KloakDB: Distributed, Scalable, Private Data Analytics with K-anonymous Query Processing

Madhav Suresh
Northwestern University
madhav@u.northwestern.edu

Zuohao She
Northwestern University
zuohaoshe2013@u.northwestern.edu

Adel Lahlau
Northwestern University
alahlou@u.northwestern.edu

William Wallace
Northwestern University
williamwallace2018@u.northwestern.edu

Jennie Rogers
Northwestern University
jennie@northwestern.edu

ABSTRACT

Private data federations enable multiple data owners to query the union of their secret data such that no party has access to the private inputs of another and the only information revealed is that which can be gleaned from the query output. Here, the data owners compute the query result amongst themselves using cryptographic techniques. *Oblivious computation* guarantees that a program's observable transcript – its instruction traces, I/Os, and network utilization – does not change as a function of a query's private inputs. Although this protects against side channel leakage, it has query runtimes that are multiple orders of magnitude higher than running the same query with no privacy guarantees. This performance overhead makes it impractical for large workloads.

We propose a framework that offers a private data federation users a decision space between these two extremes of full obliviousness and no privacy during query evaluation. KloakDB has a novel semi-oblivious query processing model, and it provides users with a fine-grained privacy-performance trade off. It generalizes k-anonymity to model the information revealed by a federation during query evaluation. k-anonymous query processing, is based on the widely deployed privacy model k-anonymity. It is the de-facto standard for data releases in many domains including healthcare and education research. K-anonymous query processing ensures that side channel information revealed during query execution for each record is indistinguishable from those of k or more records. With modest values for k, KloakDB demonstrates speedups of 15×–1060× over oblivious query processing.

PVLDB Reference Format:
Madhav Suresh, Zuohao She, Adel Lahlau, William Wallace, and Jennie Rogers. KloakDB: Distributed, Scalable, Private Data Analytics with K-anonymous Query Processing. PVLDB, 14(1): XXX-XXX, 2020.
doi:XX.XX/XXX.XX

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at http://vldb.org/pvldb/format_vol14.html.

Proceedings of the VLDB Endowment, Vol. 14, No. 1 ISSN 2150-8097.
doi:XX.XX/XXX.XX

1 INTRODUCTION

Private data federations enable data owners to query the union of their datasets without sharing their input tuples with one another. The data owners compute the query amongst themselves using cryptographic techniques such that the only information they glean is the input size of each participant and (at most) the query output. Unfortunately, these techniques exact a breathtaking performance penalty of several orders of magnitude to achieve strong theoretical privacy guarantees. On the other hand, revealing query runtimes, intermediate operator cardinalities, memory access patterns, network traces, and other metadata provides robust side-channel information about a query's private inputs to a data owner participating in its evaluation. If we were to evaluate our queries by computing over encrypted data using standard DBMS operator algorithms, their instruction traces would leak substantial information about the query's private inputs [29, 30, 36]. In this work we propose a middle way between these two extremes to provide practical security guarantees at runtime for private data federation queries.

In prior work, data owners compute the query using oblivious algorithms. We say that an algorithm's execution is *oblivious* if its observable transcript – its instruction traces, I/Os, and network transmissions – are not predicated on the contents of its private inputs. Although oblivious query processing provides strong privacy guarantees, its performance makes it impractical for all but the smallest query workloads. For example, a fully oblivious query with multiple joins requires each join's output cardinality to be equal to the cross product of the inputs. Hence, a 3-way join will incur an overhead of $O(n^3)$ and produce an output cardinality of the same magnitude. Existing systems perform this oblivious query evaluation either in trusted hardware – via secure enclaves [3, 13, 22, 74] – or software, using secure multi-party computation (MPC) [5, 6, 66]. Our approach is agnostic to this design choice and we support both types of *secure computation back-ends* in this work.

Oblivious query processing can be too strict in many use cases. Despite that, many systems need a certain level of anonymization for privacy and regulatory compliance, such as the GDPR's pseudonymization proviso. It allows "the processing of personal data in such a manner that the personal data can no longer be attributed to a specific data subject without the use of additional information" [33]. To take advantage of this opportunity, we propose k-anonymous query processing, a *semi-oblivious query processing* model based on k-anonymity. A data release is k-anonymous [61] if the records of an individual are indistinguishable from those of at least $k-1$ others. We generalize this work by making a private data federation's instruction traces k-anonymous. Hence, when a curious data owner observes a query's evaluation – e.g., its branching and looping – the only information they can glean is in batches of k tuples or more. Note that this differs from k-anonymous data releases because the attacker only sees encrypted data throughout the query's lifecycle. This framework creates a tunable privacy-performance trade off by enabling the private data federation's stakeholders to tune the k with which we process our queries. This model provides protection against side channel leakage and significant performance benefits over oblivious query processing.

K-anonymity is currently the standard for data releases in the context of electronic health records [16, 21, 49], education research [12, 23, 59], government data releases [24, 26, 73], and others. GDPR has a data pseudonymization provisio, permitting data controllers and data processors to relax their data protection obligations by removing direct identifiers from a data set and limiting indirect identifiers so long as the threat of re-identification is not "reasonably likely" [32]. Because of its widespread use, we propose this model of semi-oblivious query processing despite the well-known shortcomings of k-anonymity [1, 18, 25, 42–44]. The widespread use of this privacy model enables this work to have immediate impact.

We present KloakDB, a private data federation that offers k-anonymous query processing to protect query inputs at runtime. KloakDB offers an end-to-end architecture to uphold its security guarantees. To do this we designed an extensible architecture that accounts for information leakage *before and after* the query runs, in addition to its core runtime guarantees. *Before* it may run its first query, the system takes in a query workload – a set of query templates – and a set of shared table definitions supported by all of its member DBMSs. It analyzes the workload to identify attributes that will alter the control flow of its queries and constructs a *k-anonymous processing view* over these columns. We propose a lightweight, oblivious algorithm to compute this view efficiently. *After* k-anonymous query processing, the data owners must still work amongst themselves to protect the output of the query from reverse engineering attacks. There are already well-known techniques for doing this for common data release mechanisms such as k-anonymity [39] and differential privacy [45]. KloakDB implements an efficient, oblivious data release mechanism based on the one in Shrinkwrap [6].

To summarize, our main contributions are:

- Formalizing the model of k-anonymous query processing as a suite of semi-oblivious database operators that offers a privacy-performance trade off for computing its queries.
- Designing and implementing KloakDB, a private data federation prototype that provides end-to-end guarantees for queries executed k-anonymously.
- Evaluating KloakDB on both synthetic and real-world workloads, verifying the performance, verifying the advantage of and privacy trade-offs k-anonymous query processing and the efficiency of our end-to-end workflow.

The rest of this paper is organized as follows. In Section 2 we review the key ingredients for our approach. After that, Section 3 introduces KloakDB with a system overview and its workflow which we illustrate with a running example. Next, we formalize k-anonymous query processing in Section 4 and go in-depth on the KloakDB architecture in Section 5. Section 6 reveals our experimental results. After that, we survey the related work and conclude.

2 BACKGROUND

KloakDB combines ideas and tools from multiple facets of privacy, security, and data management. We introduce the building blocks that we use in this section. We provide overviews of k-anonymity, private data federations, secure computation backends, side channel leakage, and oblivious query processing.

2.1 K-anonymity

K-anonymity was proposed by Sweeney [61] as a solution to *linking attacks* on publicaly released datasets. A linking attack uses a private release of a dataset in conjunction with publicaly available information in order to identify and single out individuals. Here, the existence of certain attributes, *quasi-identifiers*, available in both the privately released dataset as well as the public datasets was the root cause of the relinking attacks. K-anonymity tackles this by requiring all projections onto a databases's quasi-identifiers to have at least k tuples. Recall that, k-anonymity is the cornerstone of regulatory requirements in numerous industries.

Definition 2.1. Quasi-identifier (informal) On a relation R, a quasi-identifier (QI) is a sensitive attribute that may enable attackers to reidentify individuals in a dataset..

By the Definition 2.1, quasi-identifiers are the attributes that can be potentially used in linking attacks. The selection of quasi-identifiers follows the best practices of the domain where k-anonymity is deployed.

Definition 2.2. K-anonymity *A data set is said to satisfy k-anonymity for an integer $k > 1$ if, for each combination of values of quasi-identifier attributes, at least k records exist in the data set with that combination.*

Limitations of K-anonymity. We consider the limitations of k-anonymity. We note that common vulnerabilities to k-anonymity *do not directly apply* to our system, as follows:

- *Attribute disclosure:* The private data federation agrees upon the query workload a priori, and thus can reject queries whose execution would result in attribute disclosure. Query admission is not in the scope of this paper.
- *Data Utility:* Input data is often modified to achieve k-anonymity. This results in data utility loss. In our system, we do not modify the input data, by default.
- *Composability:* Our system does not release anonymized data sets. Additionally, KloakDB assumes a fixed query workload.

2.2 Private Data Federations

A private data federation enables multiple, autonomous database systems to pool their sensitive data for analysis while keeping their input records private. It starts with a common set of table definitions against which the client queries. The tables may correspond to data on a single host or they may be horizontally partitioned among multiple data owners. We focus on the latter scenario in this work, although the system's core architecture is amenable to both. The shared schema is publicly available, and a query workload is agreed upon before running the first query. Each data owner wishes to keep their sensitive tuples private, but they are willing to reveal the results of queries over the union of their data with that of other federation members. The data owners may optionally add a security policy for the analysis on their combined data, such as restricting the columns that are visible to the client or noising their results using differential privacy. This work does not address access control on data release.

2.3 Privacy-preserving Query Processing

Privacy-preserving query evaluation provides confidentiality for private data federations during query runtime. KloakDB utilizes hardware enclaves and secure multiparty computation to uphold its guarantees. We refer to these as *secure computation backends*. The secure computation backend implementations are interchangable in our system architechture, allowing users to tailor the system to their specific security and privacy needs.

