# $\epsilon$KTELO: A Framework for Defining Differentially-Private Computations

Dan Zhang
College of Inf. and Computer Sciences
U. of Massachusetts Amherst
dzhang@cs.umass.edu

Ryan McKenna
College of Inf. and Computer Sciences
U. of Massachusetts Amherst
rmckenna@cs.umass.edu

Ios Kotsogiannis
Dept. of Computer Science
Duke University
iosk@cs.duke.edu

Michael Hay
Computer Science Dept.
Colgate University
mhay@colgate.edu

Ashwin Machanavajjhala
Dept. of Computer Science
Duke University
ashwin@cs.duke.edu

Gerome Miklau
College of Inf. and Computer Sciences
U. of Massachusetts Amherst
miklau@cs.umass.edu

## ABSTRACT

The adoption of differential privacy is growing but the complexity of designing private, efficient and accurate algorithms is still high. We propose a novel programming framework and system, $\epsilon$KTELO, for implementing both existing and new privacy algorithms. For the task of answering linear counting queries, we show that nearly all existing algorithms can be composed from operators, each conforming to one of a small number of operator classes. While past programming frameworks have helped to ensure the privacy of programs, the novelty of our framework is its significant support for authoring accurate and efficient (as well as private) programs.

After describing the design and architecture of the $\epsilon$KTELO system, we show that $\epsilon$KTELO is expressive, that it allows for safer implementations through code reuse, and that it allows both privacy novices and experts to easily design algorithms. We demonstrate the use of $\epsilon$KTELO by designing several new state-of-the-art algorithms.

**ACM Reference Format:**
Dan Zhang, Ryan McKenna, Ios Kotsogiannis, Michael Hay, Ashwin Machanavajjhala, and Gerome Miklau. 2018. $\epsilon$KTELO: A Framework for Defining Differentially-Private Computations. In *SIGMOD'18: 2018 International Conference on Management of Data, June 10–15, 2018, Houston, TX, USA.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3183713.3196921

## 1 INTRODUCTION

As the collection of personal data has increased, many institutions face an urgent need for reliable privacy protection mechanisms. They must balance the need to protect individuals with demands to use collected data for new applications, to model their users' behavior, or share data with external partners. Differential privacy [7, 8] is a rigorous privacy definition that offers a persuasive assurance to individuals, provable guarantees, and the ability to analyze the impact of combined releases of data. Informally, an algorithm satisfies differential privacy if its output does not change too much when any one record in the input database is added or removed.

The research community has actively investigated differential privacy and algorithms are known for a variety of tasks ranging from data exploration to query answering to machine learning. However, the adoption of differentially private techniques in real-world applications remains rare. This is because implementing programs that provably satisfy privacy and ensure sufficient utility for a given task is still extremely challenging for non-experts in differential privacy. In fact, the few real world deployments of differential privacy – like OnTheMap [1, 13] (a U.S. Census Bureau data product), RAPPOR [10] (a Google Chrome extension), and Apple's private collection of emoji's and HealthKit data – have required teams of privacy experts to ensure that implementations meet the privacy standard and that they deliver acceptable utility. There are three important challenges in implementing and deploying differentially private algorithms.

The first and foremost challenge is the difficulty of designing utility-optimal algorithms: i.e., algorithms that can extract the maximal accuracy given a fixed "privacy budget." While there are a number of general-purpose differentially private algorithms, such as the Laplace Mechanism [7], they typically offer suboptimal accuracy if applied directly. A carefully designed algorithm can improve on general-purpose methods by an order of magnitude or more—without weakening privacy: accuracy is improved by careful engineering and sophisticated algorithm design.

One might hope for a single dominant algorithm for each task, but a recent empirical study [15] showed that the accuracy of existing algorithms is complex: no single algorithm delivers the best accuracy across the range of settings in which it may be deployed. The choice of the best algorithm may depend on the particular task, the available privacy budget, and properties of the input data including how much data is available or distributional properties of the data. Therefore, to achieve state-of-the-art accuracy, a practitioner currently has to make a host of complex algorithm choices, which may include choosing a low-level representation for the input data, translating their queries into that representation, choosing among available algorithms, and setting parameters. The best choices will vary for different input data and different analysis tasks.

The second challenge is that the tasks in which practitioners are interested are diverse and may differ from those considered in the literature. Hence, existing algorithms need to be adapted to new application settings, but this can be non-trivial. For instance, techniques used by modern privacy algorithms include optimizing error over

*SIGMOD'18, June 10–15, 2018, Houston, TX, USA*
© 2018 Association for Computing Machinery.
ACM ISBN 978-1-4503-4703-7/18/06…$15.00
https://doi.org/10.1145/3183713.3196921

multiple queries by identifying common sub-expressions, obtaining noisy counts from the data at different resolutions, and using complex inference techniques to reconstruct answers to target queries from noisy, inconsistent and incomplete measurement queries. But different algorithms use different specialized operators for these subtasks, and it can be challenging to adapt them to new situations. Thus, designing utility-optimal algorithms requires significant expertise in a complex and rapidly-evolving research literature.

A third equally important challenge is that correctly implementing differentially private algorithms can be difficult. There are known examples of algorithm pseudocode in research papers not satisfying differential privacy as claimed. For instance, Zhang et al [36] showed that many variants of a primitive called the sparse vector technique proposed in the literature do not satisfy differential privacy. Differential privacy can also be broken through incorrect implementations of correct algorithms. For example, Mironov [25] showed that standard implementations of basic algorithms like the Laplace Mechanism [7] can violate differential privacy because of their use of floating point arithmetic. Privacy-oriented programming frameworks such as PINQ [9, 24, 27], Fuzz [12], PrivInfer [4] and LightDP [34] help implement programs whose privacy can be verified with relatively little human intervention. While they help to ensure the privacy criterion is met, they may impose their own restrictions and offer little or no support for designing utility-optimal programs. In fact, in PINQ [24], some state-of-the-art algorithms involving inference and domain reduction cannot be implemented.

