evaluations confirm their substantial performance advantages, making differentially oblivious multi-way joins closer to a practical solution for secure query processing.

CCS Concepts: • **Security and privacy → Management and querying of encrypted data**.

Additional Key Words and Phrases: Multi-way join, Oblivious query processing, Differential obliviousness

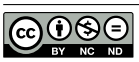
ACM Reference Format:
Zhiang Wu, Wei Dong, and Xiao Hu. 2026. Differentially Oblivious Multi-way Join. *Proc. ACM Manag. Data* 4, 2 (SIGMOD), Article 162 (June 2026), 26 pages. https://doi.org/10.1145/3802039

1 Introduction

Outsourced database systems, exemplified by commercial services such as Amazon Redshift [1], Google Cloud [4], and Microsoft Azure [5], allow clients to leverage the immense scalability of the cloud by storing their data on remote servers. To ensure data confidentiality, a standard practice is for clients to encrypt their data before outsourcing it. During query evaluation, memory access operations are delegated to the cloud, and only the resulting encrypted data is returned to the client for decryption and computation. However, this approach harbours a critical vulnerability: the server's observation of the memory access pattern[1] — the sequence of physical memory addresses accessed during query execution — constitutes a powerful side-channel. A series of attacks against

*Xiao Hu is the corresponding author.
[1]This is different from the *query access pattern* studied in [30, 56, 67], which refers to the attributes whose values are specified as constants when users access the database.

Authors' Contact Information: Zhiang Wu, z46wu@uwaterloo.ca, University of Waterloo, Waterloo, Ontario, Canada; Wei Dong, wei_dong@ntu.edu.sg, Nanyang Technological University, Singapore; Xiao Hu, xiaohu@uwaterloo.ca, University of Waterloo, Waterloo, Ontario, Canada.



ACM 2836-6573/2026/6-ART162
https://doi.org/10.1145/3802039

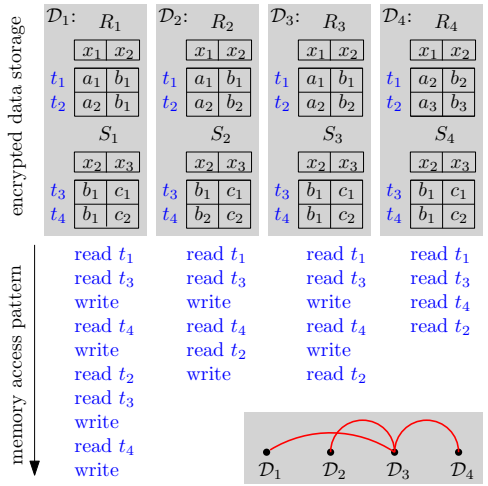


Fig. 1. An example of computing the two-way join on four different databases. The top illustrates the encrypted data in memory. The middle illustrates the memory access patterns of sort-merge join algorithms. The bottom illustrates the neighboring relationships among them.

prominent encrypted database systems [9, 16, 52] has shown that an adversary can exploit this leakage to infer sensitive information, reconstruct database contents, or recover entire queries. This threat is not merely theoretical; it undermines the core security premise of encrypted databases by revealing information even when the data itself remains encrypted.

Example 1.1. Consider computing the two-way join $R(x_1, x_2) \bowtie S(x_2, x_3)$ on four different databases $\mathcal{D}_1 = (R_1, S_1)$, $\mathcal{D}_2 = (R_2, S_2)$, $\mathcal{D}_3 = (R_3, S_3)$, and $\mathcal{D}_4 = (R_4, S_4)$, as shown in Figure 1. Each database contains 4 tuples, with 2 tuples in each relation. For simplicity, the physical positions of these four encrypted tuples in memory storage are labelled as t_1, t_2, t_3, t_4, respectively. Now, assume that the classic sort-merge join (SMJ) algorithm is applied to compute the join results. As shown, tuples in each table are already sorted by their join values on attribute B. It remains to merge these sorted tuples. On $\mathcal{D}_1 = (R_1, S_1)$, SMJ starts by reading the first tuple at t_1 from R_1 as well as the first tuple at t_3 from R_2 into the secure registers, in which the adversary cannot observe the computation. It then decrypts the data read, produces the join result (a_1, b_1, c_1), and writes the encrypted join result back to some location in memory. Next, it moves to the next tuple at t_4 in R_2 and reads it into the secure registers. Similarly, it writes the encrypted join result back to some location in memory. After reaching the end of R_2, it moves to the next tuple at t_2 and repeats the execution. The complete sequence of memory accesses by the SMJ algorithm on $\mathcal{D}_1$ is illustrated. Such a sequence of memory accesses made by the same SMJ algorithm during execution may differ on $\mathcal{D}_2$, $\mathcal{D}_3$, and $\mathcal{D}_4$. Hence, even if all data are encrypted, the adversary can still distinguish input databases solely by observing the sequence of memory addresses accessed by the SMJ algorithm.

To mitigate side-channel attacks based on memory access patterns, the gold standard is *fully oblivious* (FO) algorithms [32, 33]. An FO algorithm guarantees that, for any two databases containing the same number of tuples, their memory access patterns are indistinguishable. As a result, an adversary cannot distinguish one database from another by observing an FO algorithm's memory access pattern. From Example 1.1, the SMJ algorithm is not FO. However, FO join algorithms become inherently inefficient due to *the curse of the worst case*. Extending Example 1.1, any FO algorithm for computing two-way joins on databases that contain N tuples needs to access at least $\Omega(N^2)$ memory addresses, since there exists a worst-case database (as a generalization of $\mathcal{D}_1$) that can

produce as many as $\Omega(N^2)$ join results, and any FO algorithm needs to write its $\Omega(N^2)$ join results back to memory. This limitation can be further exacerbated when more relations participate in the join query. As shown by the AGM bound [13], for a multi-way join query, a worst-case database of N tuples can produce as many as $\Omega(N^\rho)$ join results, where ρ is the fractional edge covering number of the join. Consequently, any FO algorithm needs to incur $\Omega(N^\rho)$ memory accesses, even if the actual join result set is small or empty, rendering it impractical for real-world use.

This efficiency barrier has motivated the exploration of *differential oblivious* (DO) [20] algorithms, which offer a more pragmatic trade-off between privacy and efficiency. Drawing inspiration from differential privacy [27], a DO algorithm relaxes the security requirement: it only guarantees that for any pair of *neighboring* databases that differ in one tuple, their memory access patterns do not leak information about an individual tuple. Specifically, DO can protect against membership inference attacks on access patterns of join processing, i.e., the existence of any particular tuple is difficult to infer. Due to the relaxation of the privacy guarantee, there has been a significant speedup in the efficiency of DO algorithms over FO algorithms. For computing two-way joins over databases with N tuples, the proposed DO join algorithm [23] only incurs $O(N \log N + \mathrm{OUT})$ memory accesses, where OUT is the number of join results produced by the database. Extending Example 1.1, a one-to-one matching database (as a generalization of $\mathcal{D}_2$) only produces N join results; hence, the DO algorithm only incurs $O(N \log N)$ memory accesses, improving over the FO algorithm by a factor of $O(N)$. Note that databases containing the same number of tuples can produce dramatically different numbers of join results, varying from 1 to N^2 for the two-way join. As the value of OUT is usually much smaller (and always no larger) than the worst-case join size N^2, DO algorithms significantly speedup FO algorithms by achieving instance-dependent efficiency.

Example 1.2. Continue with the example in Figure 1. A FO algorithm should have indistinguishable memory access patterns on all of $\mathcal{D}_1, \mathcal{D}_2, \mathcal{D}_3, \mathcal{D}_4$. If one attempts to make the SMJ algorithm become FO, it at least needs to pad dummy memory accesses to $\mathcal{D}_2$, $\mathcal{D}_3$, $\mathcal{D}_4$ to match the worst-case instance $\mathcal{D}_1$, regarding the different memory addresses accessed during the execution.

In contrast, a DO algorithm does not need to have indistinguishable memory access patterns on all databases, but only neighboring databases. As shown in Figure 1, there are only three pairs of neighboring databases: $(\mathcal{D}_1, \mathcal{D}_3)$, $(\mathcal{D}_2, \mathcal{D}_3)$ and $(\mathcal{D}_3, \mathcal{D}_4)$. Hence, a DO algorithm has much better efficiency on both $\mathcal{D}_2$ and $\mathcal{D}_4$, as it does not need to pad dummy memory accesses to match $\mathcal{D}_1$.

Despite its promise, extending the DO two-way join algorithm to general multi-way joins has remained a significant challenge. The natural approach of decomposing a multi-way join into a sequence of two-way joins fails due to a fundamental lack of *composability*. While existing DO two-way join primitives provide privacy for neighboring inputs, the intermediate results they produce are not guaranteed to be neighboring. A single-tuple change in an input relation can trigger a polynomially large change in the size of an intermediate join result. When this result is fed into the next join operator in the sequence, the neighboring precondition is violated, and the privacy guarantee is lost. This composition barrier means that designing a DO multi-way join algorithm requires a completely new approach. In this paper, we take on this challenging question and achieve the following results:

- We present a general framework that transforms the DO join problem into the existing problem of differentially private (DP) counting the number of join results, under new privacy and efficiency constraints. We then examine existing sensitivity-based approaches for this DP problem and incorporate them into our general framework as baseline DO algorithms. However, these baseline solutions are not satisfactory due to their inefficiency in the DO setting. **(Section 3)**
- We propose the notion of *relaxed-residual sensitivity* (RRS), and construct our DO join algorithm based on RRS. Compared to existing sensitivities, RRS sacrifices the utility (defined in Section 3.3)

but saves the computational cost, which turns out to be the overall winner in the DO setting. Within a wide class of queries, defined as *residual-linear* joins, we theoretically prove that our RRS-based DO algorithm is optimal. **(Sections 4 and 5)**
- We prototype our RRS-based DO join algorithm and demonstrate its advantage over baseline solutions in constructing multi-way DO join algorithms on benchmark datasets. **(Section 6)**

2 Preliminaries

2.1 Multi-way Joins

Let $[m]$ denote the set $\{1, \cdots, m\}$. A multi-way (natural) join query is $Q = (\mathbf{x}, \{\mathbf{x}_i : i \in [m]\})$, where $\mathbf{x} = \{x_1, \cdots, x_n\}$ is the set of attributes, and each subset of attributes $\mathbf{x}_i \subseteq \mathbf{x}$ is the schema of a relation. Let $\mathrm{dom}(x)$ be the domain of x. An *instance* of Q is defined as a set of relations $\mathbf{I} = \{R_1^{\mathbf{I}}, \cdots, R_m^{\mathbf{I}}\}$. Each relation $R_i^{\mathbf{I}}$ consists of a set of *tuples*, where each tuple $t \in R_i^{\mathbf{I}}$ assigns a value from $\mathrm{dom}(x)$ to x for each $x \in \mathbf{x}_i$. The join result of Q over instance $\mathbf{I}$ is

$$Q(\mathbf{I}) = \left\{ t \in \prod_{x \in \mathbf{x}} \mathrm{dom}(x) \mid \forall i \in [m], \exists t_i \in R_i^{\mathbf{I}} : \pi_{\mathbf{x}_i} t = t_i \right\}.$$

i.e., all combinations of tuples, each from one relation, such that they share the same value(s) on the common attribute(s). In some contexts, we interchangeably write $Q(\mathbf{I}) = R_1^{\mathbf{I}} \bowtie \cdots \bowtie R_m^{\mathbf{I}}$.

Let $|\mathbf{I}| = \sum_{i \in [m]} |R_i^{\mathbf{I}}|$ be the *input size* of $\mathbf{I}$, i.e., the total number of tuples in all relation from $\mathbf{I}$, and $\mathrm{OUT} = |Q(\mathbf{I})|$ be the *join size*. All relations are considered private. However, this constraint can be easily relaxed to accommodate a mix of public and private relations, following the standard approach in [25, 39]. We study the data complexity [7] of join algorithms and assume the query size (such as n, m) as constant.

We use the Hamming metric to measure the distance between instances or relations. The distance between two instances $\mathbf{I}$ and $\mathbf{I}'$, denoted as $d(\mathbf{I}, \mathbf{I}')$, is the minimum number of substitutions of tuples required to transform $\mathbf{I}$ into $\mathbf{I}'$. For $i \in [m]$, let $d_i(\mathbf{I}, \mathbf{I}')$ denote the distance between $R_i^{\mathbf{I}}$ and $R_i^{\mathbf{I}'}$. For a subset $E \subseteq [m]$ of relations, let $d_E(\mathbf{I}, \mathbf{I}') = \prod_{i \in E} d_i(\mathbf{I}, \mathbf{I}')$ be the distance between $\mathbf{I}$ and $\mathbf{I}'$ with respect to E. Moreover, we abuse the expression $|\mathbf{I}| = |\mathbf{I}'|$ to indicate $|R_i^{\mathbf{I}}| = |R_i^{\mathbf{I}'}|$ for each $i \in [m]$. The superscript $\mathbf{I}, \mathbf{I}'$ is omitted when the context is clear.

We define neighboring instances without considering foreign-key (FK) constraints, consistent with prior studies [25, 31, 39, 53]: two instances $\mathbf{I}$ and $\mathbf{I}'$ are neighboring if $d(\mathbf{I}, \mathbf{I}') = 1$. Alternatively, this concept can be formulated in a FK-aware way [24, 42, 61]: two instances $\mathbf{I}$ and $\mathbf{I}'$ are neighboring if $\mathbf{I}$ can be obtained from $\mathbf{I}'$ by deleting a tuple t from a *primary private relation* and all tuples in other relations that join with t via FK constraints. These two definitions yield incomparable privacy guarantees. The FK-aware definition offers stronger, entity-level protection but typically supports only a single primary private relation. In contrast, our definition provides relationship-level guarantees applicable to any number of relations. The choice depends on the application's privacy requirements: if specific relationships (e.g., friendships or seller-buyer links) are sensitive, our definition is preferable; if entire entities — including all dependent information — must be protected, the FK-aware definition is more appropriate.

2.2 Formulation of Obliviousness

We adopt the standard word-RAM model that consists of a *trusted memory* of $O(1)$ words[2] and an *untrusted storage* of infinitely many words [23, 43]. Each word can store one tuple from an arbitrary

[2]When trusted memory becomes substantially larger, such as in advanced Intel chips supporting up to 512 GB [22, 28], the challenge of hiding access patterns shifts to hiding page-swap patterns, analogous to the difference between in-memory and external-memory algorithms. Additionally, for *doubly oblivious* algorithms [48, 68] that require access patterns within

relation. Algorithms under this model proceed in three alternating steps: (1) read encrypted data from untrusted storage into trusted memory, (2) decrypt data and compute (intermediate) results within trusted memory, and (3) encrypt and write the results back to untrusted storage. One memory access occurring in step (1) or (3) is represented as $\langle \mathrm{op}, \mathrm{ad} \rangle$, where $\mathrm{op} \in \{\mathrm{read}, \mathrm{write}\}$ is the operation and ad denotes the memory address. Each memory access takes one unit of time. As such, the runtime of an algorithm can be measured by the total number of memory accesses it produce[3]. We consider a *semi-honest* adversary that can observe the memory accesses made by the algorithms, but cannot observe or interfere with the contents of the data or the computation within the trusted memory, as in [20, 23, 43, 55].