Secure Multiparty Computation. Secure multiparty computation (MPC) enables multiple parties to compute functions on secret data without revealing the underlying inputs. Given a function $f(x, y) = z$, two parties can computer over their private data x, y and get the result of the function z. KloakDB uses EMP Toolkit [68] as our MPC backend, for the speed that it offers, as well as ease of interface. EMP Toolkit implements a *semi-honest* two party protocol. In the *semi-honest* setting, we assume that parties will honestly follow the protocol, however they remain curious and monitor side channels to gain unauthorized information about the private inputs of other protocol participants. EMP-Toolkit implements a malicious two party protocol, we leave it to future work to extend KloakDB to support this. Our current MPC implementation computes among two parties. Recent work has demonstrated efficient MPC protocols for more than two parties [69], we leave it to future work to extend our implementation to support more than two parties.

Hardware Enclaves. Trusted execution environments such as Intel SGX [10] and AMD Memory Encryption [35] are available on most new commodity systems. The code and data associated with an enclave is *sealed*; the system in which it is executing may not view or change its contents. This enclave uses remote attestation to prove to an authority, e.g., that the code and data it is running has not been tampered with and that the code executes on trusted hardware alone. Once an enclave has attested its code, this opens up a secure communication channel to which the parties to send their sensitive data. Secure enclaves have a protected region of memory, the encrypted page cache (EPC), that is not accessible by the host operating system or hypervisor. ntel SGX's memory protection has received substantial interest from the security community, with side channel attacks being discovered [10, 38, 64, 65, 67] and fixed [11, 58, 60] on a regular basis. Addressing the shortfalls of present-day hardware enclave implementations is beyond the scope of this work.

2.4 Side Channel Leakage

Side channel leakage is the information that is revealed through observing a program's execution behavior. Memory access patterns, network transmissions,and CPU instruction traces are examples of the execution behavior of a program. With query processing, output cardinalities and execution time are examples of execution behavior which leak side channel information. Adversarial parties observe these side channels in order to infer sensitive information about the input data. While these adversaries are considered *passive*, these side channels can lead to powerful attacks on private data. Xu et al [71] showed that access pattern leakage can extract complete text documents and outlines of JPEG images. Through repeated querying and with access to the query distribution and the output cardinalities, an adversary can determine secret attributes of individual records in a database [36]. On a macro level, query processing can reveal secret distributional information about the input data [3].

Oblivious Query Processing. Existing work solves the problem of side channel leakage with oblivious query processing. Here the parties computing the query learn nothing more about the inputs of others than they would if all of the members of the federation uploaded their data to a trusted third party that ran the query and returned its results to the client. However the substantial overheads associated with running a leak-free program make it impractical to use on large data sets or complex queries. For example an oblivious equi-join on two relations of length n will unconditionally output n^2 output tuples. Oblivious operator algorithms exist that don't have $O(n^2)$ complexity, and we discuss these further in Section 7. Oblivious query processing's all or nothing approach is inflexible, not offering federation members tunable trade-offs for privacy and performance, while upholding regulatory compliance in relevant domains.

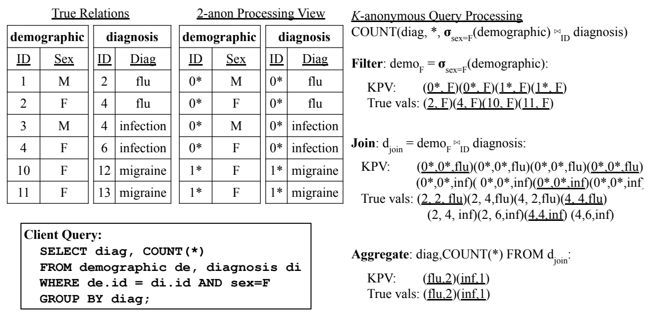
3 KLOAKDB

In this section we describe the preliminaries of KloakDB. First we define the system's trust model and security guarantees. We then describe the architecture of the private data federation and walk through the life of a query.

3.1 The KloakDB Private Data Federation

The KloakDB private data federation consists of three classes of parties:

- *Query Client:* The client has access to the shared schema, and accepted query workload that the federation has agreed upon. The client can issue any query from the query workload to the federation. The client receives an encrypted query result that has been passed through a data release mechanism, from the federation. It can observe the amount of time the query takes, however it does not have access to intermediate results.
- *Data Owners:* Data owners agree upon a shared set of table definitions, and queries before joining the data federation. All query processing and computation happens within the pooling of the data owners' resources. Data owners receive



True Relations

demographic		diagnosis	
ID	Sex	ID	Diag
1	M	2	flu
2	F	4	flu
3	M	4	infection
4	F	6	infection
10	F	12	migraine
11	F	13	migraine

2-anon Processing View

demographic		diagnosis	
ID	Sex	ID	Diag
0*	M	0*	flu
0*	F	0*	flu
0*	M	0*	infection
0*	F	0*	infection
1*	F	1*	migraine
1*	F	1*	migraine

```
Client Query:
  SELECT diag, COUNT(*)
  FROM demographic de, diagnosis di
  WHERE de.id = di.id AND sex=F
  GROUP BY diag;
```

K-anonymous Query Processing
COUNT(diag, *, $\boldsymbol{\sigma}_{sex=F}$(demographic) $\bowtie_{ID}$ diagnosis)

Filter: demo$_F$ = $\boldsymbol{\sigma}_{sex=F}$(demographic):

KPV: $\underline{\text{(0*, F)}}\underline{\text{(0*, F)}}\underline{\text{(1*, F)}}\underline{\text{(1*, F)}}$
True vals: $\underline{\text{(2, F)}}\underline{\text{(4, F)}}\underline{\text{(10, F)}}\underline{\text{(11, F)}}$

Join: d$_{join}$ = demo$_F$ $\bowtie_{ID}$ diagnosis:

KPV: $\underline{\text{(0*,0*,flu)}}$(0*,0*,flu)(0*,0*,flu)$\underline{\text{(0*,0*,flu)}}$
(0*,0*,inf)(0*,0*,inf)$\underline{\text{(0*,0*,inf)}}$(0*,0*,inf)
True vals: $\underline{\text{(2, 2, flu)}}$(2, 4,flu)(4, 2,flu)$\underline{\text{(4, 4,flu)}}$
(2, 4, inf)(2, 6,inf)$\underline{\text{(4,4,inf)}}$ (4,6,inf)

Aggregate: diag,COUNT(*) FROM d$_{join}$:

KPV: $\underline{\text{(flu,2)}}\underline{\text{(inf,1)}}$
True vals: $\underline{\text{(flu,2)}}\underline{\text{(inf,1)}}$

Figure 1: *K*-anonymous query processing example. True (non-dummy) tuples are underlined.

encrypted secret inputs from each other. The only unencrypted data they have have access to is their own. Through MPC or trusted hardware, they are able to compute over encrypted data shared by other data owners, while combining their secret data. Data owners will snoop on all side channels.

- *Coordinator:* The coordinator is the bridge between the client and the data owners, the coordinator is an intermediary who assists in query planning, distributed query execution planning, and data release. The coordinator only has access to encrypted data from the data owners. We note that the coordinator can be chosen amongst the data owners, or as an external third party.

3.2 Running Example

Consider the query in Figure 1. It counts the times a woman is diagnosed with a given ailment. It first filters the demographic table for women, joins the selected tuples with the diagnosis table, and counts the times each condition appears in the join result. Sex and diagnosis have a k-anonymous security policy where $k = 2$. We demonstrate this in the single database setting such as that of outsourced operations in the cloud.

The query planner first creates a 2-anonymous processing view. To illustrate this, we use generalization–omitting the least significant digit of the IDs–instead of KloakDB's freeform k-anonymization here. The view must have at least two individuals (IDs) in each equivalence class. All of the tuples in an equivalence class are indistinguishable from one another during query processing. We divide the ID column by 10 in both relations to suppress the least significant digit. Each relation has three equivalence classes, and for demographic they are: (0*, F) (0*, M), and (1*, F). Each equivalence class has a bitmask with a bit or dummy tag for each tuple denoting if it is a placeholder to mask the role of individual tuples in the group. When we run the query, the filter first examines each equivalence class and either 1) outputs it in its entirety if it contains at least one match–obliviously marking a dummy tag on each tuple to denote if it met the selection criteria; or 2) produces an empty set. The filter outputs two of the three demographic equivalence classes.

Next, we join the filtered demographics tuples with the diagnoses using the same all-or nothing logic to uphold tuple indistinguishability over an equivalence class. When the join compares two equivalence classes its output is either size of their cross-product of its inputs or an empty set. This join outputs two equivalence classes: (0*, flu) and (0*, infection) and three true tuple matches. If we ran this obliviously the join would output the cross-product or 36 tuples, instead of the 8 shown here. Clearly, k-anonymous query processing has an opportunity to substantially boost the performance of private data federation queries.

After the join, the aggregate iterates on its results one equivalence class at a time to count up the diagnoses for each ailment. For a given group-by bin, an aggregate outputs either: 1) a single tuple if >= k individuals contributed to it; or 2) a dummy-padded set of tuples equal in length to the source equivalence class. An observer can learn about no fewer than k individuals at a time by observing these outcomes because they either learn that all of the tuples in the class had the same group-by value or that we processed the equivalence class obliviously. At first glance, it may appear that the group-by of (0*, infection) would be processed obliviously. As we will see in the coming sections, the anonymity of an equivalence class is transitive as it passes through a k-anonymous operator. Because the join compares all tuples in an equivalence class to its potential matches in the joining relation, its output is fully padded. Hence, the join did not reveal its selectivity over this equivalence class and the groups with which it was paired. Then the count operator visits all four tuples in the infection group and emits a single tuple with the true count.