To address the aforementioned challenges, we propose $\epsilon$KTELO, a programming framework and system that aids programmers in developing differentially private programs with high utility. $\epsilon$KTELO programs can be used to solve a core class of statistical tasks that involve answering counting queries over a table of arbitrary dimension (described in Sec. 3). Tasks supported by $\epsilon$KTELO include releasing contingency tables, multi-dimensional histograms, answering OLAP and range queries, and even implementing private machine learning algorithms. This paper makes five main contributions.

First, we recognize that, for the tasks we consider, virtually all algorithms in the research literature can be described as combinations of a small number of operators that perform basic functions. Our first contribution is to abstract and unify key subroutines into a small set of operator classes in $\epsilon$KTELO– tranformations, query selection, partition selection, measurement and inference. Different algorithms differ in (a) the sequence in which these operations are performed on the data, and (b) the specific implementation of operations from these classes. In our system, differentially private programs are described as *plans* over a high level library of *operator* implementations supported by $\epsilon$KTELO. Plans described in $\epsilon$KTELO are expressive enough to reimplement all state-of-the-art algorithms from DPBench [15].

Second, if operator implementations are vetted to satisfy differential privacy, then every plan implemented in $\epsilon$KTELO comes with a proof of privacy. This proof requires a non-trivial extension of a formal analysis of a past framework [9]. This relieves the algorithm designer of the burden of proving their programs are private. By isolating privacy critical functions in operators, $\epsilon$KTELO reduces the amount of code that needs to be verified for privacy. In fact, in future work, we hope to implement operators in $\epsilon$KTELO using programming frameworks like LightDP to eliminate this burden too.

Third, the operator-based approach to implementing differentially private programs has the following benefits:

- *Modularity:* $\epsilon$KTELO enables code reuse as the same operator can be used in multiple algorithms. This helps safety, as there is less code to verify the correctness of an implementation, and amplifies innovation, as any improvement to an operator is inherited by all plans containing it.
- *Transparency:* By expressing algorithms as plans with operators from operator classes, differences/similarities of competing algorithms can be discerned. It also makes it easier to explore algorithm modifications. Further, it is possible to identify general rules for restructuring plans (like heuristics in query optimizers).
- *Flexibility:* The practitioner can now use existing operators from different algorithms and recombine them in arbitrary ways – allowing them to invent new algorithms that borrow the best ideas from the state-of-art – without the need for privacy analysis.

Fourth, as testament to benefits of $\epsilon$KTELO, we introduce three improvements to the state-of-art, which we believe would have been difficult to identify without the $\epsilon$KTELO framework. These improvements, which may be of independent interest, are:

- a general-purpose, efficient and scalable inference engine that subsumes customized inference subroutines from the literature;
- a new dimensionality reduction operator that is applicable to any plan that answers a workload of linear counting queries, and can reduce error by as much as 3× and runtime by as much as 5×;
- a new algorithm that, when expressed as a plan in $\epsilon$KTELO, looks similar to the MWEM algorithm [14] but with a few key operators replaced, which empirically lowers error up to 8×.

Finally, we demonstrate the flexibility of $\epsilon$KTELO through two case studies. For a use-case of releasing Census data tabulations, we define a new algorithm that can offer a 10× improvement over the best competitor from the literature. For building a private classifier, we used $\epsilon$KTELO to design algorithms that beat all available baselines.

We provide an overview of $\epsilon$KTELO and highlight its design in the next section. After providing background in Sec. 3, we describe the system fully in Sec. 4. We show the expressiveness of $\epsilon$KTELO plans in Sec. 5 by re-implementing existing algorithms. Algorithmic innovations are described in Sec. 6 and cases studies are examined in Sec. 7. The experimental evaluation of $\epsilon$KTELO is provided in Sec. 8 and we discuss related work and conclude in Secs. 9 and 10. The appendix includes proofs of theorems, additional algorithm background, and detailed plan descriptions.

## 2 OVERVIEW AND DESIGN PRINCIPLES

In this section we provide an overview of $\epsilon$KTELO by presenting an example algorithm written in the framework. Then we discuss the principles guiding the design of $\epsilon$KTELO.

### 2.1 An example plan: CDF estimation

In $\epsilon$KTELO, differentially private algorithms are described using *plans* composed over a rich library of *operators*. Most of the plans described in this paper are linear sequences of operators, but $\epsilon$KTELO also supports plans with iteration, recursion, and branching. Operators supported by $\epsilon$KTELO perform a well defined task and typically

**Algorithm 1** ϵKTELO CDF Estimator

```
1:  $D \leftarrow$ PROTECTED(source_uri)                              ▷ Init
2:  $D \leftarrow$ WHERE($D$, sex == 'M' AND age $\in [30, 39]$)        ▷ Transform
3:  $D \leftarrow$ SELECT(salary)                                     ▷ Transform
4:  $\mathbf{x} \leftarrow$ T-VECTORIZE($D$)                           ▷ Transform
5:  $\mathbf{P} \leftarrow$ AHPPARTITION ($\mathbf{x}$, $\epsilon/2$)   ▷ Partition Select
6:  $\bar{\mathbf{x}} \leftarrow$ V-REDUCEBYPARTITION ($\mathbf{x}$, $\mathbf{P}$)   ▷ Transform
7:  $\mathbf{M} \leftarrow$ IDENTITY($|\bar{\mathbf{x}}|$)              ▷ Query Select
8:  $\mathbf{y} \leftarrow$ VECLAPLACE($\bar{\mathbf{x}}$, $\mathbf{M}$, $\epsilon/2$)   ▷ Query
9:  $\hat{\mathbf{x}} \leftarrow$ NNLS($\mathbf{P}$, $\mathbf{y}$)      ▷ Inference
10: $\mathbf{W_{pre}} \leftarrow$ PREFIX($|\mathbf{x}|$)                ▷ Query Select
11: return $\mathbf{W_{pre}} \cdot \hat{\mathbf{x}}$                    ▷ Output
```

capture a key algorithm design idea from the state-of-the-art. Each operator belongs to one of five *operator classes* based on its input-output specification. These are: (a) transformation, (b) query, (c) inference, (d) query selection, and (e) partition selection. Operators are fully described in Sec. 4 and listed in Fig. 1.

