$\mathcal{E}$psolute: Efficiently Querying Databases While Providing Differential Privacy

Dmytro Bogatov
Boston University
Boston, MA, USA
dmytro@bu.edu

Georgios Kellaris
Canada
kellaris@bu.edu

George Kollios
Boston University
Boston, MA, USA
gkollios@cs.bu.edu

Kobbi Nissim
Georgetown University
Washington, D.C., USA
kobbi.nissim@georgetown.edu

Adam O'Neill
University of Massachusetts, Amherst
Amherst, MA, USA
adamo@cs.umass.edu

ABSTRACT

As organizations struggle with processing vast amounts of information, outsourcing sensitive data to third parties becomes a necessity. To protect the data, various cryptographic techniques are used in outsourced database systems to ensure data privacy, while allowing efficient querying. A rich collection of attacks on such systems has emerged. Even with strong cryptography, just communication volume or access pattern is enough for an adversary to succeed.

In this work we present a model for *differentially private outsourced database system* and a concrete construction, $\mathcal{E}$psolute, that provably conceals the aforementioned leakages, while remaining efficient and scalable. In our solution, differential privacy is preserved at the record level even against an untrusted server that controls data and queries. $\mathcal{E}$psolute combines Oblivious RAM and differentially private sanitizers to create a generic and efficient construction.

We go further and present a set of improvements to bring the solution to efficiency and practicality necessary for real-world adoption. We describe the way to parallelize the operations, minimize the amount of noise, and reduce the number of network requests, while preserving the privacy guarantees. We have run an extensive set of experiments, dozens of servers processing up to 10 million records, and compiled a detailed result analysis proving the efficiency and scalability of our solution. While providing strong security and privacy guarantees we are less than an order of magnitude slower than range query execution of a non-secure plain-text optimized RDBMS like MySQL and PostgreSQL.

CCS CONCEPTS

• **Security and privacy → Database and storage security**; **Management and querying of encrypted data**.

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484786

KEYWORDS

Differential Privacy; ORAM; differential obliviousness; sanitizers;

ACM Reference Format:
Dmytro Bogatov, Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'Neill. 2021. $\mathcal{E}$psolute: Efficiently Querying Databases While Providing Differential Privacy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3460120.3484786

1 INTRODUCTION

Secure outsourced database systems aim at helping organizations outsource their data to untrusted third parties, without compromising data confidentiality or query efficiency. The main idea is to encrypt the data records before uploading them to an untrusted server along with an index data structure that governs which encrypted records to retrieve for each query. While strong cryptographic tools can be used for this task, existing implementations such as CryptDB [56], Cipherbase [2], StealthDB [70] and TrustedDB [3] try to optimize performance but do not provide strong security guarantees when answering queries. Indeed, a series of works [9, 17, 34, 37, 40, 41, 43, 45, 51] demonstrate that these systems are vulnerable to a variety of reconstruction attacks. That is, an adversary can fully reconstruct the distribution of the records over the domain of the indexed attribute. This weakness is prominently due to the *access pattern leakage*: the adversary can tell if the same encrypted record is returned on different queries.

More recently, [33, 35, 43–45] showed that reconstruction attacks are possible even if the systems employ heavyweight cryptographic techniques that hide the access patterns, such as homomorphic encryption [30, 69] or Oblivious RAM (ORAM) [31, 32], because they leak the size of the result set of a query to the server (this is referred to as *communication volume leakage*). Thus, even some recent systems that provide stronger security guarantees like ObliDB [28], Opaque [75] and Oblix [50] are susceptible to these attacks. This also means that no outsourced database system can be both optimally efficient and privacy-preserving: secure outsourced database systems should not return the exact number of records required to answer a query.

We take the next step towards designing secure outsourced database systems by presenting novel constructions that strike a provable balance between efficiency and privacy. First, to combat the

access pattern leakage, we integrate a layer of ORAM storage in our construction. Then, we bound the communication volume leakage by utilizing the notion of differential privacy (DP) [24]. Specifically, instead of returning the exact number of records per query, we only reveal perturbed query answer sizes by adding random encrypted records to the result so that the communication volume leakage is bounded. Our construction guarantees privacy of any single record in the database which is necessary in datasets with stringent privacy requirements. In a medical HIPAA-compliant setting, for example, disclosing that a patient exists in a database with a rare diagnosis correlating with age may be enough to reveal a particular individual.

The resulting mechanism achieves the required level of privacy, but implemented naïvely the construction is prohibitively slow. We make the solution practical by limiting the amount of noise and the number of network roundtrips while preserving the privacy guarantees. We go further and present a way to parallelize the construction, which requires adapting noise-generation algorithms to maintain differential privacy requirements.

Using our system, we have run an extensive set of experiments over cloud machines, utilizing large datasets — that range up to 10 million records — and queries of different sizes, and we report our experimental results on efficiency and scalability. We compare against best possible solutions in terms of efficiency (conventional non-secure outsourced database systems on unencrypted data) and against an approach that provides optimal security (retrieves the full table from the cloud or runs the entire query obliviously with maximal padding). We report that our solution is very competitive against both baselines. Our performance is comparable to that of unsecured plain-text optimized database systems (like MySQL and PostgreSQL): while providing strong security and privacy guarantees, we are only 4 to 8 times slower in a typical setting. Compared with the optimally secure solution, a linear scan (downloading all the records), we are 18 times faster in a typical setting and even faster as database sizes scale up.

To summarize, our contributions in this work are as follows:

- We present a new model for a *differentially private* outsourced database system, CDP-ODB, its security definition, query types, and efficiency measures. In our model, the adversarial honest-but-curious server cannot see the record values, access patterns, or exact communication volume.
- We describe a novel construction, $\mathcal{E}$psolute, that satisfies the proposed security definition, and provide detailed algorithms for both range and point query types. In particular, to conceal the access pattern and communication volume leakages, we provide a secure storage construction, utilizing a combination of Oblivious RAM [31, 32] and differentially private sanitization [10]. Towards this, we maintain an index structure to know how many and which objects we need to retrieve. This index can be stored locally for better efficiency (in all our experiments this is the case), but crucially, it can also be outsourced to the adversarial server and retrieved on-the-fly for each query.
- We improve our generic construction to enable parallelization within a query. The core idea is to split the storage among multiple ORAMs, but this requires tailoring the overhead required for differential privacy proportionally to the number of ORAMs, in order to ensure privacy. We present practical improvements and optimization techniques that dramatically reduce the amount of fetched noise and the number of network roundtrips.
- Finally, we provide and open-source a high-quality C++ implementation of our system. We have run an extensive set of experiments on both synthetic and real datasets to empirically assess the efficiency of our construction and the impact of our improvements. We compare our solutions to the naïve approach (linear scan downloading all data every query), oblivious processing and maximal padding solution (Shrinkwrap [5]), and to a non-secure regular RDBMS (PostgreSQL and MySQL), and we show that our system is very competitive.

1.1 Related Work

We group the related secure databases, engines, and indices into three categories (i) systems that are oblivious or volume-hiding and do not require trusted execution environment (TEE), (ii) constructions that rely on TEE (usually, Intel SGX), (iii) solutions that use property-preserving or semantically secure encryption and target primarily a snapshot adversary. *We claim that $\mathcal{E}$psolute is the most secure and practical range- and point-query engine in the outsourced database model, that protects both access pattern (AP) and communication volume (CV) using Differential Privacy, while not relying on TEE, linear scan or padding result size to the maximum.*

Obliviousness and volume-hiding without enclave. This category is the most relevant to $\mathcal{E}$psolute, wherein the systems provide either or both AP and CV protection without relying on TEE. Cryptϵ [59] is a recent end-to-end system executing "DP programs". Cryptϵ has a different model than $\mathcal{E}$psolute in that it assumes two non-colluding servers, an adversarial querying user (the analyst), and it uses DP to protect the privacy of an individual in the database, which includes volume-hiding for aggregate queries. Cryptϵ also does not consider oblivious execution and attacks against the AP. Shrinkwrap [5] (and its predecessor SMCQL [4]) is an excellent system designed for complex queries over federated and distributed data sources. In Shrinkwrap, AP protection is achieved by using oblivious operators (linear scan and sort) and CV is concealed by adding fake records to intermediate results with DP. Padding the result to the maximum size first and doing a linear scan over it afterwards to "shrink" it using DP, is much more expensive than in $\mathcal{E}$psolute, however. In addition, in processing a query, the worker nodes are performing an $O(n \log n)$ cost oblivious sorting, where n is the maximum result size (whole table for range query), since they are designed to answer more general complex queries. SEAL [21] offers adjustable AP and CV leakages, up to specific bits of leakage. SEAL builds on top of Logarithmic-SRC [22], splits storage into multiple ORAMs to adjust AP, and pads results size to a power of 2 to adjust CV. $\mathcal{E}$psolute, on the other hand, fully hides the AP and uses DP with its guarantees to pad the result size. PINED-RQ [60] samples Laplacian noise right in the B+ tree index tree, adding fake and removing real pointers according to the sample. Unlike $\mathcal{E}$psolute, PINED-RQ allows false negatives (i.e., result records not

included in the answer), and does not protect against AP leakage. On the theoretical side, Chan et al. [18] (followed by Beimel et al. [8]) treat the AP itself as something to protect with DP. [18] introduces a notion of differential obliviousness that is admittedly weaker than the full obliviousness used in $\mathcal{E}$psolute. Most importantly, [18] ensures differential privacy w.r.t. the ORAM only, while $\mathcal{E}$psolute ensures DP w.r.t. the entire view of the adversary.

Enclave-based solutions. Works in this category use trusted execution environment (usually, SGX enclave). These works are primarily concerned with the AP protection for both trusted and untrusted memory, unlike $\mathcal{E}$psolute which also protects CV. Cipherbase [1, 2] was a pioneer introducing the idea of using TEE (FPGA at that time) to assist with DBMS security. HardIDX [29] simply puts the B+ tree in the enclave, while StealthDB [70] symmetrically encrypts all records and brings them in the enclave one at a time for processing. EnclaveDB [57] assumes somewhat unrealistic 192 GB enclave and puts the entire database in it. ObliDB [28] and Opaque [75] assume fully oblivious enclave memory (not available as of today) and devise algorithms that use this fully trusted portion to obliviously execute common DBMS operators, like filters and joins. Oblix [50] provides a multimap that is oblivious both in and out of the enclave. HybrIDX claims protection against both AP and CV leakages, but unlike $\mathcal{E}$psolute it only obfuscates them. $\mathcal{E}$psolute offers an indistinguishability guarantee for AP and a DP guarantee for CV, while HybrIDX hides the exact result size and only obfuscates the AP. Lastly, Hermetic [74] takes on the SGX side-channel attacks, including AP. It provides oblivious primitives, however, it only offers protection against software and not physical attacks (e.g., it trusts a hypervisor to disable interrupts).

Solutions against the snapshot adversary. Works in this category protect against the snapshot adversary, which takes a snapshot of the data at a fixed point in time (e.g., stolen hard drive). We stress that $\mathcal{E}$psolute provides semantic security against the snapshot adversary on top of AP and CV protection. CryptDB [56] is a seminal work in this direction offering computations over encrypted data. It has since been shown (e.g. [9, 41, 51]) that the underlying property-preserving schemes allow for reconstruction attacks. Arx [55] provides strictly stronger security guarantees by using only semantically secure primitives. Seabed [54] uses an additively symmetric homomorphic encryption scheme for aggregates and certain filter queries. Samanthula et al. [62] offer a method to verify and apply a predicate (a junction of conditions) using garbled circuits or homomorphic encryption without revealing the predicate itself. SisoSPIR [39] presents a mechanism to build an oblivious index tree such that neither party learns the pass taken. See [15] for a survey of range query protocols in this category.

2 BACKGROUND

In this section we describe *an outsourced database system* adapted from [43], a base for our own model (Section 3), and the constructions we will use as building blocks in our solution.