3.3 Threat Model

We delegate the trust model to the secure computation backend. In this setting we consider two possible trust models: 1) honest-but-curious, 2) malicious. Our modular architecture relies on the secure computation backend for data protection , and so the trust model reduces to the trust model of the secure computation backend.

Honest-but-curious is considered the standard trust model for the untrusted cloud setting [46]. Honest-but-curious and *semi-honest* can be used interchangeably to describe the same trust model. The honest-but-curious trust model assumes that all parties will honestly follow the protocol defined by the data federation, but will curiously monitor side channels to gain access private information of other federation members. The trust model implies that data owners will provide their genuine datasets into the data federation; maliciously crafted inputs are not allowed. Data owners and clients will attempt to learn as much as possible from observing the query's execution. Data owners will monitor memory, CPU, and network side channels, in addition to operator execution time and intermediate result sizes with local and distributed operators. For example, a federation member upholding honest-but-curious trust assumptions will faithfully execute enclave code, but will also snoop on memory channels to gather memory access pattern information.

In the malicious security model, participants are allowed to deviate arbitrarily from the protocol. The protocol still guarantees that the privacy of participant inputs is maintained. While the privacy guarantees are significantly stronger, malicious protocols have significant overhead compared to honest-but-curious protocols. By default KloakDB implements honest-but-curious protocols.

3.4 Problem Statement

We consider a private data federation $\mathcal{F}$ consisting of one client, $\mathbb{C}$, and n data owners $\mathcal{DO} = \{D_1, ..., D_n\}$ with a shared schema,

$\mathcal{T}$, consisting of relations $\mathcal{R} = \{R_1, ..., R_m\}$. The data owners agree upon a query workload $Q = \{Q_1, ..., Q_p\}$, where Q_i is a query template. The client, $\mathbb{C}$, may issue any query instance, q_i that is based on template $Q_i \in Q$. The client issues the query to a coordinator.

Consider the following query lifecycle. A query q_i is issued to the coordinator, who plans the query and constructs a distributed query execution plan. With the secure computation backends, the data owners run the distributed query execution plan, and then return the encrypted results to the coordinator. The coordinator then runs a data release mechanism in its secure computation backend, and then returns the final result to the client.

We now introduce the security guarantees of k-anonymous query processing. We reason about these properties in terms of the database columns that will impact the control flow of a private data federation query. Hence:

Definition 3.1. **Control Flow Attribute**: For a query instance q_i on schema $\mathcal{T}$, the *control flow attributes* are the set of attributes $C \in \mathcal{T}$ that change the instruction traces of the operators in q_i's query tree.

Control flow attributes are the private attributes that affect the query's observable transcript from the data owners' point-of-view. It encapsulates the query's instruction traces – including branching, loop iterations, early termination – as well as its hardware utilization, e.g., memory access patterns, network traces, and CPU time. Metadata associated with the query's evaluation, such as its intermediate result cardinalities, are also a part of its side-channel leakage. The control flow attributes of joins are their join keys. Filters use the columns referenced in their predicates for this, aggregates have their transcript informed by their group-by columns, sorts use the order-by expression, and set operations reference all columns in their schema. As tuples move up the query tree, KloakDB projects out any unnecessary columns after each operator. We union the control flow attributes of each operator in the query tree to derive the query's end-to-end column set. KloakDB's query evaluation is k-anonymous with respect to each query's control flow attributes.

We now introduce the anonymity guarantee KloakDB offers data owners during query execution. KloakDB queries have instruction traces that have the same distribution as they would if the query were executing over a view of the dataset that is k-anonymous with respect to its control flow attributes. With this in mind, KloakDB guarantees:

Definition 3.2. K**-anonymous Query Processing**: Consider a query instance,q_i, that accesses relations specified by the schema $\mathcal{T}$. It has control flow attributes, C_i where $C_i \subseteq \mathcal{T}$. When the system evaluates q_i on a dataset, $\mathcal{D}$, there exists a function $\mathcal{V}_C$ for creating a k-anonymous view of $\mathcal{D}$ with respect to C_i. A semi-oblivious algorithm for running the query, Q', satisfies the requirements of k-anonymous query processing iff its instruction traces are computationally indistinguishable from those of a simulator running q_i over $\mathcal{V}_C$ ($\mathcal{D}$) for a probabilistic polynomial time adversary. In other words:

$$Trace(Q(\mathcal{V}_C(\mathcal{D}))) \overset{c}{\equiv} Trace(Q'(\mathcal{D}))$$

This definition guarantees that the instruction trace of a tuple will be indistinguishable from at least $k-1$ other tuples. Consider our query from the running example in shown in Figure 1. The query's execution is anonymous with respect to `demographic.sex`, `demographic.id`, diagnosis.id, and `diagnosis.diag`. A curious data owner may observe the size of the intermediate results after the join and use the instruction traces to deduce how many tuples contributed to each group-by bin. At the high level, an adversary may be able to trace a *single tuple* from start to finish for a federated query evaluation.

K-anonymous query processing ensures that each tuple that flows through cannot be singled out: the definition above guarantees that the traces of any single tuple will be shared amongst at least $k-1$ others. The definition is a *bottom up* definition: we focus on an adversary monitoring a single tuple from ingest until query release. The k-anonymous processing view's tunable security parameter, k, enables a private data federation to choose and enforce a security policy based on the workload.

4 K-ANONYMOUS QUERY PROCESSING

We will now dive into how KloakDB upholds its anonymity guarantees in Definition 3.2. Let's begin with a instance q_i that evaluates over the federation's shared schema, $\mathcal{T}$. The schema is a lossless, dependency-preserving join decomposition. To capture the information revealed as tuples flow up the query tree, we model our query processing in terms of a schema-level k-anonymous view comprised of natural joins as $R_* = R_1 \bowtie \ldots \bowtie R_n$. K-anonymous data releases partition the data into equivalence classes where each tuple in a class is indistinguishable from its peers with respect to its quasi-identifiers. Each equivalence class must contain k or more tuples. In KloakDB, our equivalence classes are k-anonymous with respect to the quasi-identifiers in $\mathcal{T}$ that are referenced in q_i.

Definition 4.1. **Multi-relation k-anonymity** Consider a k-anonymous processing view, $\mathcal{V}_C$, over relations $\mathcal{R}$ that is anonymized with respect to C. We say that this view of a database instance $\mathcal{D}$ is k-anonymous iff for every valid value $t_i \in C$, $\sigma_{C=t_i}(V_C(R_*))$ produces either $\geq k$ tuples or an empty set.

This is a generalization of [48], we extend it to take into account how the control flow attributes may be greater than the size of the private attributes in our schema definition. Because we do taint analysis on the query tree for each template, C_i includes all of the private columns we computed over as well as any attributes that change the control flow of the program after we compute on our first private value. In order for us to maintain a k-anonymous view of $\mathcal{D}$ among the data owners, an execution transcript of q_i running over $\mathcal{D}$ may reveal no more information than we could glean from observing an execution of q_i over $\mathcal{V}_C$ (R_*). Individual k-anonymous operators in KloakDB do this by obliviously evaluating over each equivalence class discretely. This upholds the k-anonymous view among the data owners owing to the following property:

Subset Property [39]: If $\mathcal{D}$ is k-anonymous with respect to C, then $\mathcal{D}$ is k-anonymous with respect to any set of attributes C^* such that $C^* \subseteq C$.

Proof: Consider the frequency set of $\mathcal{D}$ with respect to C. If we remove any attribute C_i from C, each of its equivalence classes will either remain the same or it will coalesce with another one. Thus each frequency set will be greater than or equal to its previous size. □

Hence, every $C_i \subseteq C$ is itself k-anonymous. In addition, we need to ensure that as we sequentially run operators in the query tree such that composing them will uphold our security guarantees:
Transitivity Property: Given a relation R_i that is k-anonymous with respect to C, the execution and the intermediate cardinalities of any transformations predicated on C or $C_i \subseteq C$ are themselves k-anonymous.
Proof: In $\mathcal{V}_C$ $(\mathcal{D}$) each tuple is indistinguishable from at least $k-1$ others. Thus the transcript of transformations on a k-anonymous relation cannot reveal information that is not present in the source view. $\square$

Owing to the transitivity property, we reason about the view over which we compute each operator in the query tree. Each input relation, R_j, is anonymized with respect to $C_i \subseteq C$, its subset of the control flow attributes. Since every k-anonymous operator that computes on R_j will reveal information about its control flow attributes – or a subset thereof – it will uphold the federation's anonymized view of the data. Hence for all $R_j \in R$, we create a *k-anonymous processing view.*

Definition 4.2. K-**anonymous Processing View** Consider a relation R_j, and its control flow attributes, $C\ _j = C \cap R_j$. All computation over R_j – either alone or in conjunction with other relations – is at a minimum k-anonymous with respect to $C\ _j$. The relation is anonymized as $R'_j = V_{C_j}(R_j)$. R'_j satisfies our requirements for k-anonymous query processing iff for every valid value $t_j \in C$, $\sigma_{C=t_j}(V_C(R_*))$ produces either $\geq k$ tuples or an empty set. Its output admits duplicate rows.

When we compute over a k-anonymous processing view, we run a query Q over a subset of the relations, $Q(V_{C_1}(R_1) \bowtie \ldots \bowtie V_{C_i}(R_i))$. Since we are eagerly anonymizing the control flow attributes, our execution traces will protect at a level greater than or equal to that of running $Q(V_{C_i}(R_*))$.

Before we describe how KloakDB's operators provide the invariants above, we extend k-anonymous processing views to the federated setting. When considering anonymized views in Definition 3.2, the data is not combined with tuples from other hosts. If a k-anonymous processing view satisfies Definition 4.3, then it will also uphold the guarantees of k-anonymous query processing regardless of how much data was contributed by each host for a given operator. In other words, if the host does a what-if analysis of removing his or her tuples from the equivalence class, it will not expose data about fewer than k tuples.