We begin by describing an example ϵKTELO plan and use it to introduce the different operator classes. Algorithm 1 shows the pseudocode for a plan authored in ϵKTELO, which takes as input a table $D$ with schema [Age, Gender, Salary] and outputs the differentially private estimate of the empirical cumulative distribution function (CDF) of the Salary attribute, for males in their 30's. The plan is fairly sophisticated and it works in multiple steps. First the plan uses *transformation* operators on the input table $D$ to filter out records that do not correspond to males in their 30's (Line 2), selecting only the salary attribute (Line 3). Then it uses another transformation operator to construct a vector of counts $\mathbf{x}$ that contains one entry for each value of salary. $\mathbf{x}[i]$ represents the number of rows in the input (in this case males in their 30's) with salary equal to $i$.

Before adding noise to this histogram, the plan uses a *partition selection* operator, AHPPARTITION (Line 5). Operators in this class choose a partition of the data vector which is later used in a transformation. AHPPARTITION uses the sensitive data to identify a partition $\mathbf{P}$ of the counts in $\mathbf{x}$ such that counts within a partition group are close. Since AHPPARTITION uses the input data, it expends part of the privacy budget (in this case $\epsilon/2$). AHPPARTITION is a key subroutine in AHP [37], which was shown to have state-of-the-art performance for histogram estimation [15].

Next the plan uses V-REDUCEBYPARTITION (Line 6), another transformation operator on $\mathbf{x}$, to apply the partition $\mathbf{P}$ computed by AHPPARTITION. This results in a new reduced vector $\bar{\mathbf{x}}$ that contains one entry for each partition group in $\mathbf{P}$ and the entry is computed by adding up counts within each group.

The plan now specifies a set of measurement queries $\mathbf{M}$ on $\bar{\mathbf{x}}$ using the IDENTITY *query selection* operator (Line 7). The identity matrix corresponds to querying all the entries in $\bar{\mathbf{x}}$ (since $\mathbf{M} \cdot \bar{\mathbf{x}} = \bar{\mathbf{x}}$). Query selection operators do not answer any query, but rather specify which queries should be estimated. (This is analogous to how partition selection operators only select a partition but do not apply it.) Next, VECTOR LAPLACE returns differentially private answers to all the queries in $\mathbf{M}$. It does so by automatically calculating the sensitivity of the vectorized queries – which depends on all upstream data transformations – and then using the standard Laplace mechanism (Line 8) to add noise. This operator consumes the remainder of the privacy budget (again $\epsilon/2$).

So far the plan has computed an estimated histogram of partition group counts $\mathbf{y}$, while our goal is to return the empirical CDF on the original salary domain. Hence, the plan uses the noisy counts on the reduced domain $\mathbf{y}$ to infer non-negative counts in the original vector space of $\mathbf{x}$ by invoking an *inference* operator NNLS (short for non-negative least squares) (Line 9). NNLS$(\mathbf{P}, \mathbf{y})$ finds a solution, $\hat{\mathbf{x}}$, to the problem $\mathbf{P}\hat{\mathbf{x}} = \mathbf{y}$, such that all the entries in $\hat{\mathbf{x}}$ are non-negative. Finally, the plan constructs the set of queries, $\mathbf{W_{pre}}$, needed to compute the empirical CDF (a lower triangular $k \times k$ matrix representing the prefix sums) by invoking the query selection operator PREFIX($k$) (Line 10), and returns the output $\mathbf{W_{pre}} \cdot \hat{\mathbf{x}}$ (Line 11).

## 2.2 ϵKTELO design principles

The design of ϵKTELO is guided by the following principles. With each principle, we include references to future sections of the paper where the consequent benefits are demonstrated.

***Expressiveness*** ϵKTELO is designed to be expressive, meaning that a wide variety of state-of-the-art algorithms can be written succinctly as ϵKTELO plans. To ensure expressiveness, we carefully designed a foundational set of operator classes that cover features commonly used by leading differentially private algorithms. We demonstrate the expressiveness of our operators by showing in Sec. 5 that the algorithms from the recent DPBench benchmark [15] can be readily re-implemented in ϵKTELO.

***Privacy "for free"*** ϵKTELO is designed so that any plan written in ϵKTELO automatically satisfies differential privacy. The formal statement of this privacy property is in Sec. 4.3. This means that plan authors are not burdened with writing privacy proofs for each algorithm they write. Furthermore, when invoking privacy-critical operators that take noisy measurements of the data, the magnitude of the noise is automatically calibrated. As described in Sec. 4, this requires tracking all data transformations and measurements and using this information to handle each new measurement request.

***Reduced privacy verification effort*** Ensuring that an algorithm implementation satisfies differential privacy requires verifying that it matches the algorithm specification. The design of ϵKTELO reduces the amount of code that must be vetted each time an algorithm is crafted. First, since an algorithm is expressed as a plan, and all plans automatically satisfy differential privacy, the code that must be vetted is solely the individual operators. Second, each operator needs to be vetted only once but may be reused across multiple algorithms. Finally, it is not necessary to vet every operator: only the privacy-critical operators (as shown in Sec. 4, ϵKTELO mandates a clear distinction between privacy-critical and non-private operators). The end result means that verifying the privacy of an algorithm requires checking fewer lines of code. In Sec. 5, we compare the verification effort to vet the DPBench codebase[1] against the effort required to vet these algorithms when expressed as plans in ϵKTELO.

