SAQE: Practical Privacy-Preserving Approximate Query Processing for Data Federations

Johes Bater
Northwestern University
johes@u.northwestern.edu

Yongjoo Park
University of Illinois (UIUC)
yongjoo@illinois.edu

Xi He
University of Waterloo
xi.he@uwaterloo.ca

Xiao Wang
Northwestern University
wangxiao@northwestern.edu

Jennie Rogers
Northwestern University
jennie@northwestern.edu

ABSTRACT

A private data federation enables clients to query the union of data from multiple data providers without revealing any extra private information to the client or any other data providers. Unfortunately, this strong end-to-end privacy guarantee requires cryptographic protocols that incur a significant performance overhead as high as 1,000× compared to executing the same query in the clear. As a result, private data federations are impractical for common database workloads. This gap reveals the following key challenge in a private data federation: offering significantly fast and accurate query answers without compromising strong end-to-end privacy.

To address this challenge, we propose SAQE, the Secure Approximate Query Evaluator, a private data federation system that scales to very large datasets by combining three techniques — differential privacy, secure computation, and approximate query processing — in a novel and principled way. First, SAQE adds novel secure sampling algorithms into the federation's query processing pipeline to speed up query workloads and to minimize the noise the system must inject into the query results to protect the privacy of the data. Second, we introduce a query planner that jointly optimizes the noise introduced by differential privacy with the sampling rates and resulting error bounds owing to approximate query processing.

Our research shows that these three techniques are synergistic: sampling within certain accuracy bounds improves both query privacy and performance, meaning that SAQE executes over less data than existing techniques without sacrificing efficiency, privacy, or accuracy. Using our optimizer, we leverage this counter-intuitive result to identify an inflection point that maximizes all three criteria prior query evaluation. Experimentally, we show that this result enables SAQE to trade-off among these three criteria to scale its query processing to very large datasets with accuracy bounds dependent only on sample size, and not the raw data size.

PVLDB Reference Format:
Johes Bater, Yongjoo Park, Xi He, Xiao Wang, Jennie Rogers. SAQE: Practical Privacy-Preserving Approximate Query Processing for Data Federations. *PVLDB*, 13(11): 2691-2705, 2020.
DOI: https://doi.org/10.14778/3407790.3407854

Proceedings of the VLDB Endowment, Vol. 13, No. 11
ISSN 2150-8097.
DOI: https://doi.org/10.14778/3407790.3407854

1. INTRODUCTION

Querying the union of multiple private data stores is challenging due to the need to compute over the combined datasets without data providers disclosing their secret query inputs to anyone. Here, a *client* issues a query against the union of these private records and he or she receives the output of their query over this shared data. Presently, systems of this kind use a trusted third party to securely query the union of multiple private datastores. For example, some large hospitals in the Chicago area offer services for a certain percentage of the residents; if we can query the union of these databases, it may serve as invaluable resources for accurate diagnosis, informed immunization, timely epidemic control, and so on. Yet, their databases stay siloed by default; these hospitals (i.e., data providers) are reticent to share their data with one another, fearing the sensitive health records of their patients may be breached in the process of carelessly data sharing. To address this, we need a principled approach to querying over multiple private datastores.

Private Data Federations. A private data federation [9, 58] is a database system that offers end-to-end privacy guarantees for querying the union of multiple datastores. In other words, a private data federation ensures that the secret inputs of each data provider are accessible only to him or her 1) before, 2) during, and 3) after query processing. *Before* query processing, the system makes all query optimization decisions in a data-independent manner; no one can learn anything about the data by examining the query's operators or parameters. *During* query processing, the system uses secure computation protocols that compute over the private data of multiple parties such that none of them learn the secret inputs of their peers; the only information that is revealed is that which can be deduced from the query output. Finally, *after* query processing, the system adds carefully calibrated noise to the query results using *differential privacy* [22] such that an adversary viewing this output will get essentially the same answer whether or not an individual chose to make his or her data available for querying. This system has an integrated suite of differential privacy mechanisms for modeling this controlled information leakage both during and after query processing.

Existing differential privacy systems [36,46] with a single trusted data curator do not have to protect data during query processing. Conversely, a private data federation must add noise using secure computation [10] so that no party can deduce the true query answer and any client that colludes with a data provider is unable to break the system's privacy guarantees, even under repeated querying of a single dataset. To uphold this strong privacy guarantee of the input data, private data federations must add noise to query results, reducing accuracy, and thereby compromising their utility.

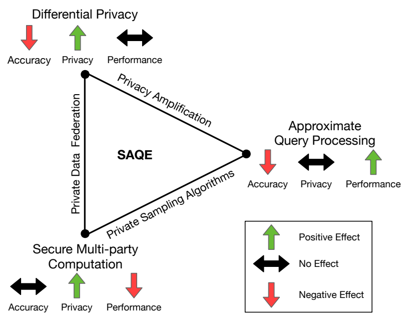


Figure 1: Key components of SAQE query processing

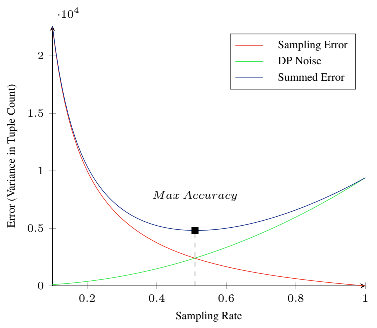


Figure 2: Expected error for `COUNT` query results in SAQE, $n = 10,000$, $\epsilon = 0.05$, and $\delta = 10^{-5}$.

Secure Computation.. We use secure computation to obliviously compute over the data of two or more data providers. An oblivious program's observable transcript (i.e., its program counter, memory access patterns, network transmissions) leaks nothing about its input data because these signals are data-independent. Oblivious query processing incurs extremely high overhead owing to its use of heavyweight cryptographic protocols to encrypt and process input data. In practice, an oblivious query execution runs in worst-case time to uphold rigorous security guarantees. For example, a join with inputs of length n will have an output cardinality of n^2. Oblivious database operators produce worst-case output cardinalities by padding their results with dummy tuples. These cryptographic protocols also incur significant computation and network overhead to run with their data encrypted in flight. Queries over secure computation have runtimes that are multiple orders of magnitude slower than running the same query with no security. Hence, systems that use secure computation alone to protect data under computation [9, 58] do not scale to datasets that are greater than hundreds of megabytes.

Approximate Query Processing. When executing queries over extremely large data sets, big data systems speed up execution by using approximate query processing [3, 20, 35, 38, 49], evaluating a query over a sample of its input data rather than computing the query exhaustively. On the face of it, approximate query processing offers an attractive way to scale up private data federation queries to large datasets. However, naïvely executing private data federation queries over a small data sample fails on two fronts: privacy and accuracy. Privacy suffers because when creating a sample, data providers reveal exact sizes of their data. An unauthorized observer can use this knowledge, along with public information, to decrypt a data provider's secure computation. Accuracy suffers because unlike regular approximate query processing, additional noise must be added for differential privacy, which can make the end-to-end error unnecessarily (and extremely) high.

In Figure 2, we plot the noise contributions of approximate query processing and differential privacy as a function of the sampling rate. This plot reveals an interesting relationship between the two. Under certain conditions, sampling actually *improves* accuracy as well as efficiency, meaning that including less data in the computation increases accuracy. We see this result because of the dependency between differential privacy and sample size. A lower sampling rate reduces the size of the sample and introduces more sampling error, thus requiring less additional noise to satisfy differential privacy. When we do not sample the data, differential privacy requires more noise to avoid information leakage. Balancing approximate query processing and differential privacy within a private data federation to maximize query result accuracy is a difficult problem and requires careful sampling and query optimization.

Our Approach. We propose SAQE, the Secure Approximate Query Evaluator, a system that brings scalability to privacy-preserving SQL querying over private data federations by generalizing approximate query processing to this setting. Unlike the naïve approach, SAQE generalizes approximate query processing to this setting by carefully composing it with differential privacy and secure computation, making their interactions synergistic through an integrated model of their performance profiles and information leakage. We illustrate SAQE's composition of these techniques in Figure 1.

First, SAQE ensures sampling-time privacy by introducing two *private sampling algorithms*. That is, the naïve approach— sampling locally at each data provider then computing on all the samples using secure computation—reveals the exact size of each sample and the data provider knows their precise contribution to the query results. A curious data provider could use this knowledge to deduce unauthorized information about private data from query results. Our private sampling algorithms prevent this as follows. In our first private sampling algorithm (i.e., an oblivious sampling algorithm), the data providers work together in secure computation to winnow down their input data to a set of samples for use in query evaluation without leaking any information. It outperforms a straw-man method by reducing the sampling cost from $O(n \log^2 n)$ to $O(n \log n)$, where n is the size of the input data. The second algorithm uses computational differential privacy to add dummy tuples to the samples, hiding their true cardinalities. Hence, the revealed sample sizes use differential privacy to control the information leakage associated with running a faster sampling algorithm while upholding SAQE's end-to-end privacy guarantees.

Second, SAQE introduces a model for composing approximate query processing and differential privacy to maximize the accuracy of its query results. Differential privacy adds noise to the query results to protect the secret input data. In addition, approximate query processing naturally has imprecise query results since it computes over samples of the source data. In SAQE, we reveal how to leverage these two sources of noise synergistically; the error introduced by approximate query processing reduces the noise SAQE must add to the query results to uphold the privacy of their source data. To optimally harness these correlated errors, we first model the composition of those two noise sources by generalizing *privacy amplification* [8, 15, 24] to private data federations. This composition makes it possible for our noise sources to be synergistic. By modeling approximate query processing and differential privacy jointly sampling does not need to reduce our query accuracy – since differential privacy's noise is a sunk cost for this figure. At the same time, SAQE realizes faster query runtimes via sampling. SAQE extends privacy amplification with a novel cost model and optimizer that accounts for the error introduced by each technique during and after query execution. Our optimizer identifies the inflection point between differential privacy and approximate query processing that minimizes query result error before runtime, choosing the optimal sampling rate for a given query.

Contributions. In this paper, we introduce SAQE, the first system for practical privacy-preserving SQL query processing. The main technical contributions in this work are:

- Identify and formalize privacy-preserving SQL query processing over sampled data with provable privacy, accuracy, and performance guarantees.
- Synergistic cost model and optimization framework that composes differential privacy with approximate query processing to identify optimal sampling rates, enabling scaling to massive data sizes for private data federations.
- Novel oblivious and DP sampling algorithms for SQL queries that combine secure computation and approximate query processing with controlled information leakage, ensuring efficient end-to-end privacy throughout query processing.
- An end-to-end system evaluation using both real hospital data and workloads as well as synthetic data and queries.

Paper Organization. In Section 2 we provide background on private data federations and SAQE's security primitives: secure computation, differential privacy, and approximate query processing. Section 3 gives an overview of SAQE's goals and architecture. Section 4 articulates our novel privacy-preserving sampling algorithms. After that, Section 5 describes the system's cost model and query optimizer. We evaluate SAQE over real-world and synthetic data in Section 6. Lastly, we discuss related work and conclude.