Definition 4.3. **Federated** K-**anonymous Processing View** $\mathcal{D} = V_C(R_*)$ is horizontally partitioned over n hosts, $\mathcal{D} = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$. To ensure that no data owner learns about fewer than k tuples at a time, for every data owner i, their view of a KloakDB query runtime is k-anonymous in the absence of their contributed tuples. Hence, $\forall i \in n$, and for each $t_j \in$ the domain of C, $\sigma_{t_j=C}(V_C(\mathcal{D} - \mathcal{D}_i))$, produces either $\geq k$ tuples or an empty set. Its output rows may include duplicates.

4.1 K-anonymous Query Processing Operators

For this section, we consider the query Q and it's operator decomposition $O = \{o_1, ..., o_n\}$. K-anonymous operators take as input a k-anonymous processing view. These views have been anonymized with respect to the anonymization attribute set C, and Definition 4.3. The views are thus are partitioned into equivalence classes, where each equivalence class has at least k tuples. Each operator, o_i has a set of control flow attributes $c_{o_i} \subset C$.

Definition 4.4. **Anonymized Equivalence Class** *For a relation R_i with anonymized attribute set C_i, and equivalence class e_p with respect to attribute $p \in \mathcal{P}$ is a multi-set of tuples who share the same anonymized value for attribute p. Note: In order for a view to be a k-anonymous processing view, it must hold that the projection onto any anonymized equivalence class e_p have $\geq k$ tuples.*

Definition 4.5. K-**anonymous Query Processing Operators** An operator, O_j computes over 1-2 input relations $\{R_i\}$, with quasi-identifiers (or private attributes) P. P consists of both private attributes and any additional ones that compute over the output of a k-anonymous query evaluation. O_j is a k-anonymous query processing operator if the input relations and output relation form k-anonymous views, $V_{P\cup C}(\{R_i\})$, and $V_{P\cup C}(R_{output})$,respectively.

***Intuition for KQP**.* K-anonymous query processing guarantees that the instruction trace of a tuple will be indistinguishable from at least $k-1$ other tuples. Consider the following aggregate query:

```
SELECT COUNT(*) FROM DIAGNOSIS di, DEMOGRAPHIC dem
WHERE DIAG="heart_disease", di.id=dem.id
GROUP BY RACE
```

For a distributed query execution plan, an honest but curious observer could monitor the size of the group by bins, and gain information about tuples present in the bins. In particular, if an honest but curious observer monitored network traces, memory access patterns, and execution time, they may be able to trace a *single tuple* from start to finish of a query. If an observer had gained demographic information, an observer could single out individuals with heart disease based on group by bin size.

We now describe the implementation and design of the sort, project, filter, join, and aggregate operators.

Sort. We implement bitonic sort, an oblivious sort algorithm. The anonymized attribute set are the sort keys used. However, we do not implement a specific k-anonymous sort algorithm.

Project. By the subset and transitivity properties listed above, the project operator can implement a standard projection algorithm.

Filter. A filter with predicate f can leak access pattern information based on which tuples match the predicate f. The predicate f is the control flow attribute, and thus is in the anonymized attribute set, $f \in C$. Our filter operator obliviously iterates over all input tuples, and marks tuples matching the filter predicate as "non-dummy", and non-matching tuples as "dummy". With this, the input and output cardinalities are equal, thus leaking no information about the selectivity of the filter predicate. The input relation has at least k tuples in each equivalence class, and thus the output relation also must have at least k tuples in each equivalence class. A filter on attributes k

Join. Only equi-inner joins on a single attribute are supported. A join with predicate j on two relations R_1, R_2, has control flow attribute j on both relations. The join operator takes in anonyimzed equivalence classes partitioned by the anonymized attribute j from

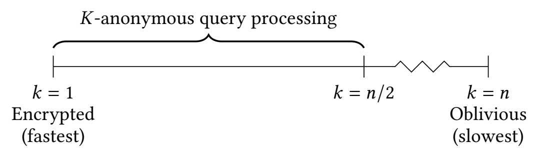


Figure 2: The privacy-performance decision space for use in k-anonymous processing view generation.

each relation, and obliviously joins them together. For attribute j each tuple has the real value of j as well as the anonymized value. If the anonymized values for j from each relation are equal, the join algorithm unconditionally outputs a joined tuple, otherwise it does not output a tuple. The output tuple is obliviously marked as a "non-dummy" tuple if the non-anonymized attribute values are equal, and a "dummy" tuple if they are not. The dummy values remain secret, since the output tuples are encrypted in the trusted backend. The input relations to join are k-anonymous processing views, and for each anonymized attribute, the respective anonymized equivalence classes per relation output the cross product, it follows that

Aggregate. Our system only allows aggregates at the top of the operator tree. An aggregate with group by attributes $\{g_i\}$ and aggregation attribute a proceeds in two steps. Aggregates are first run obliviously per anonymized equivalence class. Next, the aggregate obliviously combines the output of the aggregation on each equivalence class. In order to maintain obliviousness, when the aggregate operator encounters a "dummy tuple", it still unconditionally writes to the aggregate output. This ensures that a curious observer cannot infer which tuples are real or not real by observing memory access patterns and execution time.

4.2 Discussion

Privacy-Performance Trade Off. K-anonymous query processing enables federation members to achieve a profitable trade off between performance and privacy. We visualize this decision space in Figure 2. Consider a medical researcher querying their electronic health records that are stored using encryption in the cloud. She wishes to set her k to a higher value when she is querying highly sensitive data. For example, many states require records pertaining to the treatment of HIV and other sexually transmitted infections have greater k-values than more common diagnoses [28]. When accessing these records, she would use oblivious querying. For more common ailments, she is willing to forgo stronger privacy guarantees in exchange for faster query runtimes.

This decision space, a range of k values for anonymization, arises in many settings. In clinical data research, guidelines for k-anonymization vary. A k from 5-11 is recommended for most health contexts [37, 47, 53], although some data providers suggest $k = 3$ [41] and other, more sensitive studies call for $k = 30$ [9]. For educational data, the US's FERPA has various k-anonymization guidelines for a variety of data release scenarios in [12, 59]. Energy data is also has a range of k values for its release from $k = 5$ [14] to $k = 15$ [19].

By tapping into the expertise of the data federation, we will realize substantial performance gains by adjusting k to the sensitivity workload at hand. In practice, a private data federation may have heterogeneous security policies on client queries to address these domain-specific nuances.

Comparisons to k-anonymity. Our system focus on side channel information means that it does not release the k-anonymous view metadata. This stands in contrast with k-anonymity. K-anonymity's goal is to protect a *data release*. Thus, the vulnerabilities concerning k-anonymity are in the context of an attacker having full access to the k-anonymous private data release. As we will describe in Section 5, anonymization information is kept secret in the secure backends, so neither clients nor data owners *ever* have access to the unencrypted metadata. This is an additional reason why the vulnerabilities of k-anonymity *do not directly apply* to k-anonymous query processing.

We note that if the data owners were to run a k-anonymous data release on their dataset, unmodified database operators run on the k-anonymous release would fulfill Definition 3.2. However, this loses the benefits of our system: 1) our system allows for federation participants to choose their data release mechanism after query execution, k-anonymous data releases must modify the input dataset, thus lowering data utility , 2) high performance; efficient algorithms for k-anonymous data release carry significant overhead [39, 40, 48].

5 SYSTEM ARCHITECTURE

Combining a secure computation backend, our anonymization scheme, and oblivious partitioning we discuss below, we claim that our system fulfills our definition of k-anonymous query processing found in Definition 3.2.

5.1 Federation Setup

KloakDB initializes the private data federation with all data owners agreeing upon a query workload Q, schema $\mathcal{T}$, privacy parameter k, and quasi-identifier set $\mathbb{Q}$. The coordinator analyzes the query workload to determine the set of control flow attributes C, and unions them with the quasi-identifier set $\mathbb{Q}$ to form the anonymization attribute set $\mathcal{P}$. With the anonymization attribute set $\mathcal{P}$, the coordinator iterates over all attributes and obliviously generates k-anonymous processing views. Each data owner ends with their input relations combined with anonymization metadata. An attribute in a tuple contains both the real value as well as the anonymized value. Anonymized values are mappings into anonymized equivalence classes for each attribute in the anonymized attribute set, thus constructing k-anonymous processing views.

Coordinator Requirements. The tasks delegated to the coordinator are 1) k-anonymous view generation, 2) query planning, 3) query result release. The coordinator exclusively runs oblivious algorithms and handles all data within the secure computation backend. We abstract out the notion of the coordinator for ease of of presentation, but note that the coordinator can be chosen amongst the data owners. The coordinator is assumed to have the same level of trust as the data owners. This departs from previous work which required the coordinators to have more trust in order to provide speedups to query execution [66].

Oblivious K-anonymous View Generation. The goal of oblivious k-anonymous view generation is to generate a mapping between each

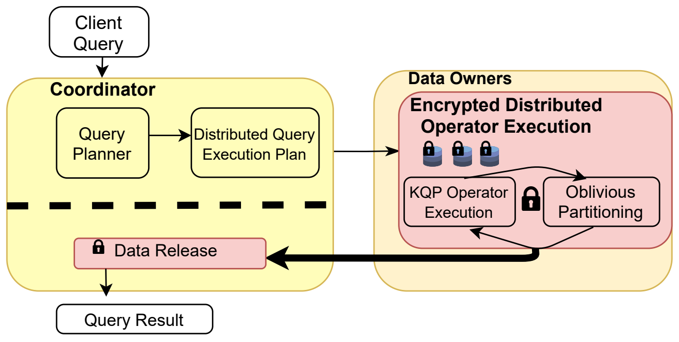


Figure 3: KloakDB query workflow

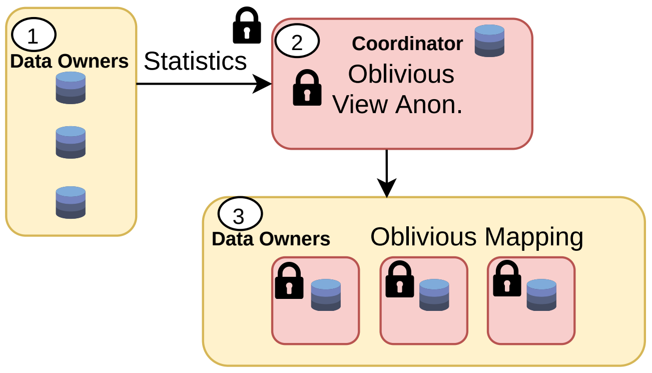


Figure 4: View Anonymization Workflow

attribute in the anonymized attribute set and anonymized *value* for the attribute. This maps each tuple into the appropriate anonymized equivalence class, per attribute. The coordinator iterates over each attribute $p \in \mathcal{P}$. The steps presented below are repeated for each attribute. Figure 4 illustrates the steps of this process.