***Transparency*** In ϵKTELO, all algorithms are expressed in the same form: each is a plan, consisting a sequence of operators where each operator is selected from a class of operators based on common functionality. This facilitates algorithm comparison and makes differences between algorithms more apparent. In Sec. 5, we summarize the plan signatures of a number of state-of-the-art algorithms (pictured in Fig. 2). These plan signatures reveal similarities and

[1] Available at: https://github.com/dpcomp-org/dpcomp_core

common idioms in existing algorithms. These are difficult to discover from the research literature or through code inspection.

We believe that $\epsilon$KTELO, by supporting the design principles described above, provides an improved platform for designing and deploying differentially private algorithms.

# 3 BACKGROUND

The input to $\epsilon$KTELO is a database instance of a single-relation schema $T(A_1, A_2, \ldots, A_\ell)$. Each attribute $A_i$ is assumed to be discrete (or suitably discretized). A *condition formula*, $\phi$, is a Boolean condition that can be evaluated on any tuple of $T$. We use $\phi(T)$ to denote the number of tuples in $T$ for which $\phi$ is true. A number of operators in $\epsilon$KTELO answer linear queries over the table. A linear query is the linear combination of any finite set of condition counts:

*Definition 3.1 (Linear counting query (declarative)).* A linear query $q$ on $T$ is defined by conditions $\phi_1 \ldots \phi_k$ and coefficients $c_1 \ldots c_k \in \mathbb{R}$ and returns $q(T) = c_1\phi_1(T) + \cdots + c_k\phi_k(T)$.

It is common to consider a vector representation of the database, denoted $\mathbf{x} = [x_1 \ldots x_n]$, where $x_i$ is equal to the number of tuples of type $i$ for each possible tuple type in the relational domain of $T$. The size of this vector, $n$, is the product of the attribute domains. Then it follows that any linear counting query has an equivalent representation as a vector of $n$ coefficients, and can be evaluated by taking a dot product with $\mathbf{x}$. Abusing notation slightly, let $\phi(i) = 1$ if $\phi$ evaluates to true for the tuple type $i$ and 0 otherwise.

*Definition 3.2 (Linear counting query (vector)).* For a linear query $q$ defined by $\phi_1 \ldots \phi_k$ and $c_1 \ldots c_k$, its equivalent vector form is $\vec{q} = [q_1 \ldots q_n]$ where $q_i = c_1\phi_1(i) + \cdots + c_k\phi_k(i)$. The evaluation of the linear query is $\vec{q} \cdot \mathbf{x}$, where $\mathbf{x}$ is vector representation of $T$.

In the sequel, we will use vectorized representations of the data frequently. We refer to the *domain* as the size of $\mathbf{x}$, the vectorized table. This vector is sometimes large and a number of methods for avoiding its materialization are discussed later.

Let $T$ and $T'$ denote two tables of the same schema, and let $T \oplus T' = (T - T') \cup (T' - T)$ denote the symmetric difference between the two tables. We say that $T$ and $T'$ are neighbors if $|T \oplus T'| = 1$.

*Definition 3.3 (Differential Privacy [7]).* A randomized algorithm $\mathcal{A}$ is $\epsilon$-differentially private if for any two instances $T$, $T'$ such that $|T \oplus T'| = 1$, and any subset of outputs $S \subseteq Range(\mathcal{A})$,

$$Pr[\mathcal{A}(T) \in S] \leq \exp(\epsilon) \times Pr[\mathcal{A}(T') \in S]$$

Differentially private algorithms can be composed with each other and other algorithms using composition rules, such as sequential and parallel composition [24] and post-processing [8]. Let $f$ be a function on tables that outputs real numbers. The *sensitivity* of the function is defined as: $max_{|T \oplus T'|=1}|f(T) - f(T')|$.

*Definition 3.4 (Stability).* Let $g$ be a transformation function that takes a data source (table or vector) as input and returns a new data source (of the same type) as output. For any pair of sources $S$ and $S'$ let $|S \oplus S'|$ denote the distance between sources. If the sources are both tables, then this distance is the size of the symmetric difference; if the sources are both vectors, then this distance is the $L_1$ norm; if the sources are of mixed type, it's undefined. Then the stability of $g$ is: $\max_{S,S':|S \oplus S'|=1} |g(S) \oplus g(S')|$. When the stability of $g$ is at most $c$ for some constant $c$, we say that $g$ is $c$-stable.

# 4 OPERATOR FRAMEWORK

This section describes the components of $\epsilon$KTELO: the execution environment (Sec. 4.1), which consists of an untrusted client space and a protected kernel, where the private data is located; the operators (Sec. 4.2), which are grouped into classes based on their functionality and assigned one of three types based on their interactions with the protected kernel; and finally, the formal privacy guarantee (Sec. 4.3).

## 4.1 Protected Kernel and Client Space

Recall that a private computation is defined to be a *plan* consisting of a sequence of *operators*. Plans run in an unprotected *client* space. All interactions with the private data are mediated by the *protected kernel*. The protected kernel encapsulates protected data sources and ensures that any sequence of operators satisfies formal privacy properties. The distinction between the client space and the protected kernel is a fundamental one in $\epsilon$KTELO. It allows authors to write plans that consist of operator calls embedded in otherwise arbitrary code (which may freely include conditionals, loops, recursion, etc.).

The protected kernel is initialized by specifying a single protected data object—an input table $T$—and a global privacy budget, $\epsilon$. Note that requests for data transformations may cause the protected kernel to derive additional data sources. Thus, the protected kernel maintains a *data source environment*, which consists of a mapping between data source variables, which are exposed to the client, and the protected data objects, which are kept private. In addition, the data source environment tracks the transformation lineage of each data source. Associated with each data source is a stability constant, which records the stability of the transformation that produced the source (defined in Sec. 3). Note that in describing operators below, we speak informally of operators having data sources as inputs and outputs rather than data source *variables*. A layer of indirection is always maintained in the implementation but sometimes elided in our descriptions to simplify the presentation.