2.1 Outsourced Database System

We abstract a database as a collection of n records r, each with a unique identifier r^{ID}, associated with search keys SK: $\mathcal{D} = \{(r_1, r_1^{\mathsf{ID}}, \mathsf{SK}_1), \ldots, (r_n, r_n^{\mathsf{ID}}, \mathsf{SK}_n)\}$. We assume that all records have an identical fixed bit-length, and that search keys are elements of the domain $\mathcal{X} = \{1, \ldots, N\}$ for some $N \in \mathbb{N}$. Outsourced database systems support search keys on multiple attributes, with a set of search keys for each of the attributes of a record. For the ease of presentation, we describe the model for a single indexed attribute and then show how to extend it to support multiple attributes.

A query is a predicate $q : \mathcal{X} \to \{0, 1\}$. Evaluating a query q on a database $\mathcal{D}$ results in $q(\mathcal{D}) = \{r_i : q(\mathsf{SK}_i) = 1\}$, all records whose search keys satisfy q.

Let $\mathcal{Q}$ be a set of queries. An *outsourced database system* for queries in $\mathcal{Q}$ consists of two protocols between two *stateful* parties: a user $\mathcal{U}$ and a server $\mathcal{S}$ (adapted from [43]):

Setup protocol Π_{setup}**:** $\mathcal{U}$ receives as input a database $\mathcal{D} = \{(r_1, r_1^{\mathsf{ID}}, \mathsf{SK}_1), \ldots, (r_n, r_n^{\mathsf{ID}}, \mathsf{SK}_n)\}$; $\mathcal{S}$ has no input. The output for $\mathcal{S}$ is a data structure $\mathcal{DS}$; $\mathcal{U}$ has no output besides its state.

Query protocol Π_{query}**:** $\mathcal{U}$ has a query $q \in \mathcal{Q}$ produced in the setup protocol as input; $\mathcal{S}$ has as input $\mathcal{DS}$ produced in the setup protocol. $\mathcal{U}$ outputs $q(\mathcal{D})$; $\mathcal{S}$ has no formal output. (Both parties may update their internal states.)

For correctness, we require that for any database $\mathcal{D} = \{(r_1, r_1^{\mathsf{ID}}, \mathsf{SK}_1), \ldots, (r_n, r_n^{\mathsf{ID}}, \mathsf{SK}_n)\}$ and query $q \in \mathcal{Q}$, it holds that running Π_{setup} and then Π_{query} on the corresponding inputs yields for $\mathcal{U}$ the correct output $\{r_i : q(\mathsf{SK}_i) = 1\}$ with overwhelming probability over the coins of the above runs. We call the protocol η-wrong if this probability is at least $1 - \eta$.

2.2 Differential Privacy and Sanitization

Differential privacy is a definition of privacy in analysis that protects information that is specific to individual records. More formally, we call databases $\mathcal{D}_1 \in \mathcal{X}^n$ and $\mathcal{D}_2 \in \mathcal{X}^n$ over domain $\mathcal{X}$ *neighboring* (denoted $\mathcal{D}_1 \sim \mathcal{D}_2$) if they differ in exactly one record.

Definition 2.1 ([23, 24]). *A randomized algorithm A is (ϵ, δ)-differentially private if for all $\mathcal{D}_1 \sim \mathcal{D}_2 \in \mathcal{X}^n$, and for all subsets O of the output space of A,*

$$\Pr[A(\mathcal{D}_1) \in O] \leq \exp(\epsilon) \cdot \Pr[A(\mathcal{D}_2) \in O] + \delta .$$

The probability is taken over the random coins of A.

When $\delta = 0$ we omit it and say that A preserves *pure* differential privacy, otherwise (when $\delta > 0$) we say that A preserves *approximate* differential privacy.

We will use mechanisms for answering count queries with differential privacy. Such mechanisms perturb their output to mask out the effect of any single record on their outcome. The simplest method for answering count queries with differential privacy is the Laplace Perturbation Algorithm (LPA) [24] where random noise drawn from a Laplace distribution is added to the count to be published. The noise is scaled so as to hide the effect any single record can have on the count. More generally, the LPA can be used to approximate any statistical result by scaling the noise to the *sensitivity* of the statistical analysis.[1]

[1]The *sensitivity* of a query q mapping databases into $\mathbb{R}^N$ is defined to be $\Delta(q) = \max_{\mathcal{D}_1 \sim \mathcal{D}_2 \in \mathcal{X}^n} \|q(\mathcal{D}_1) - q(\mathcal{D}_2)\|_1$.

Theorem 2.2 (adapted Theorem 1 from [24]). *Let $q : \mathcal{D} \to \mathbb{R}^N$. An algorithm A that adds independently generated noise from a zero-mean Laplace distribution with scale $\lambda = \Delta(q)/\epsilon$ to each of the N coordinates of $q(\mathcal{D})$, satisfies ϵ-differential privacy.*

While Theorem 2.2 is an effective and simple way of answering a single count query, we will need to answer a sequence of count queries, ideally, without imposing a bound on the length of this sequence. We will hence make use of *sanitization* algorithms.

Definition 2.3. *Let $\mathcal{Q}$ be a collection of queries. An $(\epsilon, \delta, \alpha, \beta)$-differentially private sanitizer for $\mathcal{Q}$ is a pair of algorithms (A, B) such that:*

- *A is (ϵ, δ)-differentially private, and*
- *on input a dataset $\mathcal{D} = d_1, \ldots, d_n \in \mathcal{X}^n$, A outputs a data structure $\mathcal{DS}$ such that with probability $1 - \beta$ for all $q \in \mathcal{Q}$, $|B(\mathcal{DS}, q) - \sum_i q(d_i)| \leq \alpha$.*

Remark 2.4. *Given an $(\epsilon, \delta, \alpha, \beta)$-differentially private sanitizer as in Definition 2.3 one can replace the answer $B(\mathcal{DS}, q)$ with $B'(\mathcal{DS}, q) = B(\mathcal{DS}, q) + \alpha$. Hence, with probability $1 - \beta$, for all $q \in \mathcal{Q}$, $0 \leq B'(\mathcal{DS}, q) - \sum_i q(d_i) \leq 2\alpha$. We will hence assume from now on that sanitizers have this latter guarantee on their error.*

The main idea of *sanitization* (a.k.a. private data release) is to release specific noisy statistics on a private dataset once, which can then be combined in order to answer an arbitrary number of queries without violating privacy. Depending on the query type and the notion of differential privacy (i.e., pure or approximate), different upper bounds on the error have been proven. Omitting the dependency on ϵ, δ, in case of point queries over domain size N, pure differential privacy results in $\alpha = \Theta(\log N)$ [6], while for approximate differential privacy $\alpha = O(1)$ [7]. For range queries over domain size N, these bounds are $\alpha = \Theta(\log N)$ for pure differential privacy [10, 25], and $\alpha = O\big((\log^* N)^{1.5}\big)$ for approximate differential privacy (with an almost matching lower bound of $\alpha = \Omega(\log^* N)$) [7, 16, 42]. More generally, Blum et al. [10] showed that any finite query set $\mathcal{Q}$ can be sanitized, albeit non-efficiently.

Answering point and range queries with differential privacy. Utilizing the LPA for answering point queries results in error $\alpha = O(\log N)$. A practical solution for answering range queries with error bounds very close to the optimal ones is the hierarchical method [25, 36, 72]. The main idea is to build an aggregate tree on the domain, and add noise to each node proportional to the tree height (i.e., noise scale logarithmic in the domain size N). Then, every range query is answered using the minimum number of tree nodes. Qardaji et al. [58] showed that the hierarchical algorithm of Hay et al. [36], when combined with their proposed optimizations, offers the lowest error.

Composition. Finally, we include a *composition* theorem (adapted from [47]) based on [23, 24]. It concerns executions of multiple differentially private mechanisms on non-disjoint and disjoint inputs.

Theorem 2.5. *Let $A_1, \ldots, A_r$ be mechanisms, such that each A_i provides ϵ_i-differential privacy. Let $\mathcal{D}_1, \ldots, \mathcal{D}_r$ be pairwise non-disjoint (resp., disjoint) datasets. Let A be another mechanism that executes $A_1(\mathcal{D}_1), \ldots, A_r(\mathcal{D}_r)$ using independent randomness for each A_i, and returns their outputs. Then, mechanism A is $\left(\sum_{i=1}^r \epsilon_i\right)$-differentially private (resp., $\left(\max_{i=1}^r \epsilon_i\right)$-differentially private).*

2.3 Oblivious RAM

Informally, Oblivious RAM (ORAM) is a mechanism that lets a user hide their RAM access pattern to remote storage. An adversarial server can monitor the actual accessed locations, but she cannot tell a read from a write, the content of the block or even whether the same logical location is being referenced. The notion was first defined by Goldreich [31] and Goldreich and Ostrovsky [32].

More formally, a (η_1, η_2)-ORAM protocol is a two-party protocol between a user $\mathcal{U}$ and a server $\mathcal{S}$ who stores a RAM array. In each round, the user $\mathcal{U}$ has input (o, a, d), where o is a RAM operation ($\mathbf{r}$ or $\mathbf{w}$), a is a memory address and d is a new data value, or $\perp$ for read operation. The input of $\mathcal{S}$ is the current array. Via the protocol, the server updates the memory or returns to $\mathcal{U}$ the data stored at the requested memory location, respectively. We speak of a sequence of such operations as a program $\mathbf{y}$ being *executed under the ORAM*.

An ORAM protocol must satisfy correctness and security. Correctness requires that $\mathcal{U}$ obtains the correct output of the computation except with at most probability η_1. For security, we require that for every user $\mathcal{U}$ there exists a simulator $\textsc{Sim}_{\textsc{ORAM}}$ which provides a simulation of the server's view in the above experiment given only the number of operations. That is, the output distribution of $\textsc{Sim}_{\textsc{ORAM}}(c)$ is indistinguishable from $\textsc{View}_{\mathcal{S}}$ with probability at most η_2 after c protocol rounds.

ORAM protocols are generally stateful, after each execution the client and server states are updated. *For brevity, throughout the paper we will assume the ORAM state updates are implicit, including the encryption key K generated and maintained by the client.*

Some existing efficient ORAM protocols are Square Root ORAM [31], Hierarchical ORAM [32], Binary-Tree ORAM [63], Interleave Buffer Shuffle Square Root ORAM [73], TP-ORAM [64], Path-ORAM [65] and TaORAM [61]. For detailed descriptions of each protocol, we recommend the work of Chang et al. [19]. The latter three ORAMs achieve the lowest communication and storage overheads, $O(\log n)$ and $O(n)$, respectively.

3 DIFFERENTIALLY PRIVATE OUTSOURCED DATABASE SYSTEMS

In this section we present our model, *differentially private outsourced database system*, CDP-ODB, its security definition, query types and efficiency measures. It is an extension of the ODB model in Section 2.

3.1 Adversarial model

We consider an honest-but-curious polynomial time adversary that attempts to breach differential privacy with respect to the input database $\mathcal{D}$. We observe later in Section 3.1.1 that it is impossible to completely hide the number of records returned on each query without essentially returning all the database records on each query. This, in turn, means that different query sequences may be distinguished, and, furthermore, that differential privacy may not be preserved if the query sequence depends on the content of the database records. We hence, only require the protection of differential privacy with respect to every fixed query sequence. Furthermore, we relax to computational differential privacy (following [49]).

In the following definition, the notation $\textsc{View}_{\Pi}(\mathcal{D}, q_1, \ldots, q_m)$ denotes the view of the server $\mathcal{S}$ in the execution of protocol Π in answering queries $q_1, \ldots, q_m$ with the underlying database $\mathcal{D}$.

Definition 3.1. *We say that an outsourced database system Π is (ϵ, δ)-computationally differentially private (a.k.a. CDP-ODB) if for every polynomial time distinguishing adversary $\mathcal{A}$, for every neighboring databases $\mathcal{D} \sim \mathcal{D}'$, and for every query sequence $q_1, \ldots, q_m \in \mathcal{Q}^m$ where $m = \text{poly}(\lambda)$,*

$$\Pr\left[\mathcal{A}\left(1^\lambda, \textit{View}_\Pi\left(\mathcal{D}, q_1, \ldots, q_m\right)\right) = 1\right] \leq$$
$$\exp\epsilon \cdot \Pr\left[\mathcal{A}\left(1^\lambda, \textit{View}_\Pi\left(\mathcal{D}', q_1, \ldots, q_m\right)\right) = 1\right] + \delta + \text{negl}(\lambda)\,,$$

where the probability is over the randomness of the distinguishing adversary $\mathcal{A}$ and the protocol Π.