Step 1: Statistics Collection The coordinator requests encrypted histograms for attribute p from all data owners, for each relevant relation. The data owners encrypt the histograms and send them to the secure computation backend of the coordinator. The coordinator never sees the unencrypted histograms. Our security assumptions ensure that data owners will honestly send their statistics.

Step 2: View Generation The coordinator runs an oblivious view generation algorithm inside secure computation backend, with respect to privacy parameter k. The objective of the algorithm is to uphold Definition 4.3. The algorithm ensures that each tuple is grouped into an anonymized equivalence class with at least k tuples, over every $\binom{n}{n-1}$ combination of data owners. The algorithm iterates over the combined histograms, one attribute value at a time, and obliviously generates a mapping between attribute values and anonymized equivalence classes. The algorithm creates an empty map, that is the size of the attribute domain. To remain oblivious, the algorithm unconditionally writes to each map entry for each attribute. This ensures that an honest-but-curious coordinator cannot determine which anonymized equivalence class an attribute value belongs in. Once the k-anonymized view map is complete, the coordinator sends the encrypted map to the secure computation backend of the other data owners.

Step 3: Mapping Each data owner processes the encrypted mapping between attribute values and anonymized equivalence classes within their respective secure computation backend. The mapping is never revealed to any party. Data owners load the relations into the secure computation backend. For each tuple, the mapping algorithm iterates through the entire mapping list. This ensures that a curious observer cannot gain information about which tuples map to which anonymized equivalence class.

5.2 Query Execution Workflow

Figure 3 shows how a query travels through the system. A client sends the query to the private data federation, where it is received by the coordinator. The coordinator parses the query and constructs a distributed k-anonymous query processing execution plan, annotating anonymized attributes for each operator. This distributed query execution plan is a directed acyclic graph. Per each attribute, anonymized equivalence classes are randomly assigned to data owners for query processing. This mapping is computed in secure computation backend, and is never revealed to any data owner. The coordinator recursively calls each k-anonymous operator node, starting at the leaves. Each operator has a set of anonymized attributes it must honor in order to uphold k-anonymous query processing. Before each operator, the query planner will call an *oblivious partitioning* (Section 5.2) step to ensure that tuples in the anonymized equivalence classes corresponding to the anonymized attributes for the operator are collocated on the same data owner. K-anonymous query processing operators as described in Section 4.1 are executed on each data owners over the anonymized equivalence classes.

Oblivious Partitioning. Naively partitioning tuples across data owners by their anonymized equivalence class could result in sensitive information being exposed. Naive partitioning allows data owner to determine the origin of tuples it recieves for distributed query execution. Combined with knowledge of which tuples it keeps for query execution, a data owner can infer the domain of tuples it is computing on. To mitigate this, our system partitions tuples in two steps. All operations occur within the secure computation backend. In the first step all tuples are randomly assigned, and sent to a data owner. In the second step the tuples are sent to the data owner that has been mapped to the anonymized equivalence class for the current operator's anonymized attribute set. Tuples are sent in batches to each data owner in each step, ensuring that the size of tuple batch is $> k$. Our two step oblivious partitioning decouples the location of anonymized equivalence classes over a sequence of operators - this ensures data owners do not learn of the mapping between attribute values and their respective anonymized equivalence class.

Data Release. At the end of operator execution, the results are sent to a data release engine in the secure computation backend of the coordinator. The coordinator then sends the private release of the data back to the client. Our system architecture allows for the data release mechanism to be modular and easy to modify and replace. By default we use the differential privacy data release mechanism presented in Shrinkwrap [6].

5.3 Utilization of Secure Computation Backend

Secure computation backends in the form of secure enclaves and secure multiparty computation are critical for KloakDB providing it's security guarantees. We consider the life of a tuple to better understand how and where a secure computation backend is used. A tuple starts owned by a single data owner. It is loaded into the secure computation backend when the federation is initialized. After this point, all interactions with the tuple are with the secure computation backend. In the secure computation backend, the anonymization process occurs - this ensures that the anonymization map is never revealed to any data owner or client. Tuples are shipped between data owners across the network during query processing, and are always encrypted with respect to the secure computation backend of the receiving party. As stated above, the entire anonymization process occurs within the secure computation backend. The only time data leaves the secure computation backend is after query release.

6 EXPERIMENTAL RESULTS

We consider a suite of micro-benchmarks with benchmarks on synthetic and real world workloads. Our micro-benchmarks analyze the performance of critical pieces of the architecture, while the workload benchmarks demonstrate the performance of our system in an end-to-end fashion. Our experiments demonstrate KloakDB has scalable and adjustable performance, and that is significantly outperforms the oblivious baseline.

6.1 Implementation

KloakDB is implemented as an in memory distributed query execution engine. We implement KloakDB in 4000 SLOC of C++. Our prototype supports uses Intel SGX as the hardware enclave backend and EMP-Toolkit [68] for secure multiparty computation. The enclave backend supports arbitrary number of data owners, and the MPC backend supports two data owners. Our prototype takes as input a SQL query and runs it through a standard SQL parser. The implementation uses off the shelf RPC libraries with SSL, and encrypts all data processed outside of the trusted backends.

6.2 Experimental Setup:

We run experiments on two testbeds. Our hardware Intel SGX benchmarks are run on 4 Ubuntu 16.04 servers running Intel Core i7 7700k processors, with 32 GB RAM, and a 1 TB 7200 RPM HDD. The MPC experiments are run on two machines from the same test bed. Our benchmarks utilize KloakDB in four modes of query processing: *plain*, *encrypted*, *k-anonymous*, *oblivious*. Encrypted query processing mode does not run the queries obliviously, and does not pad the intermediate result sizes. Oblivious query processing mode runs the queries obliviously and *fully pads* the output sizes. Plain mode runs using PostgreSQL's Foreign Data Wrapper (FDW) [55] to simulate a conventional data federation.

6.3 Workloads

HealthLNK: We test KloakDB over electronic health records from the *HealthLNK* data repository [52]. This clinical data research contains records from seven healthcare sites. The data repository contains about six million electronic health records from a diverse institutions–including research hospitals, clinics, and a county hospital–from 2006 to 2012. This dataset has significant skew, for example, a patient A with a disease X, might have more vital recordings than patient B with disease Y. Running KloakDB on this datatset enables us to stress our model in the presence of significant skew. We map each site in the federation to a machine in our four-node testbed. The size of the HealthLNK dataset on our testbed is approximately 39 GB.

We experiment with queries that are based on real clinical data research protocols for c. diff infections and heart disease [31, 51]. We use public patient registries for common ailments to bound the duration of our experiments. A registry lists the patient identifiers associated with a condition with no additional information about the individual. We maintain a patient registry for heart disease sufferers (*hd_cohort*) and one for individuals affected by c.diff (*cdiff_cohort*), an infection that is frequently antibiotic-resistant. Our queries are shown in Table 1.

TPC-H: TPC-H is a standard synthetic workload and dataset which simulates an OLAP environment [62]. We choose different scale factors depending on the specific experimental setup, using varying scale factors depending on the experiment. We run use queries 3,5, and 10.

6.4 Anonymized View Generation Scalability

We run anonymized view generation on one, two, three, and four relations in secure enclaves with four data owners. We scale the data size with TPC-H scale factors .1, 1, and 10. We use the customers, orders, lineitem, and supplier relations from the TPC-H schema. Orders is anonymized on the "(o_custkey)", lineitem on "(l_suppkey, l_orderkey)", supplier on "(s_suppkey)", and customer on "(c_custkey)". We choose these attributes since they are common join keys for many TPC-H queries. The results are presented in Figure 5.

The anonymization time scales roughly linearly with the data size: with a scale factor of .1 is 5s, with scale factor 1 is 44s , with scale factor 10 is 580s. The anonymization time is not uniform across relations, depending on data size and range. Gathering histograms requires running a $COUNT(*)$ type query on the relation, where the runtime will depend on size and range. For example, processing the lineitem relation takes 60% - 70% of the time of the overall anonymization. Anonymization has substantial network costs since both the histograms have to be gathered at the coordinator, and then the anonymization maps must be distributed to all hosts.

The oblivious algorithm we implement for generating the k-anonymous processing views takes approximately 25% of the time of anonymization. The anonymization time can be amortized over the lifetime of the federation.

6.5 K-anonymous Join Study

In this experiment we run a join on two hosts with TPC-H tables $lineitem$ and $orders$. For MPC we run the experiment at SF .01, for SGX we run it at both SF .01 and 1. We anonymize the input relations with respect to the join key, *orderkey*. As the k-anonymous parameter increases, join execution time increases proportional to the k parameter. For a single join, we achieve nearly linear performance degradation as a parameter of k. We realize this very efficient result with the following intuition: given n input tuples in

Name	Query
aspirin profile	`SELECT gender, race, avg(pulse) FROM demographics de, diagnosis di, vitals v, medications m WHERE m.med = 'aspirin' ∧ di.diag = 'hd' ∧ dd.pid = di.pid ∧ di.pid = v.pid ∧ m.pid = di.pid;`
comorbidity	`SELECT diag, COUNT(*) cnt FROM diagnoses WHERE pid ∈ cdiff_cohort ∧ diag <> 'cdiff' ORDER BY cnt DESC LIMIT 10;`
dosage study	`SELECT pid FROM diagnoses d, medications m WHERE d.pid = m.pid AND medication = 'aspirin' AND icd9 = 'internal bleeding' AND dosage = '325mg'`

Table 1: HealthLNK query workload.