## 4.2 Operators

Operators are organized into classes based on their functionality. We also assign operators to one of three types, based on their interaction with the protected kernel. The first type is a **Private** operator, which requests that the protected kernel perform some action on the private data (e.g., a transformation) but receives only an acknowledgement that the operation has been performed. The second type is a **Private→Public** operator, which receives information about the private data (e.g., a measurement) and thus consumes privacy budget. The last type is a **Public** operator, which does not interact with the protected kernel at all and can be executed entirely in client space. An example of a public operator would be the inference operator LEAST SQUARES (see Sec. 4.2.5), which performs a computation on the noisy measurements received from the protected kernel.

A full list of operators, and some of the plans they support, is shown in Fig. 1 and Fig. 2, respectively. In Fig. 1, operators are arranged into classes and color-coded by type, which illustrates that operator type and class are orthogonal dimensions. Next we describe in detail the operators and operator classes.

### 4.2.1 Transformation Operators.

Transformation operators take as input a data source variable (either a table or a vector) and

| Transform | |
|---|---|
| TV | T-Vectorize |
| TP | V-SplitByPartition |
| TR | V-ReduceByPartition |

| Inference | |
|---|---|
| LS | Least squares |
| NLS | Nneg Least squares |
| MW | Mult Weights |
| HR | Thresholding |

| Partition selection | |
|---|---|
| PA | AHPpartition |
| PG | Grid |
| PD | Dawa |
| PW | Workload-based |
| PS | Stripe(attr) |
| PM | Marginal(attr) |

| Query | |
|---|---|
| LM | Vector Laplace |

| Query selection | |
|---|---|
| SI | Identity |
| ST | Total |
| SP | Privelet |
| SH2 | H2 |
| SHB | HB |
| SG | Greedy-H |
| SU | UniformGrid |
| SA | AdaptiveGrids |
| SQ | Quadtree |
| SW | Worst-approx |
| SPB | PrivBayes select |

**Figure 1: The operators currently implemented in $\epsilon$KTELO. Private operators are red, Private→Public operators are orange, and Public operators are green.**

output a transformed data source (again, either a table or vector). Transformation operators modify the data held in the kernel, without returning answers. So while they do not expend privacy budget, they can affect the privacy analysis through their *stability* (Sec. 3). Every transformation in $\epsilon$KTELO has a well-established stability.

*Table Transformations* $\epsilon$KTELO supports table transformations Select, Where, SplitByPartition , and GroupBy. Their stabilities are 1, 1, 1 and 2, respectively. The definitions of the operators are nearly identical to those described in PINQ [24] and are not repeated here. Since $\epsilon$KTELO currently handles only programs that use linear queries on single tables, the Join operator is not yet supported.

*Vectorization* All of the plans in $\epsilon$KTELO start with table transformations and typically transform the resulting table into a vector using T-Vectorize (and all later operations happen on vectors).

The T-Vectorize operator is a transformation operator that takes as input a table $T$ and outputs a vector $\mathbf{x}$ that has as many cells as the number of elements in the table's domain (recall the discussion of domain Sec. 3). Each cell in $\mathbf{x}$ represents the number of records in the table that correspond to the domain element encoded by the cell. T-Vectorize is a 1-stable transformation.

The vectorize operation can significantly impact the performance of the code, especially in high-dimensional cases, as we represent one cell per element in the domain. This is the reason we allow for table transformations to reduce the domain size before running T-Vectorize. One of the primary reasons for working with the vector representation is to allow for inference operators downstream. Once in vector form, the data can be further transformed as described next.

*Vector Transformations* $\epsilon$KTELO supports transformations on vector data sources. Each vector transformation takes as input a vector $\mathbf{x}$ and a matrix $\mathbf{M}$ and produces a vector $\mathbf{x}' = \mathbf{M}\mathbf{x}$. The linearity of vector transformations is an important feature that is leveraged by downstream inference operators. The stability of vector transformations is equal to the largest $L_1$ column norm of $\mathbf{M}$.

The V-ReduceByPartition operator is a 1-stable vector transformation operator that reduces the dimensionality of the data vector $\mathbf{x}$ by eliminating cells from $\mathbf{x}$ or grouping together cells in $\mathbf{x}$. Such transformations are useful to (a) filter out parts of the domain that are uninteresting for the analyst, (b) reduce the size of the $\mathbf{x}$ vector so that algorithm performance can be improved, and (c) reduce the number of cells in $\mathbf{x}$ so that the amount of noise added by measurement operators is reduced.

| ID | Cite | Algorithm name | Plan signature |
|---|---|---|---|
| 1 | [8] | Identity | SI LM |
| 2 | [39] | Privelet | SP LM LS |
| 3 | [17] | Hierarchical (H2) | SH2 LM LS |
| 4 | [34] | Hierarchical Opt (HB) | SHB LM LS |
| 5 | [22] | Greedy-H | SG LM LS |
| 6 | - | Uniform | ST LM LS |
| 7 | [15] | MWEM | I:( SW LM MW ) |
| 8 | [42] | AHP | PA TR SI LM LS |
| 9 | [22] | DAWA | PD TR SG LM LS |
| 10 | [6] | Quadtree | SQ LM LS |
| 11 | [33] | UniformGrid | SU LM LS |
| 12 | [33] | AdaptiveGrid | SU LM LS TP[ SA LM] LS |
| 13 | NEW | DAWA-Striped | PS TP[ PD TR SG LM] LS |
| 14 | NEW | HB-Striped | PS TP[ SHB LM] LS |
| 15 | NEW | PrivBayesLS | SPB LM LS |
| 16 | NEW | MWEM variant b | I:( SW SH2 LM MW ) |
| 17 | NEW | MWEM variant c | I:( SW LM NLS ) |
| 18 | NEW | MWEM variant d | I:( SW SH2 LM NLS ) |