2. PRIVATE DATA FEDERATIONS

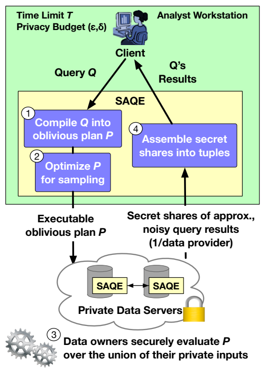
In this section, we describe detailed setup of SAQE, including the architecture, end-to-end privacy guarantee, and query lifecycle.

2.1 System Properties

A data federation makes multiple, autonomous database systems available for querying via a unified SQL interface. Here, a client queries the union of these databases as if all of their records are stored in a single engine. A federation has a query coordinator that rewrites client queries for distributed execution and orchestrates their processing among the data providers. Data federations are either homogeneous – wherein the data providers support a single schema and the federation's data is horizontally partitioned amongst them – or heterogeneous such that the data providers each may bring different tables to the federation. We operate in the homogeneous setting for this work.

A *private data federation* [9] is a generalization of conventional data federations in which m data providers, $D_1, \dots D_m$, pool their private data for analysis while each site maintains exclusive access to their inputs, $D_1, \dots D_m$, respectively. Each data provider wishes to keep individual tuples in their private datastore confidential. They are willing to let a client see the results of a query over the union of their datasets, $D = \cup_{i=1}^{m} D_i$. All $D_i \in D$ share a public schema over k relations $(R_1, ..., R_k)$. The private data, D, is horizontally partitioned over the data providers. The private data federation's client-side coordinator plans and orchestrates the execution of its queries to ensure that queries uphold the security policy while running efficiently.

Configuration. Before a private data federation accepts its first query, it performs a two-stage setup process. First, data providers do private record linkage [29] over individuals or entities that have records spanning two or more databases so that each one resolves to a single identifier. This provides a weaker privacy guarantee that is composable with differential privacy to protect non-matching records. Our paper focuses on privacy loss during execution and leaves optimization of private record linkage to future work.



Time Limit *T*
Privacy Budget (ε,δ)
Analyst Workstation
Client
Query *Q*
Q's Results
SAQE
1 Compile *Q* into oblivious plan *P*
2 Optimize *P* for sampling
4 Assemble secret shares into tuples
Executable oblivious plan *P*
Secret shares of approx., noisy query results (1/data provider)
SAQE
SAQE
Private Data Servers
3 Data owners securely evaluate *P* over the union of their private inputs

Figure 3: SAQE query workflow

Second, the data providers work together to create a security policy based on the best practices of their domain and any applicable regulations. They initialize a common data model with this. These shared table definitions, supported by all data providers are annotated with one security policy per column. This determines the information data providers are permitted to learn when processing a SAQE query. A column may be *public*, or readable by all parties. Otherwise it is *private* and data providers running queries over these columns learn either a) nothing about the data of their peers or b) differentially private information about the data distribution by observing the intermediate cardinalities of its operators. To do this, the private data federation executes query operators in plaintext or obliviously. As tuples move up the query tree, we use the information flow analysis of SMCQL [9] to determine the execution mode each operator needs to run with protect the columns upon which they compute and any private outputs from their children. Hence, after a query evaluates an operator obliviously it cannot go back to public query execution. Client access control, or determining the queries to admit from a query writer and the query results to which they have access, are outside the scope of this work. In addition, the engine computes differentially-private metadata over the unioned records of all data providers using a discrete slice of the overall privacy budget, $(\epsilon_{setup}, \delta_{setup})$. It generates a noisy table size for the query inputs, $\tilde{n}$. Also, if the query workload uses group-by aggregation, the system generates noisy histograms for the group-by attributes to support stratified sampling. If we have a group-by clause over g, then our metadata has a set privacy-preserving strata sizes, $(\tilde{n}_{g,1}, \dots, \tilde{n}_{g,N_{strata}})$.

2.2 End-to-end Privacy Guarantee

A private data federation must protect data before, during, and after query processing. SAQE's optimizer protects the contents of its member databases prior to query processing by making query planning decisions without accessing query's inputs or statistics. Next, it protects private query inputs during query evaluation by ensuring that any information that the data providers or clients glean from observing a query execution is differentially private. Existing systems [9, 10] address this by evaluating queries using secure computation, which we describe in Section 2.3, but they do so very slowly due to processing the entirety of the query's input data. Lastly, the system protects the query's output so that the results do not leak information about the query's inputs. Prior work [10] leverages differential privacy mechanisms, described in Section 2.3 to add noise to query results, thwarting attacks that may use this information to deduce the contents of the federation's inputs. SAQE generalizes on this to add less noise to the query's results by exploiting the inherent noise in approximate query processing.

SAQE provides end-to-end privacy guarantees in the private data federation (see Figure 3). In detail:

- *Clients* only learn the differentially-private output of a query.
- *Data providers* may only learn noisy information about the distribution of the data provided by their peers. They observe this controlled information leakage provided by a differential privacy mechanism during query evaluation.
- If a data provider colludes with a client, neither party will learn additional private information owing to SAQE's composition of differential privacy techniques in the query's evaluation and its noising of its results within secure computation.

This system supports a semi-honest adversary who may corrupt any subset of the data providers and the clients. SAQE guarantees that the inputs from the uncorrupted parties remains private. Client queries, as well as the common data model, are public.

2.3 System Primitives

Secure Computation refers to cryptographic protocols with which a set of mutually distrusting parties jointly compute a function without revealing any individual's input. Secure multi-party computation has witnessed a huge improvement in efficiency and security in recent years, with real-life applications such as auctions [13], distributed key management [40], and anonymous aggregation [52]. Intuitively, secure computation enables function evaluation over multiple private data providers without requiring them to share data with a trusted third party that unions their records for query evaluation. This security comes at a cost – evaluating a program using secure computation remains orders of magnitude slower than non-private evaluation. A major task of this paper is to explore database techniques that accelerate the performance of secure computation over realistic OLAP workloads.

The vast majority of secure computation protocols represent the function they compute as a circuit. This circuit representation enables secure programs to run with an observable transcript that is data-independent. In other words, the order of operations is static and independent of the input. More generally, we say that a function f is *data-oblivious* if for any valid inputs x_1 and x_2, the induced distribution on the data access pattern is indistinguishable.

In this paper, we design SAQE to: 1) use secure computation efficiently by avoiding and/or reducing the use of heavyweight cryptographic protocols as much as possible; and 2) use more efficient oblivious algorithms (for sampling in this context) to reduce the number of gates that a query must run within secure computation. Designing efficient oblivious algorithms is a difficult task because their operations need to be deterministic. This frequently introduces dummy operations to prevent a curious observer from deducing information about a query's secret inputs by observing the program counter during branching, looping, and other program flow. Indeed, oblivious algorithms always run slower than non-oblivious ones and sometimes are even asymptotically less efficient.

Differential Privacy [22, 23] provides a strong privacy guarantee to individuals in the database while supporting multiple releases of statistics about the data. It has been deployed by government organizations such as the US Census Bureau [42] and industry including Google [25], Uber [34], and others. SAQE uses (ϵ, δ)-differential privacy, and formally, its guarantees are as follows:

DEFINITION 1 ((ϵ, δ)-DIFFERENTIAL PRIVACY). *A randomized mechanism $M : \mathcal{D} \rightarrow \mathcal{R}$ satisfies (ϵ, δ)-differential privacy if for any pair of neighboring databases $D, D' \in \mathcal{D}$ such that D and D' differ by adding or removing a row and any set $O \subseteq \mathcal{R}$,*

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

In SAQE, we consider a computationally bounded, polynomial time adversary, so we relax our guarantee to computational differential privacy as in [47]. To achieve differential privacy, SAQE injects noises in two different places: the query output and differential privacy sampling. The differential privacy guarantee degrades gracefully when invoked multiple times. In the simplest form, the overall privacy loss of multiple deferentially private mechanisms is bounded with sequential composition [21].

THEOREM 1 (SEQUENTIAL COMPOSITION). *If M_1 and M_2 are (ϵ_1, δ_1) and (ϵ_2, δ_2) differentially-private algorithms respectively, and both use independent randomness, then releasing the outputs $M_1(D)$ and $M_2(D)$ over database D satisfies $(\epsilon_1+\epsilon_2, \delta_1+\delta_2)$-differential privacy.*

There exist advanced composition theorems that give tighter bounds on privacy loss under certain conditions [23], but we use sequential composition as defined to maximize the generality of this system.

In this work, we use differential privacy to provide strong bounds on information leakage during query processing. We refer to these bounds as our privacy budget, $(\epsilon_{total}, \delta_{total})$. By adding noise before and after query processing, we protect the private data of the data providers, but reduce the final query result accuracy. SAQE's query planner maximizes query accuracy while adding noise to uphold the guarantees of differential privacy using a constraint solver when it receives a query for optimization.

Approximate Query Processing is a mechanism that trades the accuracy of query answers in exchange for shorter latency. Aggregate queries, such as COUNT, SUM, AVG, benefit the most from approximate query processing since they can achieve an attractive accuracy-performance trade-off by exploiting the statistical properties of aggregate operations. Although there are numerous approaches to approximate query processing (e.g., wavelet, histograms, sketching), this work focuses on sampling due to its generality in supporting ad-hoc selection predicates. Although prior systems [3, 17, 31, 35, 39] have introduced approximate query processing for answering relational queries, SAQE is the first to bring this mechanism to private data federations.

One useful property of sampling-based approximate query processing is that the error bounds in its answers quickly diminishes as the sample size increases,even when the original dataset is quite large. Specifically, for a sample of size n_s records and an original dataset of size n, the standard error of the mean is proportional to $(\sqrt{n - n_s})/\sqrt{n_s(n-1)}$ with an upper bound of $1/\sqrt{n_s}$. Hence, the upper bound on the error of the approximate query result is independent of the size of the original dataset. This principle generalizes to all aggregates including COUNT, SUM, and AVG.

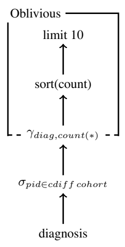
2.4 Query Lifecycle

A query runs in SAQE using the steps shown in Figure 3. This shows the two main facilities of SAQE and its two main facilities: query planning and query execution. The system plains and optimizes its queries on the client side, where the private data federation parses the input SQL query Q and compiles a secure executable query plan P that translates SQL to secure computation for all of the steps that run obliviously over the inputs of two or more data providers. The query execution occurs on the server side, amongst the data owners, where they jointly execute P over their respective private databases and return the final result to the client.