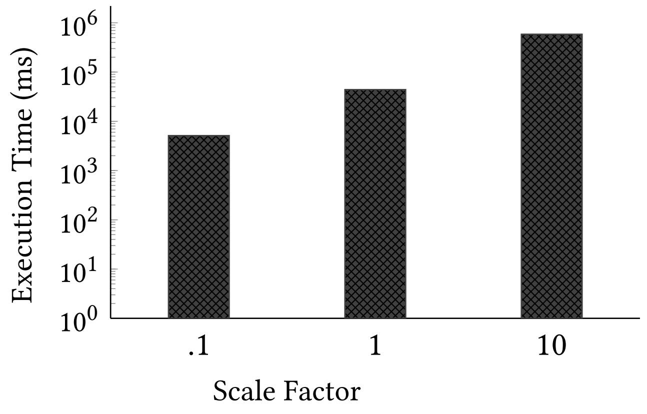


Figure 5: TPC-H Anonymization(Enclave)
Tables: *lineitem, orders, customer, supplier*

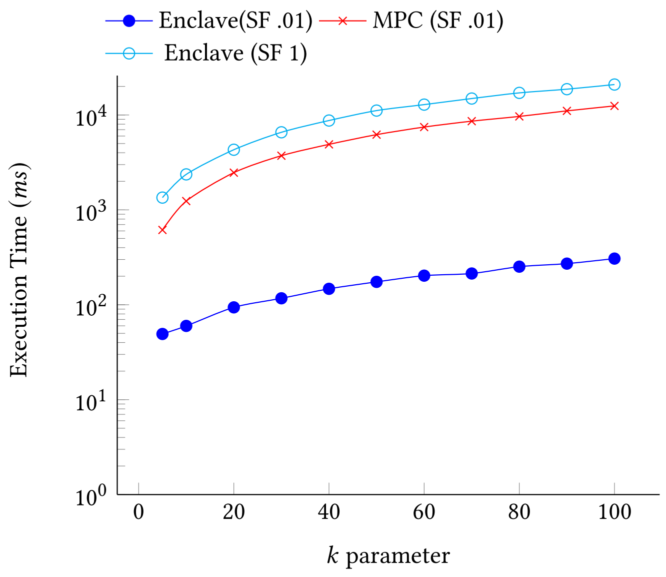


Figure 6: TPC-H Join (MPC, Enclave)
Tables: *lineitem, orders*

each relation, and an anonymization parameter of k, each matched equivalence class produces O(k^2) tuples. The anonymized view generator produces approximately n/k equivalence classes per relation. Hence, the output size of a k-anonymous join is $O(nk)$. The join's execution time is proportional to the size of its inputs, therefore as k increases, we see a commensurate linear increase in execution time. In MPC, as we increase k, performance increases linearly. With $k = 5$, the MPC join takes 614ms, $k = 100$, 12457ms - a 20X performance penalty. The SGX backend also has performance linear with k. At SF .01, with $k = 5$, the SGX join takes 49ms, $k = 100$, 306ms - a 6.2X performance penalty. At SF 1, the performance is more linear: with $k = 5$, the join takes 1347ms, $k = 100$ 20928ms - a 15x performance penalty.

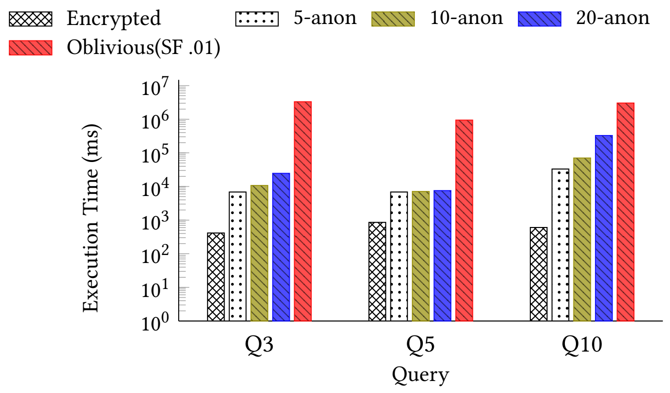


Figure 7: TPC-H Workload. Encrypted, k-anon mode run with SF 1. Oblivious run with SF .01

6.6 TPC-H Query Workload

We run KloakDB on three TPC-H workload queries 3,5,10 in SGX with four data owners. We choose these queries because of their complex aggregates combined with multiple joins. We run with scale factors 1 for encrypted mode, and $k = 5, 10, 20$. Running oblivious query processing with TPC-H SF 1 exceeded our deadline of 4000 seconds. We use scale factor .01 in order to allow the oblivious baseline to complete. Even with a substantially larger datasize, k-anonymous query processing significantly outperforms oblivious computation. This experiment demonstrates two properties of KloakDB: 1) Tunable performance with k, 2) performance over oblivious baseline. As we increase the k from 5-20, the performance scales roughly linearly with the security parameter.

6.7 HealthLNK Query Workload

We run KloakDB on our real-world workload in SGX with four data owners, omitting the anonymization setup time in the presentation. For the four relations used in our queries, the anonymization time for a year of data took a little over two seconds. Our experiments validate the viability of KQP on a dataset with significant skew.
Aspirin Profile Operator Performance We analyze the per operator overhead of KloakDB with the *aspirin profile* query in Figure 8. We measure this query's runtime in encrypted, k-anonymous ($k = 5$), and oblivious mode. We randomly select 25 patients from the *HealthLNK* dataset from one year of data.

Figure 8 presents the operator runtime in each execution mode. The sequence of three joins is where KQP assumes a substantial performance gains over oblivious query processing. In oblivious mode first join emits n^2 tuples, the second produces n^3, and so on. In contrast, the expected cardinality of the first KQP join output is $O(nk)$ tuples, the second join $O(nk^2)$ and so on. The third join takes approximately 6 ms in encrypted mode, 650 ms in k-anonymous, and 93000 ms for oblivious processing. This is a 103x slowdown between k-anonymous and encrypted, and a 143x slowdown between oblivious and k-anonymous execution. The aggregate in encrypted mode takes approximately 5ms, in k-anonymous 6700ms, oblivious 27900ms. The performance gap between k-anonymous mode and encrypted mode is due an unoptimized implementation, however the k-anonymous aggregate is 5x faster than oblivious execution.

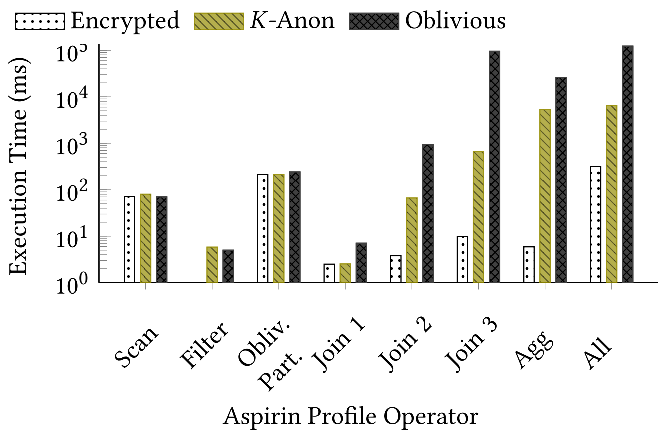


Figure 8: Aspirin Profile, Operator Perf. (Enclave)

The overall runtime for encrypted execution, k-anonymous, and oblivious is 320 ms, 6600ms, 123,000 ms respectively. The slowdown incurred by k-anonymous execution compared to encrypted execution is 21X, and the speedup of k-anonymous execution in comparison to oblivious execution is 18X. Due to the prohibitively expensive overhead of oblivious execution, we sampled only 25 patients for *aspirin profile*. As the data size increases, we expect the gap between oblivious and k-anonymous execution to widen.

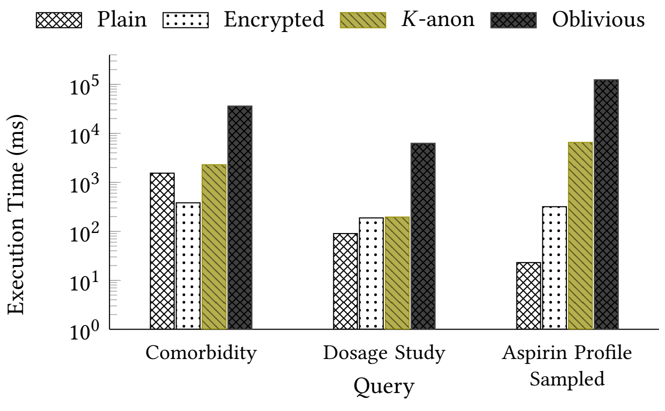


Figure 9: HealthLNK Query Workload (Enclave)

Full Distributed Workload In this section we run the full query workload in Table 1. We demonstrate that k-anonymous query processing provides substantial performance improvements over oblivious query processing while providing data protection in comparison to encrypted execution. We run the queries in four modes: *plain*, encrypted , k-anonymous, and oblivious. For the *comorbidity* and *dosage study* queries we run the queries on a full year of data in all four modes. However, the *aspirin profile* queries was unable to complete in oblivious mode on a full year of data, therefore we sample 25 unique *patients* per host. Figure 9 has the results of the full workload. The *comorbidity* query demonstrates that even with a simple query, k-anonymous query processing is an attractive alternative to oblivious query processing. K-anonymous execution has a 15X speedup compared to oblivious execution, and 6X slowdown compared to encrypted. The *dosage study* query sees a 31X speedup in k-anonymous execution compared to oblivious execution, and a 1.03X slowdown in k-anonymous execution compared to encrypted. We detailed the performance for *aspirin profile* in this query in Section 6.7.