**Figure 2: The high-level signatures of plans implemented in $\epsilon$KTELO (referenced in the paper by ID). All plans begin with a vectorize transformation, omitted for readability. We also omit parameters of operators, including $\epsilon$ budget shares. I(*subplan*) refers to iteration of a subplan and TP[*subplan*] means that *subplan* is executed on each partition produced by TP .**

V-ReduceByPartition takes as input a partition defining a grouping of the cells in the $\mathbf{x}$. It can be carried out by representing the partition as a $(p \times n)$ matrix $\mathbf{P}$ where $n$ is the number of cells in $\mathbf{x}$, $p$ is the number of groups in the partition, and $\mathbf{P}[i, j] = 1$ if cell $j$ in $\mathbf{x}$ is mapped to group $i$, and 0 otherwise.

The V-SplitByPartition operator is the vector analogue of the tabular SplitByPartition operator. It takes as input a partition and splits the data vector $\mathbf{x}$ into $k$ vectors, $\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(k)}$, one for each group in the partition, each representing a disjoint subset of the original domain. This operator allows us to create different subplans for disjoint parts of the domain. This is a 1-stable vector transform. (Note: V-SplitByPartition can be expressed as $k$ linear transforms with matrices that select the appropriate elements of the domain for each partition.)

### 4.2.2 Query Operators.

Query operators are responsible for computing noisy answers to queries on a particular data source. Since answers are returned, query operators necessarily expend privacy budget. Query operators take a data source variable and $\epsilon$ as input.

For tables, the NoisyCount operator takes as input a table $D$ and $\epsilon$ and returns $|D| + \eta$, where $\eta$ is drawn from the Laplace distribution with scale $1/\epsilon$. For vectors, the Vector Laplace operator takes as input a vector $\mathbf{x}$, epsilon, and a set of linear counting queries $\mathbf{M}$ represented in matrix form. Let $\mathbf{M}$ be a matrix of size $(m \times n)$. Vector Laplace returns $\mathbf{M}\mathbf{x} + \frac{\sigma(\mathbf{M})}{\epsilon} b$ where $b$ is a vector of $m$ independently drawn Laplace random variables with scale 1 and $\sigma(\mathbf{M})$ is the maximum $L_1$ norm of the columns of $\mathbf{M}$.

For both query operators, it is easy to show they satisfy $\epsilon$-differential privacy with respect their data source input [23, 24]. Note, however, in the case when the source is derived from other data sources through transformation operators, the total privacy loss could be higher. The cumulative privacy loss depends on the stability of the transformations and is tracked by the protected kernel.

*4.2.3 Query Selection Operators.* Since each query operation consumes privacy budget, the plan author must be judicious about what queries are being asked. Recent privacy work has shown that if the plan author's goal is to answer a *workload* of queries, simply asking these queries directly can lead to sub-optimal accuracy (e.g., when workload queries ask about overlapping regions of the domain). Instead, higher accuracy can be achieved by designing a *query strategy*, a collection of queries whose answers can be used to reconstruct answers to the workload. This approach was formalized by the matrix mechanism [23], and has been a key idea in many state-of-the-art algorithms [6, 16, 20, 22, 29, 32].

A query selection operator is distinguished by its output type: all such operators output a set of linear counting queries **M** represented in matrix form (i.e., the matrix input to the VECTOR LAPLACE operator described above). As Fig. 1 indicates, $\epsilon$KTELO supports a large number of query selection operators, most of which are extracted from algorithms proposed in the literature.

While these operators agree in terms of their output, they vary considerably in terms of their inputs: some employ fixed strategies that depend only on the size of **x** (e.g. IDENTITY and PREFIX in Algorithm 1), some adapt to the workload (e.g., GREEDY-H), some depend on previous measurements (e.g., ADAPTIVEGRIDS), etc.

Most query selection operators only rely on non-private information (domain size, workload) and therefore are of Public type. But there are a few that consult the private data, and thus have the Private→Public type. For example, WORST-APPROX is an operator that picks the query from a workload that is the worst approximated by a current estimate of the data. Such an operator is used by iterative algorithms like MWEM [14]. Another is PRIVBAYES SELECT, an operator that privately constructs a Bayes net over the attributes of the data source, and then returns a matrix corresponding to the sufficient statistics for fitting the parameters of Bayes net. This was used as a subroutine in PrivBayes [35].

*4.2.4 Partition Selection Operators.* Partition selection operators compute a matrix **P** which can serve as the input to the V-REDUCEBYPARTITION and V-SPLITBYPARTITION operators described earlier. Of course the matrix **P** must be appropriately structured to be a valid partition of **x**.

This is an important kind of operator since much of recent innovation into state-of-the-art algorithms for answering histograms and range queries has used partitions to either (1) reduce the domain size of the data vector by grouping together cells with similar counts, or (2) split the data vector into smaller vectors and leverage the parallel composition of differential privacy to process each subset of the domain independently. $\epsilon$KTELO includes partition selection operators AHPPARTITION and DAWA which are subroutines from the AHP [37] and DAWA [20] algorithms, respectively. Both of these operators are data adaptive, and hence are Private→Public. This paper also introduces a new partition selection operators, WORKLOAD-BASED and STRIPE, described in Secs. 6.2 and 7.1 respectively.

*4.2.5 Inference Operators.* An inference operator derives new estimates to queries based on the history of transformations and query answers. Inference operators never use the input data directly and hence are Public. Plans typically terminate with a call to an inference operator to estimate a final set of query answers reflecting all available information computed during execution of the plan. Some plans may also perform inference as the plan executes.