Step 1: Oblivious Query Planning. The client first receives a SQL query, Q, and parses the query into a directed acyclic graph of database operators. Next, the private data federation analyzes the query tree bottom-up using information flow analysis to determine the minimal subtree that must run in secure computation based on

```
SELECT diag, COUNT(*) cnt
FROM diagnosis
WHERE patient_id
    IN cdiff_cohort
GROUP BY diag
ORDER BY cnt
LIMIT 10
```

(a) SQL Query



(b) Oblivious Query Plan

Figure 4: *Comorbidity* query and plan

the column-level security policy. We call this the *oblivious subtree*. When taken together with the rest of the operators in the query tree, we have our *oblivious query plan*. Figure 4 shows an example query, *Comorbidity*, and its corresponding oblivious query execution plan. Figure 4b places the oblivious subtree inside the box, meaning that the GROUP BY, SORT, and LIMIT operators must be jointly executed by all data providers using secure computation, while the SELECT and FILTER operators are executed locally on each data provider in plaintext. Note that this phase uses no private information to determine how to run the query.

Step 2: Secure Query Execution. The data providers locally run any plaintext operators that exist in the query plan to prepare their inputs for sampling. The data providers perform this step in parallel before unioning their data into a single array obliviously. This union encrypts the query inputs as secret shares by running a cryptographic protocol. At the end of the protocol, each party has one share of each tuple. At least k out of m parties must work together to recover the secret input from their shares. After secret sharing the query's inputs, the data providers jointly compute the query by executing the query plan bottom up.

Step 3: Query Results Release. After the private data federation evaluates the query, but before sending its results to the client, the data providers noise its output with a differential privacy mechanism to prevent anyone from learning the precise result. The data providers then each send their noisy cryptographic shares to the client, who assembles the shares to reveal the noisy query result.

3. SAQE OVERVIEW

We outline our research goal s and the challenges therein. We then provide a roadmap for how SAQE addresses these goals with a novel query compilation and execution pipeline. In addition, we show the SAQE architecture and present the SAQE client API.

3.1 Research Aims

The overarching goal of this research is to scale private data federations large datasets while upholding their end-to-end privacy guarantees and providing query results with high accuracy. To date, the bottleneck for query processing in this setting has been oblivious query evaluation over secure computation. The main bottleneck of existing private data federations is the secure computation required to protect the data during query execution. To achieve this speedup with minimal loss in accuracy and privacy, we generalize approximate query processing to obliviously sample query inputs while providing tight accuracy bounds on their results. Moreover, we integrate approximate query processing with differential privacy. SAQE exploits the uncertainty inherent in query results over sampled data to introduce less noise into the query's final output. We frame this research goal as an optimization problem: *identify the query plan with the highest accuracy subject to a client-supplied privacy budget and an optional time constraint.*

Naïvely, one could apply approximate query processing to this setting by taking a uniform random sample of private data from each data owner, and then executing the query over secure computation on the sampled data. This would improve performance since the system would process a smaller set of inputs. This approach, however, introduces two new problems: 1) *Privacy:* approximate query processing does not provide privacy guarantees, so sampling with existing algorithms will compromise the end-to-end privacy guarantee because each data provider knows what tuples he or she contributed to the query, and 2) *Accuracy:* in the absence of additional mechanisms the interaction between the noise introduced by sampling and the noise introduced by differential privacy is not well-defined and may cause query answers to have extremely imprecise query results.

To address these challenges we introduce SAQE, a private data federation system that extends approximate query processing to private data federations to improve performance while ensuring privacy and accuracy. SAQE introduces novel sampling algorithms prevent uncontrolled information leakage thereby upholding the system's end-to-end privacy guarantee. In addition, we introduce a novel query optimizer to capture the interaction between SAQE's two noise sources. This maximizes the accuracy of our query results by not "double counting" the information leakage from these mechanisms. We describe the sampling algorithms in depth in Section 4. We then formalize this relationship between approximate query processing and differential privacy and use this to optimize the accuracy of our query results in Section 5.

3.2 Architecture

Figure 3 depicts the SAQE architecture and how it augments the capabilities of a private data federation. SAQE has two components: the server and the client. Each data provider runs an instance of the SAQE server to evaluate queries on their private dataset as part of the unioned query workload. The SAQE client performs query planning and optimization. It also assembles the cryptographic shares of the query results from the data providers to reveal the noisy query output.

SAQE Client. On the client's workstation, SAQE begins by parsing her SQL query into a directed acyclic graph of operators with their security policy for running either in the clear or obliviously as described in Section 2.1. The client-side software next adds one of the sampling operators from Section 4 to the query tree each time the federation unions the secret inputs of multiple parties. The sampling algorithms may be full-oblivious or differentially private. After that, the client-side software optimizes the sampling rate to maximize query accuracy while meeting a client-provided time limit T using the algorithm in Section 5. Recall that the system completes this query planning process without using any information about the data providers' private query inputs. In contrast to previous work that maximizes the performance versus privacy trade-off [10], SAQE optimizes within a three dimensional decision space of efficiency, privacy, and accuracy. This optimization process is performed independently of the query's private inputs, hence the plan leaks no information about this data. Note that the planner does not reorder commutative operators when it optimizes a query plan, we leave this challenge to future work.

SAQE Server. As in existing private data federations [9, 10] the SAQE Server takes in the oblivious query plan generated by the private data federation and executes it locally on a host running the private DBMS for each data provider. Next, all of the data owners compute the public subtree(s) of the query plan in parallel. For each leaf in the oblivious subtree, SAQE Server runs a sampling operator selected based on the query type, e.g., stratified sampling for GROUP BY queries. It then runs both the remaining database operators and noises the query results within secure computation.

Supported Operators. SAQE supports a large class of database operators: selection, projection, aggregation (i.e., SUM, AVG, and COUNT) optionally with group-by, sorting, limit and denormalized joins. We denormalize by precomputing re-usable intermediate join results without sampling, as in [3, 36], to simplify our sampling accuracy bounds.

3.3 Query Syntax

Let's look at the SAQE system architecture as shown in Figure 3 in terms of a medical research use case. In this scenario, each hospital hosts a database that contains private data, such as individual patient records. We refer to these hospital databases as the private databases. In addition, there is a medical researcher that wants to execute a query Q over the union of the data in the private databases within some time limit T and privacy budget (ϵ, δ). This extended SAQE query syntax is:

```
SELECT AGGREGATE([DISTINCT] *)
FROM <table>
WHERE privacy = <epsilonResult,deltaResult,
                 epsilonSample,deltaSample>
    [AND <selection criteria>]
[AND time_limit = <time limit>]
[GROUP BY <attribute>]
```

In our running example, a medical researcher specifies the desired tables and attributes, the output privacy budget, the sampling budget, as well as an optional time limit. In the absence of a time limit, we choose a sampling rate that minimizes the total query error. Note that the researcher does not need any knowledge of the source databases besides the shared database schema. This researcher is the client and she is working from a machine we refer to as the analyst workstation. Throughout the rest of the paper, we examine how SAQE parses the input query Q, compiles an oblivious plan P according the provided budgets, and executes P over the union of the data from each hospital.

4. PRIVATE SAMPLING

Our core idea in this work is to use secure sampling algorithms to improve performance and enhance security, while providing provable guarantees on result accuracy. However, if we naïvely sample our input data without careful examination, we may end up compromising the accuracy or privacy of our system. In this section, we outline several sampling techniques employed in our system, including uniform, stratified, and distinct sampling and reveal our oblivious algorithms for each technique. We then generalize these fully-oblivious sampling methods—which leak no information about the query's inputs nor the tuples selected in the sample—to differentially-private sampling. The latter leaks bounded information about the query's inputs, and runs more efficiently by executing part of the sampling algorithm on each data provider locally.

4.1 Oblivious Sampling

We now introduce a suite of oblivious sampling algorithms. They offer the same properties as traditional sampling for approximate query processing but with one important addition: they guarantee that as we sample the union of the data of two or more mutually distrustful parties that none of them learn about tuples that were selected – either from their query inputs or those of their peers. SAQE performs its sampling using secure computation to provide this protection. In the private data federation setting, our algorithm obliviously samples the combined data of all participating parties and creates a single encrypted sampled query input for use in the remaining query processing. This sampling requires additional time in our query execution pipeline, and the system amortizes this cost with the reduced operator runtimes over the sampled query inputs.

We present sampling algorithms that support a broad class of SQL queries. Uniform sampling offers tight accuracy bounds for linear queries (e.g., `COUNT`) without group-by or distinct operators. Stratified sampling adds tighter accuracy bounds for group-by queries. Last, distinct sampling brings in support for queries with `DISTINCT` aggregation.

4.1.1 Uniform Sampling

In approximate query processing, a system commonly uses uniform sampling for its inputs when the following two conditions are satisfied: 1) a query does not have joins on fact tables; and 2) if a query includes the group-by clause, the support for every group (i.e., the number of items that belong to each group) must be large enough to ensure a certain accuracy guarantee. These conditions are commonly satisfied when the database schema follows the star schema (or the snowflake schema) for which joins are mainly performed between a (large) fact table and (smaller) dimension tables. If rare groups are present in a dataset or fact tables are joined, other sampling mechanisms discussed later provide higher accuracy.

We cannot release the true number of input records n without revealing private information about the unioned input data. Instead, we use a noisy estimate of the number of records $\tilde{n}$. We add dummy tuples to the query inputs to protect their true cardinality. We generate this noisy input cardinality using the truncated Laplace mechanism, $TLap(\epsilon_{setup}, \delta_{setup})$ [10]. It adds a non-negative integer to the source cardinality. It slightly increases our input size, and guarantees SAQE samples over all of the true input records.

This privacy-preserving input cardinality, $\tilde{n}$ informs our expected sample size, n_s. The probability of selecting a given input record during sampling is $p = \frac{n_s}{\tilde{n}}$. This figure does not affect the privacy amplification or the noise we inject into the query answer for the system's differential privacy guarantee because it only impacts the information revealed from observing the query's execution by altering the public sample size. Since the system does not disclose the true input table size, n, we estimate the error in the query answer owing to sampling using the approximate data input size $\tilde{n}$ (minus the expected number of dummies based on $(\epsilon_{setup}, \delta_{setup})$) instead of the true data input size n. Before any queries are executed by SAQE, the noisy metadata is calculated and cached for future use. Hence, it does not affect the system's per-query privacy budget.

Oblivious Algorithm. Creating uniform samples obliviously can be done using an oblivious random shuffling, which in turn can be implemented using oblivious sorting. However, this approach is slow since we need to provide a random index for each element. In order to speedup the oblivious uniform sampling, we assign each element a random bit and use an oblivious compaction algorithm, which outperforms random shuffling. We show the details of our implementation in Algorithm 1. The time complexity of this sampling implementation is $O(n \log n)$ due to the use of oblivious compaction. We use an oblivious compaction algorithm [28] that runs in $O(n \log n)$ time. Recent work [7] devised an asymptotically linear-time oblivious compaction; however, the actual efficiency is not higher than what we use due to high fixed setup costs.