Remark 3.2 (Informal). *We note that security and differential privacy in this model imply protection against communication volume and access pattern leakages and thus prevent a range of attacks, such as [17, 43, 51].*

3.1.1 On impossibility of adaptive queries. Non-adaptivity in our CDP-ODB definition does not reflect a deficiency of our specific protocol but rather an inherent source of leakage when the queries may depend on the decrypted data. Consider an adaptive CDP-ODB definition that does not fix the query sequence $q_1, \ldots, q_m$ in advance but instead an arbitrary (efficient) user $\mathcal{U}$ chooses them during the protocol execution with $\mathcal{S}$. As before, we ask that the $\mathcal{S}$'s view is DP on neighboring databases for every such $\mathcal{U}$. We observe that this definition cannot possibly be satisfied by *any* outsourced database system without unacceptable efficiency overhead. Note that non-adaptivity here does not imply that the client knows all the queries in advance, but rather can choose them at any time (e.g., depending on external circumstances) as long as they do not depend on true answers to prior queries.

To see this, consider two neighboring databases $\mathcal{D}, \mathcal{D}'$. Database $\mathcal{D}$ has 1 record with key = 0 and $\mathcal{D}'$ has none. Furthermore, both have 50 records with key = 50 and 100 records with key = 100. User $\mathcal{U}$ queries first for the records with key = 0, and then if there is a record with key = 0 it queries for the records with key = 50, otherwise for the records with key = 100. Clearly, an efficient outsourced database system cannot return nearly as many records when key = 50 versus key = 100 here. Hence, this allows distinguishing $\mathcal{D}, \mathcal{D}'$ with probability almost 1.

To give a concrete scenario, suppose neighboring medical databases differ in one record with a rare diagnosis "Alzheimer's disease". A medical professional queries the database for that diagnosis first (point query), and if there is a record, she queries the senior patients next (range query, age $\geq$ 65), otherwise she queries the general population (resulting in more records). We leave it open to meaningfully strengthen our definition while avoiding such impossibility results, and we defer the formal proof to future work.

3.2 Query types

In this work we are concerned with the following query types:

Range queries Here we assume a total ordering on $\mathcal{X}$. A query $q_{[a,b]}$ is associated with an interval $[a, b]$ for $1 \leq a \leq b \leq N$ such that $q_{[a,b]}(c) = 1$ iff $c \in [a, b]$ for all $c \in \mathcal{X}$. The equivalent SQL query is:

```
SELECT * FROM table WHERE attribute BETWEEN a AND b;
```

Point queries Here $\mathcal{X}$ is arbitrary and a query predicate q_a is associated with an element $a \in \mathcal{X}$ such that $q_a(b) = 1$ iff $a = b$. In an ordered domain, point queries are degenerate range queries. The equivalent SQL query is:

```
SELECT * FROM table WHERE attribute = a;
```

3.3 Measuring Efficiency

We define two basic efficiency measures for a CDP-ODB.

Storage efficiency is defined as the sum of the bit-lengths of the records in a database relative to the bit-length of a corresponding encrypted database. Specifically, we say that an outsourced database system has *storage efficiency* of (a_1, a_2) if the following holds. Fix any $\mathcal{D} = \{(r_1, r_1^{\text{ID}}, \text{SK}_1), \ldots, (r_n, r_n^{\text{ID}}, \text{SK}_n)\}$ and let $n_1 = \sum_{i=1}^n |r_i|$. Let $\mathcal{S}_{\text{state}}$ be an output of $\mathcal{S}$ on a run of Π_{setup} where $\mathcal{U}$ has input $\mathcal{D}$, and let $n_2 = |\mathcal{S}_{\text{state}}|$. Then $n_2 \leq a_1 n_1 + a_2$.

Communication efficiency is defined as the sum of the lengths of the records in bits whose search keys satisfy the query relative to the actual number of bits sent back as the result of a query. Specifically, we say that an outsourced database system has *communication efficiency* of (a_1, a_2) if the following holds. Fix any q and $\mathcal{DS}$ output by Π_{setup}, let $\mathcal{U}$ and $\mathcal{S}$ execute Π_{query} where $\mathcal{U}$ has inputs q, and output R, and $\mathcal{S}$ has input $\mathcal{DS}$. Let m_1 be the amount of data in bits transferred between $\mathcal{U}$ and $\mathcal{S}$ during the execution of Π_{query}, and let $m_2 = |R|$. Then $m_2 \leq a_1 m_1 + a_2$.

Note that $a_1 \geq 1$ and $a_2 \geq 0$ for both measures. We say that an outsourced database system is *optimally storage efficient* (resp., *optimally communication efficient*) if it has storage (resp., communication) efficiency of $(1, 0)$.

4 $\mathcal{E}$PSOLUTE

In this section we present a construction, $\mathcal{E}$psolute, that satisfies the security definition in Section 3, detailing algorithms for both range and point query types. We also provide efficiency guarantees for approximate and pure DP versions of $\mathcal{E}$psolute.

4.1 General construction

Let $\mathcal{Q}$ be a collection of queries. We are interested in building a differentially private outsourced database system for $\mathcal{Q}$, called $\mathcal{E}$psolute. Our solution will use these building blocks.

- A (η_1, η_2)-ORAM protocol ORAM$(\cdot)$.
- An $(\epsilon, \delta, \alpha, \beta)$-differentially private sanitizer (A, B) for $\mathcal{Q}$ and negligible β, which satisfies the non-negative noise guarantee from Remark 2.4.
- A pair of algorithms CreateIndex and Lookup. CreateIndex consumes $\mathcal{D}$ and produces an index data structure $\mathcal{I}$ that maps a search key SK to a list of record IDs r^{ID} corresponding to the given search key. Lookup consumes $\mathcal{I}$ and q and returns a list $T = r_1^{\text{ID}}, \ldots, r_{|T|}^{\text{ID}}$ of record IDs matching the supplied query.

Our protocol $\Pi = (\Pi_{\text{setup}}, \Pi_{\text{query}})$ of $\mathcal{E}$psolute works as shown in Algorithm 1. Hereafter, we reference lines in Algorithm 1. See Fig. 1 for a schematic description of the protocol.

Setup protocol Π_{setup}. Let $\mathcal{U}$'s input be a database $\mathcal{D} = \{(r_1, r_1^{\text{ID}}, \text{SK}_1), \ldots, (r_n, r_n^{\text{ID}}, \text{SK}_n)\}$ (line 2). $\mathcal{U}$ creates an index $\mathcal{I}$ mapping

Algorithm 1 $\mathcal{E}$psolute protocol. ORAM $(\cdot)$ denotes an execution of ORAM protocol (Section 2.3), where $\mathcal{U}$ plays the role of the client. ORAM protocol client and server states are implicit. $S \setminus T$ represents a set of valid record IDs S that are not in the true result set T.

Π_{setup}

	User $\mathcal{U}$		Server $\mathcal{S}$
1:	**User** $\mathcal{U}$		**Server** $\mathcal{S}$
2:	**Input:** $\mathcal{D}$		**Input:** $\emptyset$
3:	$\mathcal{I} \leftarrow \textsc{CreateIndex}(\mathcal{D})$		
4:	$\mathbf{y} = (\mathbf{w}, r_i^{\text{ID}}, r_i)\big\|_{i=1}^{n}$		
5:		$\xleftrightarrow{\text{ORAM}(\mathbf{y})}$	
6:	$\mathcal{DS} \leftarrow \text{A}(\text{SK}_1, \ldots, \text{SK}_N)$	$\xrightarrow{\mathcal{DS}}$	
7:	**Output:** $\mathcal{I}$		**Output:** $\mathcal{DS}$

Π_{query}

	User $\mathcal{U}$		Server $\mathcal{S}$
1:	**User** $\mathcal{U}$		**Server** $\mathcal{S}$
2:	**Input:** $q, \mathcal{I}$		**Input:** $\mathcal{DS}$
3:	$T \leftarrow \textsc{Lookup}(\mathcal{I}, q)$	$\xrightarrow{q}$	$c \leftarrow \text{B}(\mathcal{DS}, q)$
4:	$\mathbf{y}_{\text{true}} = (\mathbf{r}, r_i^{\text{ID}}, \perp)\big\|_{i \in T}$	$\xleftarrow{c}$	
5:	$\mathbf{y}_{\text{noise}} = (\mathbf{r}, S \setminus T, \perp)\big\|_1^{c-\lvert T\rvert}$		
6:	R	$\xleftrightarrow{\text{ORAM}(\mathbf{y}_{\text{true}} \Vert \mathbf{y}_{\text{noise}})}$	
7:	**Output:** R		**Output:** $\emptyset$

search keys to record IDs corresponding to these keys (line 3). $\mathcal{U}$ sends over the records to $\mathcal{S}$ by executing the ORAM protocol on the specified sequence (lines 4 to 5). $\mathcal{U}$ generates a DP structure $\mathcal{DS}$ over the search keys using sanitizer A, and sends $\mathcal{DS}$ over to $\mathcal{S}$ (line 6). The output of $\mathcal{U}$ is $\mathcal{I}$ and of $\mathcal{S}$ is $\mathcal{DS}$; final ORAM states of $\mathcal{S}$ and $\mathcal{U}$ are implicit, including encryption key K (line 7).

Query protocol Π_{query}. $\mathcal{U}$ starts with a query q and index $\mathcal{I}$, $\mathcal{S}$ starts with a DP structure $\mathcal{DS}$. One can think of these inputs as outputs of Π_{setup} (line 2). $\mathcal{U}$ immediately sends the query to $\mathcal{S}$, which uses the sanitizer B to compute the total number of requests c, while $\mathcal{U}$ uses index $\mathcal{I}$ to derive the true indices of the records the query q targets (line 3). $\mathcal{U}$ receives c from $\mathcal{S}$ and prepares two ORAM sequences: $\mathbf{y}_{\text{true}}$ for real records retrieval, and $\mathbf{y}_{\text{noise}}$ to pad the number of requests to c to perturb the communication volume. $\mathbf{y}_{\text{noise}}$ includes valid non-repeating record IDs that are not part of the true result set T (lines 4 to 5). $\mathcal{U}$ fetches the records, both real and fake, from $\mathcal{S}$ using the ORAM protocol (line 6). The output of $\mathcal{U}$ is the filtered set of records requested by the query q; final ORAM states of $\mathcal{S}$ and $\mathcal{U}$ are implicit (line 7).

The protocols for point and range queries only differ in sanitizer implementations, see Sections 4.5 and 4.6. Note above that in any execution of Π_{query} we have $c \geq q(\mathcal{D})$ with overwhelming probability $1 - \beta$ (by using sanitizers satisfying Remark 2.4), and thus the protocol is well-defined and its accuracy is $1 - \beta$. Also note that the DP parameter δ is lower-bounded by β because sampling negative noise, however improbable, violates privacy, and therefore the final construction is (ϵ, β)-DP.

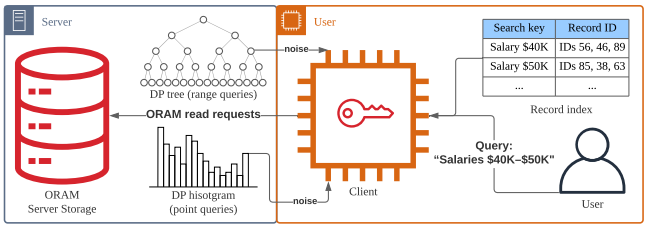

Figure 1: $\mathcal{E}$psolute construction

4.2 Security

Theorem 4.1. *$\mathcal{E}$psolute is $(\beta \cdot m)$-wrong and (ϵ, δ)-CDP-ODB where the negligible term is* $\text{negl}(\lambda) = 2 \cdot \eta_2$.