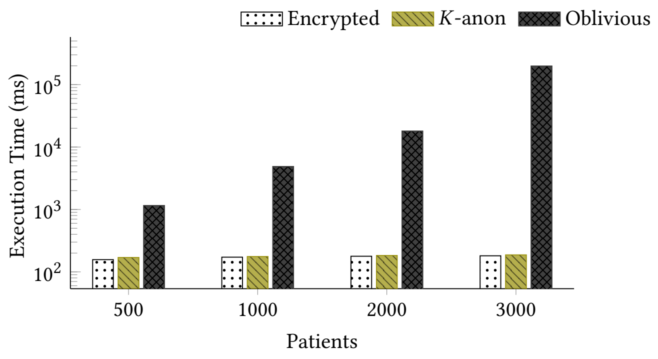


Figure 10: Dosage Study Scale (Enclave)

Dosage Study Scale-up In this section we verify that as the input tuple size increases, the gap between k-anonymous execution and oblivious execution widens. We use *dosage study* so that oblivious execution may complete. We vary the number of patients we sample to measure performance changes for data of increasing size in encrypted, 5-anonymous, and oblivious mode. Figure 10 shows the runtime of this query. K-anonymous execution is slightly slower than oblivious execution with 500 sampled patients owing to the overhead of its$k$-anonymous processing view setup. As the input size increases, k-anonymous query processing offers significant performance benefits over full-oblivious query processing. With 3,000 patients, the runtime for encrypted, k-anonymous, and full-oblivious query processing respectively are approximately 181ms, 187ms, and 198369. This yields 1.03X slowdown for k-anonymous mode compared to encrypted, and 1060X speedup for it in comparison to oblivious. The stark slowdown for oblivious mode is due to the substantial memory pressure imposed by exploding cardinalities, leading the join output to spill to disk one equivalence class at a time. In k-anonymous mode, scaling the input size from 500 to 3000 patients yields a 1.1X slowdown. This stands in contrast with the 171X slowdown we observe in oblivious mode. This experiment highlights an important feature of this system: KloakDB enables substantial speedups for query processing as input data scales.

7 RELATED WORK

KloakDB builds on principles in secure query processing, oblivious computation, and encrypted computation. We survey the existing research in these areas.

KloakDB builds on principles in query processing, applied security, and automated access control policies. There is substantial active research in all of these areas and we survey them in this section.

Speaking broadly, there are two common methods for methods for general-purpose computing over the data of two or more mutually distrustful parties: in software with secure multi-party computation [27, 72] and in hardware using hardware enclaves [10, 35]. The former is possible on any system, but exacts a substantial overhead in making the computation oblivious and encrypting its contents. The latter requires specialized hardware, but is more efficient. We chose hardware enclaves for this work, and the principles of k-anonymous query processing readily generalize to secure multi-party computation.

There has been substantial work on oblivious query processing using hardware enclaves [3, 4, 22, 50, 56, 74]. In this setting a curious observer of an enclave learns nothing about the data upon which they compute by observing its instruction traces. We build on this work by offering semi-oblivious query processing for querying data of moderate sensitivity.

KloakDB is a private data federation. This challenge was researched with the use of secure multi-party computation to combine the private data of multiple parties in [5, 8, 13, 66, 70]. We extend this work, but examine how to do it semi-obliviously in exchange for faster query runtimes. Shrinkwrap [6] considers a similar semi-oblivious model through reducing the output of joins with differential privacy. Our work differs in that Shrinkwrap still requires executing the full cross product for joins and only after runs the Shrinkwrap protocol. Oblivious Coopetitive Analytics [13] provides a framework to reduce the burden of oblivious computation through utilizing publically available constraints. However, the system remains oblivious with respect to the public information. Conclave [66] focuses on query rewriting to avoid computatation in MPC, our work can be used in conjunction with the methods of Conclave. Similary, ObliDB's [22] methods of accelerated oblivious query operators using SGX can be used with our work to provide faster oblivious processing over equivalence classes.

K-anonymous data releases were proposed in [57]. There has been substantial work on efficiently generating k-anonymous views of a given dataset [7, 17, 34, 39, 40, 61]. KloakDB extends the techniques in [15] to build k-anonymous processing views. Automatically enforcing k-anonymous access control policies in a dataset was researched in [20].

Most of the prior work on oblivious query processing focuses on outsourced computation from a single data provider, either in software with secure multi-party computation [2] or in hardware with hardware enclaves [3, 74]. Some of them [5, 8, 66, 70] offer interoperability for multiple data owners.

There is also work about computing queries in the cloud over data stored with fully homomorphic encryption [54, 63]. Encrypted databases have reduced expressiveness since they cannot readily compose operators for nested blocks of select statements.

8 CONCLUSIONS

We presented a semi-oblivious query processing model, k-anonymous query processing for private data federations. With KQP, data owners have a fine grained knob with which to trade off privacy and performance. Our formalization of k-anonymous query processing allows for complex queries, while protecting against unauthorized privacy leakage. This is an important step towards more approaches that strike a balance between security and performance for querying private data. Our model is grounded in k-anonymity, a relevant and widely deployed privacy model. We utilize a privacy model which is the cornerstone of regulatory requirements in a few industries, which should allow our work to have immediate impact. We built and tested a prototype KloakDB, which implements k-anonymous query processing using either secure multiparty computation or hardware enclaves.

Our evaluation shows KQP provides fine-grained tunablility for increasing privacy. Our join study demonstrates a linear tradeoff between privacy and performance. On a real-world dataset with a real-world workload we demonstrate speedups of 15X-1060X. Our results show that if KQP is the appropriate query processing model for a data federation, there is significant room for performance and scalability gains.

REFERENCES

[1] Charu C Aggarwal. 2005. On k-anonymity and the curse of dimensionality. In *VLDB*. VLDB Endowment, 901–909.

[2] Gagan Aggarwal, Mayank Bawa, Prasanna Ganesan, Hector Garcia-Molina, Krishnaram Kenthapadi, Rajeev Motwani, Utkarsh Srivastava, Dilys Thomas, and Ying Xu. 2005. Two can keep a secret: A distributed architecture for secure database services. *CIDR* (2005).

[3] Arvind Arasu and Raghav Kaushik. 2014. Oblivious query processing. *ICDT* (2014).

[4] Sumeet Bajaj and Radu Sion. 2013. TrustedDB : A Trusted Hardware based Database with Privacy and Data Confidentiality. *TKDE* 26, 3 (2013), 1–14.

[5] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. 2016. SMCQL: Secure Querying for Federated Databases. *VLDB* 10, 6 (2016), 673–684. https://doi.org/10.14778/3055330.3055334

[6] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2019. Shrinkwrap: Differentially-Private Query Processing in Private Data Federations. *VLDB* 12, 3 (2019), 307–320.

[7] Roberto J Bayardo and Rakesh Agrawal. 2005. Data privacy through optimal k-anonymization. In *ICDE*. IEEE, 217–228.

[8] Dan Bogdanov, Sven Laur, and Jan Willemson. 2008. Sharemind: A framework for fast privacy-preserving computations. In *EuroSec*. Springer, 192–206.

[9] CDC Youth Risk Behavior Surveillance System. 2017. 2015 YRBS National, State, and District Combined Datasets User's Guide. (2017). https://www.cdc.gov/healthyyouth/data/yrbs/pdf/2015/2015_yrbs_sadc_documentation.pdf

[10] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint* 2016 (2016), 86. https://eprint.iacr.org/2016/086.pdf

[11] Victor Costan, Ilia A Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation.. In *USENIX Security*. 857–874.

[12] Jon P Daries, Justin Reich, Jim Waldo, Elise M Young, Jonathan Whittinghill, Daniel Thomas Seaton, Andrew Dean Ho, and Isaac Chuang. 2014. Privacy, anonymity, and big data in the social sciences. *ACM Queue* 12, 7 (2014), 30.

[13] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. 2020. Oblivious Coopetitive Analytics Using Hardware Enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece) *(EuroSys '20)*. Association for Computing Machinery, New York, NY, USA, Article 39, 17 pages. https://doi.org/10.1145/3342195.3387552

[14] DC Department of Energy & Environment. 2018. Energy Benchmarking Data Collection. https://doee.dc.gov/page/energy-benchmarking-data-collection

[15] Katerina Doka, Mingqiang Xue, Dimitrios Tsoumakos, and Panagiotis Karras. 2015. k-Anonymization by freeform generalization. In *CCS*. ACM, 519–530.

[16] Khaled El Emam. 2011. Methods for the de-identification of electronic health records for genomic research. *Genome Medicine* 3, 4 (2011), 25.

[17] Khaled El Emam, Fida Kamal Dankar, Romeo Issa, Elizabeth Jonker, Daniel Amyot, Elise Cogo, Jean-Pierre Corriveau, Mark Walker, Sadrul Chowdhury, Regis Vaillancourt, Tyson Roffey, and Jim Bottomley. 2009. A globally optimal k-anonymity method for the de-identification of health data. *JAMIA* 16, 5 (2009), 670–682.

[18] Khaled El Emam, Elizabeth Jonker, Luk Arbuckle, and Bradley Malin. 2011. A systematic review of re-identification attacks on health data. *PloS one* 6, 12 (2011), e28071.

[19] Elevate Energy. 2014. Aggregated Data Access: The 15/15 Rule in Illinois and Beyond. http://www.elevateenergy.org/wp/wp-content/uploads/1515-Rule-Factsheet-FINAL.pdf

[20] Mohamed Y Eltabakh, Jalaja Padma, Yasin N Silva, Pei He, Walid G Aref, and Elisa Bertino. 2012. Query processing with K-anonymity. *International Journal of Data Engineering* 3, 2 (2012), 48–65.