Utility Analysis. In Algorithm 1, the utility of our implementation relies on the the accuracy of our input parameters: the input data size $\tilde{n}$ and the sampling rate p. SAQE generates $\tilde{n}$ using a differentially private mechanism. Since $\tilde{n}$ is noisy metadata, it can be used across multiple queries. Hence, generating $\tilde{n}$ only consumes the privacy budget once, when data is inserted into the database

Algorithm 1: Oblivious uniform random sampling

```
Input: Dataset $D$ with $\tilde{n}$ dummy-padded records secret shared
    across $m$ parties as $d_1, ..., d_{\tilde{n}}$, sampling rate $p$
Output: Uniform random sample $O$ of expected size $n_s$, where
    $n_s = p\tilde{n}$
Use secure computation to:
1. Append the input lists from all parties together.
2. For each element in the list, securely sample a bit that equals '1'
   with probability $p$ and output as $r_i$.
3. Obliviously select all records $D_i$ from $D$ such that $r_i = 1$
4. Collect all selected records into a single relation $O$
```

and its accuracy is set by the database administrator. On the other hand, choosing p to maximize accuracy depends on the constraints of each query. We discuss our approach in Section 5.

Privacy Analysis. When using oblivious sampling, all computation on private data is carried out using secure computation. As such, we do not leak any information when executing the sampling operator. The input to the sampler is the union-ed data from each data provider while the output is the noisy result of the query execution, just as in existing private data federations. As such, these two steps also do not leak information. With oblivious sampling, neither the input, computation, nor output compromise privacy.

4.1.2 Stratified Sampling

Stratified sampling is a biased sampling mechanism that preserves rare groups (e.g., items with `eye_color = red`) by constructing a sample as follows. Given a column (e.g., `eye_color`), the mechanism first partitions a dataset into multiple strata based on the attribute values in the column; that is, tuples with `eye_color = black` and tuples with `eye_color = brown` are assigned to different strata. Then, uniform sampling is performed independently for each stratum. The sampling rate for each stratum is set such that each strata has a sufficient number of samples. For example, if there are very few tuples with `eye_color = brown`, then all these tuples will be included in the samples. After sampling, the samples from each stratum are concatenated to construct the full sample. Hence, the contribution of each group to the full sample is proportional to the size of their stratum.

To prepare stratified sampling, first, we must partition our input data into separate strata. Given a set of values $\{v_1, \ldots, v_{N_{\mathsf{strata}}}\}$ and a privacy budget $(\epsilon_{setup}, \delta_{setup})$, each party partitions its data into N_{strata} stratum and apply TLap$(\epsilon_{setup}, \delta_{setup})$ to add dummy records to each strata. A public label v_i is assigned to each record if it falls into strata with value v_i (even the record is a dummy). Then these records are secretly shared among the k parties while keeping the labels public. With these labels, we can now estimate the size of each stratum in D in the stratified sampling.

Oblivious Implementation. In Algorithm 2, we detail our oblivious stratified sampling implementation. For each stratum, we carry out oblivious uniform random sampling as in Algorithm 1 and collect the selected records. Note that we must construct a sample for every stratum in the domain, even if the input data does not contain records in that stratum. If we do not, then we leak the information about which stratum are present in the private input data. Finally, we concatenate all samples to form our final sample O.

Utility Analysis. In order to determine the accuracy guarantees of a stratified sample, we need to bound the error according to our sampling rates $\{p_1, \ldots, p_{N_{\mathsf{strata}}}\}$. For the error, we define our bound as the maximum variance from any single stratum. If we used uniform sampling, there is no guarantee that a records in specific stratum will appear in the sample, resulting in an error of 100%. Conversely, our stratified sampling algorithm reduces error by ensuring

Algorithm 2: Oblivious stratified sampling algorithm

```
Input: Set of data records $D$ stratified over $g$ with size
    $\tilde{n} = \sum_{i=1}^{N_{strata}} \tilde{n}_{g,i}$ distributed across $k$ parties and
    partitioned into strata $D_{g,1}, ..., D_{g,N_{\mathsf{strata}}}$, and sampling
    rates $\{p_1, \ldots, p_{N_{\mathsf{strata}}}\}$
Output: Stratified random sample $O$ of expected size $n_s$, where
    $n_s = \sum_{i=1}^{N_{\mathsf{strata}}} \tilde{n}_{g,i} \cdot p_i$
Partition $D$ into $\{A_1, \ldots, A_{N_{\mathsf{strata}}}\}$ based on their labels
for $i \leftarrow 1$ to $N_{\mathsf{strata}}$ do
    1. $O_i \leftarrow$ ObliviousUniform($A_{g,i}$, $\tilde{n}_{g,i}$, $\tilde{p}_i$)
2. Collect all stratum results $O_i$ into a single relation $O$
```

Algorithm 3: Oblivious distinct sampling algorithm

```
Input: Set of data records $D$ with size $\tilde{n}$ distributed across $k$
    parties as $D_1, ..., D_{\tilde{n}}$, a maximum frequency $f$, sampling
    rate $p$
Output: Random sample $O$ of expected size $\tilde{n}pf$
1. Each party $P_i$ creates two lists, an input list $L_i$ that contains all
   input records from $D_i$ and a de-duplicated list $L'_i$.
Use secure computation to:
2. Append the list $L'_i$ from all parties and obliviously de-duplicate
   the list using oblivious sorting and a linear scan. The resulting list
   is $L'$.
3. $S \leftarrow$ ObliviousUniform($L'$, $\tilde{n}$, $p$)
4. Obliviously select all records $D_i$ from $\{L_i\}_{i \in [k]}$ such that the
   value is in $S$
5. Collect all selected records into a single relation $O$
```

that all strata are present in the sample. We determine our sampling rates $\{p_1, \ldots, p_{N_{\mathsf{strata}}}\}$, and our error, by applying the optimization discussed in Section 5.

Privacy Analysis. For stratified sampling, we apply oblivious uniform sampling within each stratum, guaranteeing that privacy leakage within each stratum is bounded by the differential privacy guarantees given by oblivious uniform sampling. When creating the final result, we concatenate the sampled records from all strata within secure computation, leaking no additional information.

4.1.3 Distinct Sampling

Distinct sampling [27] is used to sample the domain of an attribute rather than the attribute tuples themselves. Given a set of tuples with multiple possible values (e.g., `eye_color = black` and `eye_color = brown`), a distinct sampler uses a collision-resistant hash function H to hash the attribute values of the column, where H returns a value between 0 and 1; then, distinct sampling chooses the tuples if their hash values are smaller than a sampling probability p. Hence, it is most effective when the query requires finding distinct values for a given attribute.

Oblivious Implementation. Algorithm 3 details our distinct sampling implementation. In order to carry out distinct sampling with secure computation, we cannot apply a hash function H due to prohibitive costs [5]. Instead, we utilize secure computation to generate a de-duplicated list of records L' and sample L' using Algorithm 1 to create a list of sampled records S. Then, we oblivious select records from D whose values are present in S. We use the public maximum frequency f to correctly bound the size of the sampled output.

Utility Analysis. Consider a set of tuples for a single attribute, where each distinct value appears some unknown number of times. If we apply uniform random sampling, we may severely undercount the number of distinct values due to a small number of values appearing over a large fraction of the tuples. Applying stratified sampling would return distinct values, but in the case where each tuple had a different value, it would return the entire set of tuples, effectively not sampling at all. Conversely, distinct sampling samples from the attribute domain, ensuring that we obtain a subset of possible attribute values. As such, we treat error as the number of records with unique attribute values missing, rather than the total records missing. With this definition, we determine our sampling rate p and our error by applying the optimization in Section 5.

Privacy Analysis. In our implementation, all computation over private data is carried out using secure computation, protecting the data during computation. At the output, the client only receives the decrypted final result, guaranteeing that they do not learn additional information about the private inputs. Note that our implementation holds similarities with universe sampling [35]. Unlike universe sampling, we use distinct sampling only for Distinct aggregate queries, so all duplicates are removed at the query output. As such, any change to a single input tuple only affects the output by 1, i.e., query sensitivity equals 1.

4.2 Differentially-Private Sampling

In some cases, such as for extremely large datasets, oblivious sampling may be too expensive due to its reliance on processing all input records within secure computation. An alternative approach is using differential privacy to generate local samples first. Differentially private sampling creates samples at the source database using plaintext uniform, stratified or distinct samplers, then combines the samples using secure computation. This means that the initial sampling is executed in plaintext, with much less performance degradation compared to oblivious sampling. In exchange, local sampling leaks information about the source data. Previous work has shown that we can bound information leakage within differential privacy guarantees by adding noise to the resulting sample in the form of dummy records [10] using the $TLap(\epsilon, \delta)$ mechanism.

In our implementation, each party i samples their records locally, without secure computation. Then, using a multiparty protocol for generating noise as in [48], the parties add dummy records to their samples to create noisy samples. The magnitude of this noise depends on the privacy budget $(\epsilon_{sample}, \delta_{sample})$ allocated to the Laplace noise mechanism $TLap$. With secure computation, we combine the noisy samples from all the parties to create an oblivious, uniform random sample. The time complexity of this sampling implementation is $O(n)$ due to the use of oblivious union.

With this sampling approach, we avoid costly operations in secure computation, at the expense of query accuracy. We use a portion of our total budget to hide our input sample sizes, leaving a smaller privacy budget for the query output, which requires SAQE to add more noise at the output, reducing our query accuracy.

Privacy Analysis. For sampling with differential privacy, each data provider submits their noisy sampled data to the private data federation. If their sample is not noised, then the client could learn additional information about an data provider's private data. For example, if the parties send data to the sampler without noise, then they reveal the true size of their data. When this data is private, the client can use the input data size to learn private information, such as the operator selectivity, and deduce the private data values.

5. QUERY OPTIMIZATION

We now discuss the SAQE optimizer and how it chooses a sampling rate by balancing the contributions from our two noise sources: approximate query processing and differential privacy. We describe our optimization problem and discuss our two modes of operation, max accuracy mode and time bound mode. In addition, we clarify the statistics used during optimization to arrive at an approximately optimal sampling rate.

5.1 Modeling Query Result Accuracy

The goal of the SAQE query optimizer is to maximize query result accuracy. The first source of inaccuracy in SAQE is the error introduced by sampling the query inputs. Since we execute a query over a sample of the true data, our answer will suffer reduced accuracy. The second noise source is due to the noisy query results needed to satisfy differential privacy. Using a differentially private mechanism, we add noise to the query results to prevent attackers from learning unauthorized information. In SAQE, we want the optimizer to reason about these two noise sources *a priori*, meaning that SAQE can choose a sampling rate during query planning that maximizes result accuracy. As such, we have to define each of these noise sources and show how to combine them.