Proof. We consider a sequence of views

$$\text{View}_1 \to \text{View}_2 \to \text{View}_3 \to \text{View}_4 .$$

View_1 is $\text{View}_\Pi(\mathcal{D}, q_1, \ldots, q_m)$. View_2 is produced only from $\mathcal{DS} \leftarrow \text{A}(\text{SK}_1, \ldots, \text{SK}_N)$. Namely, compute $c_i \leftarrow \text{A}(\mathcal{DS}, q_i)$ for all i and run ORAM simulator on $\sum_i c_i$. By ORAM security,

$$\Pr[\mathcal{A}(\text{View}_1)] - \Pr[\mathcal{A}(\text{View}_2)] \leq \eta_2 .$$

View_3 is produced similarly but $\mathcal{DS} \leftarrow \text{A}\left(\text{SK}'_1, \ldots, \text{SK}'_N\right)$ instead. Note that the c_i are simply post-processing on $\mathcal{DS}$ via B so

$$\Pr[\mathcal{A}(\text{View}_2)] = \exp(\epsilon) \cdot \Pr[\mathcal{A}(\text{View}_3)] + \delta .$$

$\text{View}_4 = \text{View}_\Pi(\mathcal{D}', q_1, \ldots, q_m)$. It follows by ORAM security

$$\Pr[\mathcal{A}(\text{View}_3)] - \Pr[\mathcal{A}(\text{View}_4)] \leq \eta_2 .$$

Putting this all together completes the proof. □

4.3 Efficiency

For an ORAM with communication efficiency (a_1, a_2) and an (α, β)-differentially private sanitizer, the $\mathcal{E}$psolute communication efficiency is $(a_1, a_2 \cdot \alpha)$. The efficiency metrics demonstrate how the total storage or communication volume (the number of stored or transferred bits) changes additively and multiplicatively as the functions of data size n and domain N. We therefore have the following corollaries for the efficiency of the system in the cases of approximate and pure differential privacy.

Corollary 4.2. *$\mathcal{E}$psolute is an outsourced database system with storage efficiency $(O(1), 0)$. Depending on the query type, assume it offers the following communication efficiency.*

Range queries $\left(O(\log n), O\left(2^{\log^* N} \log n\right)\right)$

Point queries $(O(\log n), O(\log n))$

Then, there is a negligible δ such that $\mathcal{E}$psolute satisfies (ϵ, δ)-differential privacy for some ϵ.[2]

[2]Note that the existence of ϵ in this setting implies that the probability of an adversary breaking the DP guarantees is bounded by it.

Proof. By using ORAM, we store only the original data once and hence, we get optimal storage efficiency.

The communication efficiency depends on the upper bound of the error for each sanitizer when $\delta > 0$, as described in Section 2.2 and Remark 2.4. The most efficient ORAM protocol to date has $O(\log n)$ communication overhead (see Section 2.3). □

Corollary 4.3. *$\mathcal{E}$psolute is an outsourced database system with storage efficiency $(O(1), 0)$. Depending on the query type, assume it offers the following communication efficiency.*

Range queries $(O(\log n), O(\log N \log n))$
Point queries $(O(\log n), O(\log N \log n))$

Then, $\mathcal{E}$psolute satisfies ϵ-differential privacy for some ϵ.

Proof. Similarly, we derive the proof by considering the use of ORAM and the upper bound of the error for each sanitizer when $\delta = 0$ in Section 2.2. □

4.4 Extending to multiple attributes

We will now describe how $\mathcal{E}$psolute supports multiple indexed attributes and what the privacy and performance implications are. The naïve way is to simply duplicate the entire stack of states of $\mathcal{U}$ and $\mathcal{S}$, and during the query use the states whose attribute the query targets. However, $\mathcal{E}$psolute design allows to keep the most expensive part of the state — the ORAM state — shared for all attributes and both types of queries. Specifically, the index $\mathcal{I}$ and DP structure $\mathcal{DS}$ are generated per attribute and query type, while $\mathcal{U}$ and $\mathcal{S}$ ORAM states are generated once. This design is practical since $\mathcal{DS}$ is tiny and index $\mathcal{I}$ is relatively small compared to ORAM states, see Section 6.

We note that in case the indices grow large in number, it is practical to outsource them to the adversarial server using ORAM and download only the ones needed for each query. In terms of privacy, the solution is equivalent to operating different $\mathcal{E}$psolute instances because ORAM hides the values of records and access patterns entirely. Due to Theorem 2.5 for non-disjoint datasets, the total privacy budget of the multi-attribute system will be the sum of individual budgets for each attribute / index.

Next, we choose two DP sanitizers for our system, for point and for range queries, and calculate the α values to make them output positive values with high probability, consistent with Remark 2.4.

4.5 $\mathcal{E}$psolute for point queries

For point queries, we use the LPA method as the sanitizer to ensure pure differential privacy. Specifically, for every histogram bin, we draw noise from the Laplace distribution with mean α_p and scale $\lambda = 1/\epsilon$. To satisfy Remark 2.4, we have to set α_p such that if values are drawn from Laplace $(\alpha_p, 1/\epsilon)$ at least as many times as the number of bins N, they are all positive with high probability $1 - \beta$, for negligible β.

We can compute the exact minimum required value of α_p in order to ensure drawing positive values with high probability by using the CDF of the Laplace distribution. Specifically, α_p should be equal to the minimum value that satisfies the following inequality.

$$\left(1 - \frac{1}{2}e^{-\alpha_p \cdot \epsilon}\right)^N \leq 1 - \beta$$

which is equivalent to

$$\alpha_p = \left\lceil -\frac{\ln\left(2 - 2\sqrt[N]{1-\beta}\right)}{\epsilon} \right\rceil$$

4.6 $\mathcal{E}$psolute for range queries

For range queries, we implement the aggregate tree method as the sanitizer. Specifically, we build a complete k-ary tree on the domain, for a given k. A leaf node holds the number of records falling into each bin plus some noise. A parent node holds sum of the leaf values in the range covered by this node, plus noise. Every time a query is issued, we find the minimum number of nodes that cover the range, and determine the required number of returned records by summing these node values. Then, we ask the server to retrieve the records in the range, plus to retrieve multiple random records so that the total number of retrieved records matches the required number of returned records.

The noise per node is drawn from the Laplace distribution with mean α_h and scale $\lambda = \frac{\log_k N}{\epsilon}$. Consistent with Remark 2.4, we determine the mean value α_h in order to avoid drawing negative values with high probability. We have to set α_h such that if values are drawn from Laplace $\left(\alpha_h, \frac{\log_k N}{\epsilon}\right)$ at least as many times as the number of nodes in the tree, they are all positive with high probability $1 - \beta$, for negligible β.

Again, we can compute the exact minimum required value of α_h in order to ensure drawing positive values with high probability by using the CDF of the Laplace distribution. Specifically, α_h should be equal to the minimum value that satisfies the following inequality.

$$\left(1 - \frac{1}{2}e^{-\frac{\alpha_h \cdot \epsilon}{\log_k N}}\right)^{\text{nodes}} \leq 1 - \beta$$

which is equivalent to

$$\alpha_h = \left\lceil -\frac{\ln\left(2 - 2\sqrt[\text{nodes}]{1-\beta}\right) \cdot \log_k N}{\epsilon} \right\rceil \tag{1}$$

where $\text{nodes} = \frac{k^{\lceil \log_k (k-1) + \log_k N - 1 \rceil} - 1}{k-1} + N$ is the total number of tree nodes.

5 AN EFFICIENT PARALLEL $\mathcal{E}$PSOLUTE

While the previously described scheme is a secure and correct CDP-ODB, a single-threaded implementation may be prohibitively slow in practice. To bring the performance closer to real-world requirements, we need to be able to scale the algorithm horizontally. In this section, we describe an upgrade of $\mathcal{E}$psolute — a scalable parallel solution.

We suggest two variants of parallel $\mathcal{E}$psolute protocol. Both of them work by operating m ORAMs and randomly assigning to each of them n/m database records. For each query, we utilize the index $\mathcal{I}$ to find the required records from the corresponding ORAMs. For each ORAM, we execute a separate thread to retrieve the records. The threads work in parallel and there is no need for locking, since each ORAM works independently from the rest. We present two methods that differ in the way they build and store DP structure $\mathcal{DS}$, and hence the number of ORAM requests they make.

Algorithm 2 Parallel $\mathcal{E}$psolute for Π_γ, extends Algorithm 1. m is the number of parallel ORAMs. H is a random hash function $\mathrm{H}: \{0,1\}^* \to \{1,\ldots,m\}$. γ and $\tilde{k}_0$ are computed as in Section 5.2. $\mathcal{U}$ and $\mathcal{S}$ maintain m ORAM states implicitly.

Π_{setup} of Π_γ

	User $\mathcal{U}$		**Server** $\mathcal{S}$
1:	**Input:** $\mathcal{D}$		**Input:** $\emptyset$
3:	$\mathcal{I} \leftarrow \textsc{CreateIndex}(\mathcal{D}, m)$		
	..... **for** $j \in \{1,\ldots,m\}$ **do** (in parallel)		
4:	$\left\langle \bar{r}, \overline{r^{\mathsf{ID}}} \right\rangle$ s.t. $\mathrm{H}\left(r^{\mathsf{ID}}\right) = j$		
5:	$\mathbf{y} = \left\langle (\mathbf{w}, \overline{r^{\mathsf{ID}}}, \bar{r}) \right\rangle$		
6:		$\xleftrightarrow{\mathrm{ORAM}_j(\mathbf{y})}$	
	..... **endfor**		
7:	$\mathcal{DS} \leftarrow \mathrm{A}(\mathsf{SK}_1,\ldots,\mathsf{SK}_N)$	$\xrightarrow{\mathcal{DS}}$	
8:	**Output:** $\mathcal{I}$		**Output:** $\mathcal{DS}$

Π_{query} of Π_γ

	User $\mathcal{U}$		**Server** $\mathcal{S}$
1:	**Input:** $q, \mathcal{I}$		**Input:** $\mathcal{DS}$
3:	$T_1,\ldots,T_m \leftarrow \textsc{Lookup}(I, q)$	$\xrightarrow{q}$	$k \leftarrow \mathrm{B}(\mathcal{DS}, q)$
4:		$\xleftarrow{c}$	$c \leftarrow (1+\gamma)\frac{\tilde{k}_0}{m}$
	..... **for** $j \in \{1,\ldots,m\}$ **do** (in parallel)		
5:	$\mathbf{y}_{\mathsf{true}} = (\mathbf{r}, r_i^{\mathsf{ID}}, \perp)\big\vert_{i \in T_j}$		
6:	$\mathbf{y}_{\mathsf{noise}} = (\mathbf{r}, S \setminus T_j, \perp)\big\vert_1^{c-\lvert T_j \rvert}$		
7:	R_j	$\xleftrightarrow{\mathrm{ORAM}_j(\mathbf{y}_{\mathsf{true}} \Vert \mathbf{y}_{\mathsf{noise}})}$	
	..... **endfor**		
8:	**Output:** $R_j\big\vert_{j=1}^m$		**Output:** $\emptyset$

5.1 No-γ-method: DP structure per ORAM

In $\Pi_{\mathsf{no}-\gamma}$, for each ORAM / subset of the dataset, we build a DP index the same way as described in Section 4. We note that Theorem 2.5 for disjoint datasets applies to this construction: the privacy budget ϵ for the construction is the largest (least private) among the ϵ's of the DP indices for each ORAM / subset of the dataset.

The communication efficiency changes because (i) we essentially add m record subsets in order to answer a query, each having at most α extra random records, and (ii) each ORAM holds fewer records than before, resulting in a tree of height $\log \frac{n}{m}$.

However, we cannot expect that the records required for each query are equally distributed among the different ORAMs in order to reduce the multiplicative communication cost from $\log n$ to $\frac{\log n}{m}$. Instead, we need to bound the worst case scenario which is represented by the maximum number of records from any ORAM that is required to answer a query. This can be computed as follows.

Let X_j be 1 if a record for answering query q is in a specific ORAM_j, and 0 otherwise. Due to the random assignment of records to ORAMs, $\Pr\left[X_j = 1\right] = 1/m$. Assume that we need k_0 records in order to answer query q. The maximum number of records from ORAM_j in order to answer q is bounded as follows.

$$\Pr\left[\sum_{i=1}^{k_0} X_i > (1+\gamma)\frac{k_0}{m}\right] \leq \exp\left(-\frac{k_0\gamma^2}{3m}\right) \tag{2}$$

Finally, we need to determine the value of γ such that $\exp\left(-\frac{k_0\gamma^2}{3m}\right)$ is smaller than the value β. Thus, $\gamma = \sqrt{\frac{-3m\log\beta}{k_0}}$. The communication efficiency for each query type is described in the following corollary.