Definition 2.1 (Access Pattern [32, 33]). For a query Q, the *access pattern* of a join algorithm $\mathcal{A}$ over an instance $\mathbf{I}$ of Q is the sequence $\mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I}) = \left(\langle \mathrm{op}_i^{\mathbf{I}}, \mathrm{ad}_i^{\mathbf{I}} \rangle\right)_{i=1}^{\ell}$ produced by $\mathcal{A}$, where $\ell = |\mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I})|$ is the number of memory accesses.

For a randomized algorithm, its access pattern is the distribution over the set of all possible sequences generated by the algorithm on the input instance.

Definition 2.2 (FO [32, 33]). For a query Q, a join algorithm $\mathcal{A}$ is *fully oblivious* if for any two instances $\mathbf{I}$ and $\mathbf{I}'$ with $|\mathbf{I}| = |\mathbf{I}'|$,

$$\mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I}) = \mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I}')$$

i.e., $|\mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I})| = |\mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I}')|$, and for every possible i, $\langle \mathrm{op}_i^{\mathbf{I}}, \mathrm{ad}_i^{\mathbf{I}} \rangle = \langle \mathrm{op}_i^{\mathbf{I}'}, \mathrm{ad}_i^{\mathbf{I}'} \rangle$.

Definition 2.3 (DO [20]). For a query Q, a join algorithm $\mathcal{A}$ is (ϵ, δ)*-differentially oblivious* if for any pair of neighboring instances $\mathbf{I}$ and $\mathbf{I}'$, and any subset of memory access patterns $\mathcal{Y}$,

$$\Pr[\mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I}) \in \mathcal{Y}] \leq e^{\epsilon} \cdot \Pr[\mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I}') \in \mathcal{Y}] + \delta.$$

Definition 2.4 (DP [27]). An algorithm $\mathcal{A}$ is (ϵ, δ)-differentially private if for any pair of neighboring instances $\mathbf{I}$ and $\mathbf{I}'$, and any subset of possible outputs $\mathcal{Y}$:

$$\Pr[\mathcal{A}(\mathbf{I}) \in \mathcal{Y}] \leq e^{\epsilon} \cdot \Pr[\mathcal{A}(\mathbf{I}') \in \mathcal{Y}] + \delta.$$

The notion of DO is analogous to DP, except that the adversary can only observe the algorithm's memory access pattern rather than the algorithm's output. It is not hard to see that DO weakens the privacy guarantee of FO from two perspectives: (1) it only provides guarantees for neighbouring instances with the same input size, instead of all instances with the same input size, and (2) it only provides (ϵ, δ)-indistinguishability guarantees on neighboring instances. Moreover, FO can be interpreted as a special case of DO when $\epsilon = \delta = 0$, since in this case the DO guarantee can be extended to all databases containing the same number of tuples via the neighboring relationship.

Leakage of FO and DO Algorithms. We also compare FO and DO algorithms in terms of their leakage functions. The definition via simulation-based security is formally presented in [20]. Intuitively, an algorithm $\mathcal{A}$ is *oblivious with respect to the leakage function* $\mathsf{Leak}_{\mathcal{A}}(\cdot)$, if its access pattern on instance $\mathbf{I}$ only depends on $\mathsf{Leak}_{\mathcal{A}}(\mathbf{I})$. Then, an FO algorithm has its leakage function defined as $\mathsf{Leak}_{\mathcal{A}}(\mathbf{I}) = |\mathbf{I}|$ for instance $\mathbf{I}$, i.e., the input size of $\mathbf{I}$ [21, 23].

As established in [20], if an algorithm $\mathcal{A}$ is oblivious with respect to the leakage function $\mathsf{Leak}_{\mathcal{A}}(\cdot)$, and moreover, $\mathsf{Leak}_{\mathcal{A}}(\cdot)$ is (ϵ, δ)-DP, then $\mathcal{A}$ is (ϵ, δ)-DO.

---

the trusted memory to be oblivious, limiting the trusted memory to $O(1)$ words significantly simplifies the design and analysis of such algorithms.

[3]The ability of computing over encrypted data does not ease the difficulty of protecting access patterns. We model the encryption and decryption schemes as free-variable parameters and do not pursue this direction further.

As an example, in the DO two-way join algorithm [23], $\mathsf{Leak}_{\mathcal{A}}(\mathbf{I})$ comprises the input size $|\mathbf{I}|$, a noisy join size, and a list of noisy join key frequencies, which satisfies (ϵ, δ)-DP. As another example, in our DO multi-way join algorithm (presented in Section 5), $\mathsf{Leak}_{\mathcal{A}}(\mathbf{I})$ consists of the input size $|\mathbf{I}|$ and a (ϵ, δ)-DP upper bound on the join size. As a counter example, in some instantiation of the algorithms proposed by [10, 43, 46], $\mathsf{Leak}_{\mathcal{A}}(\mathbf{I})$ consists of the input size $|\mathbf{I}|$ and the true join size, violating the (ϵ, δ)-DP, hence these instantiations are not (ϵ, δ)-DO.

3 Framework of DO Join

This section presents a general framework that connects DO join to a classic DP problem: *given a join query and a database instance, how can the join size be released under DP?* We show how the additional constraints imposed by DO are injected into this DP problem and decompose the design of DO join algorithms into two subproblems.

3.1 Framework

Our starting point is insecure output-sensitive join algorithms. For a multi-way join Q and an instance $\mathbf{I}$, the join result $Q(\mathbf{I})$ can be computed in $O(N^w + |Q(\mathbf{I})|)$ time, where w is a width measurement of Q [35, 49, 65]. The cost (in terms of the total number of memory accesses) can be abstracted as a function of the input size and the join size.

As neighboring instances could have dramatically different join size, the leakage of true join size $|Q(\mathbf{I})|$ from the total number of memory accesses will violate the DO constraint. One may wonder whether we can protect one obvious leakage of $|Q(\mathbf{I})|$ by computing a DP upper bound $\widetilde{\mathrm{OUT}}$ on $|Q(\mathbf{I})|$, and then padding dummy accesses to match $\widetilde{\mathrm{OUT}}$. However, this could lead to new possible leakage from the computation of $\widetilde{\mathrm{OUT}}$, which now is one part of the whole execution. Moreover, the actual memory accesses in terms of read and write operations made during the execution should also satisfy the DO constraint.

To mitigate these possible leakages, we decompose the original DO join problem into the following two subproblems:

- **(Problem 1)** For a join query Q and instance $\mathbf{I}$, how to design a DO algorithm that computes a DP upper bound $\widetilde{\mathrm{OUT}}$ on $|Q(\mathbf{I})|$?
- **(Problem 2)** For a join query Q, instance $\mathbf{I}$ and a DP upper bound $\widetilde{\mathrm{OUT}}$ on $|Q(\mathbf{I})|$, how to design an algorithm that computes the join result $Q(\mathbf{I})$ correctly while leaks at most $|\mathbf{I}|$ and $\widetilde{\mathrm{OUT}}$?

As a heads-up, $\widetilde{\mathrm{OUT}}$ will be used as *advice* in padding dummy memory accesses to avoid leaking $|Q(\mathbf{I})|$ directly. We require any advice to be an upper bound on $|Q(\mathbf{I})|$, so that the join evaluation can finish correctly. We denote the algorithmic solutions to **Problem 2** as *advised FO* algorithms. Let $\mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I}, \Lambda)$ be the *access pattern* of such an algorithm $\mathcal{A}$, taking a join query Q, an instance $\mathbf{I}$, and an advice parameter $\Lambda \geq |Q(\mathbf{I})|$ as input.

Definition 3.1 (Advised FO). For a join query Q, an algorithm $\mathcal{A}$ is *advised FO* if for any two instances $\mathbf{I}$ and $\mathbf{I}'$ of same input size, for any input advice $\Lambda \geq \max\{|Q(\mathbf{I})|, |Q(\mathbf{I}')|\}$,

$$\mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I}, \Lambda) = \mathsf{Access}_{\mathcal{A}}(Q, \mathbf{I}', \Lambda)$$

If $\mathcal{A}$ taking as input an instance $\mathbf{I}$ and an advice Λ, then $\mathsf{Leak}_{\mathcal{A}}(\mathbf{I})$ comprises the input size $|\mathbf{I}|$ and the advice Λ. This is different from a FO algorithm $\mathcal{A}'$, where $\mathsf{Leak}_{\mathcal{A}'}(\mathbf{I})$ comprises only the input size $|\mathbf{I}|$. Putting everything together, we come to the meta algorithm:

Theorem 3.2. *Algorithm 1 is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-DO.*

The formal proof of Theorem 3.2 is deferred to the full version [3]. Intuitively, the output size of (ϵ_1, δ_1)-DO algorithm is fixed, since it exclusively computes a single value $\widetilde{\mathrm{OUT}}$ across any instance.

Algorithm 1: MetaDOJoin($Q, \mathbf{I}$)

1 $\widetilde{\text{OUT}} \leftarrow$ an (ϵ_2, δ_2)-DP upper bound on the join size $|Q(\mathbf{I})|$ via an (ϵ_1, δ_1)-DO algorithm;
2 Invoke an advised FO algorithm with advice $\widetilde{\text{OUT}}$ to compute the join results $Q(\mathbf{I})$;

The $\widetilde{\text{OUT}}$-advised FO algorithm is (ϵ_2, δ_2)-DO, as $\widetilde{\text{OUT}}$ is (ϵ_2, δ_2)-DP. A DO composition theorem can be developed specifically for the above scenario so that Algorithm 1 is $(\epsilon_1 + \epsilon_2, \delta_1 + \delta_2)$-DO.

There can be many solutions to each subproblem, and any combination of them forms a valid DO join algorithm, thanks to the privacy guarantee from Theorem 3.2. Each problem itself is an independently interesting question, and we do not pursue all possibilities in this work. We make the following assumption in building the first DO multi-way join algorithm: $\epsilon_1 = \delta_1 = 0$. In other words, we seek FO algorithms for **Problem 1**, and leave all privacy budgets to the (ϵ_2, δ_2)-DP upper bound $\widetilde{\text{OUT}}$ of the true join size $|Q(\mathbf{I})|$.

Below, we first examine **Problem 2** with some solutions from the literature. Our main focus is on **Problem 1**. Due to the intricacy of this problem, we first review background knowledge and investigate the limitations of existing approaches in Section 3.3.

3.2 Solutions for Problem 2

We review two lines of work for advised FO join algorithms.

Oblivious RAM (ORAM)-based Advised FO Algorithms [21, 32, 33]. ORAM is a generic framework that sanitizes the memory access patterns of insecure algorithms via obfuscation, translating each (logical) memory access into a polylogarithmic number of random physical memory accesses. Hence, we can take an insecure output-sensitive join algorithm $\mathcal{A}$ and turn it into an advised FO algorithm as follows.

Suppose Λ is the input advice, which should be an upper bound of the true join size. We first incorporate $\mathcal{A}$ into the ORAM framework to compute the join result $Q(\mathbf{I})$, and then pad dummy memory accesses until reaching Λ logical accesses in total. For insecure join algorithms, we can choose among the textbook sorting-based and the index-based ones [59]. For the ORAM framework, we can use the constructions in [11, 12].

Non-ORAM-based Advised FO Algorithms [10, 43, 46] Another line of work transforms insecure output-sensitive join algorithms into advised FO counterparts using efficient oblivious primitives. These algorithms take an upper bound on the join size as advice Λ. Their access patterns depend only on the input size and the advice Λ.

The runtime of both approaches is linear (up to a logarithmic factor) in that of the insecure output-sensitive join algorithms and the value of Λ. In some instantiations, one can first compute the true join size using an FO algorithm and then feed the true join size (as the advice) into advised FO algorithms. Such instantiations leak both the input size and the join size during execution. Moreover, if the worst-case join size [13] is fed into advised FO algorithms, such an instantiation becomes FO.

In our implementation, we use the non-ORAM-based advised FO algorithm [10, 43, 46] and adopt a DP upper bound of the true join size as the advice Λ for **Problem 2**.

3.3 How Problem 1 Matters for DO Join

The vanilla version of **Problem 1** without the DO constraints has been studied in the DP setting. The most common approach is based on *sensitivity*, a quantity measuring how much the join size changes across neighboring databases. [23, 40, 50, 64] demonstrate that drawing random noise from

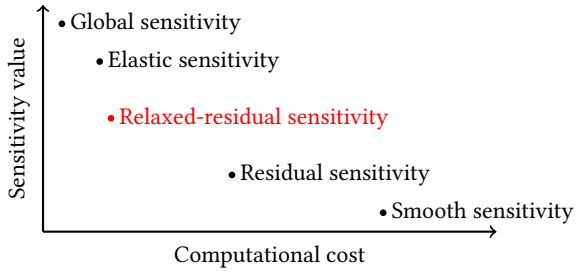


Fig. 2. Comparison of different sensitivities in terms of sensitivity value and computational cost.

a certain distribution calibrated by the sensitivity and masking the join size by the noise yields a valid DP upper bound. In the DP setting, sensitivity is evaluated by two independent criteria: (1) *utility*: since the noise to be added is proportional to the sensitivity, a higher sensitivity corresponds to a larger $\widetilde{\mathrm{OUT}}$ and thus lower utility; (2) *computational cost*: the time of computing the sensitivity. However, the utility and computational cost of a sensitivity jointly affect the runtime of DO join algorithms. Since $\widetilde{\mathrm{OUT}}$ is used to guide the number of dummy accesses padded in **Problem 2** and the runtime of **Problem 2** is proportional to $\widetilde{\mathrm{OUT}}$. Hence, instead of separating the utility and computation of a sensitivity, we now study them together through the lens of DO.

Below, we review the preliminaries of constructing sensitivities for counting the join size, and investigate how existing sensitivities introduced in [25, 39, 50] behave in the DO setting. Figure 2 presents a conceptual comparison of different sensitivities in terms of sensitivity value and computational cost.

Local sensitivity. *Local sensitivity* of an instance $\mathbf{I}$ is $\mathsf{LS}(\mathbf{I}) = \max_{\mathbf{I}':d(\mathbf{I},\mathbf{I}')=1} ||Q(\mathbf{I})| - |Q(\mathbf{I}')||$, capturing the maximum difference in the join size caused by changing one tuple in $\mathbf{I}$. It has been shown that calibrating noise with $\mathsf{LS}(\mathbf{I})$ is not DP [50].