Error from Approximate Query Processing. We use well-known statistical formulas for sampling without replacement to determine the expression for the variance in our sampling error: Var(X)= $\tilde{n}^2[r(1-r)/n_s][1-(n_s-1)/(\tilde{n}-1)]$, where r is the query selectivity, $\tilde{n}$ is the population size, and n_s is the sample size. Note that Var(X) depends on the selectivity r of a query, which is typically unknown before processing the query. We remove this dependency on r by working with its upper bound as follows: $r(1-r) \leq 1/4$. We further simplify the bound by relying on: $1-(n_s-1)/(\tilde{n}-1) \leq 1 - n_s/\tilde{n} = 1-p$. Finally, we obtain a simplified upper bound on the sampling variance: $\tilde{n}(1-p)/(4p)$. For stratified sampling, the bound becomes $\tilde{n}_i(1-p_i)/(4p_i)$ where i is the current stratum. For distinct sampling, the bound is $\tilde{n}_{distinct}(1-p)/(4p)$ where $\tilde{n}_{distinct}$ is the estimated number of unique values in the input.

Noise from Differential Privacy. In order to combine our noise from differential privacy with our sampling noise, we noise our query results using a Gaussian Mechanism [23]:

THEOREM 2 (GAUSSIAN MECHANISM). *Given function $f : \mathcal{D} \to \mathbb{R}^d$, define its l_2 sensitivity be $\Delta_2 f = \max_{|D \ominus D'|=1} \|f(D) - f(D')\|_2$. Let $\epsilon \in (0,1)$ be arbitrary. The Gaussian Mechanism with parameter $\sigma \geq c\Delta_2 f/\epsilon$ adds noise to each of the d components of the output scaled to $\mathcal{N}(0, \sigma^2)$. It achieves (ϵ, δ)-differential privacy when $c^2 > 2\ln(1.25/\delta)$.*

The l_2 sensitivity measures the largest possible change to the function output when adding or removing a row. For instance, the sensitivity of COUNT(*) is 1 and the sensitivity for Sum(*) is the maximum domain value. When adding sufficient noise to match the sensitivity of the function, differential privacy can be achieved [23].

Given a query plan consisting of a DAG of operators where the last operator is an aggregate function $f : \mathcal{D} \to \mathbb{R}^d$, we apply an (ϵ, δ)-differentially private mechanism M such as the Gaussian mechanism to the last operator. The overall privacy loss on the input dataset at the leaf of the query plan can be analyzed using the *stability* of the operators in the query plan. In SAQE, we use a general stability notion from prior work [24].

DEFINITION 2 (PROBABILISTIC STABILITY). *Let α and β be functions in $\mathbb{R}^{\geq 0} \to \mathbb{R}^{\geq 0}$. A randomized transformation RS is (α, β)-probabilistic stable if for any (ϵ, δ)-differentially private mechanism M, $M \circ RS$ satisfies $(\alpha(\epsilon), \beta(\delta))$-differential privacy.*

Prior work for private databases [10,36,46] only consider deterministic transformations , i.e., their stability functions $(\alpha(\cdot), \beta(\cdot))$ are linear. For instance, for Selection, and Projection, Count(*), and Count Distinct(*), $\alpha(\cdot)$ and $\beta(\cdot)$ are identity functions, i.e., the privacy parameters are unchanging. For GroupBy, $\alpha(\epsilon) = 2\epsilon$ and $\beta(\delta) = 2\delta$. OrderBy and Limit usually have very high stabilities, and hence we consider a query plan that applies these operators after a differentially private mechanism as a post-processing step.

Table 1: Error Contributions of COUNT and SUM queries

Query	Var(X)	Var(Y)
Count	$n(1-p)/(4p)$	$2\ln(1.25p/\delta_{result})/\ln(1+(e^{\epsilon_{result}}-1)/p)^2$
Sum	$n\Delta^2(1/p-1)$	$2\Delta^2\ln(1.25p/\delta_{result})/\ln(1+(e^{\epsilon_{result}}-1)/p)^2$

On the other hand, sampling operators are randomized transformations and their stability depends on many factors [8]. Uniform sampling used by SAQE applies Bernoulli sampling (or Poisson subsampling) with sampling rate p and is (σ, β)-stable [8, 24], where $\alpha(\epsilon) = \ln(1 + p(e^\epsilon - 1))$ and $\beta(\delta) = p \cdot \delta$, As $p < 1$, this sampling operator actually tightens the privacy parameter, i.e., $\alpha(\epsilon) < \epsilon, \beta(\delta) < \delta$, and hence strengthens or *amplifies* the privacy guarantee, unlike deterministic transformations. We also show that the stability results for the other two sampling techniques. Distinct sampling shared similar stability as uniform sampling. Stratified sampling has different sampling rates among strata and hence its stability is analyzed based on the maximum sampling rate. We show the full proof in the full paper.

THEOREM 3. *Stratified sampling (Algorithm 2) has a stability* $\alpha(\epsilon) = \ln(1 + p_{\max}(e^\epsilon - 1))$ *and* $\beta(\delta) = p_{\max} \cdot \delta$, *where* $p_{\max} = \max(p_1, \ldots, p_{N_{strata}})$. *Distinct sampling (Algorithm 3) has a stability* $\alpha(\epsilon) = \ln(1 + p(e^\epsilon - 1))$ *and* $\beta(\delta) = p \cdot \delta$.

Then we can bound the privacy loss of a complex query plan and derive the corresponding noise parameter of the the Gaussian mechanism that applies to the final aggregate in the query plan.

PROPOSITION 1. *For a query plan with l operators* $(RS_1, \ldots, RS_l)$ *with an aggregate function f, for which* RS_i *is* $(\alpha_i(\cdot), \beta_i(\cdot))$*-probabilistic stable, if the mechanism applied is* $(\epsilon_{result}, \delta_{result})$*-differentially private, then the overall privacy loss is* (ϵ_0, δ_0), *where* $\epsilon_0 = \alpha_1 \circ \cdots \circ \alpha_l(\epsilon_{result})$ *and* $\delta_0 = \beta_1 \circ \cdots \circ \beta_l(\delta_{result})$.

Take a counting query with uniform sampling as an example. If a sampling operator with sampling rate p is applied, to achieve $(\epsilon_{result}, \delta_{result})$-differential privacy on the input data, the Gaussian mechanism applied to the final count aggregate operator only needs to be (ϵ_0, δ_0)-differentially private, where $\epsilon_0 = \ln(1 + (e^{\epsilon_{result}} - 1)/p)$ and $\delta_0 = \delta_{result}/p$. This requires a Gaussian noise with a variance Var(Y)= $2\ln(1.25p/\delta_{result})/(\ln(1 + (e^{\epsilon_{result}} - 1)/p)^2$, which is smaller than the case without sampler.

Approximately Maximizing Query Result Accuracy. Now that we have our two Gaussian noise distributions, as well as their amplification, we can combine them as follows. Let X be a random variable standing for the sampling error from approximate query processing, and let Y be a random variable standing for the output error from differential privacy. We use the mean squared error and represent the total output error of the query by the sum of the variances of X and Y, i.e. Var(X) + Var(Y), as X and Y are independent and have zero mean. We show the equations for X and Y for Count and Sum queries in Table 1.

5.2 The SAQE Optimizer

With the noise source definitions in Table 1 , we can approximately optimize the accuracy of our query results. The SAQE optimizer offers two modes. In the first, *max accuracy*, it takes in a SQL query and a privacy budget constraint. The optimizer identifies a query plan that maximizes accuracy and only samples the data until no additional accuracy gains are possible owing to the fixed noise requirements of differential privacy. In order to maximize accuracy, we use an objective function that incorporates our two noise sources: sampling and differential privacy. In the second mode, *time bound*, we maximize accuracy within fixed time and privacy constraints. We handle our time constraint using our cost model from Section 5.2 and our privacy constraint according to existing differential privacy analyses on relational databases [10,36,46]. We use a solver to identify the optimal query plan in both modes where the goal of our objective function is to determine the optimal sampling rate p for a given secure leaf operator λ.

Given a query, SAQE constructs an objective function to determine the sampling rate used during query execution. Using an off-the-shelf solver, we can solve this function to maximize query accuracy, while satisfying our privacy constraints. Through query optimization, SAQE creates query plans that balance differential privacy, approximate query processing, and secure computation.

Max Accuracy: Uniform Sampling. For *max accuracy* mode, we combine the expressions for our noise sources shown in Table 1 to determine the optimal sampling rate p using the following optimization problem:

$$\begin{aligned} \min_p \quad & n(1-p)/(4p) + 2\ln(1.25/\delta_{result})/\epsilon_{result}^2 \qquad (1) \\ \text{s.t.} \quad & p\delta_{result} \le \delta_0 \\ & \ln(1 + p(e^{\epsilon_{result}} - 1)) \le \epsilon_0 \end{aligned}$$

The above analysis details a linear count query utilizing a uniform sampler. In Table 1, we show the result of similar error analyses on queries involving Sums, where Δ = maximum domain value.

Max Accuracy: Stratified Sampling. For stratified sampling, we adjust our optimization problem to minimize the maximum error from any single stratum. We optimize:

$$\begin{aligned} \min_{p_1,\ldots,p_{N_{\text{strata}}}} \quad & \max_i Error(\tilde{n}_i, p_i, \epsilon_{result}, \delta_{result}) \qquad (2) \\ \text{s.t.} \quad & \max_i p_i\delta_{result} \le \delta_0 \\ & \max_i \ln(1 + p_i(e^{\epsilon_{result}} - 1)) \le \epsilon_0 \end{aligned}$$

Here $Error(\tilde{n}_i, p_i, \epsilon_{result}, \delta_{result})$ is similar to uniform sampling, except the optimization is carried out for all strata. Since the optimization is over disjoint data, we can carry out the optimization in parallel, avoiding additional latency overhead.

Max Accuracy: Distinct Sampling. In distinct sampling, we sample from the space of unique values in the source data. To handle this setting, we adjust our optimization problem as follows:

$$\begin{aligned} \min_p \quad & Error(\tilde{n}_{distinct}, p, \epsilon_{result}, \delta_{result}) \qquad (3) \\ \text{s.t.} \quad & p\delta_{result} \le \delta_0 \quad \text{and} \quad \ln(1 + p(e^{\epsilon_{result}} - 1)) \le \epsilon_0 \end{aligned}$$

Again, $Error(\tilde{n}_{distinct})$ here is the same equation as in uniform sampling, except that we use the number of possible distinct values $\tilde{n}_{distinct}$ instead of the number of input records $\tilde{n}$. We estimate $\tilde{n}_{distinct}$ as $\tilde{n}/f$, where f is the maximum frequency. Since we use distinct sampling only for queries that measure accuracy by the number of distinct values in the output, our accuracy equations are identical to the uniform sampling setting.

Time Bound Mode. We extend our optimization problem in Equation 2 to *time bound* optimization by adding the constraint $T \le T_{MAX}$ where we solve for T using our SAQE cost model.