Corollary 5.1. *Let $\Pi_{\mathsf{no}-\gamma}$ be an outsourced database system with storage efficiency $(O(1), 0)$. Depending on the query type, $\Pi_{\mathsf{no}-\gamma}$ offers the following communication efficiency.*

Range queries $\left(O\left(\left(1+\sqrt{\frac{-3m\log\beta}{k_0}}\right)\log\frac{n}{m}\right), O\left(\frac{\log^{1.5} N}{\epsilon} m \log n\right)\right)$

Point queries $\left(O\left(\left(1+\sqrt{\frac{-3m\log\beta}{k_0}}\right)\log\frac{n}{m}\right), O\left(\frac{\log N}{\epsilon} m \log n\right)\right)$

Then, $\Pi_{\mathsf{no}-\gamma}$ satisfies ϵ-differential privacy for some ϵ.

In our experiments, we set m as a constant depending on the infrastructure. However, if m is set as $O(\log n)$, the total communication overhead of the construction will still exceed the lower-bound presented in [46].

5.2 γ-method: shared DP structure

In Π_γ, we maintain a single shared DP structure $\mathcal{DS}$. When a query is issued, we must ensure that the number of records retrieved from every ORAM is the same. As such, depending on the required noisy number of records $\tilde{k}_0$, we need to retrieve at most $(1+\gamma)\frac{\tilde{k}_0}{m}$ records from each ORAM, see Eq. (2), for $\gamma = \sqrt{\frac{-3m\log\beta}{\tilde{k}_0}}$. Setting $\tilde{k}_0 = k_0 + \frac{\log^{1.5} N}{\epsilon}$ for range queries and $\tilde{k}_0 = k_0 + \frac{\log N}{\epsilon}$ for point queries, the communication efficiency is as follows.

Corollary 5.2. *Let Π_γ be an outsourced database system with storage efficiency $(O(1), 0)$. Depending on the query type, Π_γ offers the following communication efficiency.*

Range queries $\left(O\left(\left(1+\sqrt{\frac{-3m\log\beta}{k_0+\frac{\log^{1.5} N}{\epsilon}}}\right)\log\frac{n}{m}\left(1+\frac{\log^{1.5} N}{\epsilon}\right)\right), 0\right)$

Point queries $\left(O\left(\left(1+\sqrt{\frac{-3m\log\beta}{k_0+\frac{\log N}{\epsilon}}}\right)\log\frac{n}{m}\left(1+\frac{\log N}{\epsilon}\right)\right), 0\right)$

Then, Π_γ satisfies ϵ-differential privacy for some ϵ.

Π_γ is depicted in Algorithm 2. There are a few extensions to the subroutines and notation from Algorithm 1. CreateIndex and Lookup now build and query the index which maps a search key to a pair — the record ID and the ORAM ID (1 to m) which stores the record. Lines 4 to 6 of Algorithm 2 Π_{setup} repeat for each ORAM and operate on the records partitioned for the given ORAM using hash function H on the record ID. A shared DP structure is created

with the sanitizer A (line 7). In Algorithm 2 Π_{query}, the total number of ORAM requests is computed once (line 4). Lines 5 to 7 repeat for each ORAM and operate on the subset of records stored in the given ORAM. Note that $\mathcal{U}$ and $\mathcal{S}$ implicitly maintain m ORAM states, and the algorithm uses the (A, B) sanitizer defined in Section 4.

Note that we guarantee privacy and access pattern protection on a record level. Each ORAM gets accessed at least once (much more than once for a typical query) thus the existence of a particular result record in a particular ORAM is hidden.

5.3 Practical improvements

Here we describe the optimizations aimed at bringing the construction's performance to the real-world demands.

5.3.1 ORAM request batching. We have noticed that although the entire set of ORAM requests for each query is known in advance, the requests are still executed sequentially. To address this inefficiency, we have designed a way to combine the requests in a batch and reduce the number of network requests to the bare minimum. We have implemented this method over PathORAM, which we use for the (η_1, η_2)-ORAM protocol, but the idea applies to most tree-based ORAMs (similar to [20]).

Our optimization utilizes the fact that all PathORAM leaf IDs are known in advance and paths in a tree-based storage share the buckets close to the root. The core idea is to read all paths first, processes the requests and and then write all paths back. This way the client makes a single `read` request, which is executed much faster than many small requests. Requests are then processed in main memory, including re-encryptions. Finally, the client executes the `write` requests using remapped leaves as a single operation, saving again compared to sequential execution.

This optimization provides up to **8 times** performance boost in our experiments. We note that the gains in speed and I/O overhead are achieved at the expense of main memory, which is not an issue given that the memory is released after a batch, and our experiments confirm that. The security guarantees of PathORAM are maintained with this optimization, since the security proof in [65, Section 3.6] still holds. Randomized encryption, statistically independent remapping of leaves, and stash processing do not change.

5.3.2 Lightweight ORAM servers. We have found in our experiments that naïve increase of the number of CPU cores and gigabytes of RAM does not translate into linear performance improvement after some threshold. Investigating the observation we have found that the $\mathcal{E}$psolute protocol, executing parallel ORAM protocols, is highly intensive with respect to main memory access, cryptographic operations and network usage. The bottleneck is the hardware — we have confirmed that on a single machine the RAM and network are saturated quickly preventing the linear scaling.

To address the problem, we split the user party $\mathcal{U}$ into multiple lightweight machines that are connected locally to each other and reside in a single trust domain (e.g., same data center). Specifically, we maintain a *client machine* that receives user requests and prepares ORAM `read` requests, and up to m lightweight *ORAM machines*, whose only job is to run the ORAM protocols in parallel. See Fig. 2 for the schematic representation of the architecture. We

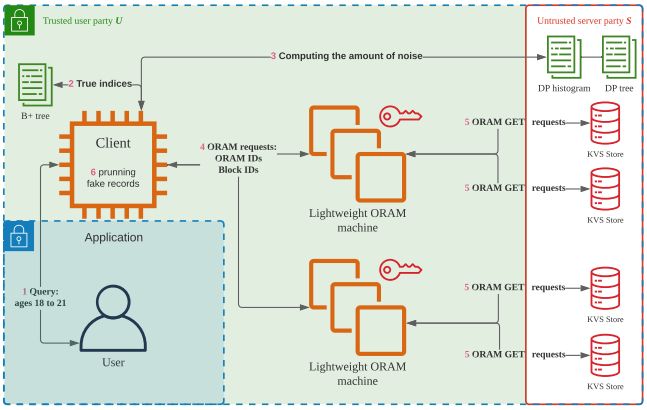

Figure 2: Lightweight ORAM machines diagram. A *user* sends a query to $\mathcal{U}$ modeled as the *client* machine, which uses local *data index* and *DP structures* to prepare a set of ORAM requests, which are sent to respective *ORAM machines*. These machines execute the ORAM protocol against the *untrusted storage* of $\mathcal{S}$.

emphasize that $\mathcal{U}$ is still a single party, therefore, the security and correctness guarantees remain valid.

The benefit of this approach is that each of the lightweight machines has its own hardware stack. Communication overhead among $\mathcal{U}$ machines is negligible compared to the one between $\mathcal{U}$ and $\mathcal{S}$. The approach is also flexible: it is possible to use up to m ORAM machines and the machines do not have to be identical. Our experiments show that when the same number of CPU cores and amount of RAM are consumed the efficiency gain is up to **5 times**.

6 EXPERIMENTAL EVALUATION

We have implemented our solution as a modular client-server application in C++. We open-sourced all components of the software set: PathORAM [14] and B+ tree [11] implementations and the main query executor [12]. We provide PathORAM and B+ tree components as C++ libraries to be used in other projects; the code is documented, benchmarked and tested (228 tests covering 100 % of the code). We have also published our datasets and query sets [13].

For cryptographic primitives, we used OpenSSL library (version 1.1.1i). For symmetric encryption in ORAM we have used AES-CBC algorithm [26, 27] with a 256-bits key (i.e., $\eta_2 = 2^{-256}$), for the hash algorithm H used to partition records among ORAMs we have used SHA-256 algorithm [52]. Aggregate tree fanout k is 16, proven to be optimal in [58].

We designed our experiments to answer the following questions:

Question-1 How practical is our system compared to the most efficient and most private real-world solutions?
Question-2 How practical is the storage overhead?
Question-3 How different inputs and parameters of the system affect its performance?
Question-4 How well does the system scale?
Question-5 What improvements do our optimizations provide?
Question-6 What is the impact of supporting multiple attributes?

To address **Question-1** we have run the default setting using conventional RDBMS (MySQL and PostgreSQL), Linear Scan approach and Shrinkwrap [5]. To target **Question-2**, we measured the exact storage used by the client and the server for different data, record and domain sizes. To answer **Question-3**, we ran a default setting and then varied all parameters and inputs, one at a time. For **Question-4** we gradually added vCPUs, ORAM servers and KVS instances and observed the rate of improvement in performance. For **Question-5** we have run the default setting with our optimizations toggled. Lastly, for **Question-6** we have used two datasets to construct two indices and then queried each of the attributes.

6.1 Data sets

We used two real and one synthetic datasets — California public pay pension database 2019 [67] (referred to as "CA employees"), Public Use Microdata Sample from US Census 2018 [68] (referred to as "PUMS") and synthetic uniform dataset. We have used salary / wages columns of the real datasets, and the numbers in the uniform set also represent salaries. The `NULL` and empty values were dropped.

We created three versions of each dataset — 10^5, 10^6 and 10^7 records each. For uniform dataset, we simply generated the target number of entries. For PUMS dataset, we picked the states whose number of records most closely matches the target sizes (Louisiana for 10^5, California for 10^6 and the entire US for 10^7). Uniform dataset was also generated for different domain sizes — number of distinct values for the record. For CA employees dataset, the set contains 260 277 records, so we contracted it and expanded in the following way. For contraction we uniformly randomly sampled 10^5 records. For expansion, we computed the histogram of the original dataset and sampled values uniformly within the bins.

Each of the datasets has a number of corresponding query sets. Each query set has a selectivity or range size, and is sampled either uniformly or following the dataset distribution (using its CDF).

6.2 Default setting

The default setting uses the Π_γ from Section 5 and lightweight ORAM machines from Section 5.3.2 and Fig. 2. We choose the Π_γ because it outperforms $\Pi_{\text{no}-\gamma}$ in all experiments (see **Question-4** in Section 6.5). In the setting, there are 64 Redis services (8 services per one Redis server VM), 8 ORAM machines communicating with 8 Redis services each, and the client, which communicates with these 8 ORAM machines. We have empirically found this configuration optimal for the compute nodes and network that we used in the experiments. ORAM and Redis servers run on GCP `n1-standard-16` VMs (Ubuntu 18.04), in regions `us-east4` and `us-east1` respectively. Client machine runs `n1-highmem-16` VM in the same region as ORAM machines. The ping time between the regions (i.e. between trusted and untrusted zones) is 12 ms and the effective bandwidth is 150 MB/s. Ping within a region is negligible.

Default DP parameters are $\epsilon = \ln(2) \approx 0.693$ and $\beta = 2^{-20}$, which are consistent with the other DP applications proposed in the literature [38]. Buckets number is set as the largest power of $k = 16$ that is no greater than the domain of the dataset N.

Default dataset is a uniform dataset of 10^6 records with domain size 10^4, and uniformly sampled queries with selectivity 0.5 %. Default record size is 4 KiB.

6.3 Experiment stages

Each experiment includes running 100 queries such that the overhead is measured from loading query endpoints into memory to receiving the exact and whole query response from all ORAM machines. The output of an experiment is, among other things, the overhead (in milliseconds), the number of real and noisy records fetched and communication volume averaged per query.

6.4 RDBMS, Linear Scan and Shrinkwrap

On top of varying the parameters, we have run similar workloads using alternative mechanisms — extremes representing highest performance or highest privacy. Unless stated otherwise, the client and the server are in the trusted and untrusted regions respectively, with the network configuration as in Section 6.2.