Global sensitivity. *Global sensitivity* GS is the maximum local sensitivity over all instances of the same size. This quantity is independent of the input instance and hence can be used for noise calibration. However, GS is prohibitively large and damages its utility for releasing join sizes. To address these issues, the *β-smooth upper bound on the local sensitivity* [50] is introduced as an instance-specific upper bound of local sensitivity, where the parameter β depends on ϵ and δ. Detailed configuration of β is deferred to Section 5.3. We treat $\beta = O(1)$ for now.

Definition 3.3 (Smooth Upper Bound on the Local Sensitivity [50]). For $\beta > 0$, $\mathsf{S}(\cdot)$ is a β-smooth upper bound on local sensitivity if (**upper bound**) for any instance $\mathbf{I}$, $\mathsf{S}(\mathbf{I}) \geq \mathsf{LS}(\mathbf{I})$, and (**smoothness**) for any neighboring instances $\mathbf{I}$ and $\mathbf{I}'$, $\mathsf{S}(\mathbf{I}) \leq e^{\beta} \cdot \mathsf{S}(\mathbf{I}')$.

Smooth sensitivity. *Smooth sensitivity* SS is the smallest smooth upper bound on the local sensitivity, which is built on *k-distance local sensitivity* $\mathsf{LS}_k(\mathbf{I}) = \max_{\mathbf{I}':d(\mathbf{I},\mathbf{I}')=k} \mathsf{LS}(\mathbf{I}')$, i.e., the maximum difference in the join size between two instances at distance k [50]. Specifically, $\mathsf{SS}(\mathbf{I}) = \max_{k\geq 0} e^{-\beta k} \cdot \mathsf{LS}_k(\mathbf{I})$. However, computing SS is rather expensive. The best insecure algorithm incurs $N^{O(\log N)}$ time [25], since it enumerates all instances within distance $O(\log N)$ to maximize $e^{-\beta k} \cdot \mathsf{LS}_k(\mathbf{I})$. This bottleneck motivates the study of computationally simple upper bounds on $\mathsf{LS}_k(\mathbf{I})$ and leads to two independent constructions: *residual sensitivity* [25] and *elastic sensitivity* [39]. Their constructions are presented below to serve as the stage for our proposed new sensitivity.

Residual sensitivity. *Residual sensitivity* RS is built on *maximum boundaries* over $\mathbf{I}$ with respect to *subjoins*. Consider a join query $Q = (\mathbf{x}, \{\mathbf{x}_i : i \in [m]\})$. Given a subset of relations $E \subseteq [m]$, let $Q_E = (\cup_{i\in E}\mathbf{x}_i, \{\mathbf{x}_i : i \in E\})$ be the subjoin induced by E. The *boundary attributes* ∂E are the

Notations	Meaning
Q	Join query
$\mathbf{x}$	All attributes of Q
$\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_m$	Attributes in each relation of Q
$\mathbf{I}$	Join instance
$Q(\mathbf{I}), \lvert Q(\mathbf{I})\rvert$	Join result and its size
$\widetilde{\mathrm{OUT}}$	A DP upper bound of true join size
E	A subset of relations
∂E	Boundary attributes of relations in E
Q_E	Subjoin induced by relations in E
T_E	Maximum boundary of E
$\mathsf{RST}_E, \mathsf{EST}_E$	Two candidate upper bounds of T_E
$\hat{\mathsf{T}}_E$	Upper bound of T_E in hybrid construction
$\mathbf{y}$	Boundary attributes (to be dropped)
$\mathsf{LS}_k(\cdot), \hat{\mathsf{LS}}_k(\cdot)$	k-distance local sensitivity and its upper bound
$\mathsf{S}(\cdot)$	Smooth upper bound on local sensitivity
$\mathsf{ES}(\cdot)$	Elastic sensitivity
$\mathsf{RS}(\cdot)$	Residual sensitivity
$\mathsf{RRS}(\cdot)$	Relaxed residual sensitivity

Table 1. Main notations used in this paper.

attributes that belong to relations both inside and outside E, i.e., $\partial E = \{x : x \in \mathbf{x}_i \cap \mathbf{x}_j, i \in E, j \notin E\}$. The maximum boundary of E over $\mathbf{I}$ is defined as

$$\mathsf{T}_E(\mathbf{I}) = \max_{t \in \mathrm{dom}(\partial E)} |Q_E(\mathbf{I}) \ltimes t|. \tag{1}$$

See Figure 3 for examples. Intuitively, if changing one tuple in any relation from $[m] - E$, at most $\mathsf{T}_E(\mathbf{I})$ tuples in $Q(\mathbf{I})$ will be affected. The structural properties and computational cost of maximum boundary queries will be discussed in Section 4.1.

[25] further proved in Lemma 3.5 that the maximum boundary (or any smooth upper bound on the maximum boundary defined below) can be used to construct an upper bound of $\mathsf{LS}_k(\mathbf{I})$, and therefore a smooth upper bound on the local sensitivity.

Definition 3.4 (Smooth Upper Bound on Maximum Boundary [25]). $\hat{\mathsf{T}}(\cdot)$ is a smooth upper bound on the maximum boundary $\mathsf{T}(\cdot)$ if

- **(upper bound)** for any instance $\mathbf{I}$, and any subset of relations $E \subseteq [m]$, $\hat{\mathsf{T}}_E(\mathbf{I}) \geq \mathsf{T}_E(\mathbf{I})$;
- **(smoothness)** for any subset of relations $E \subseteq [m]$, any $i \in E$, and any neighboring instances $\mathbf{I}$ and $\mathbf{I}'$ where $d(R_i^{\mathbf{I}}, R_i^{\mathbf{I}'}) = 1$ and $R_j^{\mathbf{I}} = R_j^{\mathbf{I}'}$ for any $j \in E - \{i\}$, $\left|\hat{\mathsf{T}}_E(\mathbf{I}) - \hat{\mathsf{T}}_E(\mathbf{I}')\right| \leq \hat{\mathsf{T}}_{E-\{i\}}(\mathbf{I})$.

As its name suggests, $\hat{\mathsf{T}}_E(\cdot)$ is always an upper bound on $\mathsf{T}_E(\cdot)$, and does not vary significantly over neighboring instances.

Lemma 3.5 ([25]). *Let $\hat{\mathsf{T}}(\cdot)$ be a smooth upper bound on the maximum boundary. Then,*

$$\hat{\mathsf{LS}}_k(\mathbf{I}) = \max_{\mathbf{I}': d(\mathbf{I},\mathbf{I}')=k} \max_{i \in [m]} \sum_{E \subseteq [m]-\{i\}} \hat{\mathsf{T}}_{[m]-\{i\}-E}(\mathbf{I}) \cdot d_E(\mathbf{I}, \mathbf{I}') \tag{2}$$

is an upper bound on $\mathsf{LS}_k(\mathbf{I})$, and

$$\mathsf{S}(\mathbf{I}) = \max_{k \geq 0} e^{-\beta k} \cdot \hat{\mathsf{LS}}_k(\mathbf{I}) \tag{3}$$

is a smooth upper bound on local sensitivity.

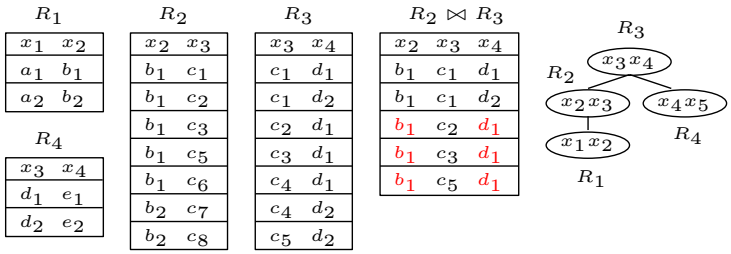


Fig. 3. An instance of line-4 join $Q = R_1(x_1, x_2) \bowtie R_2(x_2, x_3) \bowtie R_3(x_3, x_4) \bowtie R_4(x_4, x_5)$. For $E = \{2\}$, the boundary attributes are $\partial E = \{x_2, x_3\}$ and the maximum boundary is $\mathsf{T}_E(\mathbf{I}) = \max_{(b,c)\in \mathrm{dom}(x_2,x_3)} |R_2 \ltimes (b, c)| = 1$. For $E = \{2, 3\}$, the boundary attributes are $\partial E = \{x_2, x_4\}$ and the maximum boundary query is $\mathsf{T}_E(\mathbf{I}) = \max_{(b,d)\in \mathrm{dom}(x_2,x_4)} |R_2 \bowtie R_3 \ltimes (b, d)| = 3$, as highlighted in red. The rightmost is a join tree of Q.

Intuitively, $\hat{\mathsf{LS}}_k(\mathbf{I})$ approximates $\mathsf{LS}_k(\mathbf{I})$ by a weighted combination of the maximum change in the join size caused by changing $d_E(\mathbf{I}, \mathbf{I}')$ tuples from relations in E,and the fact that changing such a tuple affects at most $\hat{\mathsf{T}}_{[m]-\{i\}-E}(\mathbf{I})$ join results. Although $\hat{\mathsf{LS}}_k(\mathbf{I})$ is defined on all instances that differ by k tuples from $\mathbf{I}$, its computation does not depend on the actual tuples in $\mathbf{I}'$, but instead on the distance between $\mathbf{I}$ and $\mathbf{I}'$ induced by each subset $E \subseteq [m]$ of relations. This significantly reduces the computational cost of exhaustively enumerating each instance $\mathbf{I}'$ with $d(\mathbf{I}, \mathbf{I}') = k$.

$\mathsf{RS}(\mathbf{I})$ is derived by incorporating the maximum boundary $\mathsf{T}(\cdot)$ as Equation (2) to obtain $\hat{\mathsf{LS}}_k(\mathbf{I})$, which is subsequently applied into Equation (3). It has been proved that $\mathsf{RS}(\mathbf{I})$ is at most a constant factor larger than $\mathsf{SS}(\mathbf{I})$ and leads to satisfactory utility [25]. However, computing $\mathsf{RS}(\mathbf{I})$ is still expensive, where the bottleneck lies in computing $\hat{\mathsf{LS}}_k(\mathbf{I})$ and further boils down to computing the maximum boundaries $\mathsf{T}_E(\cdot)$. These queries are essentially join queries with aggregations (the formal definition is deferred to Section 4.1). Even insecure algorithms can incur superlinear runtime when computing some such queries, not to mention FO algorithms.

Elastic Sensitivity. Alternatively, degree information — the maximum frequency of a join value appears in the instance — can be exploited to derive an upper bound on $\mathsf{LS}_k(\mathbf{I})$ and leads to *elastic sensitivity* $\mathsf{ES}(\mathbf{I})$ [39]. As a result, $\mathsf{ES}(\mathbf{I})$ can be efficiently computed by collecting maximum frequencies of input relations in linear time of the input size $|\mathbf{I}|$ under data complexity. However, $\mathsf{ES}(\mathbf{I})$ is significantly larger than $\mathsf{RS}(\mathbf{I})$.

4 Relaxed-residual Sensitivity (RRS)

This section presents a new notion of sensitivity, called *relaxed-residual sensitivity* (RRS), following the construction framework of Lemma 3.5. To achieve, we focus on building a smooth upper bound on the maximum boundary; this immediately yields an upper bound $\hat{\mathsf{LS}}_k(\mathbf{I})$, and thus a smooth upper bound on the local sensitivity. The resulted sensitivity RRS achieves a more favorable trade-off between utility and computation in DO join evaluation. As shown in Figure 2, RRS sits between RS and ES in terms of both metrics: RRS is strictly smaller than ES, while can also be computed in linear time under data complexity; and RRS produces a marginally larger bound than RS but achieves a polynomial-time speed-up in the computation.

All missing proofs are provided in the full version [3]. To help understanding, we continue running the example in Figure 3.

4.1 Insights from Evaluating Join-max Query

We start by examining the maximum boundaries in Equation (1) and provide the following formal characterization. A *join-max query* is defined as $\max_{\mathbf{y}} Q = (\mathbf{x}, \{\mathbf{x}_i : i \in [m]\})$, where $Q = (\mathbf{x}, \{\mathbf{x}_i :$

$i \in [m]\}$) is a join query and $\mathbf{y} \subseteq \mathbf{x}$ is the set of aggregation attributes. The query result of $\max_{\mathbf{y}} Q$ on an instance $\mathbf{I}$ is

$$\max_{\mathbf{y}} Q(\mathbf{I}) = \max_{t \in \mathrm{dom}(\mathbf{y})} |Q(\mathbf{I}) \ltimes t|. \tag{4}$$

A join-max query (semantically) first computes the full join results $Q(\mathbf{I})$. Then it partitions $Q(\mathbf{I})$ into groups by the attributes in $\mathbf{y}$, and for each group, it counts the number of join results in that group. Finally, it returns the maximum count over all groups. If $\mathbf{y} = \emptyset$, the join-max query simply computes the join size $|Q(\mathbf{I})|$. A maximum boundary query, as in Equation (1), can be written as $\max_{\partial E} Q_E(\mathbf{I})$, where the underlying join is the subjoin induced by E and the aggregation attributes are boundary attributes of E. We introduce the following preliminaries before discussing how to evaluate join-max queries:

Definition 4.1 (Acyclic Join [65]). A join $Q = (\mathbf{x}, \{\mathbf{x}_i : i \in [m]\})$ is acyclic if one can organize its relations into a join tree $\mathcal{T}$ such that (1) there is a one-to-one correspondence between relations in Q and the nodes in $\mathcal{T}$; and (2) for each attribute $x \in \mathbf{x}$, the set of nodes containing x forms a connected subtree.

Definition 4.2 (Free-connex Join-Max Query [15]). A join-max query $\max_{\mathbf{y}} Q$ with $Q = (\mathbf{x}, \{\mathbf{x}_i : i \in [m]\})$, is free-connex if Q is acyclic, and the induced join by adding one more relation containing aggregation attributes, i.e., $(\mathbf{x}, \{\mathbf{x}_i : i \in [m]\} \cup \{\mathbf{y}\})$, is also acyclic.

Lemma 4.3 ([10]). *For a free-connex join-max query* $\max_{\mathbf{y}} Q$ *and any instance* $\mathbf{I}$*, there is an FO algorithm that computes* $\max_{\mathbf{y}} Q(\mathbf{I})$ *in* $\widetilde{O}(|\mathbf{I}|)$ *time under data complexity.*

Example 4.4. For $E = \{2, 3\}$, its induced join $(\{x_2, x_3, x_4\}, \{\{x_2, x_3\}, \{x_3, x_4\}, \{x_2, x_4\}\})$ is not acyclic and then T_E is not free-connex. For $E = \{4\}$, T_E is free-connex since both $(\{x_4, x_5\}, \{\{x_4, x_5, \}\})$ and $(\{x_4, x_5\}, \{\{x_4, x_5\}, \{x_4\}\})$ are acyclic. It can be checked that as long as $E \neq \{2, 3\}$, T_E is always free-connex for line-4 join.