For a single operator, λ_i, we express the cost as function of the cardinality $card(\lambda_i, \mathbf{N_i})$ and the cost of oblivious operator evaluation $c_o(\lambda_i, card(\lambda_i, \mathbf{N_i}))$. We take the sum of our two cost sources, bounded by our fixed time limit T_{max}, to arrive at our cost model:

$$T = \sum_{i=1}^{\ell} c_p(p_i, N_i) + c_o(\lambda_i, card(\lambda_i, \mathbf{N_i})) \le T_{max} \qquad (4)$$
$$\textit{where } c_p(p_i, N_i) = 0 \textit{ for non-secure leaf operators}$$

Since the time cost T varies depending on client and host hardware, we use I/Os as a stand-in for time. This fits well with our

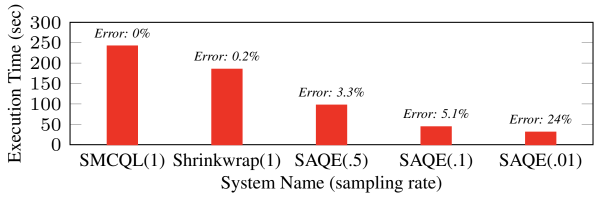


Figure 5: End-to-end performance comparison with existing systems. HealthLNK *Comorbidity*, $\epsilon_{total} = 0.5$, $\delta_{total} = 10^{-6}$.

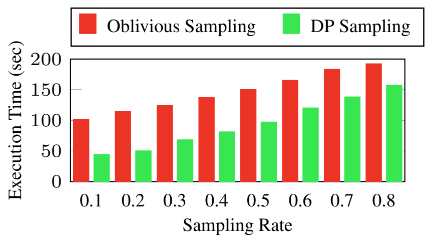


Figure 6: Sampling approach performance comparison, HealthLNK *Comorbidity*, $\epsilon_{total} = 0.001$, $\delta_{total} = 10^{-6}$.

oblivious execution environment due to memory accesses being the overriding cost source during execution [10]. When communicating the cost to the user, the system will need an I/O to time conversion factor. This conversion factor will account for the hardware-dependent costs of a given execution environment.

6. EXPERIMENTAL RESULTS

In this section, we evaluate SAQE using real-world hospital data and analytics queries, as well as a selected TPC-H query workload. We begin by describing our experimental design and configuration, including our query workloads. Next, we examine the end-to-end performance of SAQE against both a fully oblivious, non-sampled execution and a non-secure, plaintext execution. Finally, we examine the effectiveness of our cost model and optimization in relation to the trade-offs between accuracy, privacy, and efficiency.

6.1 Experimental Setup

TPC-H Workload We evaluate SAQE on TPC-H, scale factor 1. We configure the private data federation as if we were running an international online shop. Businesses of this kind are increasingly subject to data residency requirements such as that of the European Union's GDPR. Hence, we partition the suppliers and customers by their nationkey and collocate a customer with his or her orders and lineitems. The primary keys for order and customer are public owing to their known domain of (1... <table cardinality>). Likewise, we partition suppliers and the partsupp table by their country of origin and have public primary keys. Since the company's website makes their catalog, prices, and countries served visible to anyone who accesses it, we replicate the parts, nation, and region table on all nodes and set the supplycost as public. Lineitem, the fact table, has all of its attributes set to private. This guards against a curious observer attempting to deduce the contents of individual shopping carts. In the same vein, the foreign key relationship between customer and order and between partsupp and lineitem are also private. For our experiments, we denormalize the schema as described in Section 3.2. SAQE use secure computation to create to pre-join the source relations securely, and the system maintains a secret-shared mapping table from this that is not accessible to anyone.

HealthLNK Workload We compare SAQE with existing systems by using HealthLNK [50], a clinical data research network that provides a repository containing records from Chicago-area healthcare institutions. Our experiments use one year's worth of data from two hospitals, totalling 500,000 patient records, or 15 GB of data. This data set contains a public patient registry with anonymized patient identifiers. HealthLNK provides public schema with security annotations denoting public and private attributes.

Configuration We implemented SAQE on top of a two-party private data federation using an off-the-shelf secure computation library, EMP-Toolkit [59]. We ran SAQE using 6 servers running in pairs. Each machine has PostgreSQL 9.6 running on Ubuntu Linux, as well as 64 GB of memory and 7200 RPM NL-SAS hard drives on a dedicated 10Gbps network. We use $(\epsilon_{total}, \delta_{total})$ to represent the sum of the result and sample per query budgets. Unless otherwise specified, our figures show the average runtime of three runs per experiment, apply differentially private sampling, and use a default per query privacy budget of ($\epsilon_{total} = 0.001$, $\delta_{total} = 1.0 \times 10^{-6}$). Under oblivious sampling, the default privacy budget is used entirely for the result, while for differentially private sampling, the budget is split evenly between result and sample. For each database, we release noisy table and group sizes using $\epsilon_{setup} = 0.1$ for each. As the metadata is used for all queries, it does not consume our per query budget.

Performance Analysis We motivate SAQE by comparing overall execution times against existing systems and provide an analysis of our two sampling strategies. Since we can set our execution time *a priori* using our cost model, our performance numbers are not meant to show that SAQE outperforms previous systems in all ways. Instead, we want to provide a sense of the efficiency design space and how we can tune SAQE to adjust our trade-offs.

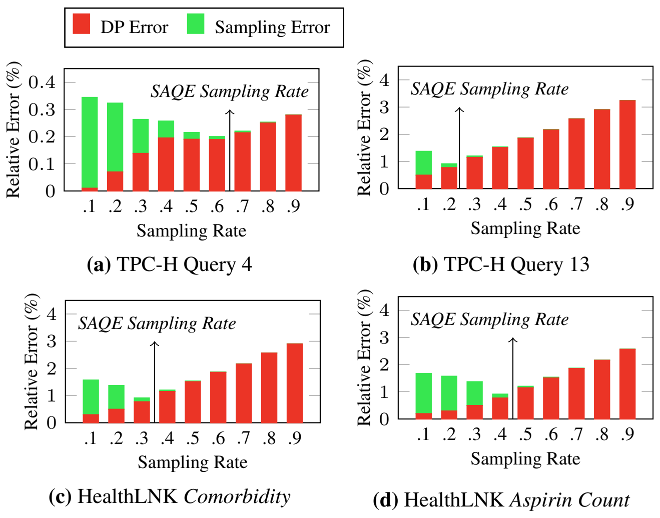
6.2 End-to-end Performance

First, we look at the end to end performance of SAQE in comparison with two existing private data federations, SMCQL [9] and Shrinkwrap [10]. SMCQL works in a similar setting and provides privacy guarantees on the input data and on all computation. For a fair comparison, we implemented secure operators in SMCQL using EMP, an MPC implementation much faster than the one used in SMCQL. Note that SMCQL has no output error since it does not add output noise to protect query result information leakage.

Shrinkwrap combines differential privacy guarantees with secure computation to speed up query execution. By relaxing privacy of computation guarantees, Shrinkwrap reduces the intermediate result cardinalities, speeding up execution by processing fewer tuples. Shrinkwrap bounds this information leakage according to a privacy budget (ϵ, δ). Note that Shrinkwrap introduces error by using a portion of the privacy budget to add noise to the query output. To provide an apples to apples comparison, we use oblivious sampling for SAQE and fix both the Shrinkwrap and SAQE privacy budgets to ($\epsilon_{total} = 0.5$, $\delta_{total} = 10^{-6}$).

In Figure 5, we see the performance comparison between SMCQL, Shrinkwrap, and SAQE using the HealthLNK *Comorbidity* query, shown in Figure 4a. For SAQE, we provide results from three different sampling rates. We see that by adjusting the sampling rate we can tune the execution time of the system. In exchange for performance, SAQE's use of approximate query processing introduces error into the query result, with the lower sampling rates providing higher error along with better execution times.

We also see a weakness of Shrinkwrap. In Shrinkwrap, performance is improved by minimizing the oblivious intermediate result cardinalities needed for secure computation. This works best for queries where the worst case, oblivious cardinality is extremely large compared to the true cardinality, such as in foreign key-foreign key joins where the output cardinality can be the cross product of the inputs. For linear queries, such as *Comorbidity*, the oblivious cardinality is, at worst, equal to the input size n. In this case,



(a) TPC-H Query 4 **(b)** TPC-H Query 13

(c) HealthLNK *Comorbidity* **(d)** HealthLNK *Aspirin Count*

Figure 7: Result error as sampling rate increases, $\epsilon_{total} = 0.001, \delta_{total} = 10^{-6}$.

the benefit of Shrinkwrap is greatly reduced. For this class of queries, SAQE provides performance improvements that far exceed the lower bound of Shrinkwrap. In fact, we can combine SAQE sampling with Shrinkwrap to further improve performance on other classes of queries, such as those with foreign key-foreign key joins.

6.3 Sampling Approaches

Now we compare the performance of oblivious and differentially-private sampling from Section 4. Here, differential privacy sampling uses half of the privacy budget during sampling, reducing query answer accuracy but improving performance. Figure 6 shows the two approaches with varying sampling rates.

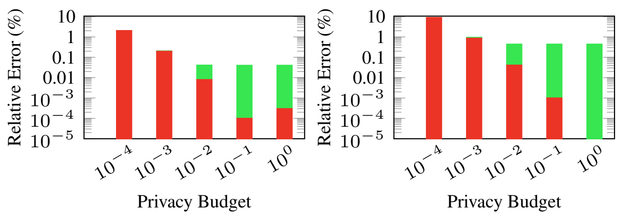
We see that, as expected, differential privacy sampling outperforms oblivious sampling for all sampling rates. However, the difference between the two narrows as the sampling rate increases. Oblivious sampling takes all tuples from data providers as input to secure computation so that the selected tuples are not disclosed to anyone. This means that the oblivious sampler's performance is dependent on the source data size, rather than the sample size.

Differential privacy sampling on the other hand, samples before any secure computation, so its performance is a function of the sampling rate, p instead of the raw data size. As the sampling rate increases, and the sample size approaches that of the source relation, the gap between these two approaches reduces proportionally. Note that the performance after sampling is better with oblivious sampling, since differential privacy sampling introduces more dummy tuples that must be processed within secure computation.

6.4 Accuracy Analysis

Now we examine the effects of the sampling rate and the privacy budget on the final result error. In these experiments, we fix the performance of SAQE using the time bound optimizer from Section 5 and collect results over the queries workload. We use the relative error – $\sum$ |(released - true) / true| – as a measure of the utility of SAQE's privacy-preserving query answers with respect to its true, unnoised output.

Result Accuracy and Sampling Rate. In this experiment, we show the relationship between error and sampling rate. We set our privacy budget to ($\epsilon_{total} = 0.001, \delta_{total} = 0.00001$) and show the relative error as our sampling rate changes. We include our SAQE-optimized sampling rate selected using Table 1.



(a) TPC-H Query 4 **(b)** TPC-H Query 13

Figure 8: Result error as privacy budget increases, $\epsilon_{total} = 0.001, \delta_{total} = 10^{-6}$.