Relational databases. Conventional RDBMS represents the most efficient and least private and secure solution in our set. While MySQL and PostgreSQL offer some encryption options and no differential privacy, for our experiments we turned off security features for maximal performance. We have run queries against MySQL and PostgreSQL varying data and record sizes. We used `n1-standard-32` GCP VMs in `us-east1` region, running MySQL version 14.14 and PostgreSQL version 10.14.

Linear Scan. Linear scan is a primitive mechanism that keeps all records encrypted on the server then downloads, decrypts and scans the entire database to answer every query. This method is trivially correct, private and secure, albeit not very efficient. There are RDBMS solutions, which, when configured for maximum privacy, exhibit linear scan behavior (e.g., MS-SQL Always Encrypted with Randomized Encryption [48] and Oracle Column Transparent Data Encryption [53]). For a fair comparison we make the linear scan even more efficient by allowing it to download data via parallel threads matching the number of threads and bytes per request to that of our solution. Although linear scan is wasteful in the amount of data it downloads and processes, compared to our solution it has a benefit of not executing an ORAM protocol with its logarithmic overhead and network communication in both directions.

Shrinkwrap. Shrinkwrap [5] is a construction that answers federated SQL queries hiding both access pattern and communication volume. Using the EMP toolkit [71] and the code Shrinkwrap authors shared with us, we implemented a prototype that only answers range queries. This part of Shrinkwrap amounts to making a scan over the input marking the records satisfying the range, sorting the input, and then revealing the result set plus DP noise to the client. For the latter part we have adapted Shrinkwrap's Truncated Laplace Mechanism [5, Definition 4] to hierarchical method [58] in order to be able to answer an unbounded number of all possible range queries. We have emulated the outsourced database setting by using two `n1-standard-32` servers in different regions (12 ms ping and 150 MB/s bandwidth) executing the algorithm in a circuit model (the faster option per Shrinkwrap experiments) and then revealing the result to the trusted client. We note that although the complexity of a Shrinkwrap query is $O(n \log n)$ due to the sorting step, its functionality is richer as it supports more relational operators, like `JOIN`, `GROUP BY` and aggregation. We also note that since MySQL,

PostgreSQL and Shrinkwrap are not parallelized within the query, experiments using more CPUs do not yield higher performance.

6.5 Results and Observations

After running the experiments, we have made the following observations. Note that we report results based on the default setting.

- $\mathcal{E}$psolute is efficient compared to a strawman approach, RDBMS and Shrinkwrap: it is three orders of magnitude faster than Shrinkwrap, 18 times faster than the scan and only 4–8 times slower than a conventional database. In fact, for different queries, datasets, and record sizes, our system is much faster than the linear scan, as we show next.
- $\mathcal{E}$psolute's client storage requirements are very practical: client size is just below 30 MB while the size of the offloaded data is over 400 times larger.
- $\mathcal{E}$psolute scales predictably with the change in its parameters: data size affects performance logarithmically, record size — linearly, and privacy budget ϵ — exponentially.
- $\mathcal{E}$psolute is scalable: using Π_γ with the lightweight ORAM machines, the increase in the number of threads translates into linear performance boost.
- The optimizations proposed in Section 5.3 provide up to an order of magnitude performance gain.
- $\mathcal{E}$psolute efficiently supports multiple indexed attributes. The overhead and the client storage increase slightly due to a lower privacy budget and extra local indices.

For the purposes of reproducibility we have put the log traces of all our experiments along with the instructions on how to run them on a publicly available page epsolute.org. Unless stated otherwise, the scale in the figures is linear and the x-axis is categorical.

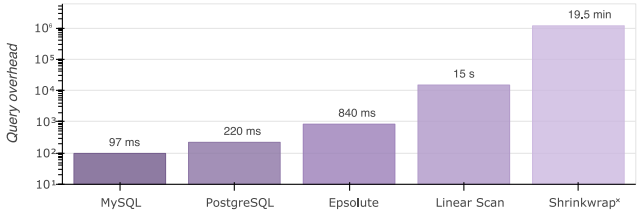

Figure 3: Different range-query mechanisms (log scale). Default setting: 10^6 4 KiB uniformly-sampled records with the range 10^4.

Question-1: against RDBMS, Linear Scan and Shrinkwrap. The first experiment we have run using $\mathcal{E}$psolute is the default setting in which we observed the query overhead of **840 ms**. To put this number in perspective, we compare $\mathcal{E}$psolute to conventional relational databases, the linear scan and Shrinkwrap.

For the default setting, MySQL and PostgreSQL, configured for no privacy and maximum performance, complete in 97 ms and 220 ms respectively, which is just **8 to 4 times** faster than $\mathcal{E}$psolute, see Fig. 3. Conventional RDBMS uses efficient indices (B+ trees) to locate requested records and sends them over without noise and encryption, and it does so using less hardware resources. In our experiments RDBMS performance is linearly correlated with the result and record sizes.

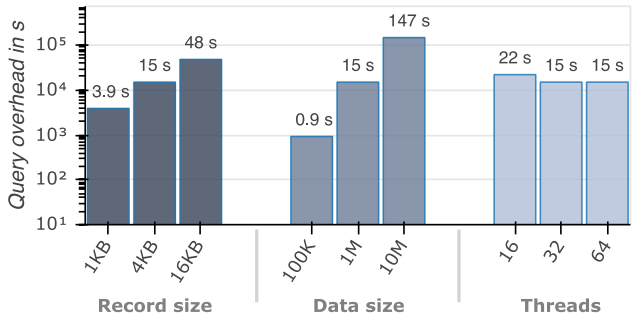

Figure 4: Linear scan performance, logarithmic scale. The experiments are run for the default setting of 10^6 records of size 4 KiB and 64 threads, with one of the three parameters varying.

Linear scan experiments demonstrate the practicality of $\mathcal{E}$psolute compared to a trivial "download everything every time" approach, see Fig. 4. Linear scan's overhead is $O(n)$ regardless of the queries, while $\mathcal{E}$psolute's overhead is $O(\log n)$ times the result size. According to our experiments, $\mathcal{E}$psolute eclipses the linear scan at 4 KiB, 64 threads and only *ten thousand records* (both mechanisms complete in about 120 ms). For a default setting (at a million records), the difference is **18 times**, see Fig. 4.

Because Shrinkwrap sorts the input obliviously in a circuit model, it incurs $O(n \log n)$ comparisons, each resulting in multiple circuit gates, which is much more expensive than the linear scan. Unlike linear scan, however, Shrinkwrap does not require much client memory as the client merely coordinates the query. While Shrinkwrap supports richer set of relational operators, for range queries alone $\mathcal{E}$psolute is **three orders of magnitude** faster.

n \ Record	1 KiB		4 KiB		16 KiB	
10^5	400 KiB	400 B	400 KiB	102 KiB	400 KiB	1.6 MB
	396 MB	4.6 MB	**1.5 GB**	14 MB	**6.2 GB**	51 MB
10^6	3.9 MB	400 B	3.9 MB	102 KiB	3.9 MB	1.6 MB
	3.2 GB	15 MB	**12 GB**	25 MB	**48 GB**	62 MB
10^7	40 MB	400 B	40 MB	102 KiB	*40 MB*	*1.6 MB*
	24 GB	99 MB	**96 GB**	109 MB	***384 GB***	*146 MB*
n \ N	100		10^4		10^6	

Table 1: Storage usage for varying data, record and domain sizes. The values are as follows. Left top: index $\mathcal{I}$ (B+ tree), right top: aggregate tree $\mathcal{DS}$, right bottom: ORAM $\mathcal{U}$ state and left bottom (bold): ORAM $\mathcal{S}$ state. *Italic* indicates that the value is estimated.

Question-2: storage. While $\mathcal{E}$psolute storage efficiency is near-optimal $(O(1), 0)$, it is important to observe the absolute values. Index $\mathcal{I}$ is implemented as a B+ tree with fanout 200 and occupancy 70 %, and its size, therefore, is roughly $5.7n$ bytes. Most of the ORAM client storage is the PathORAM stash with its size chosen in a way to bound failure probability to about $\eta_1 = 2^{-32}$ (see [65, Theorem 1]). In Table 1, we present $\mathcal{E}$psolute storage usage for the parameters

that affect it — data, record and domain sizes. We measured the sizes of the index $\mathcal{I}$, DP structure $\mathcal{DS}$, and ORAM client and server states. Our observations are: (i) index size expectedly grows only with the data size, (ii) $\mathcal{DS}$ is negligibly small in practice, (iii) small $\mathcal{I}$ and $\mathcal{DS}$ sizes imply the efficiency of supporting multiple indexed attributes, (iv) $\mathcal{S}$ to $\mathcal{U}$ storage size ratio varies from **85** in the smallest setting to more than **2 000** in the largest, and (v) one can trade client storage for ORAM failure probability. We conclude that the storage requirements of $\mathcal{E}$psolute are practical.

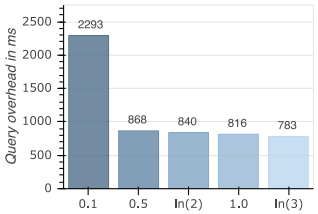

Figure 5: Privacy budget ϵ

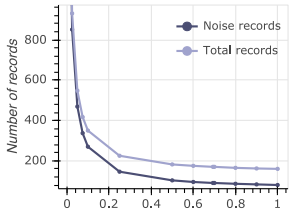

Figure 6: Effect of ϵ

Question-3: varying parameters. To measure and understand the impact of configuration parameters on the performance of our solution we have varied ϵ, record size, data size n, domain size N, selectivities, as well as data and query distributions. The relation that is persistent throughout the experiments is that for given data and record sizes, the performance (the time to completely execute a query) is strictly proportional to the total number of records, fake and real, that are being accessed per query. Each record access goes through the ORAM protocol, which, in turn, downloads, re-encrypts and uploads $O(\log n)$ blocks. These accesses contribute the most to the overhead and all other stages (e.g., traversing index or aggregate tree) are negligible.

Privacy budget ϵ and its effect. We have run the default setting for $\epsilon = \{0.1, 0.5, \ln 2, 1.0, \ln 3\}$. ϵ strictly contributes to the amount of noise, which grows exponentially as ϵ decreases, see Fig. 5, observe sharp drop. As visualized on Fig. 6, at high ϵ values the noise contributes a fraction of total overhead, while at low values the noise dominates the overhead entirely.

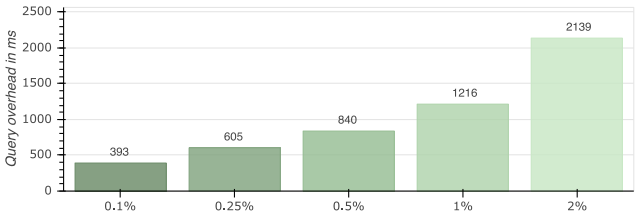

Figure 7: Selectivity

Selectivity. We have ranged the selectivity from 0.1 % to 2 % of the total number of records, see Fig. 7. Overhead expectedly grows with the result size. For smaller queries, and thus for lower overhead, the relation is positive, but not strictly proportional. This phenomena, observed for the experiments with low resulting per-query time, is explained by the variance among parallel threads. During each query the work is parallelized over m ORAMs and the query is completed when the *last* thread finishes. The problem, in distributed systems known as "the curse of the last reducer" [66], is when one thread takes disproportionally long to finish. In our case, we run 64 threads in default setting, and the delay is usually caused by a variety of factors — blocking I/O, network delay or something else running on a shared vCPU. This effect is noticeable when a single thread does relatively little work and small disruptions actually matter; the effect is negligible for large queries.

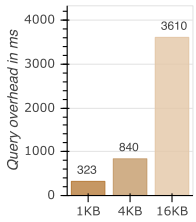

Figure 8: Record size

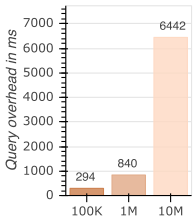

Figure 9: Data size

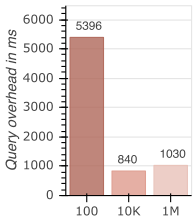

Figure 10: Domain size

Record, data and domain sizes. We have tried 1 KiB, 4 KiB and 16 KiB records, see Fig. 8. Trivially, the elapsed time is directly proportional to the record size.