4.2 Dropping Boundary Attributes

Inspired by the join-max query evaluation, we can drop some attributes in ∂E so that the resulting join-max query is free-connex, hence can be computed in linear time. We define

$$\mathsf{RST}_E^{\mathbf{y}}(\mathbf{I}) = \max_{\partial E - \mathbf{y}} Q_E(\mathbf{I}) = \max_{t \in \mathrm{dom}(\partial E - \mathbf{y})} |Q_E(\mathbf{I}) \ltimes t|\,, \tag{5}$$

where $\mathbf{y} \subseteq \partial E$. As long as the join-max query $\max_{\mathrm{dom}(\partial E - \mathbf{y})} Q_E(\mathbf{I})$ is free-connex, it can be computed in linear time. Note that $\mathsf{RST}_E^{\mathbf{y}}(\mathbf{I})$ captures $\mathsf{T}_E(\mathbf{I})$ when $\mathbf{y} = \emptyset$, and $|Q_E(\mathbf{I})|$ when $\mathbf{y} = \partial E$.

Example 4.5. For any $E \neq \{2, 3\}$, T_E is free-connex and hence can be computed in linear time. The bottleneck is to compute T_E for $E = \{2, 3\}$, which requires quadratic time [8]. But, we can drop some boundary attributes from $\partial E = \{x_2, x_4\}$. If dropping attribute x_2, we get an upper bound of $\mathsf{T}_E(\mathbf{I})$ as $\max_{d \in \mathrm{dom}(x_4)} |R_2 \bowtie R_3 \ltimes (d)|$. Similarly, we can also drop attribute x_4 to get another upper bound $\max_{b \in \mathrm{dom}(x_2)} |R_2 \bowtie R_3 \ltimes (b)|$, or drop both attributes x_2, x_4 to get a even looser upper bound $|R_2 \bowtie R_3|$. All these join-max queries are free-connex, thus can be computed in linear time.

Lemma 4.6 intuitively implies that the more attributes $\mathbf{y}$ are dropped, the more $\mathsf{RST}_E^{\mathbf{y}}$ relaxes from the maximum boundary T_E. Then, any resulting $\mathsf{RST}_E^{\mathbf{y}}$ is an upper bound on T_E. Lemma 4.7 shows the smoothness of $\mathsf{RST}_E^{\mathbf{y}}$ for some fixed $\mathbf{y}$. As such, $\mathsf{RST}_E^{\mathbf{y}}$ is a smooth upper bound on the maximum boundary as defined in 3.4.

Lemma 4.6. *For any instance* $\mathbf{I}$*, any* $E \subsetneq [m]$ *and any pair of* $\mathbf{y}, \mathbf{y}' \subseteq \partial E$ *with* $\mathbf{y}' \subseteq \mathbf{y}$, $\mathsf{T}_E(\mathbf{I}) \leq \mathsf{RST}_E^{\mathbf{y}'}(\mathbf{I}) \leq \mathsf{RST}_E^{\mathbf{y}}(\mathbf{I})$.

Lemma 4.7. *For any $E \subsetneq [m]$, $i \in E$, $\mathbf{y} \subseteq \partial E$, and two neighboring instances $\mathbf{I}, \mathbf{I}'$ such that $d(R_i^{\mathbf{I}}, R_i^{\mathbf{I}'}) = 1$ and $R_j^{\mathbf{I}} = R_j^{\mathbf{I}'}$ for any $j \in E - \{i\}$, $\left|\mathsf{RST}_E^{\mathbf{y}}(\mathbf{I}) - \mathsf{RST}_E^{\mathbf{y}}(\mathbf{I}')\right| \le \mathsf{RST}_{E-\{i\}}^{\mathbf{y}}(\mathbf{I})$.*

Besides, Lemma 4.7 guides the choice of $\mathbf{y}$ — the attributes to be dropped — over different subsets of relations in E. If an attribute x is dropped from T_E for some $E \subsetneq [m]$, then x must be dropped from T_E for each $E' \subseteq E$ if $x \in \partial E'$, to guarantee the smoothness.

4.3 Incorporating Degree Information

Inspired by the construction of elastic sensitivity ES, we also exploit the degree information in constructing a smooth upper bound on the maximum boundary. We adapt the original construction of ES into the framework in Lemma 3.5, and extend it to accommodate the idea of dropping boundary attributes.

For any relation $R_j \in [m]$ and a subset of attributes $\mathbf{y} \subseteq \mathbf{x}_j$, the maximum frequency of $\mathbf{y}$ in R_j is $\deg(\mathbf{y}, R_j) = \max_{t \in \mathrm{dom}(\mathbf{y})} |R_j \ltimes t|$, i.e., the maximum number of tuples in R_j that can be joined with some $t \in \mathrm{dom}(\mathbf{y})$. When $\mathbf{y} = \emptyset$, $\deg(\emptyset, R_j) = |R_j|$. To connect the maximum degrees in different relations, we organize all relations into a tree such that there is a one-to-one correspondence between relations and nodes[4]. By specifying a relation R_r as the root, let $p(j, r)$ be the parent node of relation R_j in this rooted tree. By incorporating the idea of dropping boundary attributes, for any $\mathbf{y} \subseteq \partial E$ and $E \subsetneq [m]$, we define

$$\mathsf{EST}_E^{\mathbf{y}}(\mathbf{I}) = \max_{r \in [m]-E} \prod_{j \in E} \deg\left((\mathbf{x}_j \cap \mathbf{x}_{p(j,r)}) \cup \partial E - \mathbf{y}, R_j\right),$$

As all the maximum frequencies can be computed in each input table, this takes $O(N)$ time under data complexity. Hence, $\mathsf{EST}_E^{\mathbf{y}}(\mathbf{I})$ can be computed in such an amount of time as well.

Example 4.8. In Figure 3, the join tree is rooted at R_3 with $p(1,3) = 2$, $p(2,3) = 3$ and $p(4,3) = 3$. On this instance, we have $\deg(\{x_2\}, R_2) = 5$, $\deg(\{x_2, x_3\}, R_3) = 2$, $\deg(\{x_2, x_3\}, R_2) = 1$, and $\deg(\{x_2, x_4\}, R_3) = 4$. For $E = \{2, 3\}$, $\mathsf{T}_E(\mathbf{I}) = 3$, which can be relaxed as $(\mathbf{y} = \{x_4\})$:

$$\mathsf{EST}_E^{\mathbf{y}}(\mathbf{I}) = \max\left\{ \begin{array}{l} \deg(\{x_2\}, R_2) \cdot \deg(\{x_3\}, R_3) \\ \deg(\{x_2, x_3\}, R_2) \cdot \deg(\{x_4\}, R_3) \end{array} \right\} = 10.$$

Following a similar analysis of elastic sensitivity, we can prove both the upper bound (Lemma 4.9) and smoothness (Lemma 4.10) of EST as desired in Definition 3.4.

Lemma 4.9. *For any instance $\mathbf{I}$, any $E \subsetneq [m]$ and any pair of $\mathbf{y}, \mathbf{y}' \subseteq \partial E$ with $\mathbf{y}' \subseteq \mathbf{y}$, $\mathsf{T}_E(\mathbf{I}) \le \mathsf{EST}_E^{\mathbf{y}'}(\mathbf{I}) \le \mathsf{EST}_E^{\mathbf{y}}(\mathbf{I})$.*

Lemma 4.10. *For any $E \subsetneq [m]$, $i \in E$, $\mathbf{y} \subseteq \partial E$, and two neighboring instances $\mathbf{I}, \mathbf{I}'$ such that $d(R_i^{\mathbf{I}}, R_i^{\mathbf{I}'}) = 1$ and $R_j^{\mathbf{I}} = R_j^{\mathbf{I}'}$ for any $j \in E - \{i\}$, $\left|\mathsf{EST}_E^{\mathbf{y}}(\mathbf{I}) - \mathsf{EST}_E^{\mathbf{y}}(\mathbf{I}')\right| \le \mathsf{EST}_{E-\{i\}}^{\mathbf{y}}(\mathbf{I})$.*

4.4 New Smooth UB on Maximum Boundary

Now, we will combine these two ideas together to construct our final smooth upper bound on the maximum boundary, denoted as $\hat{\mathsf{T}}_E$, where $\hat{\mathsf{T}}_E$ can take RST_E or EST_E together with some dropped attributes $\mathbf{y}$. Recall Definition 3.4: the upper bound property of $\hat{\mathsf{T}}_E$ is easily preserved as this is preserved by each individual choice; and we will focus on the smoothness property of $\hat{\mathsf{T}}_E$. We also mention a helpful lemma on the relative ordering between RST and EST:

[4]Compared with the notion of join tree in Definition 4.1, incorporating degree information only requests a tree that satisfies condition (1) but not (2). Hence, for cyclic joins (where a join tree does not exist), such a tree is also feasible. We also note that there could be many different ways to construct such a tree, some of which will lead to a smaller upper bound on the maximum boundary. For example, for acyclic joins, it is always good to choose a join tree to incorporate degree information.

Algorithm 2: HybridConstruct(Q)

1 **foreach** $i \in [m]$ **do**
2 **foreach** $E' \subseteq [m]$ *such that* $|E'| = m - i$ **do**
3 $\mathfrak{y} \leftarrow$ all boundary attributes dropped from T_E for any $E \subsetneq [m]$ such that $E' \subseteq E$ and $|E'| + 1 = |E|$;
4 $\mathbf{y}' \leftarrow$ any subset of $\partial E'$ such that $\mathfrak{y} \cap \partial E' \subseteq \mathbf{y}'$;
5 **if** $\hat{\mathsf{T}}_E = \mathsf{EST}_E^{\mathbf{y}}$ *or* $\max_{\partial E' - \mathbf{y}'} Q_{E'}$ *is not free-connex* **then** $\hat{\mathsf{T}}_{E'} \leftarrow \mathsf{EST}_{E'}^{\mathbf{y}'}$;
6 **else** $\hat{\mathsf{T}}_{E'} \leftarrow$ either $\mathsf{RST}_{E'}^{\mathbf{y}'}$ or $\mathsf{EST}_{E'}^{\mathbf{y}'}$;

Lemma 4.11. *For any* $\mathbf{I}$, $E \subsetneq [m]$ *and* $\mathbf{y} \subseteq \partial E$, $\mathsf{RST}_E^{\mathbf{y}}(\mathbf{I}) \le \mathsf{EST}_E^{\mathbf{y}}(\mathbf{I})$.

Smoothness Analysis. Consider two neighboring instances $\mathbf{I}, \mathbf{I}'$ such that $d(R_i^{\mathbf{I}}, R_i^{\mathbf{I}'}) = 1$ for some $i \in E$. Suppose we already drop the attributes $\mathbf{y}$ from ∂E for T_E. Now, we attempt to dropping the attributes $\mathbf{y}'$ from $\partial(E - \{i\})$ for $T_{E-\{i\}}$.

- **Case 1: when $\hat{\mathsf{T}}_E$ takes $\mathsf{RST}_E^{\mathbf{y}}$.** The changes on $\mathsf{RST}_E^{\mathbf{y}}$ over instances $\mathbf{I}, \mathbf{I}'$ can be bounded by taking an arbitrary choice for $E - \{i\}$, as long as $\mathbf{y} \cap \partial(E - \{i\}) \subseteq \mathbf{y}'$:

$$\left|\mathsf{RST}_E^{\mathbf{y}}(\mathbf{I}) - \mathsf{RST}_E^{\mathbf{y}}(\mathbf{I}')\right| \le \mathsf{RST}_{E-\{i\}}^{\mathbf{y}'}(\mathbf{I}) \le \mathsf{EST}_{E-\{i\}}^{\mathbf{y}'}(\mathbf{I}).$$

  Following Lemma 4.11, if taking the first choice for E by dropping boundary attributes $\mathbf{y}$, then for the subset $E - \{i\}$, we can safely take either choice by dropping all boundary attributes also in $\mathbf{y}$.
- **Case 2: when $\hat{\mathsf{T}}_E$ takes $\mathsf{EST}_E^{\mathbf{y}}$.** However, in this case, the changes on $\mathsf{EST}_E^{\mathbf{y}}$ over neighboring instances $\mathbf{I}, \mathbf{I}'$ can be bounded only by taking the second choice for $E - \{i\}$, and $\mathbf{y} \cap \partial(E - \{i\}) \subseteq \mathbf{y}'$:

$$\left|\mathsf{EST}_E^{\mathbf{y}}(\mathbf{I}) - \mathsf{EST}_E^{\mathbf{y}}(\mathbf{I}')\right| \le \mathsf{EST}_{E-\{i\}}^{\mathbf{y}}(\mathbf{I}) \le \mathsf{EST}_{E-\{i\}}^{\mathbf{y}'}(\mathbf{I}).$$

  This means that if taking the second choice for T_E by dropping attributes $\mathbf{y}$, then for the subset $E - \{i\}$, we should stick to the second choice by dropping all boundary attributes also in $\mathbf{y}$.

Hybrid Construction. Based on analysis above, we present Algorithm 2 to construct $\hat{\mathsf{T}}$. **(Line 1–2)** Algorithm 2 proceeds in a partial order on all subsets of $[m]$: for any pair of subsets $E, E' \subsetneq [m]$, if $E' \subsetneq E$, an upper bound on T_E is picked before on $\mathsf{T}_{E'}$. As such, it starts by enumerating subsets $E' \subsetneq [m]$ in a decreasing order of their sizes. When considering subsets of size $m - i$, Algorithm 2 have made choices for all subsets of size $m - i - 1$. **(Line 3–4)** Algorithm 2 collects the attributes dropped by T_E as $\mathfrak{y}$ for all supersets $E \supsetneq E'$. Then, it chooses a subset $\mathbf{y}'$ from $\partial E'$, such that if a boundary attribute $x \in \partial E'$ is in $\mathfrak{y}$, i.e., x has been dropped by T_E for some superset $E \supsetneq E'$, then x is included into $\mathbf{y}'$ and will also be dropped by $\mathsf{T}_{E'}$. **(Line 5–7)** If $\hat{\mathsf{T}}_E$ is $\mathsf{EST}_E^{\mathbf{y}}$, then $\hat{\mathsf{T}}_{E'}$ can only take $\mathsf{EST}_E^{\mathbf{y}'}$, as discussed in the smoothness analysis. If the query $\max_{\partial E' - \mathbf{y}'} Q_{E'}$ is not free-connex, then $\hat{\mathsf{T}}_{E'}$ also cannot take $\mathsf{RST}_E^{\mathbf{y}'}$. Otherwise, $\hat{\mathsf{T}}_{E'}$ is free to take either choice.