Figure 7 plots sampling rates against the relative error for TPC-H queries 4 and 13, as well as HealthLNK Comorbidity and Aspirin Count queries [10]. Aspirin Count uses distinct sampling, while the other queries use stratified sampling. In traditional approximate query processing, we expect the relative error to decrease monotonically as the sampling rate rises. This reflects that as a greater proportion of the source data is used, the query result is more accurate. With SAQE, this is no longer the case due to the presence of differential privacy guarantees. As discussed in Section 5, a larger sample size requires more noise to obscure the result and guarantee the same level of differential privacy. Our two noise sources, sampling and differential privacy, work in opposition. Which source dominates depends on the sampling rate and privacy budget. In Figure 7, we show that the SAQE-optimized sampling rate points out the inflection point where the contributions of each source are minimized. This point is where we maximize our trade-off between accuracy and performance according to our optimization problem in Equation 2. Our results show that we can identify the inflection point for all our selected sampling strategies. If we choose the incorrect strategy, such as uniform sampling for Q13, our error increases due to strata with lower counts being dropped. Experimentally, we see that 20% of groups in Q13 are dropped at the maximum accuracy sampling rate (0.24).

Error and Privacy Budget. Now we fix our sampling rate to the SAQE-optimized value and use oblivious sampling to see the role of the privacy budget. In Figure 8, we see see a different view of sampling versus privacy noise effect shown in Figure 7. The larger the privacy budget, the less noise SAQE needs to add to guarantee a differential privacy query result. This means that when the budget is large enough, the majority of noise in the system is due to sampling. Since we fix the sampling rate in our experiment, the relative error levels off once we reach this inflection point.

6.5 Data Size Scaling

We now look at how SAQE adjusts as the data input size scales. We show two different execution modes: max accuracy mode and time bound mode. In this experiment, we hold constant that SAQE maintains max accuracy by always executing at the inflection point shown in Figure 7. Figure 9 shows the end-to-end execution times as we increase our source table size up to 1 TB in max accuracy mode. We compare SAQE with the SMCQLand Shrinkwrap systems. Note that the gray shaded regions represent estimated execution times as the prior systems do not have enough available memory to execute the query. SAQE successfully executes previously un-executable queries with maximum accuracy.

Figure 10 compares the execution time for Shrinkwrap, SMCQL, and SAQE over a synthetically scaled version of the HealthLNK dataset. In the baseline, 1X, case, all three systems execute well under the time limit, but as the data size increases, the non-SAQE systems fall out. We see that SMCQL can only execute successfully in the 1X case and Shrinkwrap only runs in the the 1X and 2X

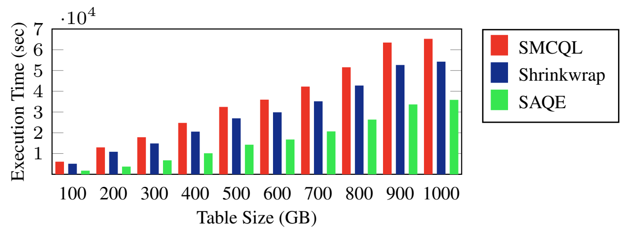


Figure 9: Scaling up in max accuracy mode, HealthLNK *Comorbidity*, ϵ_{total}=0.001, δ_{total}=10^{-6}.

cases. This limitation is due to the time budget, where the systems cannot return a result within the specified time. With SAQE, we can continue executing well above the 1X or 2X cases. In exchange, SAQE sacrifices final result accuracy. Our experiments show that we only incur a modest decrease in accuracy, with a 4.6% relative error at 7X the baseline data size. By allowing the client to tune the sampling rate to their specific requirements, SAQE provides significant performance flexibility with minimal accuracy cost.

7. RELATED WORK

SAQE builds upon research in approximate query processing, differential privacy, secure computation, and private data federations. We highlight the state-of-the-art in each area and describe how SAQE reveals novel synergies among them.

Private data federations offer privacy-preserving query evaluation over the union of multiple private data stores. This is a natural extension to research on secure querying for data outsourced to an untrusted cloud service provider. There were many approaches to solving this challenge including storing and querying the private outsourced data with homomorphic encryption [51,56], secure computation [4,30] or a trusted hardware module [6,64]. Private data federations [9,58] offer *in-situ* SQL evaluation among the multiple private data providers where each one wishes to maintain exclusive access to his or her dataset. Shrinkwrap [10] generalized this by offering differentially-private query processing and results for private data federation queries, but it relies on using the privacy budget to accelerate each query, whereas SAQE speeds up query performance with oblivious sampling—requiring no additional privacy use for its faster runtime. SAQE advances this line with its general-purpose, hardware-agnostic, SQL analytics with provable privacy guarantees with increased scalability owing to its generalization of approximate query processing to private data federations.

Approximate query processing makes it possible for a system to meaningfully estimate query results by sampling their inputs with tight accuracy bounds. Some systems [3,49] compute their query results from a sample set on the initial, larger dataset. Others sample the data for a given query at runtime [31,35,39]. We take the latter approach with this system to support ad-hoc querying, although offline approaches offer fertile ground for future research for well-known workloads. This work expands on the state of the art in approximate query processing to support privacy-preserving analytics by introducing oblivious and differential privacy sampling algorithms. SAQE's oblivious uniform sampler is a natural extension to the one proposed in [53], although the previous algorithm required the use of a trusted execution environment to work efficiently and our algorithms are hardware-independent. The remaining oblivious and differentially-private algorithms described in this work are the first of their kind for processing analytical queries and readily generalize to the untrusted cloud setting.

There has been significant research in differential privacy query processing [33,34,36,37,45,46,60] that provides strong privacy guarantees while minimizing query result noise, as well as accuracy-

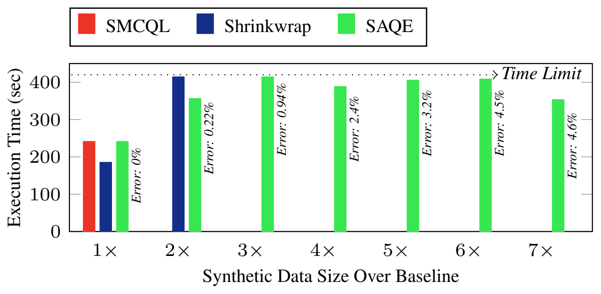


Figure 10: Data-size scaling comparison, HealthLNK *Comorbidity*, 150 MB baseline

aware approaches [26,41] that constrain accuracy rather than optimize it as in SAQE. Additional research [10,48] revealed how to add differentially-private noise to query results with secure computation to ensure that the recipient of the result does not have access to private inputs. Using secure computation in private federated databases [16,19,44,47] or private federated learning [14,55,63] has a smaller error, in the query results or trained model, than local differentially private mechanisms that assume no trusted data curators. Prior approaches to combining differential privacy with approximate query processing in centralized setting [1,8,24] or in federated learning [2,54] quantify the privacy amplification a query receives from sampling. However, they do not investigate the performance improvement and the accuracy optimization brought by sampling. SANNS [18] offers approximate KNN queries over secure computation. It is efficient, but lacks the formal guarantees of DP. Moreover, its techniques do not generalize to SQL queries.

We use secure computation [62] for privacy-preserving query evaluation over the union of data from multiple data providers such that none can learn the secret inputs of their peers. Although researchers proved the feasibility of this technology more than 30 years ago, in the past 15 years the cryptography community has improved its efficiency by more than five orders of magnitude [11,32,43]. Owing to these advances we increasingly see these protocols used in big data workflows on untrusted servers [12,57,58].

Opaque [64], Shrinkwrap [10], and Hermetic [61] proposed analytical cost models to optimize the performance of private query evaluation. These approaches supported oblivious query processing with the latter two incorporating differential privacy into their analysis to minimize query runtime. Similarly, BlinkDB [3] and VerdictDB [49] optimize the sampling rates of their queries to provide high performance with strong accuracy guarantees, but do not offer any privacy guarantees. SAQE takes the best of both worlds. SAQE's cost model estimates the performance of approximate query processing over secure computation and creates execution plans that meet user-set deadlines while maximizing accuracy.

8. CONCLUSIONS

SAQE is a private data federation that answers SQL queries over the union of multiple datasets without requiring data providers to disclose their private records. It introduces a novel generalization of approximate query processing for private query processing, optimizing the sampling rate and query result noise to maximize query accuracy while leaking no information about a query's private inputs. We validate this work on synthetic and real-world workloads to verify that SAQE identifies the inflection point between sampling and differential privacy that maximizes the query result accuracy.

9. ACKNOWLEDGEMENTS

This work was supported by National Science Foundation under the grant CNS-1846447 and NSERC through a Discovery Grant.

10. REFERENCES

[1] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 308–318. ACM, 2016.

[2] N. Agarwal, A. T. Suresh, F. X. Yu, S. Kumar, and B. McMahan. cpsgd: Communication-efficient and differentially-private distributed SGD. In *NeurIPS*, 2018.

[3] S. Agarwal, B. Mozafari, A. Panda, H. Milner, S. Madden, and I. Stoica. BlinkDB. In *Proceedings of the 8th ACM European Conference on Computer Systems - EuroSys '13*, page 29, New York, New York, USA, 2013. ACM Press.

[4] G. Aggarwal, M. Bawa, P. Ganesan, H. Garcia-Molina, K. Kenthapadi, R. Motwani, U. Srivastava, D. Thomas, and Y. Xu. Two can keep a secret: A distributed architecture for secure database services. *CIDR*, 2005.

[5] M. R. Albrecht, C. Rechberger, T. Schneider, T. Tiessen, and M. Zohner. Ciphers for MPC and FHE. In E. Oswald and M. Fischlin, editors, *EUROCRYPT 2015, Part I*, volume 9056 of *LNCS*, pages 430–454. Springer, Heidelberg, Apr. 2015.

[6] A. Arasu, S. Blanas, K. Eguro, R. Kaushik, D. Kossmann, R. Ramamurthy, and R. Venkatesan. Orthogonal Security with Cipherbase. *CIDR 2013, Sixth Biennial Conference on Innovative Data Systems Research, Asilomar, CA, USA, January 6-9, 2013, Online Proceedings*, 2013.

[7] G. Asharov, I. Komargodski, W.-K. Lin, K. Nayak, E. Peserico, and E. Shi. Optorama: Optimal oblivious ram. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 403–432. Springer, 2020.

[8] B. Balle, G. Barthe, and M. Gaboardi. Privacy amplification by subsampling: Tight analyses via couplings and divergences. In *Proceedings of the 32Nd International Conference on Neural Information Processing Systems*, NIPS'18, pages 6280–6290, USA, 2018. Curran Associates Inc.

[9] J. Bater, G. Elliott, C. Eggen, S. Goel, A. Kho, and J. Rogers. SMCQL: secure querying for federated databases. *PVLDB*, 10(6):673–684, 2017.

[10] J. Bater, X. He, W. Ehrich, A. Machanavajjhala, and J. Rogers. Shrinkwrap: efficient sql query processing in differentially private data federations. *PVLDB*, 12(3):307–320, 2018.