We set n to 10^5, 10^6 and 10^7, see Fig. 9. The observed correlation of overhead against the data size is positive but non-linear, 10 times increment in n results in less than 10 times increase in time. This is explained by the ORAM overhead — when n changes, the ORAM storage gets bigger and its overhead is logarithmic.

For synthetic datasets we have set N to 100, 10^4 and 10^6, see Fig. 10. The results for domain size correlation are more interesting: low and high values deliver worse performance than the middle value. Small domain for a large data set means that a query often results in a high number of real records, which implies significant latency regardless of noise parameters. A sparse dataset, on the other hand, means that for a given selectivity wider domain is covered per query, resulting in more nodes in the aggregate tree contributing to the total noise value.

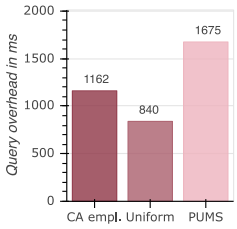

Figure 11: Data distribution

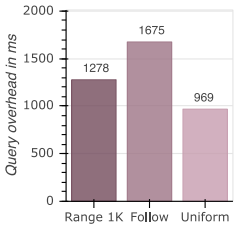

Figure 12: Query distribution

Data and query distributions. Our solution performs best on the uniform data and uniform ranges, see Figs. 11 and 12. Once a skew of any kind is introduced, there appear sparse and dense regions that contribute more overhead than uniform regions. Sparse regions span over wider range for a given selectivity, which results in more

noise. Dense regions are likely to include more records for a given range size, which again results in more fetched records. Both real datasets are heavily skewed towards smaller values as few people have ultra-high salaries.

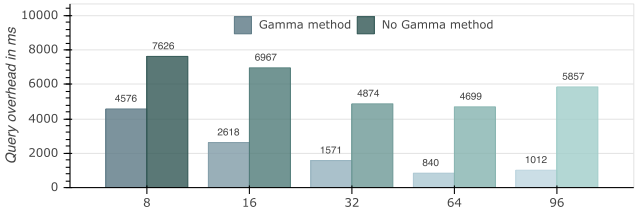

Figure 13: Scalability measurements for Π_γ and $\Pi_{\text{no}-\gamma}$

Question-4: scalability. Horizontal scaling is a necessity for a practical system, this is the motivation for the parallelization in the first place. Ideally, performance should improve proportionally to the parallelization factor, number of ORAMs in our case, m.

For scalability experiments we run the default setting for both $\Pi_{\text{no}-\gamma}$ and Π_γ (*no-γ-method* and *γ-method* respectively) varying the number of ORAMs m, from 8 to 96 (maximum vCPUs on a GCP VM). The results are visualized on Fig. 13. We report two positive observations: (i) the γ-method provides substantially better performance and storage efficiency, and (ii) when using this method the system scales linearly with the number of ORAMs. ($m = 96$ is a special case because some ORAMs had to share a single KVS.)

Improvement (section)	Enabled	Disabled	**Boost**
ORAM batching (5.3.1)	840 ms	6 978 ms	**8.3x**
Lightweight ORAM machines (5.3.2)	840 ms	4 484 ms	**5.3x**
Both improvements	840 ms	8 417 ms	***10.0x***

Table 2: Improvements over parallel $\mathcal{E}$psolute

Question-5: optimizations benefits. Table 2 demonstrates the boosts our improvements provide; when combined, the speedup is up to an order of magnitude.

ORAM request batching (Section 5.3.1) makes the biggest difference. We have run the default setting with and without the batching. The overhead is substantially smaller because far fewer I/O requests are being made, which implies benefits across the full stack: download, re-encryption and upload.

Using lightweight ORAM machines (Section 5.3.2) makes a difference when scaling. In the default setting, 64 parallel threads quickly saturate the memory access and network channel, while spreading computation among nodes removes the bottleneck.

Question-6: multiple attributes. $\mathcal{E}$psolute supports multiple indexed attributes. In Section 4.4 we described that the performance implications amount to having an index $\mathcal{I}$ and a DP structure $\mathcal{DS}$ per attribute and sharing the privacy budget ϵ among all attributes. As shown in Table 1, $\mathcal{I}$ and $\mathcal{DS}$ are the smallest components of the client storage. To observe the query performance impact, we have used the default dataset with domains 10^4 and 10^6 as indexed

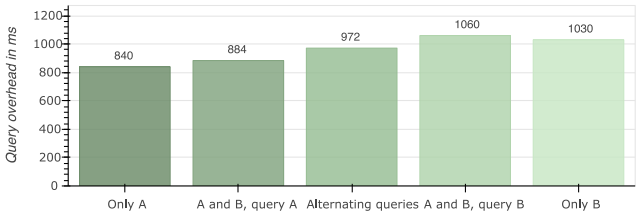

Figure 14: Query overhead when using multiple attributes. *Only A* and *Only B* index one attribute. *A and B* indexes both attributes and then queries one of them. *Alternating* indexes both attributes and runs half of the queries against *A* and another half against *B*.

attributes A and B respectively. We ran queries against only A, only B and against both attributes in alternating fashion. Each of the attributes used $\epsilon = \frac{\ln 2}{2}$ to match the default privacy budget of $\ln(2)$.

Fig. 14 demonstrates the query overhead of supporting multiple attributes. The principal observation is that the overhead increases only slightly due to a lower privacy budget. The client storage went up by just 9 MB, and still constitutes only 3.3 % of the server storage, which is not affected by the number of indexed attributes.

7 CONCLUSION AND FUTURE WORK

In this paper, we present a system called $\mathcal{E}$psolute that can be used to store and retrieve encrypted records in the cloud while providing strong and provable security guarantees, and that exhibits excellent query performance for range and point queries. We use an optimized Oblivious RAM protocol that has been parallelized together with very efficient Differentially Private sanitizers that hide both the access patterns and the exact communication volume sizes and can withstand advanced attacks that have been recently developed. We provide a prototype of the system and present an extensive evaluation over very large and diverse datasets and workloads that show excellent performance for the given security guarantees.

In our future work, we plan to investigate methods to extend our approaches to use a trusted execution environment (TEE), like SGX, in order to improve the performance even further. We will also explore a multi-user setting without the need for a shared stateful client, and enabling dynamic workloads with insertions and updates. We will also consider how adaptive and non-adaptive security models would change in the case of dynamic environments. One would presumably also require DP of the server's view in this setting. Lastly, we plan to explore other relational operations like `JOIN` and `GROUP BY`.

ACKNOWLEDGMENTS

We thank anonymous reviewers and Arkady Yerukhimovich for valuable feedback. We also thank Daria Bogatova for devising the name $\mathcal{E}$psolute and helping with the plots, diagrams and writing. Finally, we thank Johes Bater for sharing Shrinkwrap code and reviewing the prototype. Kobbi Nissim was supported by NSF Grant No. 2001041, "Rethinking Access Pattern Privacy: From Theory to Practice". Dmytro Bogatov and George Kollios were supported by NSF CNS-2001075 Award.

REFERENCES

[1] Arvind Arasu, Spyros Blanas, Ken Eguro, Manas Joglekar, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, Prasang Upadhyaya, and Ramarathnam Venkatesan. 2013. Secure Database-as-a-Service with Cipherbase. In *Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data (SIGMOD '13)*. Association for Computing Machinery, 1033–1036. https://doi.org/10.1145/2463676.2467797

[2] Arvind Arasu, Spyros Blanas, Ken Eguro, Raghav Kaushik, Donald Kossmann, Ravi Ramamurthy, and Ramarathnam Venkatesan. 2013. Orthogonal Security With Cipherbase. In *6th Biennial Conference on Innovative Data Systems Research (CIDR'13)*.

[3] Sumeet Bajaj and Radu Sion. 2013. TrustedDB: A trusted hardware-based database with privacy and data confidentiality. *IEEE Transactions on Knowledge and Data Engineering* 26, 3 (2013), 752–765. https://doi.org/10.1109/TKDE.2013.38

[4] Johes Bater, Gregory Elliott, Craig Eggen, Satyender Goel, Abel Kho, and Jennie Rogers. 2017. SMCQL: Secure Querying for Federated Databases. *Proc. VLDB Endow.* 10, 6 (Feb. 2017), 673–684. https://doi.org/10.14778/3055330.3055334

[5] Johes Bater, Xi He, William Ehrich, Ashwin Machanavajjhala, and Jennie Rogers. 2018. Shrinkwrap: efficient sql query processing in differentially private data federations. *Proceedings of the VLDB Endowment* 12, 3 (2018), 307–320. https://doi.org/10.14778/3291264.3291274

[6] Amos Beimel, Hai Brenner, Shiva Prasad Kasiviswanathan, and Kobbi Nissim. 2014. Bounds on the sample complexity for private learning and private data release. *Machine learning* 94, 3 (2014), 401–437. https://doi.org/10.1007/s10994-013-5404-1

[7] Amos Beimel, Kobbi Nissim, and Uri Stemmer. 2013. Private learning and sanitization: Pure vs. approximate differential privacy. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques*. Springer, 363–378. https://doi.org/10.1007/978-3-642-40328-6_26

[8] Amos Beimel, Kobbi Nissim, and Mohammad Zaheri. 2019. Exploring Differential Obliviousness. In *Approximation, Randomization, and Combinatorial Optimization. Algorithms and Techniques (APPROX/RANDOM 2019) (Leibniz International Proceedings in Informatics (LIPIcs), Vol. 145)*. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 65:1–65:20. https://doi.org/10.4230/LIPIcs.APPROX-RANDOM.2019.65

[9] Vincent Bindschaedler, Paul Grubbs, David Cash, Thomas Ristenpart, and Vitaly Shmatikov. 2018. The Tao of Inference in Privacy-protected Databases. *PVLDB* 11, 11 (2018), 1715–1728. https://doi.org/10.14778/3236187.3236217

[10] Avrim Blum, Katrina Ligett, and Aaron Roth. 2013. A learning theory approach to non-interactive database privacy. *Journal of the ACM (JACM)* 60, 2 (2013), 1–25. https://doi.org/10.1145/1374376.1374464

[11] Dmytro Bogatov. 2021. B+ tree (static). https://github.com/epsolute/b-plus-tree.

[12] Dmytro Bogatov. 2021. Epsolute. https://github.com/epsolute/epsolute.

[13] Dmytro Bogatov. 2021. Original and procesed datasets used in this paper. http://csr.bu.edu/dp-oram/. Accessed: 2021-01-24.

[14] Dmytro Bogatov. 2021. PathORAM. https://github.com/epsolute/path-oram.

[15] Dmytro Bogatov, George Kollios, and Leonid Reyzin. 2019. A comparative evaluation of order-revealing encryption schemes and secure range-query protocols. *Proceedings of the VLDB Endowment* 12, 8 (2019), 933–947. https://doi.org/10.14778/3324301.3324309

[16] Mark Bun, Kobbi Nissim, Uri Stemmer, and Salil Vadhan. 2015. Differentially private release and learning of threshold functions. In *2015 IEEE 56th Annual Symposium on Foundations of Computer Science*. IEEE, 634–649. https://doi.org/10.1109/FOCS.2015.45

[17] David Cash, Paul Grubbs, Jason Perry, and Thomas Ristenpart. 2015. Leakage-abuse attacks against searchable encryption. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 668–679. https://doi.org/10.1145/2810103.2813700

[18] T-H. Hubert Chan, Kai-Min Chung, Bruce M. Maggs, and Elaine Shi. 2019. Foundations of Differentially Oblivious Algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms* (San Diego, California) *(SODA '19)*. Society for Industrial and Applied Mathematics, USA, 2448–2467.