Example 4.12. Continue with the line-4 join. For $E = \{1, 2, 3\}$ or $\{2, 3, 4\}$, sinceT_E is free-connex, $\hat{\mathsf{T}}_E$ can be $\mathsf{RST}_E^{\emptyset}$. For $E = \{2, 3\}$, as T_E is not free-connex, $\hat{\mathsf{T}}_E$ can be either $\mathsf{RST}_E^{\{x_2\}}$ or $\mathsf{EST}_E^{\{x_2\}}$. If $\hat{T}_E$ is $\mathsf{EST}_E^{\{x_2\}}$, then $\hat{\mathsf{T}}_{\{2\}}$ has to take $\mathsf{EST}_{\{2\}}^{\mathbf{y}'}$ for some $\mathbf{y}'$ with $x_2 \in \mathbf{y}'$.

Lemma 4.13. *Algorithm 2 returns a valid smooth upper bound* $\hat{\mathsf{T}}(\cdot)$ *on the maximum boundary* $\mathsf{T}(\cdot)$ *in* $O(1)$ *time under data complexity.*

Algorithm 3: DOJoin$(Q, \mathbf{I}, \epsilon, \delta)$

1 Compute the true join size $|Q(\mathbf{I})|$ [10, 38];
2 $\hat{\mathsf{T}} \leftarrow$ HybridConstruct(Q); ▶ **Algorithm 2**;
3 **foreach** $E \subseteq [m]$ **do** Compute $\hat{\mathsf{T}}_E(\mathbf{I})$ [10];
4 $\mathsf{RRS}(\mathbf{I}) \leftarrow$ the RRS constructed based on ϵ, δ and $\{\hat{\mathsf{T}}_E(\mathbf{I}) : E \subseteq [m]\}$; ▶ **Lemma 3.5**;
5 $\widetilde{\mathrm{OUT}} \leftarrow (\epsilon, \delta)$-DP upper bound of true join size $|Q(\mathbf{I})|$ based on $\mathsf{RRS}(\mathbf{I})$ [40] ▶ **Section 5.3**;
6 Compute $Q(\mathbf{I})$ by the advised-FO algorithm [10, 38, 43, 46] with advice $\widetilde{\mathrm{OUT}}$; ▶ **Section 3.2**;

The correctness of Algorithm 2 follows from the smoothness analysis and Definition 3.4. Algorithm 2 enumerates all subsets of E and constructs $\hat{\mathsf{T}}_E$ for every $E \subseteq [m]$ in $O(2^{3mn})$ time. In terms of theoretical analysis, we focus on data complexity and assume the query size to be a constant, so this remains a constant. In Section 5.3, we also provide some heuristics to speed up this algorithm.

5 RRS-based DO Join

Putting everything together, we obtain RRS-based DO algorithm.

5.1 Algorithm

Algorithm 3 takes as input the join query Q, an instance $\mathbf{I}$, and privacy parameters ϵ, δ. In line 1, Algorithm 3 first computes the true join size using the FO algorithm in [10]. In lines 2-3, Algorithm 1 constructs a smooth upper bound on the maximum boundaries using Algorithm 2, and then evaluates each maximum boundary query on the input instance using the oblivious algorithms in [10]. Then, in line 4, it constructs RRS following Lemma 3.5, where the computation based on maximum boundaries can be done within trusted memory assuming the query size as a constant. Now, in line 5, it is ready to compute a DP upper bound $\widetilde{\mathrm{OUT}}$ of the true join size, following some standard mechanism for adding noise [40]. We give more details when discussing the implementation in Section 5.3. Finally, in line 6, it invokes the advised FO join algorithm [10, 38, 43, 46] to compute the join results with the advice $\widetilde{\mathrm{OUT}}$.

Algorithm 3 indeed applies to cyclic joins by empowering lines 1 and 6 with two additional primitives. We integrate the generalized hypertree decomposition (GHD) [35] alongside the FO worst-case optimal join algorithm [38] to transform a cyclic instance into an acyclic one. Then, lines 1 and 6 run on this derived acyclic instance using [10, 43, 46]. Meanwhile, lines 2-5 remain unchanged and are directly applied to the input cyclic instance.

Theorem 5.1. *Algorithm 3 is (ϵ, δ)-DO.*

Proof of Theorem 5.1. Line 1 invokes the FO algorithm [10] to compute $|Q(\mathbf{I})|$, hence this step only depends on (or reveals) the input size $|\mathbf{I}|$. Line 2 runs within the trusted memory, hence does not incur any access to untrusted storage. Line 3 invokes FO algorithms in [10] to compute $\hat{\mathsf{T}}_E(\mathbf{I})$, hence this step only depends on (or reveals) the input size $|\mathbf{I}|$. Lines 4-5 run within the trusted memory. Line 6 computes the join results with advice $\widetilde{\mathrm{OUT}}$, which only depends on (or reveals) the input size $|\mathbf{I}|$ and the advice $\widetilde{\mathrm{OUT}}$. Hence, the access pattern of Algorithm 3 only depends on $|\mathbf{I}|$ and $\widetilde{\mathrm{OUT}}$, i.e., its leakage function $\mathsf{Leak}(\mathbf{I})$ consists of $|\mathbf{I}|$ and $\widetilde{\mathrm{OUT}}$. As $\widetilde{\mathrm{OUT}}$ satisfies (ϵ, δ)-DP, Algorithm 3 is (ϵ, δ)-DO by [20]. □

Join	FO	DO
Two-way	N^2	$N \log N + \lvert\boldsymbol{Q}(\mathbf{I})\rvert$ [23] $\Omega(N \log\log N + \lvert\boldsymbol{Q}(\mathbf{I})\rvert + \Delta \log N)$ [23]
Multi-way	$O(N^\rho \log N)$ [38]	$O\left((N + \lvert\boldsymbol{Q}(\mathbf{I})\rvert + \mathsf{RS}(\mathbf{I})) \cdot \log N\right)$ (for residual-linear joins) $O\left((N^w + \lvert\boldsymbol{Q}(\mathbf{I})\rvert + \mathsf{RRS}(\mathbf{I})) \cdot \log N\right)$ $\Omega\left(N + \lvert\boldsymbol{Q}(\mathbf{I})\rvert + \mathsf{RS}(\mathbf{I})\right)$ [26]

Table 2. Summary of theoretical results on DO join evaluation. N is the input size and $|\boldsymbol{Q}(\mathbf{I})|$ is the true join size. $\epsilon = O(1)$ and $\delta = 1/N^{O(1)}$. Δ is the maximum frequency of a join value. w is the fractional hypertree width of the join query [35].

5.2 Theoretical Analysis of Time Complexity

Theorem 5.2. *Assume $\epsilon = O(1)$ and $\delta = 1/N^{O(1)}$. Algorithm 3 runs in $O((N^w + |\boldsymbol{Q}(\mathbf{I})| + \mathsf{RRS}(\mathbf{I})) \cdot \log N)$ time under data complexity, where w is the fractional hypertree width of the join query [35].*

Proof of Theorem 5.2. For simplicity, we first consider acyclic joins for which $w = 1$. Line 2 runs in $O(1)$ time from Lemma 4.13. Line 2 computes every maximum boundary $\hat{\mathsf{T}}_E(\mathbf{I})$ in $O(N \log N)$ time. As there are $O(1)$ different subsets $E \subsetneq [m]$ under data complexity, this step takes $O(N \log N)$ time in total. Line 4 computes $\mathsf{RRS}(\mathbf{I})$ according to Equations (2) and (3), which still takes $O(1)$ time according to [25]. Line 1 computes the true join size in $O(N \log N)$ time [10]. Line 5 computes a noisy join size by calibrating noise based on $\mathsf{RRS}(\mathbf{I})$, which only takes $O(1)$ time. Finally, the advised FO join algorithm runs in $O\left((N + \widetilde{\mathrm{OUT}}) \cdot \log N\right) = O\left((N + |\boldsymbol{Q}(\mathbf{I})| + \mathsf{RRS}(\mathbf{I})) \cdot \log N\right)$ time, since $\widetilde{\mathrm{OUT}}$ is proportional to $O\left(|\boldsymbol{Q}(\mathbf{I})| + \mathsf{RRS}(\mathbf{I})\right)$ assuming $\epsilon = O(1)$ and $\delta = 1/N^{O(1)}$. Next, we turn to cyclic joins. The only difference in the analysis is that the input size of the transformed instance is enlarged from N to $O(N^w)$, which only affects the complexity of line 1 and line 6. □

Optimality on Residual-linear Joins. As observed by [23], the total number of memory accesses made by a DO algorithm must be a DP upper bound of the join size. In this way, any DP lower bound for releasing join size is a runtime lower bound for DO join evaluation. Dong and Yi [26] showed the lower bound $\Omega(N + |\boldsymbol{Q}(\mathbf{I})| + \mathsf{SS}(\mathbf{I}))$ for releasing join size under DP can be further simplified as $\Omega(N + |\boldsymbol{Q}(\mathbf{I})| + \mathsf{RS}(\mathbf{I}))$ when $\beta = O(1)$. Hence, the same lower bound carries over to the DO join evaluation. Comparing it with Theorem 5.1, the gap on the upper and lower bound comes from the gap between $O(|\mathsf{RRS}(\mathbf{I})|)$ and $\Omega(N + |\boldsymbol{Q}(\mathbf{I})| + \mathsf{RS}(\mathbf{I}))$.

Lemma 5.3 ([25]). *Let $0 < \beta < 1$ be a constant. $\mathsf{S}(\mathbf{I}) = O(\max_{E \subseteq [m]} \hat{\mathsf{T}}_E(\mathbf{I}))$ as defined in Equation (3).*

From Lemma 5.3, this gap further translates to the gap between the terms $O(\max_E \hat{\mathsf{T}}_E(\mathbf{I}))$ and $\Omega(N + |\boldsymbol{Q}(\mathbf{I})| + \max_E \mathsf{T}_E(\mathbf{I}))$. We identify the class of *residual-linear joins* on which no asymptotical gap exists between $O(\max_E \hat{\mathsf{T}}_E(\mathbf{I}))$ and $\Omega(N + \max_E \mathsf{T}_E(\mathbf{I}))$ for any instance $\mathbf{I}$, hence Algorithm 3 asymptomatically matches the lower bound.

Definition 5.4 (Residual-linear Join). A join query $\boldsymbol{Q}$ is *residual-linear* if for any $E \subsetneq [m]$, either T_E is free-connex, or there exists $j \in E$ such that $\mathbf{x}_j$ contains all non-boundary attributes of T_E, i.e., $\cup_{i \in E} \mathbf{x}_i - \partial E \subseteq \mathbf{x}_j$.

When T_E satisfies the first condition, $\hat{\mathsf{T}}_E$ can take T_E via $\mathsf{RST}_E^{\emptyset}$; when T_E satisfies the second condition, the value of $\hat{\mathsf{T}}_E$ is bounded by the input size of R_j, which is N in this case. In either case,

$\hat{\mathsf{T}}_E$ can be bounded by $O(N + \mathsf{T}_E)$. Many commonly-used joins are residual-linear with such nice properties, including line-3 join, line-4 join, line-5 join, star joins, and hierarchical joins. Details are provided in the full version [3]. We summarize the theoretical results of DO joins in Table 2.

5.3 Discussions

Below, we discuss some details in implementing Algorithm 3.

Computing $\widetilde{\mathrm{OUT}}$. Given ε, δ, true join size $|\boldsymbol{Q}(\mathbf{I})|$ and sensitivity $\mathsf{S}(\mathbf{I})$, we compute the noisy join size $\widetilde{\mathrm{OUT}}$ following [40, 50]:

- Set $\epsilon_1 = \epsilon_2 = \frac{\epsilon}{2}$, $\delta_1 = \delta_2 = \frac{\delta}{2e^{\epsilon/2}}$ and $\beta = \frac{\epsilon_1}{\ln(1/\delta_1)}$;
- Compute $\tilde{\mathsf{S}}(\mathbf{I}) = \exp\left(\frac{\beta}{\epsilon_1} \cdot (\ln \frac{1}{\delta_1} + Y)\right) \cdot \mathsf{S}(\mathbf{I})$, where $Y \sim \mathsf{Lap}(0, 1)$;
- Sample a noisy value $\widetilde{\Delta}$ from $\mathsf{TLap}\left(\frac{\tilde{\mathsf{S}}(\mathbf{I})}{\epsilon_2} \cdot \ln(1 + \frac{e^{\epsilon_2}-1}{\delta_2}), \frac{\tilde{\mathsf{S}}(\mathbf{I})}{\epsilon_2}\right)$, where $\mathsf{TLap}(\tau, b)$ denotes the shifted and truncated Laplace distribution with support $[0, 2\tau]$ and density proportional to $\exp(-\frac{|x-\tau|}{b})$, and get $\widetilde{\mathrm{OUT}} = |\boldsymbol{Q}(\mathbf{I})| + \widetilde{\Delta}$.

Following the standard noise configuration [25], we use $|\boldsymbol{Q}(\mathbf{I})| + f_{\epsilon,\delta} \cdot \mathsf{S}(\mathbf{I})$ to simulate $\widetilde{\mathrm{OUT}}$, where $f_{\epsilon,\delta} = \frac{2e^2}{\epsilon} \cdot \left(1 + \ln(1 + \frac{2(e^{\epsilon} - e^{\epsilon/2})}{\delta})\right)$.

Optimization for Hybrid Construction. The non-determinism introduced by Algorithm 2 may affect the overall runtime of the proposed DO algorithm, since different $\hat{\mathsf{T}}$ can lead to different sensitivity values. However, exhaustively enumerating all possible constructions of $\hat{\mathsf{T}}$ to identify the one that leads to the smallest sensitivity is also computationally expensive. To mitigate this, we apply the following heuristics when implementing Algorithm 2. First, for residual-linear queries, we relax T_E using only the first choice, since it always yields a tighter upper bound than the other choice. Second, as dropping more boundary attributes leads to looser smooth upper bounds, we only drop attributes when necessary. For example, at line 4 of Algorithm 2, if the join-max query $\max_{\partial E' - \mathfrak{y} \cap \partial E'} \boldsymbol{Q}_{E'}$ is already free-connex, we simply choose $\mathbf{y}' = \mathfrak{y} \cap \partial E'$.

Discussion for Cyclic Joins. We address a strawman solution that fails to achieve DO for cyclic joins: employ GHD [35] alongside FO worst-case optimal join algorithms [38] to transform a cyclic instance into an acyclic one, and then apply Algorithm 3 entirely to the acyclic instance. This approach does not achieve DO because neighboring instances can lead to significantly different acyclic instances derived by GHD; consequently, these instances are no longer neighboring, and Algorithm 3 cannot be applied.