[11] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: a system for secure multi-party computation. In P. Ning, P. F. Syverson, and S. Jha, editors, *ACM CCS 2008*, pages 257–266. ACM Press, Oct. 2008.

[12] D. Bogdanov, S. Laur, and J. Willemson. Sharemind: A framework for fast privacy-preserving computations. In S. Jajodia and J. López, editors, *ESORICS 2008*, volume 5283 of *LNCS*, pages 192–206. Springer, Heidelberg, Oct. 2008.

[13] P. Bogetoft, D. L. Christensen, I. Damgård, M. Geisler, T. Jakobsen, M. Krøigaard, J. D. Nielsen, J. B. Nielsen, K. Nielsen, J. Pagter, M. I. Schwartzbach, and T. Toft. Secure multiparty computation goes live. In R. Dingledine and P. Golle, editors, *FC 2009*, volume 5628 of *LNCS*, pages 325–343. Springer, Heidelberg, Feb. 2009.

[14] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, page 1175–1191, New York, NY, USA, 2017. Association for Computing Machinery.

[15] M. Bun, K. Nissim, U. Stemmer, and S. P. Vadhan. Differentially private release and learning of threshold functions. In *IEEE 56th Annual Symposium on Foundations of Computer Science, FOCS 2015, Berkeley, CA, USA, 17-20 October, 2015*, pages 634–649, 2015.

[16] T. H. Chan, E. Shi, and D. Song. Optimal lower bound for differentially private multi-party aggregation. In *Algorithms - ESA 2012 - 20th Annual European Symposium, Ljubljana, Slovenia, September 10-12, 2012. Proceedings*, pages 277–288, 2012.

[17] S. Chaudhuri, G. Das, and V. Narasayya. Optimized stratified sampling for approximate query processing. *ACM Transactions on Database Systems (TODS)*, 32(2):9, 2007.

[18] H. Chen, I. Chillotti, Y. Dong, O. Poburinnaya, I. Razenshteyn, and M. S. Riazi. Sanns: Scaling up secure approximate k-nearest neighbors search. *arXiv preprint arXiv:1904.02033*, 2019.

[19] A. R. Chowdhury, C. Wang, X. He, A. Machanavajjhala, and S. Jha. Cryptϵ: Crypto-assisted differential privacy on untrusted servers. In *SIGMOD*, 2020.

[20] A. Crotty, A. Galakatos, E. Zgraggen, C. Binnig, and T. Kraska. Vizdom: interactive analytics through pen and touch. *PVLDB*, 8(12):2024–2027, 2015.

[21] C. Dwork. Differential privacy. *Proceedings of the 33rd International Colloquium on Automata, Languages and Programming*, pages 1–12, 2006.

[22] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our Data, Ourselves: Privacy Via Distributed Noise Generation. *Proceedings of EUROCRYPT'06*, 4004:486–503, 2006.

[23] C. Dwork and A. Roth. The algorithmic foundations of differential privacy. *Found. Trends Theor. Comput. Sci.*, 2014.

[24] H. Ebadi, T. Antignac, and D. Sands. Sampling and partitioning for differential privacy. In *2016 14th Annual Conference on Privacy, Security and Trust (PST)*, pages 664–673, Dec 2016.

[25] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067. ACM, 2014.

[26] C. Ge, X. He, I. F. Ilyas, and A. Machanavajjhala. APEx. In *Proceedings of the 2019 International Conference on Management of Data - SIGMOD '19*, number 1, pages 177–194, New York, New York, USA, 2019. ACM Press.

[27] P. B. Gibbons. Distinct sampling for highly-accurate answers to distinct values queries and event reports. *VLDB 2001 - Proceedings of 27th International Conference on Very Large Data Bases*, pages 541–550, 2001.

[28] M. T. Goodrich. Data-oblivious external-memory algorithms for the compaction, selection, and sorting of outsourced data. In *Proceedings of the twenty-third annual ACM symposium on Parallelism in algorithms and architectures*, pages 379–388, 2011.

[29] X. He, A. Machanavajjhala, C. Flynn, and D. Srivastava. Composing differential privacy and secure computation: A case study on scaling private record linkage. In *Proceedings*

of the 2017 ACM SIGSAC Conference on Computer and Communications Security, pages 1389–1406. ACM, 2017.

[30] Z. He, W. K. Wong, B. Kao, D. W. L. Cheung, R. Li, S. M. Yiu, and E. Lo. SDB: a secure query processing system with data interoperability. *PVLDB*, 8(12):1876–1879, 2015.

[31] J. M. Hellerstein, P. J. Haas, and H. J. Wang. Online aggregation. In *Acm Sigmod Record*, volume 26, pages 171–182. ACM, 1997.

[32] Y. Huang, D. Evans, J. Katz, and L. Malka. Faster secure two-party computation using garbled circuits. In *USENIX Security 2011*. USENIX Association, Aug. 2011.

[33] N. Johnson, J. P. Near, J. M. Hellerstein, and D. Song. Chorus: Differential privacy via query rewriting, 2018.

[34] N. Johnson, J. P. Near, and D. Song. Towards practical differential privacy for sql queries. In *VLDB*, 2018.

[35] S. Kandula, A. Shanbhag, A. Vitorovic, M. Olma, R. Grandl, S. Chaudhuri, and B. Ding. Quickr: Lazily approximating complex adhoc queries in bigdata clusters. In *Proceedings of the 2016 international conference on management of data*, pages 631–646. ACM, 2016.

[36] I. Kotsogiannis, Y. Tao, X. He, M. Fanaeepour, A. Machanavajjhala, M. Hay, and G. Miklau. Privatesql: a differentially private sql query engine. *PVLDB*, 12(11):1371–1384, 2019.

[37] I. Kotsogiannis, Y. Tao, A. Machanavajjhala, G. Miklau, and M. Hay. Architecting a differentially private sql engine. In *CIDR*, 2019.

[38] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *Proceedings of the 2016 International Conference on Management of Data, SIGMOD Conference 2016, San Francisco, CA, USA, June 26 - July 01, 2016*, 2016.

[39] F. Li, B. Wu, K. Yi, and Z. Zhao. Wander join: Online aggregation via random walks. In *Proceedings of the 2016 International Conference on Management of Data*, pages 615–629. ACM, 2016.

[40] Y. Lindell. Fast secure two-party ECDSA signing. In J. Katz and H. Shacham, editors, *CRYPTO 2017, Part II*, volume 10402 of *LNCS*, pages 613–644. Springer, Heidelberg, Aug. 2017.

[41] E. Lobo-Vesga, A. Russo, and M. Gaboardi. A Programming Framework for Differential Privacy with Accuracy Concentration Bounds. pages 1–22, 2019.

[42] A. Machanavajjhala, D. Kifer, J. M. Abowd, J. Gehrke, and L. Vilhuber. Privacy: Theory meets practice on the map. In *ICDE*, 2008.

[43] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - secure two-party computation system. In M. Blaze, editor, *USENIX Security 2004*, pages 287–302. USENIX Association, Aug. 2004.

[44] A. McGregor, I. Mironov, T. Pitassi, O. Reingold, K. Talwar, and S. Vadhan. The limits of two-party differential privacy. In *2010 IEEE 51st Annual Symposium on Foundations of Computer Science*, 2010.

[45] R. McKenna, G. Miklau, M. Hay, and A. Machanavajjhala. Optimizing error of high-dimensional statistical queries under differential privacy. *PVLDB*, 11(10):1206–1219, 2018.

[46] F. D. McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data*, SIGMOD '09, pages 19–30, New York, NY, USA, 2009. ACM.

[47] I. Mironov, O. Pandey, O. Reingold, and S. Vadhan. Computational differential privacy. In *CRYPTO*, 2009.

[48] A. Narayan and A. Haeberlen. DJoin: differentially private join queries over distributed databases. *Proceedings of the 10th USENIX Symposium*, page 14, 2012.

[49] Y. Park, B. Mozafari, J. Sorenson, and J. Wang. VerdictDB: Universalizing approximate query processing. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, pages 1461–1476, 2018.

[50] PCORI. Exchanging de-identified data between hospitals for city-wide health analysis in the Chicago Area HealthLNK data repository (HDR). *IRB Protocol*, 2015.

[51] R. Popa and C. Redfield. CryptDB: protecting confidentiality with encrypted query processing. *SOSP*, pages 85–100, 2011.

[52] L. Qin, A. Lapets, F. Jansen, P. Flockhart, K. D. Albab, I. Globus-Harris, S. Roberts, and M. Varia. From usability to secure computing and back again. In *Fifteenth Symposium on Usable Privacy and Security (SOUPS 2019)*, Santa Clara, CA, Aug. 2019. USENIX Association.

[53] S. Sasy and O. Ohrimenko. Oblivious sampling algorithms for private data analysis. In *Advances in Neural Information Processing Systems*, pages 6495–6506, 2019.

[54] R. Shokri and V. Shmatikov. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-16, 2015*, pages 1310–1321, 2015.

[55] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou. A hybrid approach to privacy-preserving federated learning. In *AISec*, New York, NY, USA, 2019. Association for Computing Machinery.

[56] S. Tu, M. F. Kaashoek, S. Madden, and N. Zeldovich. Processing analytical queries over encrypted data. *PVLDB*, 6(5):289–300, 2013.

[57] N. Volgushev, M. Schwarzkopf, A. Lapets, M. Varia, and A. Bestavros. Integrating mpc in big data workflows. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 1844–1846. ACM, 2016.

[58] N. Volgushev, M. Varia, M. Schwarzkopf, A. Lapets, B. Getchell, and A. Bestavros. Conclave: Secure multi-party computation on big data. In *Proceedings of the 14th EuroSys Conference 2019*, volume 70, pages 1–18, New York, New York, USA, aug 2019. ACM Press.

[59] X. Wang, A. J. Malozemoff, and J. Katz. EMP-Toolkit: Efficient Multiparty Computation Toolkit. https://github.com/emp-toolkit, 2016.

[60] R. J. Wilson, C. Y. Zhang, W. Lam, D. Desfontaines, D. Simmons-Marengo, and B. Gipson. Differentially Private SQL with Bounded User Contribution. pages 1–20, 2019.

[61] M. Xu, A. Papadimitriou, A. Feldman, and A. Haeberlen. Using Differential Privacy to Efficiently Mitigate Side Channels in Distributed Analytics. In *Proceedings of the 11th European Workshop on Systems Security - EuroSec'18*, pages 1–6, New York, New York, USA, 2018. ACM Press.

[62] A. C.-C. Yao. How to generate and exchange secrets (extended abstract). In *27th FOCS*, pages 162–167. IEEE Computer Society Press, Oct. 1986.

[63] N. Zhang, M. Li, and W. Lou. Distributed Data Mining with Differential Privacy. In *2011 IEEE International Conference on Communications (ICC)*, pages 1–5. IEEE, jun 2011.

[64] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. *14th {USENIX} Symposium on Networked Systems Design and Implementation, {NSDI} 2017, Boston, MA, USA, March 27-29, 2017*, pages 283–298, 2017.