Ideally, an inference method should: (i) properly account for measurements with unequal noise; (ii) support inference over incomplete measurements (in which derived answers are not completely determined by available measurements); (iii) should incorporate all available information (including a prior or constraint on the input dataset); and lastly, (iv) inference should efficiently scale to large domains. Many versions of inference have been considered in the literature [2, 14, 16, 19, 21, 26, 29, 31, 37] but none meet all of the objectives above. $\epsilon$KTELO currently supports multiple inference methods, in part to support algorithms from past work and in part to offer necessary tradeoffs among the properties above.

All the inference operators supported in $\epsilon$KTELO take as input a set of queries, represented as a matrix **M**, and noisy answers to these queries, denoted **y**. The output inference is a data vector $\hat{\mathbf{x}}$ that best fits the noisy answers—i.e., an $\hat{\mathbf{x}}$ such that $\mathbf{M}\hat{\mathbf{x}} \approx \mathbf{y}$. The estimated $\hat{\mathbf{x}}$ can then be used to derive an estimate of any linear query $q$ by computing $q \cdot \hat{\mathbf{x}}$. The inference operator may optionally take as input a set of weights, one per query (row) in **M** to account for queries with different noise scales.

$\epsilon$KTELO supports two variants of least squares inference, the most widely used form of inference in the current literature [16, 21, 29]. $\epsilon$KTELO extends these methods and formulates them as general operators, allowing us to replicate past algorithms, and consider new forms of inference that support constraints. The first variant solves a classical least squares problem:

*Definition 4.1 (Ordinary least squares (LS)).* Given scaled query matrix **M** and answer **y**, the least squares estimate of **x** is:

$$\hat{\mathbf{x}} = \underset{x \in \mathbb{R}^n}{\arg\min} \|\mathbf{M}\mathbf{x} - \mathbf{y}\|_2 \tag{1}$$

Our second variant imposes a non-negativity constraint on $\hat{\mathbf{x}}$:

*Definition 4.2 (Non-negative least squares (NNLS)).* Given scaled query matrix **M** and answer vector **y**, the non-negative least squares estimate of **x** is:

$$\hat{\mathbf{x}} = \underset{x \geq 0}{\arg\min} \|\mathbf{M}\mathbf{x} - \mathbf{y}\|_2 \tag{2}$$

These inference methods can also support some forms of prior information, particularly if it can be represented as a linear query. For example, if the total number of records in the input table is publicly known, or other special queries have publicly available answers, they can be added as "noisy" answers with negligible noise scale and they will naturally incorporated into the inference process and the derivation of new query estimates.

We also support an inference method based on a multiplicative weights update rule, which is used in the MWEM [14] algorithm. This inference algorithm is closely related to the principle of maximum entropy, and is especially effective when one has measured an incomplete set of queries.

*Defining inference under vector transformations* Recall that in the discussion above we describe inference as operating on a single vector **x** with a corresponding query matrix **M**. However, plans can include an arbitrary combination of vector transformations, followed by query operators, resulting in a collection of query answers defined over various vector representations of the data. $\epsilon$KTELO handles this by taking advantage of the structure of vector transformations and query operators, both of which perform linear transformations, therefore making it possible to map measured queries back on to the original domain (i.e., a vector produced by the VECTORIZE operation) and perform inference there. This allows for the most complete form of inference but other alternatives are conceivable, for example by performing inference locally on transformed vectors and combining inferred queries. This might have efficiency advantages, but would likely sacrifice accuracy, and is left for future investigation.

REMARK 1. *The operators described above can capture a large class of algorithms from the literature which were designed for answering sets of linear queries over modest domain sizes. Yet many features of* ϵktelo *are general and, in conjunction with new operators,* ϵktelo *could support a wider array of tasks, including non-linear queries or larger domain sizes. In Sec. 10, we briefly suggest future directions for expanding the tasks implemented in* ϵktelo.

## 4.3 Privacy Guarantee

In this section, we state the privacy guarantee offered by ϵKTELO. Informally, ϵKTELO will ensure that if the system's protected kernel is initialized with a source database $T$ and a privacy budget $\epsilon$, then any plan (chosen by the client) will satisfy $\epsilon$-differential privacy with respect to $T$. Note that if the client exhausts the privacy budget, subsequent calls to Private→Public operators will return an exception, indicating that such operations are not permitted. Importantly, the system is designed so that an exception itself does not leak sensitive information – i.e., the decision to return an exception does not depend on the private state.

A *transcript* is a sequence of operator calls and their responses. Formally, let $r_k = \langle op_1, a_1, op_2, a_2, \ldots, op_k, a_k \rangle$ denote a length $k$ sequence where $op_i$ is an operator call and $a_i$ the response. We assume that the value of $op_i$ is a deterministic function of $a_1, \ldots, a_{i-1}$; however, the claim can be extended to support randomized client code. We use $R_k = r_k$ to denote the event that the first $k$ operations results in transcript $r_k$. Let $\mathcal{R}_k$ denote the set of all possible transcripts of length $k$. Note that because we assume that all Private→Public operators output values from an arbitrary, but finite set, the set of transcripts is finite. Let $P(R_k = r_k \mid \textbf{Init}(T, \epsilon_{tot}))$ denote the conditional probability of event $R_k = r_k$ given that the system was initialized with input $T$ and a privacy budget of $\epsilon_{tot}$.

THEOREM 4.3 (PRIVACY OF ϵKTELO PLANS). *Let $T, T'$ be any two instances such that $|T \oplus T'| = 1$. For all $k \in \mathbb{N}^+$ and $r_k \in \mathcal{R}_k$,*

$$P(R_k = r_k \mid \textbf{Init}(T, \epsilon_{tot})) \leq \exp(\epsilon_{tot}) \times P(R_k = r_k \mid \textbf{Init}(T', \epsilon_{tot})).$$

The proof of Theorem 4.3 appears in Appendix A.4. It extends the proof in [9] to support the V-SPLITBYPARTITION operator.