[19] Zhao Chang, Dong Xie, and Feifei Li. 2016. Oblivious RAM: A dissection and experimental evaluation. *Proceedings of the VLDB Endowment* 9, 12, 1113–1124. https://doi.org/10.14778/2994509.2994528

[20] Binyi Chen, Huijia Lin, and Stefano Tessaro. 2016. Oblivious Parallel RAM: Improved Efficiency and Generic Constructions. In *Theory of Cryptography*, Eyal Kushilevitz and Tal Malkin (Eds.). Springer Berlin Heidelberg, 205–234. https://doi.org/10.1007/978-3-662-49099-0_8

[21] Ioannis Demertzis, Dimitrios Papadopoulos, Charalampos Papamanthou, and Saurabh Shintre. 2020. SEAL: Attack Mitigation for Encrypted Databases via Adjustable Leakage. In *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, 2433–2450. https://www.usenix.org/conference/usenixsecurity20/presentation/demertzis

[22] Ioannis Demertzis, Stavros Papadopoulos, Odysseas Papapetrou, Antonios Deligiannakis, and Minos Garofalakis. 2016. Practical private range search revisited. In *Proceedings of the 2016 International Conference on Management of Data*. 185–198. https://doi.org/10.1145/2882903.2882911

[23] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 486–503. https://doi.org/10.1007/11761679_29

[24] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. 2006. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*. Springer, 265–284. https://doi.org/10.1007/11681878_14

[25] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N Rothblum. 2010. Differential privacy under continual observation. In *Proceedings of the forty-second ACM symposium on Theory of computing*. 715–724. https://doi.org/10.1145/1806689.1806787

[26] Morris Dworkin. 2001. Recommendation for Block Cipher Modes of Operation: Methods and Techniques. https://doi.org/10.6028/NIST.SP.800-38A

[27] Morris Dworkin, Elaine Barker, James Nechvatal, James Foti, Lawrence Bassham, E. Roback, and James Dray. 2001. Advanced Encryption Standard (AES). https://doi.org/10.6028/NIST.FIPS.197

[28] Saba Eskandarian and Matei Zaharia. 2019. ObliDB: Oblivious query processing for secure databases. *PVLDB* 13, 2 (2019), 169–183. https://doi.org/10.14778/3364324.3364331

[29] Benny Fuhry, Raad Bahmani, Ferdinand Brasser, Florian Hahn, Florian Kerschbaum, and Ahmad-Reza Sadeghi. 2017. HardIDX: Practical and secure index with SGX. In *IFIP Annual Conference on Data and Applications Security and Privacy*. Springer, 386–408. https://doi.org/10.1007/978-3-319-61176-1_22

[30] Craig Gentry. 2010. Computing arbitrary functions of encrypted data. *Commun. ACM* 53, 3 (2010), 97–105. https://doi.org/10.1145/1666420.1666444

[31] Oded Goldreich. 1987. Towards a theory of software protection and simulation by oblivious RAMs. In *Proceedings of the nineteenth annual ACM symposium on Theory of computing*. 182–194. https://doi.org/10.1145/28395.28416

[32] Oded Goldreich and Rafail Ostrovsky. 1996. Software protection and simulation on oblivious RAMs. *Journal of the ACM (JACM)* 43, 3 (1996), 431–473. https://doi.org/10.1145/233551.233553

[33] Paul Grubbs, Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2018. Pump up the volume: Practical database reconstruction from volume leakage on range queries. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 315–331. https://doi.org/10.1145/3243734.3243864

[34] 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*. ACM, 162–168. https://doi.org/10.1145/3102980.3103007

[35] Zichen Gui, Oliver Johnson, and Bogdan Warinschi. 2019. Encrypted databases: New volume attacks against range queries. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 361–378. https://doi.org/10.1145/3319535.3363210

[36] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the Accuracy of Differentially Private Histograms through Consistency. *Proc. VLDB Endow.* 3, 1–2 (Sept. 2010), 1021–1032. https://doi.org/10.14778/1920841.1920970

[37] Bijit Hore, Sharad Mehrotra, Mustafa Canim, and Murat Kantarcioglu. 2012. Secure multidimensional range queries over outsourced data. *VLDBJ* 21, 3 (2012), 333–358. https://doi.org/10.1007/s00778-011-0245-7

[38] Justin Hsu, Marco Gaboardi, Andreas Haeberlen, Sanjeev Khanna, Arjun Narayan, Benjamin C Pierce, and Aaron Roth. 2014. Differential privacy: An economic method for choosing epsilon. In *2014 IEEE 27th Computer Security Foundations Symposium*. IEEE, 398–410. https://doi.org/10.1109/CSF.2014.35

[39] Yuval Ishai, Eyal Kushilevitz, Steve Lu, and Rafail Ostrovsky. 2016. Private large-scale databases with distributed searchable symmetric encryption. In *Cryptographers' Track at the RSA Conference*. Springer, 90–107. https://doi.org/10.1007/978-3-319-29485-8_6

[40] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2012. Access Pattern disclosure on Searchable Encryption: Ramification, Attack and Mitigation. In *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society.

[41] Mohammad Saiful Islam, Mehmet Kuzu, and Murat Kantarcioglu. 2014. Inference attack against encrypted range queries on outsourced databases. In *Proceedings of the 4th ACM conference on Data and application security and privacy*. 235–246. https://doi.org/10.1145/2557547.2557561

[42] Haim Kaplan, Katrina Ligett, Yishay Mansour, Moni Naor, and Uri Stemmer. 2020. Privately Learning Thresholds: Closing the Exponential Gap. In *Proceedings of Thirty Third Conference on Learning Theory (Proceedings of Machine Learning Research, Vol. 125)*. PMLR, 2263–2285.

[43] Georgios Kellaris, George Kollios, Kobbi Nissim, and Adam O'neill. 2016. Generic attacks on secure outsourced databases. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. 1329–1340. https://doi.org/10.1145/2976749.2978386

[44] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. 2020. The state of the uniform: attacks on encrypted databases beyond the uniform query distribution. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1223–1240. https://doi.org/10.1109/SP40000.2020.00029

[45] Marie-Sarah Lacharité, Brice Minaud, and Kenneth G Paterson. 2018. Improved reconstruction attacks on encrypted data using range query leakage. In *2018 IEEE*

Symposium on Security and Privacy (SP). IEEE, 297–314. https://doi.org/10.1109/SP.2018.00002

[46] Kasper Green Larsen, Mark Simkin, and Kevin Yeo. 2020. Lower Bounds for Multi-server Oblivious RAMs. In *Theory of Cryptography*. Springer International Publishing, 486–503. https://doi.org/10.1007/978-3-030-64375-1_17

[47] Frank D McSherry. 2009. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*. 19–30. https://doi.org/10.1145/1559845.1559850

[48] Microsoft. 2021. MS-SQL Always Encrypted. https://docs.microsoft.com/sql/relational-databases/security/encryption/always-encrypted-database-engine.

[49] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. 2009. Computational differential privacy. In *Annual International Cryptology Conference*. Springer, 126–142. https://doi.org/10.1007/978-3-642-03356-8_8

[50] Pratyush Mishra, Rishabh Poddar, Jerry Chen, Alessandro Chiesa, and Raluca Ada Popa. 2018. Oblix: An efficient oblivious search index. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 279–296. https://doi.org/10.1109/SP.2018.00045

[51] Muhammad Naveed, Seny Kamara, and Charles V Wright. 2015. Inference attacks on property-preserving encrypted databases. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 644–655. https://doi.org/10.1145/2810103.2813651

[52] National Institute of Standards and Technology. 2015. Secure Hash Standard (SHS). https://doi.org/10.6028/NIST.FIPS.180-4

[53] Oracle. 2021. Introduction to Transparent Data Encryption. https://docs.oracle.com/database/121/ASOAG/introduction-to-transparent-data-encryption.htm.

[54] Antonis Papadimitriou, Ranjita Bhagwan, Nishanth Chandran, Ramachandran Ramjee, Andreas Haeberlen, Harmeet Singh, Abhishek Modi, and Saikrishna Badrinarayanan. 2016. Big Data Analytics over Encrypted Datasets with Seabed. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. USENIX Association, 587–602.

[55] Rishabh Poddar, Tobias Boelter, and Raluca Ada Popa. 2019. Arx: an encrypted database using semantically secure encryption. *Proceedings of the VLDB Endowment* 12, 11 (2019), 1664–1678. https://doi.org/10.14778/3342263.3342641

[56] Raluca Ada Popa, Catherine MS Redfield, Nickolai Zeldovich, and Hari Balakrishnan. 2011. CryptDB: Protecting confidentiality with encrypted query processing. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. 85–100.

[57] Christian Priebe, Kapil Vaswani, and Manuel Costa. 2018. EnclaveDB: A secure database using SGX. In *2018 IEEE Symposium on Security and Privacy (SP)*. IEEE, 264–278. https://doi.org/10.1109/SP.2018.00025

[58] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Understanding hierarchical methods for differentially private histograms. *Proceedings of the VLDB Endowment* 6, 14 (2013), 1954–1965. https://doi.org/10.14778/2556549.2556576

[59] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. 2020. Cryptε: Crypto-assisted differential privacy on untrusted servers. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*. 603–619. https://doi.org/10.1145/3318464.3380596

[60] Cetin Sahin, Tristan Allard, Reza Akbarinia, Amr El Abbadi, and Esther Pacitti. 2018. A Differentially Private Index for Range Query Processing in Clouds. In *2018 IEEE 34th International Conference on Data Engineering (ICDE)*. 857–868. https://doi.org/10.1109/ICDE.2018.00082

[61] Cetin Sahin, Victor Zakhary, Amr El Abbadi, Huijia Lin, and Stefano Tessaro. 2016. Taostore: Overcoming asynchronicity in oblivious data storage. In *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 198–217. https://doi.org/10.1109/SP.2016.20

[62] Bharath Kumar Samanthula, Wei Jiang, and Elisa Bertino. 2014. Privacy-preserving complex query evaluation over semantically secure encrypted data. In *European Symposium on Research in Computer Security*. Springer, 400–418. https://doi.org/10.1007/978-3-319-11203-9_23

[63] Elaine Shi, T-H Hubert Chan, Emil Stefanov, and Mingfei Li. 2011. Oblivious RAM with $O(\log^3 N)$ worst-case cost. In *International Conference on The Theory and Application of Cryptology and Information Security*. Springer, 197–214. https://doi.org/10.1007/978-3-642-25385-0_11

[64] Emil Stefanov, Elaine Shi, and Dawn Xiaodong Song. 2012. Towards Practical Oblivious RAM. In *Network and Distributed System Security Symposium (NDSS)*.

[65] Emil Stefanov, Marten Van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao Yu, and Srinivas Devadas. 2013. Path ORAM: an extremely simple oblivious RAM protocol. In *Proceedings of the 2013 ACM SIGSAC conference on Computer & communications security*. 299–310. https://doi.org/10.1145/3177872

[66] Siddharth Suri and Sergei Vassilvitskii. 2011. Counting triangles and the curse of the last reducer. In *Proceedings of the 20th international conference on World wide web*. 607–614. https://doi.org/10.1145/1963405.1963491

[67] Transparent California. 2019. California public pay and pension 2019 dataset. https://transparentcalifornia.com.

[68] U.S. Census Bureau. 2018. American Community Survey Public Use Microdata Sample. https://www.census.gov/programs-surveys/acs/microdata.html.

[69] Vinod Vaikuntanathan. 2011. Computing blindfolded: New developments in fully homomorphic encryption. In *2011 IEEE 52nd Annual Symposium on Foundations of Computer Science*. IEEE, 5–16. https://doi.org/10.1109/FOCS.2011.98

[70] Dhinakaran Vinayagamurthy, Alexey Gribov, and Sergey Gorbunov. 2019. StealthDB: a scalable encrypted database with full SQL query support. *Proceedings on Privacy Enhancing Technologies* 2019, 3 (2019), 370–388. https://doi.org/10.2478/popets-2019-0052

[71] Xiao Wang, Alex J. Malozemoff, and Jonathan Katz. 2016. EMP-toolkit: Efficient MultiParty computation toolkit. https://github.com/emp-toolkit.

[72] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. 2010. Differential privacy via wavelet transforms. *IEEE Transactions on knowledge and data engineering* 23, 8 (2010), 1200–1214. https://doi.org/10.1109/TKDE.2010.247

[73] Dong Xie, Guanru Li, Bin Yao, Xuan Wei, Xiaokui Xiao, Yunjun Gao, and Minyi Guo. 2016. Practical private shortest path computation based on oblivious storage. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*. IEEE, 361–372. https://doi.org/10.1109/ICDE.2016.7498254

[74] Min Xu, Antonis Papadimitriou, Andreas Haeberlen, and Ariel Feldman. 2019. Hermetic: Privacy-preserving distributed analytics without (most) side channels. (2019).

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