[21] Khaled El Emam. 2008. Heuristics for De-identifying Health Data. *IEEE Security & Privacy* 6, 4 (2008), 58–61. https://doi.org/10.1109/MSP.2008.84

[22] Saba Eskandarian and Matei Zaharia. 2017. An Oblivious General-Purpose SQL Database for the Cloud. *arXiv preprint 1710.00458* (2017).

[23] Family Education Rights and Privacy Act. 1974. USC 1232-34 CFR Part 99.

[24] Federal Committee on Statistical Methodology. 2005. Report on Statistical Disclosure Limitation Methodology. https://www.hhs.gov/sites/default/files/spwp22.pdf

[25] Dan Frankowski, Dan Cosley, Shilad Sen, Loren Terveen, and John Riedl. 2006. You are what you say: privacy risks of public mentions. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 565–572.

[26] Simson L Garfinkel. 2015. *De-Identification of Personal Information*. Technical Report. National Institute of Standards and Technology.

[27] Oded Goldreich, Sylvio Micali, and Avi Wigderson. 1987. How to Play Any Mental Game. *STOC* (1987), 218–229. https://doi.org/10.1145/28395.28420

[28] Lawrence O Gostin, Zita Lazzarini, Verla S Neslund, and Michael T Osterholm. 1996. The public health information infrastructure: a national review of the law on health information privacy. *JAMIA* 275, 24 (1996), 1921–1927.

[29] Paul Grubbs, Thomas Ristenpart, and Vitaly Shmatikov. 2017. Why your encrypted database is not secure. In *Proceedings of the 16th workshop on hot topics in operating systems*. 162–168.

[30] Paul Grubbs, Kevin Sekniqi, Vincent Bindschaedler, Muhammad Naveed, and Thomas Ristenpart. 2017. Leakage-abuse attacks against order-revealing encryption. In *2017 IEEE Symposium on Security and Privacy (SP)*. IEEE, 655–672.

[31] Adrian F Hernandez, Rachael L Fleurence, and Russell L Rothman. 2015. The ADAPTABLE Trial and PCORnet: shining light on a new research paradigm. *Annals of internal medicine* 163, 8 (2015), 635–636.

[32] Mike Hintze. 2017. Viewing the GDPR through a De-Identification Lens: A Tool for Compliance, Clarification, and Consistency. (2017).

[33] Mike Hintze and Khaled El Emam. 2018. Comparing the benefits of pseudonymisation and anonymisation under the GDPR. *Journal of Data Protection & Privacy* 2, 2 (2018), 145–158.

[34] Wei Jiang and Chris Clifton. 2006. A secure distributed framework for achieving k-anonymity. *VLDB Journal* 15, 4 (2006), 316–333. https://doi.org/10.1007/s00778-006-0008-z

[35] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD Memory Encryption. *White paper, Apr* (2016).

[36] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2016. Generic attacks on secure outsourced databases. In *CCS*. ACM, 1329–1340.

[37] Richard J Klein, Suzanne E Proctor, Manon A Boudreault, and Kathleen M Turczyn. 2002. Healthy People 2010 criteria for data suppression. *Statistical notes* 24 (2002).

[38] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In *USENIX Security*. 16–18.

[39] Kristen Lefevre, David J DeWitt, and Raghu Ramakrishnan. 2005. Incognito: Efficient full-domain k-anonymity. In *SIGMOD*. ACM, 49–60. https://tthdhmdoanmonhoc.googlecode.com/svn/trunk/Research/k-anonimity/old/incognito.pdf

[40] Kristen LeFevre, David J DeWitt, and Raghu Ramakrishnan. 2006. Mondrian multidimensional k-anonymity. In *ICDE*. IEEE, 25.

[41] Bernard Lo. 2015. Sharing clinical trial data: maximizing benefits, minimizing risk. *JAMA* 313, 8 (2015), 793–794.

[42] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. 2007. L-Diversity: Privacy beyond k-Anonymity. *ACM Trans. Knowl. Discov. Data* 1, 1 (March 2007), 3–es. https://doi.org/10.1145/1217299.1217302

[43] Bradley Malin and Latanya Sweeney. 2004. How (not) to protect genomic data privacy in a distributed network: using trail re-identification to evaluate and design anonymity protection systems. *Journal of biomedical informatics* 37, 3 (2004), 179–192.

[44] David J Martin, Daniel Kifer, Ashwin Machanavajjhala, Johannes Gehrke, and Joseph Y Halpern. 2007. Worst-case background knowledge for privacy-preserving data publishing. In *ICDE*. IEEE, 126–135.

[45] Frank D McSherry. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*. ACM, 19–30.

[46] S. Mehrotra, S. Sharma, J. Ullman, and A. Mishra. 2019. Partitioned Data Security on Outsourced Sensitive and Non-Sensitive Data. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*. 650–661. https://doi.org/10.1109/ICDE.2019.00064

[47] National Center for Health Statistics. 2012. *Disclosure Manual: Preventing Disclosure: Rules for Researchers*. Technical Report. Center for Disease Control. https://www.cdc.gov/rdc/Data/B4/DisclosureManual.pdf

[48] Mehmet Ercan Nergiz, Christopher Clifton, and Ahmet Erhan Nergiz. 2009. Multirelational k-anonymity. *IEEE Transactions on Knowledge and Data Engineering* 21, 8 (2009), 1104–1117.

[49] Office for Civil Rights, Health and Human Services. 2002. Standards for privacy of individually identifiable health information. *Federal Register* 67, 157 (2002), 53181.

[50] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious Multi-party Machine Learning on Trusted Processors. In *Proceedings of the 25th USENIX Conference on Security Symposium* (Austin, TX, USA) *(SEC'16)*. USENIX Association, Berkeley, CA, USA, 619–636. http://dl.acm.org/citation.cfm?id=3241094.3241143

[51] PCORI. 2015. Characterizing the Effects of Recurrent Clostridium Difficile Infection on Patients. *IRB Protocol* ORA: 14122 (2015).

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

[53] PCORnet Council. 2018. Governance Policies for PCORnet, the National Patient-Centered Clinical Research Network. Version 4 (2018). https://pcornet.org/wp-content/uploads/2018/02/PCORnet-Governance-Policy-v4_approved_31Jan2018.pdf

[54] Ra Popa and C Redfield. 2011. CryptDB: protecting confidentiality with encrypted query processing. *SOSP* (2011), 85–100. https://doi.org/10.1145/2043556.2043566

[55] PostgreSQL. [n.d.]. Foreign Data Wrappers. https://wiki.postgresql.org/wiki/Foreign_data_wrappers

[56] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A Secure Database using SGX. In *EnclaveDB: A Secure Database using SGX*. IEEE, 0.

[57] Pierangela Samarati and Latanya Sweeney. 1998. *Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression*. Technical Report. SRI International.

[58] Sajin Sasy, Sergey Gorbunov, and Christopher Fletcher. 2017. ZeroTrace: Oblivious memory primitives from Intel SGX. *IACR Cryptology 'Archive Report* 549 (2017), 2017.

[59] Marilyn Seastrom. 2017. Best Practices for Determining Subgroup Size in Accountability Systems While Protecting Personally Identifiable Student Information. *Institute of Education Sciences* IES 2017, 147 (2017).

[60] Ming-Wei Shih, Sangho Lee, Taesoo Kim, and Marcus Peinado. 2017. T-SGX: Eradicating controlled-channel attacks against enclave programs. In *NDSS*.

[61] Latanya Sweeney. 2002. k-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10, 05 (2002), 557–570.

[62] The Transaction Processing Council. [n.d.]. TPC-H Benchmark (Revision 2.18.0). http://www.tpc.org/tpc_documents_current_versions/pdf/tpc-h_v2.18.0.pdf

[63] Stephen Tu, M Frans Kaashoek, Samuel Madden, and Nickolai Zeldovich. 2013. Processing Analytical Queries over Encrypted Data. *Proc. VLDB Endow.* 6, 5 (3 2013), 289–300. https://doi.org/10.14778/2535573.2488336

[64] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *USENIX Security*.

[65] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2019. Breaking Virtual Memory Protection and the SGX Ecosystem with Foreshadow. *IEEE Micro* 39, 3 (2019), 66–74.

[66] Nikolaj Volgushev, Malte Schwarzkopf, Ben Getchell, Andrei Lapets, Mayank Varia, and Azer Bestavros. 2018. Conclave Worflow Manager for MPC. https://github.com/multiparty/conclave

[67] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. *CCS* (2017), 2421–2434.

[68] Xiao Wang, Alex J Malozemoff, and Jonathan Katz. 2016. EMP-Toolkit: Efficient Multiparty Computation Toolkit. https://github.com/emp-toolkit.

[69] Xiao Wang, Samuel Ranellucci, and Jonathan Katz. 2017. Global-Scale Secure Multiparty Computation. Cryptology ePrint Archive, Report 2017/189. https:

//eprint.iacr.org/2017/189.

[70] Wai Kit Wong, Ben Kao, David Wai Lok Cheung, Rongbin Li, and Siu Ming Yiu. 2014. Secure query processing with data interoperability in a cloud database environment. In *SIGMOD*. ACM, 1395–1406.

[71] Y. Xu, W. Cui, and M. Peinado. 2015. Controlled-Channel Attacks: Deterministic Side Channels for Untrusted Operating Systems. In *2015 IEEE Symposium on Security and Privacy*. 640–656. https://doi.org/10.1109/SP.2015.45

[72] Andrew C Yao. 1982. Protocols for secure computations. In *FOCS*. IEEE, 160–164.

[73] Laura Zayatz. 2007. Disclosure avoidance practices and research at the US Census Bureau: An update. *Journal of Official Statistics* 23, 2 (2007), 253.

[74] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform.. In *NSDI*. 283–298.