Alternatively, we can remove a subset of join predicates to render the join acyclic, then invoke Algorithm 3 and subsequently filter the results to satisfy the original predicates. This strategy, consistent with prior work [39], preserves the privacy guarantee since dropping predicates only increases the join size and its sensitivity.

6 Experiments

6.1 Setup

All experiments are conducted on a single machine with Intel(R) Xeon(R) Platinum CPU @ 2.60GHz and 1TB of memory. All code is available at [2]. As our primary focus is on assessing the advantage of RRS over ES and RS, we disregard the encryption and decryption steps throughout the computation, and all algorithms execute on plaintext, following the prototype setting described in [43].

Datasets. Two datasets are used in our experiments: the SNAP Deezer social network [57] and TPC-H benchmark [6]. Deezer contains 28281 vertices and 185504 directed edges. We split edges into 6 relations, each containing 32140, 29809, 30856, 31134, 30981, and 30584 edges, respectively.

Datasets		Deezer					
Queries		Q_{line3}	Q_{line4}	Q_{line5}	Q_{line6}	Q_{star4}	$Q_{\triangle}$
$\lvert Q(\mathbf{I})\rvert$		216,797	722,281	2,323,110	8,233,670	4,244,360	196
ES	Value	103,000	1.21×10^6	1.75×10^8	2.66×10^{10}	40,400	6,800
	Time (s)	0.11	0.15	0.19	0.23	0.12	0.11
	$\widetilde{\text{OUT}}_{\text{ES}}$	1.22×10^6	1.18×10^8	1.70×10^{10}	2.58×10^{12}	8.17×10^6	661,000
RS	Value	3,700	17,300	145,000	807,000	33,000	103
	Time (s)	0.330	5,290	16,000	84,800	1.14	1.51
	$\widetilde{\text{OUT}}_{\text{RS}}$	576,000	2.40×10^6	1.64×10^7	8.66×10^7	7.45×10^6	10,200
RRS	Value	3,700	23,800	145,000	2.57×10^6	33,000	103
	Time (s)	0.330	0.890	1.92	89.0	1.14	1.51
	$\widetilde{\text{OUT}}_{\text{RRS}}$	576,000	3.03×10^6	1.64×10^7	2.57×10^8	7.45×10^6	10,200
AGM		9.92×10^8	3.09×10^{13}	3.07×10^{13}	9.40×10^{17}	9.20×10^{17}	45,034

Table 3. Comparison of sensitivity values, computation time, and noisy join sizes computed of different sensitivities on the Deezer dataset. Let $\epsilon = 4$ and $\delta = 1 \times 10^{-9}$. For reference, we also provide the true join size $|Q(\mathbf{I})|$ and the worst-case join size AGM [13].

We additionally create a relation containing all triangles in the graph, denoted as R_7. To investigate how different relation sizes affect performance, we split the edges in an unbalanced way and create four datasets with ratios between the largest and the smallest relation varying among 5, 10, 50, and 100. For the TPC-H benchmark, we ignore the key constraints since we focus on tuple-level DO.

Queries. We omit two-way joins from evaluation since, in these cases, our DO algorithm degenerates to either the naive approach (Theorem 15) in [23] or the adapted ODBJ method in [55]. Furthermore, since [55] already provides a comprehensive evaluation of various DO two-way join algorithms, we refer interested readers to their work for additional details. In the following sections, we focus specifically on our general framework for DO multi-way joins.

We evaluate the following join queries on Deezer:

$$Q_{\text{line}\ell} = R_1(A_1, A_2) \bowtie \cdots \bowtie R_\ell(A_\ell, A_{\ell+1}), \quad \ell = 3, 4, 5, 6;$$
$$Q_{\text{star4}} = R_1(A_1, B) \bowtie R_2(A_2, B) \bowtie R_3(A_3, B) \bowtie R_4(A_4, B);$$
$$Q_{\triangle} = R_1(A, B) \bowtie R_2(B, C) \bowtie R_3(A, C) \bowtie R_7(A, B, C);$$

We select 6 representative queries from the TPC-H: Q_{tpc2}, Q_{tpc3}, Q_{tpc7}, Q_{tpc9}, Q_{tpc10}. Notably, Q_{tpc3} is line-3 join; Q_{tpc10} and Q_{tpc21} are line-4 joins; Q_{tpc2} and Q_{tpc7} are line-5 joins; Q_{tpc9} is the most complicated one involving 7 relations. First, Q_{line3}, Q_{tpc3}, Q_{star4}, and $Q_{\triangle}$ are residual-linear with all maximum boundaries as being free-connex join-max queries, hence RRS and RS coincide. Second, Q_{line4}, Q_{tpc10}, Q_{tpc21}, Q_{line5}, Q_{tpc2} and Q_{tpc7} are also residual-linear but without this special property. Third, Q_{line6} and Q_{tpc9} are not residual-linear. The discussion for the remaining join queries is provided in the full version [3].

Parameters Setup. The parameter ϵ determines the privacy guarantee, where a smaller ϵ provides stronger privacy. The parameter δ is set relative to the input size N, approximately $1/N^{1.2}$. The parameters β and $f_{\epsilon,\delta}$ are functions of ϵ and δ (following the construction in Section 5.3); therefore, we only need to manually set up ϵ and δ.

In Sections 6.2–6.3, we fix $\epsilon = 4$. We set $\delta = 1 \times 10^{-9}$ for Deezer and $\delta = 1 \times 10^{-8}$ for TPC-H (scale 0.1); the difference in δ reflects the different input sizes of the two datasets. In Section 6.4, we vary ϵ across $\{1, 2, 4, 8, 16\}$ to study its effect. As before, we maintain $\delta = 1 \times 10^{-9}$ for Deezer and $\delta = 1 \times 10^{-8}$ for TPC-H (scale 0.1). In Section 6.5, we vary the scale of the TPC-H dataset to evaluate scalability, fixing $\epsilon = 4$. As the scale changes from 0.01, 0.05, 0.1, 0.5, to 1, the input size changes, so we set δ to 10^{-7}, 2×10^{-8}, 10^{-8}, 2×10^{-9}, and 10^{-9} accordingly. Additionally, in investigating the

Datasets		TPC-H					
Queries		Q_{tpc3}	Q_{tpc10}	Q_{tpc21}	Q_{tpc2}	Q_{tpc7}	Q_{tpc9}
$\lvert Q(\mathbf{I})\rvert$		600,572	600,572	600,572	80,000	600,572	600,572
ES	Value	310	199,000	194,000	598,000	3.15×10^6	2.72×10^7
	Time (s)	2.07	1.28	1.96	0.220	1.63	2.53
	$\widetilde{\text{OUT}}_{\text{ES}}$	628,000	1.82×10^7	1.71×10^7	5.31×10^7	2.80×10^8	9.87×10^9
RS	Value	155	26,500	31,500	24,600	26,500	31,500
	Time (s)	3.45	12,000	5,000	277	$> 1.00 \times 10^7$	5,000
	$\widetilde{\text{OUT}}_{\text{RS}}$	614,000	2.95×10^6	3.39×10^6	2.26×10^6	2.95×10^6	3.39×10^6
RRS	Value	155	26,500	31,500	24,600	26,500	31,500
	Time (s)	3.45	4.76	10.9	1.69	17.9	34.4
	$\widetilde{\text{OUT}}_{\text{RRS}}$	614,000	2.95×10^6	3.39×10^6	2.26×10^6	2.95×10^6	3.39×10^6
AGM		9.00×10^9	2.25×10^{11}	3.75×10^9	1.00×10^8	3.75×10^9	1.50×10^7

Table 4. Comparison of sensitivity values, computation time, and noisy join sizes computed for different sensitivities on the TPC-H dataset (with scale = 0.1). Let $\epsilon = 4$ and $\delta = 1 \times 10^{-8}$. For reference, we also provide the true join size $|Q(\mathbf{I})|$ and the worst-case join size AGM [13].

effect of the unbalanced ratio of Deezer, we stick to $\delta = 1 \times 10^{-9}$ and vary ϵ across $\{1, 2, 4, 8, 16\}$ for each unbalanced ratio.

6.2 Sensitivity

This subsection compares the value, the computation cost, and the resulting noisy join size of three different sensitivities ES, RS, and RRS. Our results are reported in Tables 3 and 4.

Graph Queries. First, let's focus on Q_{line3}, Q_{star4}, and $Q_{\triangle}$, on which RRS and RS coincide. As expected, RRS and RS have identical sensitivity values and computation time as shown in Table 3. In contrast, ES is larger than RS and RRS across all queries due to the following reasons [25]: ES makes an extremely pessimistic assumption that the most frequent values on the common attributes between relations join, and even worse, ES promises the smoothness property by adding a uniform error to the maximum frequencies. Although RS and RRS collects more data statistics than ES, their computation time is comparable since both maximum frequencies and free-connex maximum boundaries can be efficiently computed.

Then, let's move to Q_{line4} and Q_{line5}, another two residual-linear queries on which where ES, RS and RRS behave diversely. RRS is significantly smaller than ES due to the pessimistic assumption mentioned. For Deezer dataset, the gap between ES and RRS is roughly $10^2\times$ for Q_{line4} and $10^3\times$ for Q_{line5}, while both ES and RRS can be computed within 2 seconds. RS and RRS happen to be equal on Q_{line5}, while RRS is only $1.38\times$ larger than RS on Q_{line4}. But it takes about $5,290$ and $16,000$ seconds to compute RS for Q_{line4} and Q_{line5}, because Q_{line4} induces one non-free-connex maximum boundary and Q_{line5} induces three such expensive maximum boundaries.

Lastly, we move to the non-residual-linear join Q_{line6}. Again, RRS is much smaller than ES. Although RRS requires 89.0 seconds to compute, the cost is worthwhile since the gap between RRS and ES is roughly $10^4\times$. Compared to RS, RRS saves $\sim 84,800$ seconds in computation due to some costly maximum boundaries (but required by RS). Not surprisingly, RRS is roughly $3\times$ larger than RS.

Relational Queries. First, let's focus on Q_{tpc3} which is equivalent to Q_{line3}. RRS again degenerates to RS, both of which are much smaller than ES. We rule out ES from subsequent analysis as it matches our observation of graph queries: EScan be computed within seconds, but it leads to the largest sensitivity values. Second, Q_{tpc10} and Q_{tpc21} are equivalent to Q_{line4}, where RRS can be computed up to $10^3\times$ and $10^2\times$ faster than RS. Third, Q_{tpc2} and Q_{tpc7} are equivalent to Q_{line5}. Notice that

RS is completely impractical on Q_{tpc7}, since Q_{tpc7} induces an non-free-connex maximum boundary query over relations Orders and Lineitem, where RS costs more than (estimated) 10^7 seconds to compute and sort the cartesian product product of these two relations (approximately 10^{10} tuples). Finally, Q_{tpc9} is non-residual-linear. RS takes about 5,000 seconds to compute while RRS only takes 34.4 seconds. The reason why RS and RRS are identical for all relational queries is that the TPC-H dataset is generated under key constraints by default, and dropping attributes from boundaries does not increase the aggregation results even for queries that are not residual-linear.

Noisy Join Size. Let $\widetilde{\text{OUT}}_{\text{ES}}$, $\widetilde{\text{OUT}}_{\text{RS}}$ and $\widetilde{\text{OUT}}_{\text{RRS}}$ denote the noisy join size built from ES, RS and RRS. Since $\widetilde{\text{OUT}}$ is proportional to the sensitivity itself, the relative ordering between ES, RS and RRS carry over to that for $\widetilde{\text{OUT}}_{\text{ES}}$, $\widetilde{\text{OUT}}_{\text{RS}}$ and $\widetilde{\text{OUT}}_{\text{RRS}}$, which also matches our observations in Table 4. Different instantiations of $\widetilde{\text{OUT}}$ correspond to different solutions to **Problem 1** in Section 3.3.

6.3 DO Join Evaluation

This subsection compares three DO join algorithms built from ES, RS, RRS, and the performance of the advised FO algorithm [10, 38, 43, 46] when taking different advice.

We first focus on the DO join evaluation. Recall the framework outlined in Section 3. In **Problem 1**, the sensitivity computation dominates the cost of computing $\widetilde{\text{OUT}}$, and the cost of constructing $\widetilde{\text{OUT}}$ following Section 5.3 is almost negligible. In **Problem 2**, we adopt advised FO join algorithms in [10, 38, 43, 46] to compute join results, by taking $\widetilde{\text{OUT}}$ as the advice. The cost is proportional to $\widetilde{\text{OUT}}$. Figures 4 to 9 report the runtime of DO join evaluation on graph queries, and Figures 9(a) to 9(f) report the runtime of DO join evaluation on relational queries. We focus on $\epsilon = 4$.

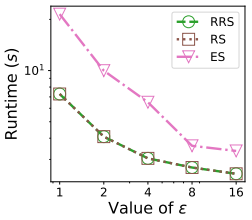


Fig. 4. Q_{line3} on Deezer.

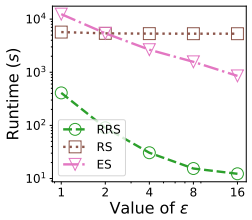


Fig. 5. Q_{line4} on Deezer.

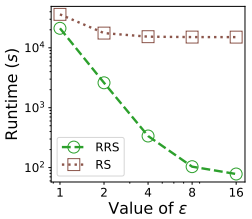


Fig. 6. Q_{line5} on Deezer.

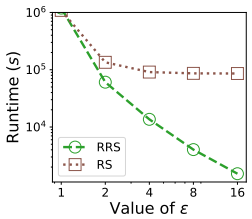


Fig. 7. Q_{line6} on Deezer.

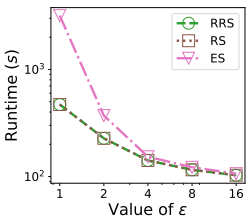


Fig. 8. Q_{star4} on Deezer.

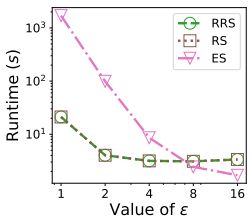
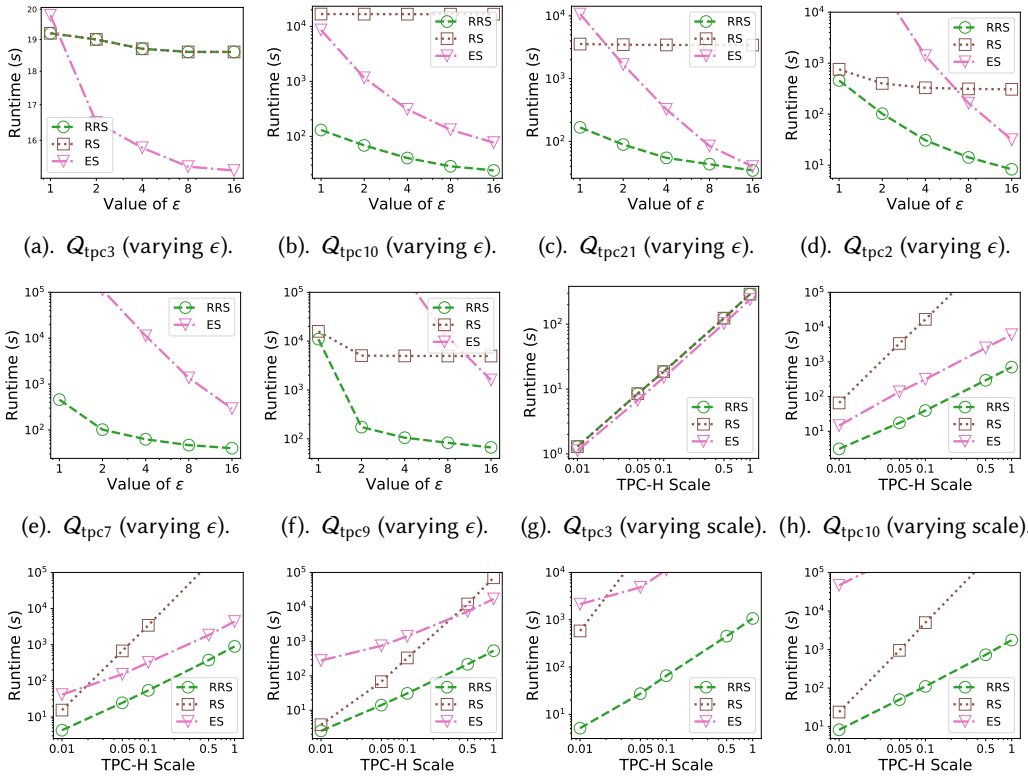


Fig. 9. $Q_{\triangle}$ on Deezer.

Graph Queries. First, Q_{line3}, Q_{star4} and $Q_{\triangle}$ have identical RS and RRS values, whereas their ES values are larger. We have $\widetilde{\text{OUT}}_{\text{RRS}} = \widetilde{\text{OUT}}_{\text{RS}} < \widetilde{\text{OUT}}_{\text{ES}}$. Therefore, RRS-based and RS-based algorithms are more efficient than the ES-based one. Moreover, their performance is dominated by the advised-FO join evaluation, since all sensitivities can be computed in linear time on these three queries.

Second, Q_{line4} and Q_{line5} illustrate the case where RRS-based algorithm outperforms the ES and RS-based ones. For these two queries, $\widetilde{\text{OUT}}_{\text{RRS}}$ and $\widetilde{\text{OUT}}_{\text{RS}}$ are close to each other and dramatically smaller than $\widetilde{\text{OUT}}_{\text{ES}}$, as shown in Table 3. However, the RS-based algorithm is bottlenecked by RS computation, due to the existence of non-free-connex maximum boundaries. As observed, it takes 5,290 seconds to compute RS but only 50 seconds to collect the join result for Q_{line4}. The performance of the ES-based algorithm is dominated by the advised-FO join evaluation, which purely depends on the magnitude of ES. As a result, both RS and ES-based algorithms are $10^2\times$ slower than the RRS-based one. For Q_{line5}, the practicality of the RS-based one is hindered by



(a). Q_{tpc3} (varying ϵ). (b). Q_{tpc10} (varying ϵ). (c). Q_{tpc21} (varying ϵ). (d). Q_{tpc2} (varying ϵ).

(e). Q_{tpc7} (varying ϵ). (f). Q_{tpc9} (varying ϵ). (g). Q_{tpc3} (varying scale). (h). Q_{tpc10} (varying scale).

(i). Q_{tpc21} (varying scale). (j). Q_{tpc2} (varying scale). (k). Q_{tpc7} (varying scale). (l). Q_{tpc9} (varying scale).

Fig. 10. Comparison of TPC-H queries under different privacy budgets ϵ and input scales.

the 16,000 seconds required to compute RS, due to the existence of non-free-connex maximum boundaries. In addition, we have $\widetilde{\text{OUT}}_{\text{ES}} \approx 10^3 \cdot \widetilde{\text{OUT}}_{\text{RRS}}$, which prevents the ES-based algorithm from finishing within our time limit. Thus, it is not shown in Figure 6.

Finally, for Q_{line6}, we exclude ES-based algorithm since $\widetilde{\text{OUT}}_{\text{ES}}$ is $10^4\times$ larger than $\widetilde{\text{OUT}}_{\text{RRS}}$. In Table 3, RRS saves $\sim 84,800$ seconds over RS in sensitivity computation, but $\widetilde{\text{OUT}}_{\text{RRS}} \approx 3 \cdot \widetilde{\text{OUT}}_{\text{RS}}$, leading to a more expensive join step. As a result, the overall runtime of RS- and RRS-based algorithms does not differ much.

Relational Queries. Our observations on relational queries closely mirror those of graph queries. First, RRS-based algorithm outperforms ES-based one on all queries since $\widetilde{\text{OUT}}_{\text{RRS}} \ll \widetilde{\text{OUT}}_{\text{ES}}$. Second, Q_{tpc3} equivalent to Q_{line3}, where three DO algorithms perform comparably. Third, the cost of RRS-based algorithm is dominated by the non-free-connex maximum boundary queries for Q_{tpc2}, Q_{tpc7}, Q_{tpc9}, Q_{tpc10} and Q_{tpc21}. As RS and RRS are same on these queries, the performance gain of RRS over RS is in the sensitivity computation, which varies with respect to the costly maximum boundaries.

Remark. An observation consistent across all queries is the varying gap between RS and RRS-based algorithms. Note that the noise scale factor $f_{\epsilon,\delta}$ of constructing a DP upper bound $\widetilde{\text{OUT}}$ is large, resulting in $\widetilde{\text{OUT}}$ being $2\times \sim 50\times$ larger than $|Q(\mathbf{I})|$. As a result, the outperformance of RRS in sensitivity computation could be diminished by the advised-FO join evaluation, due to a larger advice. We note that any efficient mechanism for computing $\widetilde{\text{OUT}}$ based on sensitivity as presented

in Section 6.1 can significantly reduce the overall runtime. In such cases, the performance gain from RRS in sensitivity computation would become more pronounced.

Breakdown of DO Join Evaluation. We present a breakdown analysis of the DO join algorithm implemented using RRS. We compare our method against the FO (AGM-advised FO) join and the OUT-advised FO join (which additionally leaks the true join size OUT) algorithms in [10, 43, 46]. As shown in Figure 11, FO is the most computationally expensive (exceeding the timeout), OUT-advised FO is the fastest, and DO falls in between. Moreover, the cost of computing OUT, $\widetilde{\text{OUT}}$, or the AGM bound is consistently negligible compared to the corresponding join evaluation step. The runtime of the advised join evaluation phase is proportional to the advice; this is corroborated by the observation that the OUT-advised join is the fastest and the AGM-advised join is the slowest, given that OUT < $\widetilde{\text{OUT}}$ < AGM.

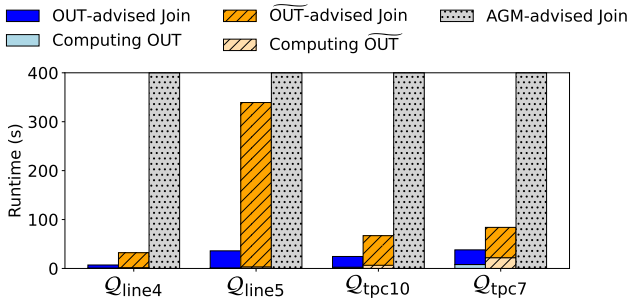


Fig. 11. Breakdown of DO join evaluation and Comparison between different instantiations of advised FO join algorithms [10, 43, 46]. Time limit is set to 400 seconds.

6.4 Privacy Budget

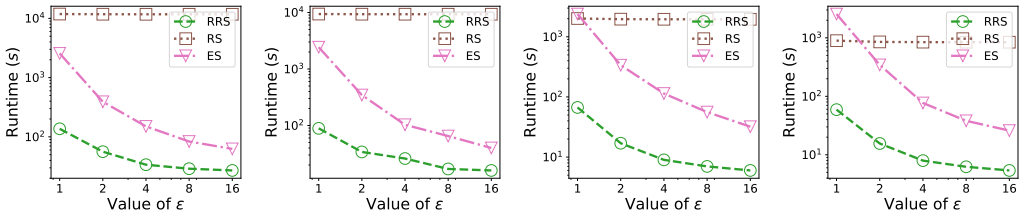
This subsection explores how ϵ affects the performance of DO join algorithms. Figures 4 to 9 and Figures 9(a) to 9(f) report the runtime on graph and relational queries with different values of ϵ.

First, the computation cost of sensitivity increases when ϵ decreases, as we need to enumerate k up to $\frac{m}{1-e^{-\beta}}$ to achieve the optimality of Equation (2) and $\frac{m}{1-e^{-\beta}}$ increases when ϵ decreases. However, such variation is usually negligible compared to the significant variation in the advised-FO join evaluation step.

Second, a smaller ϵ corresponds to a smaller β and a larger $f_{\epsilon,\delta}$. Besides, a smaller β results in a looser (larger) smooth upper bound $\mathsf{S}(\mathbf{I})$. Hence, a smaller ϵ leads to a larger noisy output size $\widetilde{\text{OUT}}$, thereby increasing the cost of the join computation step. In summary, reducing ϵ makes the DO join evaluation more expensive. Furthermore, algorithms based on ES and RRS are more sensitive to ϵ than the RS-based join algorithm. Since their overall runtime is dominated by the join evaluation, which heavily depends on the noisy output size and is particularly affected when ϵ varies. But the overall runtime of the RS-based one is additionally determined by the sensitivity computation if a non-free-connex maximum boundary is involved in the construction of RS, which incurs the same amount of time under different ϵ.

6.5 Scalability

We evaluate the performance of DO join algorithms across varying input sizes on TPC-H datasets. The runtime of as shown in Figures 9(g) to 9(l). As expected, the total runtime of ES-based, RS-based, and RRS-based algorithms increases with respect to the input size. However, the RS-based algorithm is more sensitive to scale growth, since the non-free-connex maximum boundaries in Q_{tpc7} and Q_{tpc9} requires $O(N^2)$ time to compute, which grows quadratically with respect to the input size and affects the overall runtime significantly when the noisy output size is far away from N^2. For ES and RRS-based algorithms, computing ES and RRS only takes $O(N)$ time, and their performance is determined by the noisy join size of each query, which increases almost linearly with respect to



(a). Ratio = 5. (b). Ratio = 10. (c). Ratio = 50. (d). Ratio = 100.

Fig. 12. Q_{line4} on Deezer under different unbalanced ratios.

Fig. 13. Access patterns of computing upper bounds on the maximum boundary of $E = \{2, 3\}$ by [10].

Fig. 14. Access patterns of computing the join $R_1(A_1, A_2) \bowtie R_2(A_2, A_3)$ by [10, 43, 46] with advice $\widetilde{\text{OUT}} = 8$.

input scale on TPC-H. Consequently, they display a gradual increase in overall runtime compared to the RS-based algorithm.

Additionally, we investigate how different input sizes affect the performance by evaluating Q_{line4} across four unbalanced datasets constructed from Deezer, where the largest relation is always R_3, and the smallest relation is always R_2. The trends of RRS, ES, and RS in Figures 11(a) to 11(d) match the balanced case in Figure 5. The RRS-based algorithm consistently outperforms the other two. However, the gap between RS and RRS and the gap between RS and ES decrease as the ratio increases. The reason is that the only non-free-connex maximum boundary query in RS involves R_2 and R_3, and the cardinality of $R_2 \times R_3$ decreases as the ratio increases.

6.6 Access Pattern Instrumentation

We empirically verify the obliviousness of our DO join evaluation on a small synthetic dataset for Q_{line4}. Each relation contains 5 tuples. There are 3 join result tuples. We adopt the noisy join size $\widetilde{\text{OUT}}$ as 8. As the complete access patterns is too large to show, we only select a few representative results in Figures 13 and 14. The horizontal axis represents the access order, the vertical axis represents the discretized memory addresses, each grey bar denotes read, and each black bar denotes write. In Figure 13, the maximum boundary $T_{\{2,3\}}$ is computed in an FO way. In Figure 14, the two-way join $R_1(A_1, A_2) \bowtie R_2(A_2, A_3)$ is computed by an advised-FO algorithm with the advice $\widetilde{\text{OUT}} = 8$. The complete access pattern only depends on the input size 5 and the noisy join size 8.

7 Related Work

FO Join Processing. The nested-loop join, which computes the cross product of input relations, is FO, but runs in $O(N^m)$ time. Li and Chen [44] studied the FO two-way θ-join, but their algorithm is not asymptotically faster than the nested-loop join. Meanwhile, incorporating the insecure worst-case optimal join algorithms [49] into the ORAM framework yields an FO join algorithm.

[38] recently rewrote the worst-case optimal join algorithm into an FO one without using ORAM. Wang and Yi [63] designed a circuit of $O(N^{\rho} \cdot \text{polylog}\, N)$ gates for joins, which is also FO.

Beyond FO. Relaxed notions of FO that allow specific types of leakage, such as join size, multiplicity of join values, and the size of intermediate results, have also been explored. One relevant line of work examines join algorithms while allowing the leakage of join size. Integrating an insecure output-sensitive join algorithm into an ORAM framework yields a relaxed FO algorithm; for example, [21] implemented an index-based join algorithm together with Path ORAM [58, 60], and Arasu and Kaushik [10] proposed an algorithm for acyclic multi-way joins without using ORAM. [43, 46] further explored efficient implementations of the algorithm in [10]. Opaque [69] and ObliDB [29] are two end-to-end oblivious data systems that support distributed query processing and dynamic data updates, respectively. Shrinkwarp [18] protects the sizes of intermediate results of a multi-way join, while still leaking the join size. Adore [54] delivers a set of efficient DO relational operators with $O(\log N)$ trusted memory, including selection with projection, grouping with aggregation, and foreign-key joins. Doquet [55] introduces a system that supports DO two-way joins with runtime matching [23] asymptotically and only uses $O(1)$ trusted memory.

In addition to joins, FedKNN [66] investigated differentially oblivious k-nearest neighbor search. Beimel et al. [19] studied the graph property testing problem. Other problems, such as composition theory [70], Turing machine [41], sorting and prefix sum [20], have also been discussed. There are some other notions of obliviousness [34, 36, 47] proposed, but incomparable to differential obliviousness. Besides, protecting access pattern has also been studied in the multi-party computation setting [14, 17, 37, 45, 51, 52, 62].

8 Conclusion

We present a general framework for designing DO join algorithms and introduce our first construction based on a novel concept of relaxed residual sensitivity. This is a step towards achieving a better privacy-efficiency tradeoff in join processing by mitigating side-channel attacks from memory access patterns. Several intriguing follow-up questions remain.

For instance, while our focus has been on constructing a sensitivity that can be computed in linear time, this is not always a strict requirement and can be lifted in future exploration. Moreover, we focus on the tuple-based DO setting, which does not account for foreign key constraints or bulk changes. Future work can explore a "user-based" or "group-based" DO model to redefine neighboring databases to accommodate these different application scenarios. As the concept of DO shifts toward instance-dependent algorithms, it creates new opportunities for instance-dependent query optimization in DO join evaluation, for example, choosing a good join order while preserving DO privacy. Lastly, our implementation represents just one specific instantiation of our general framework, primarily aimed at demonstrating the new notion of relaxed residual sensitivity. To facilitate the practical deployment of DO join algorithms, it will be essential to empirically evaluate various instantiations of our framework and choose the best based on the application.

We also emphasize that our DO join framework is modular; any improvement in its underlying components could automatically enhance the performance of DO join algorithms, such as the very recent work on calibrating positive noise for computing a DP upper bound on the join size [64].

Acknowledgement

This work was supported by the Natural Sciences and Engineering Research Council of Canada (NSERC) Discovery Grant, the NTU–NAP Startup Grant (024584-00001), and the Singapore Ministry of Education Tier 1 Grant (RG19/25). We would also like to thank the anonymous reviewers who have made valuable suggestions on improving the presentation of the paper.

References

[1] Amazon. https://aws.amazon.com/redshift/.
[2] Code. https://github.com/z46wu/DOJoin.
[3] Full version. https://github.com/z46wu/DOJoin.
[4] Google. https://cloud.google.com/.
[5] Microsoft. https://azure.microsoft.com/.
[6] TPC-H. https://www.tpc.org/tpch/.
[7] Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of databases*. Vol. 8. Addison-Wesley Reading.
[8] Mahmoud Abo Khamis, Hung Q Ngo, and Atri Rudra. 2016. FAQ: questions asked frequently. In *PODS*. 13–28.
[9] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravishankar Ramamurthy, and Ramarathnam Venkatesan. 2013. Orthogonal Security with Cipherbase. In *CIDR*.
[10] Arvind Arasu and Raghav Kaushik. 2013. Oblivious query processing. *ICDT* (2013).
[11] Gilad Asharov, Ilan Komargodski, Wei-Kai Lin, Kartik Nayak, Enoch Peserico, and Elaine Shi. 2020. Optorama: Optimal oblivious ram. In *Eurocrypt*. Springer, 403–432.
[12] Gilad Asharov, Ilan Komargodski, and Yehuda Michelson. 2023. FutORAMa: A Concretely Efficient Hierarchical Oblivious RAM. In *Proceedings of the 2023 ACM SIGSAC Conference on Computer and Communications Security* (Copenhagen, Denmark) *(CCS '23)*. Association for Computing Machinery, New York, NY, USA, 3313–3327. doi:10.1145/3576915.3623125
[13] Albert Atserias, Martin Grohe, and Dániel Marx. 2008. Size bounds and query plans for relational joins. In *FOCS*. IEEE, 739–748.
[14] Saikrishna Badrinarayanan, Sourav Das, Gayathri Garimella, Srinivasan Raghuraman, and Peter Rindal. 2022. Secret-shared joins with multiplicity from aggregation trees. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 209–222.
[15] Guillaume Bagan, Arnaud Durand, and Etienne Grandjean. 2007. On acyclic conjunctive queries and constant delay enumeration. In *CSL*. Springer, 208–222.
[16] Sumeet Bajaj and Radu Sion. 2013. TrustedDB: A trusted hardware-based database with privacy and data confidentiality. *TKDE* 26, 3 (2013), 752–765.
[17] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. 2017. smcql: Secure Querying for Federated Databases. *Proc. VLDB Endow.* 10, 6 (2017).
[18] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: Efficient SQL Query Processing in Differentially Private Data Federations. *Proc. VLDB Endow.* 12, 3 (2018), 307–320.
[19] Amos Beimel, Kobbi Nissim, and Mohammad Zaheri. 2019. Exploring Differential Obliviousness. In *APPROX/RANDOM*.
[20] T.-H. Hubert Chan, Kai-Min Chung, Bruce Maggs, and Elaine Shi. 2022. Foundations of Differentially Oblivious Algorithms. *J. ACM* 69, 4, Article 27 (Aug. 2022), 49 pages. doi:10.1145/3555984
[21] Zhao Chang, Dong Xie, Sheng Wang, Feifei Li, and Yulong Shen. 2024. Towards Practical Oblivious Join Processing. *IEEE Transactions on Knowledge and Data Engineering* 36, 4, 1829–1842. doi:10.1109/TKDE.2023.3310038
[22] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel TDX Demystified: A Top-Down Approach. *ACM Comput. Surv.* 56, 9, Article 238 (April 2024), 33 pages. doi:10.1145/3652597
[23] Shumo Chu, Danyang Zhuo, Elaine Shi, and T-H. Hubert Chan. 2021. Differentially Oblivious Database Joins: Overcoming the Worst-Case Curse of Fully Oblivious Algorithms. In *ITC*, Vol. 199. 19:1–19:24.
[24] 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 *Proc. ACM SIGMOD International Conference on Management of Data*.
[25] Wei Dong and Ke Yi. 2021. Residual Sensitivity for Differentially Private Multi-Way Joins. In *Proceedings of the 2021 International Conference on Management of Data*. 432–444.
[26] Wei Dong and Ke Yi. 2022. A Nearly Instance-optimal Differentially Private Mechanism for Conjunctive Queries. In *PODS*.
[27] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *TCC*. 265–284.
[28] Muhammad El-Hindi, Tobias Ziegler, Matthias Heinrich, Adrian Lutsch, Zheguang Zhao, and Carsten Binnig. 2022. Benchmarking the Second Generation of Intel SGX Hardware. In *Proceedings of the 18th International Workshop on Data Management on New Hardware* (Philadelphia, PA, USA) *(DaMoN '22)*. Association for Computing Machinery, New York, NY, USA, Article 5, 8 pages. doi:10.1145/3533737.3535098
[29] Saba Eskandarian and Matei Zaharia. 2019. ObliDB: oblivious query processing for secure databases. *Proceedings of the VLDB Endowment* 13, 2, 169–183.

[30] Daniela Florescu, Alon Levy, Ioana Manolescu, and Dan Suciu. 1999. Query optimization in the presence of limited access patterns. *SIGMOD Rec.* 28, 2 (June 1999), 311–322. doi:10.1145/304181.304210

[31] Congcong Fu, Hui Li, Jian Lou, Huizhen Li, and Jiangtao Cui. 2023. DP-starJ: A Differential Private Scheme towards Analytical Star-Join Queries. *Proc. ACM Manag. Data* 1, 4, Article 238 (Dec. 2023), 24 pages. doi:10.1145/3626725

[32] Oded Goldreich. 1987. Towards a theory of software protection and simulation by oblivious RAMs. In *STOC*. 182–194.

[33] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *JACM* 43, 3 (1996), 431–473.

[34] Dov Gordon, Jonathan Katz, Mingyu Liang, and Jiayu Xu. 2022. Spreading the privacy blanket: Differentially oblivious shuffling for differential privacy. In *International Conference on Applied Cryptography and Network Security*. Springer, 501–520.

[35] Georg Gottlob, Gianluigi Greco, Nicola Leone, and Francesco Scarcello. 2016. Hypertree Decompositions: Questions and Answers. In *Proceedings of the 35th ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (San Francisco, California, USA) *(PODS '16)*. Association for Computing Machinery, New York, NY, USA, 57–74. doi:10.1145/2902251.2902309

[36] Paul Grubbs, Anurag Khandelwal, Marie-Sarah Lacharité, Lloyd Brown, Lucy Li, Rachit Agarwal, and Thomas Ristenpart. 2020. Pancake: Frequency smoothing for encrypted data stores. In *29th USENIX Security Symposium (USENIX Security 20)*. 2451–2468.

[37] Feng Han, Lan Zhang, Hanwen Feng, Weiran Liu, and Xiangyang Li. 2022. Scape: Scalable collaborative analytics system on private database with malicious security. In *2022 IEEE 38th International Conference on Data Engineering (ICDE)*. IEEE, 1740–1753.

[38] Xiao Hu and Zhiang Wu. 2025. Optimal Oblivious Algorithms for Multi-Way Joins. In *28th International Conference on Database Theory (ICDT 2025) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 328)*, Sudeepa Roy and Ahmet Kara (Eds.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 25:1–25:19. doi:10.4230/LIPIcs.ICDT.2025.25

[39] Noah Johnson, Joseph P Near, and Dawn Song. 2018. Towards practical differential privacy for SQL queries. *Proc. VLDB Endow.* 11, 5 (2018), 526–539.

[40] Vishesh Karwa, Sofya Raskhodnikova, Adam Smith, and Grigory Yaroslavtsev. 2014. Private analysis of graph structure. *TODS* 39, 3 (2014), 1–33.

[41] Ilan Komargodski and Elaine Shi. 2021. Differentially Oblivious Turing Machines. In *12th Innovations in Theoretical Computer Science Conference (ITCS 2021) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 185)*, James R. Lee (Ed.). Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 68:1–68:19. doi:10.4230/LIPIcs.ITCS.2021.68

[42] Ios Kotsogiannis, Yuchao Tao, Xi He, Maryam Fanaeepour, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2019. PrivateSQL: a differentially private SQL query engine. *Proc. VLDB Endow.* (July 2019). doi:10.14778/3342263.3342274

[43] Simeon Krastnikov, Florian Kerschbaum, and Douglas Stebila. 2020. Efficient oblivious database joins. *Proc. VLDB Endow.* 13, 12 (July 2020), 2132–2145. doi:10.14778/3407790.3407814

[44] Yaping Li and Minghua Chen. 2008. Privacy preserving joins. In *ICDE*. IEEE, 1352–1354.

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

[46] Apostolos Mavrogiannakis, Xian Wang, Ioannis Demertzis, Dimitrios Papadopoulos, and Minos Garofalakis. 2025. OBLIVIATOR: oblivious parallel joins and other operators in shared memory environments. In *Proceedings of the 34th USENIX Conference on Security Symposium* (Seattle, WA, USA) *(SEC '25)*. USENIX Association, USA, Article 437, 20 pages.

[47] Sahar Mazloom and S Dov Gordon. 2018. Secure computation with differentially private access patterns. In *CCS*. 490–507.

[48] 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)*. 279–296. doi:10.1109/SP.2018.00045

[49] Hung Q Ngo, Ely Porat, Christopher Ré, and Atri Rudra. 2018. Worst-case optimal join algorithms. *JACM* 65, 3 (2018), 1–40.

[50] Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2007. Smooth sensitivity and sampling in private data analysis. In *STOC*. 75–84.

[51] 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 *USENIX Security*. 2129–2146.

[52] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: protecting confidentiality with encrypted query processing. In *SOSP*. 85–100.

[53] Davide Proserpio, Sharon Goldberg, and Frank McSherry. 2014. Calibrating data to sensitivity in private data analysis: a platform for differentially-private analysis of weighted datasets. *Proc. VLDB Endow.* 7, 8 (April 2014), 637–648. doi:10.14778/2732296.2732300

[54] Lianke Qin, Rajesh Jayaram, Elaine Shi, Zhao Song, Danyang Zhuo, and Shumo Chu. 2022. Adore: Differentially Oblivious Relational Database Operators. *Proceedings of the VLDB Endowment* 16, 4 (2022), 842–855.

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

[56] Anand Rajaraman, Yehoshua Sagiv, and Jeffrey D. Ullman. 1995. Answering queries using templates with binding patterns (extended abstract). In *Proceedings of the Fourteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems* (San Jose, California, USA) *(PODS '95)*. Association for Computing Machinery, New York, NY, USA, 105–112. doi:10.1145/212433.220199

[57] Benedek Rozemberczki and Rik Sarkar. 2020. Characteristic Functions on Graphs: Birds of a Feather, from Statistical Descriptors to Parametric Models. In *Proceedings of the 29th ACM International Conference on Information and Knowledge Management (CIKM '20)*. ACM, 1325–1334.

[58] Elaine Shi. 2020. Path oblivious heap: Optimal and practical oblivious priority queue. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 842–858.

[59] Avi Silberschatz, Henry F. Korth, and S. Sudarshan. 2020. *Database System Concepts, Seventh Edition*. McGraw-Hill Book Company. https://www.db-book.com/

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

[61] Yuchao Tao, Xi He, Ashwin Machanavajjhala, and Sudeepa Roy. 2020. Computing Local Sensitivities of Counting Queries with Joins. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data* (Portland, OR, USA) *(SIGMOD '20)*. Association for Computing Machinery, New York, NY, USA, 479–494. doi:10.1145/3318464.3389762

[62] Yilei Wang and Ke Yi. 2021. Secure Yannakakis: Join-Aggregate Queries over Private Data. In *SIGMOD*. 1969–1981.

[63] Yilei Wang and Ke Yi. 2022. Query Evaluation by Circuits. In *Proceedings of the 41st ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems*. 67–78.

[64] Hanshen Xiao, Jun Wan, Elaine Shi, and Srinivas Devadas. 2025. One-Sided Bounded Noise: Theory, Optimization Algorithms and Applications. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security* (Taipei, Taiwan) *(CCS '25)*. Association for Computing Machinery, New York, NY, USA, 4214–4228. doi:10.1145/3719027.3765110

[65] Mihalis Yannakakis. 1981. Algorithms for acyclic database schemes. In *VLDB*, Vol. 81. 82–94.

[66] Xinyi Zhang, Qichen Wang, Cheng Xu, Yun Peng, and Jianliang Xu. 2024. FedKNN: Secure Federated k-Nearest Neighbor Search. *Proc. ACM Manag. Data* 2, 1, Article 11 (March 2024), 26 pages.

[67] Hangdong Zhao, Shaleen Deep, and Paraschos Koutris. 2023. Space-Time Tradeoffs for Conjunctive Queries with Access Patterns. In *Proceedings of the 42nd ACM SIGMOD-SIGACT-SIGAI Symposium on Principles of Database Systems* (Seattle, WA, USA) *(PODS '23)*. Association for Computing Machinery, New York, NY, USA, 59–68. doi:10.1145/3584372.3588675

[68] Leqian Zheng, Zheng Zhang, Wentao Dong, Yao Zhang, Ye Wu, and Cong Wang. 2025. *H2O2RAM: a high-performance hierarchical doubly oblivious RAM*. USENIX Association, USA.

[69] 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 *NSDI 17*. 283–298.

[70] Mingxun Zhou, Elaine Shi, T-H Hubert Chan, and Shir Maimon. 2023. A theory of composition for differential obliviousness. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 3–34.

Received October 2025; revised January 2026; accepted February 2026