While the system ensures differential privacy, it is conceivable that private information could be leaked through side-channel attacks (e.g., timing attacks). Privacy engineers who design operators are responsible for protecting against such an attack; a careful analysis of this issue is beyond the scope of this paper.

# 5 EXPRESSING KNOWN ALGORITHMS

To highlight the expressiveness of ϵKTELO, we re-implemented state-of-the-art algorithms as ϵKTELO plans. We examined 12 differentially private algorithms for answering low dimensional counting queries that were deemed competitive[2] in a recent benchmark study [15]. Plans numbered 1 through 12 in Fig. 2 summarize the plan signatures of these algorithm.

The process of re-implementing in ϵKTELO this seemingly diverse set of algorithms consisted of breaking the algorithms down into key subroutines and translating them into operators. The translation strategy typically falls into one of three categories.

The first translation strategy was to identify specific implementations of fairly common differentially private operations and replace them with a single unified general-purpose operator in ϵKTELO. For instance, the Laplace mechanism (LM), which adds noise drawn from the Laplace distribution, appears in *every one* of the 12 algorithms. Noise addition can be implemented in a number of ways (e.g., calling a function in the numpy.random package, taking the difference of exponential random variables, etc.). In ϵKTELO, all these plans call the same VECTOR LAPLACE operator with a single unified sensitivity calculation.

Another less obvious example of this translation is for subroutines that infer an estimate of $\mathbf{x}$ using noisy query answers. With the exception of IDENTITY (plan 1) and MWEM (plan 7), each of the algorithms uses instances of least squares inference, often customized to the structure of the noisy query answers. For instance, PRIVELET (plan 2) uses Haar wavelet reconstruction, hierarchical strategies like HB and DAWA (plans 4, 9) use a tree-based implementation of inference, and others like UNIFORM and AHP (plans 6, 8) use uniform expansion. We replaced each of these custom inference methods with a single general-purpose least squares inference operator. It would still be possible to implement a specialized inference operator in ϵKTELO that exploited particular properties of a query set, but we did not find this to be beneficial.

Our second translation strategy was to identify higher-level patterns that reflect design idioms that exist across multiple algorithms in literature. In these cases, we replace one or more subroutines in the original code with a sequence of operators that capture this idiom. For example, plans 2, 3, 4, 5, 6, 10, and 11 all consist of a sequence of three operators: Query selection, Query (LM), and Inference (LS), differing only in the method for Query selection. For other algorithms, this idiom reappears as a subroutine, as in plans 8 (AHP) and 9 (DAWA).

Finally, we were left with subroutines of algorithms that represented key intellectual advances in the differential privacy literature. These were ideas that made an algorithm distinctive and typically led to its state-of-the-art performance. For instance, in the DAWA algorithm (plan 9), the key innovation was a new partition method used for reduction that works by finding a grouping of the bins in a vector and required a novel proof of privacy. We encapsulate these subroutines as new operators in our framework (in the case above, we added partition selection operator, which we call DAWA and denote as PD in plan 9).

Once the necessary operators are implemented, the plan definition for an existing algorithm is typically a few lines of code responsible for combining operators and managing parameters. We performed extensive testing to confirm that reimplementations in ϵKTELO of existing algorithms provide statistically equivalent outputs.

[2] This is the subset of algorithms that offered the best accuracy for at least one of the input settings of the benchmark.

***Code reuse*** Once reformulated in $\epsilon$KTELO, nearly all algorithms use the VECTOR LAPLACE operator and least squares inference. This means that any improvements to either of these operators will be inherited by all the plans. We show such an example in Sec. 6.1.

***Reduced privacy verification effort*** Code reuse also reduces the number of critical operators that must be carefully vetted. The operators that require careful vetting are ones that consume the privacy budget, which are the Private→Public operators in Fig. 1. These are: VECTOR LAPLACE, the partition selection operators for both DAWA [20] and AHP [37], a query selection operator used by PrivBayes [35], and a query selection operator used by the MWEM [14] algorithm that privately derives the worst-currently-approximated workload query. In contrast, for the DPBench code base, the entire code has to be vetted to audit the use and management of the privacy budget. The end result is that verifying the privacy of an algorithm requires checking fewer lines of code. For example, to verify the QuadTree algorithm in the DPBench codebase requires checking 163 lines of code. However, with $\epsilon$KTELO, this only requires vetting the 30-line VECTOR LAPLACE operator. (Furthermore, by vetting just this one operator, we have effectively vetted 10 of the 18 algorithms in Fig. 2, since the only privacy sensitive operator these algorithms use is VECTOR LAPLACE.). When we consider all of the DPBench algorithms in Fig. 2, algorithms 1-12, verifying the DPBench implementation requires checking a total of 1837 lines of code while vetting all the privacy-critical operators in $\epsilon$KTELO requires checking 517 lines of code.

***Transparency*** As noted above, $\epsilon$KTELO plans make explicit the typical patterns that result in accurate differentially private algorithms. Moreover, $\epsilon$KTELO plans help clarify the distinctive ingredients of state-of-the-art algorithms. For instance, DAWA and AHP (plans 9 and 8 respectively in Fig. 2) have the same structure but differ only in two operators: partition selection and query selection.

# 6 ALGORITHMIC INNOVATIONS

In this section, we describe three algorithmic innovations. Each innovation is an instance of a *type* of innovation that is facilitated by the design of $\epsilon$KTELO. The first is an example of *operator inception*, which occurs when a new operator is proposed for an operator class; we introduce a general and highly scalable inference operator. The second is both a new operator and an example of *plan restructuring*, in which a plan is systematically restructured by applying a general design principle or heuristic rule. The operator is a partition selection operator, used for reduction, that minimizes the domain based on the workload in a way that we prove can never hurt error (but may improve it and reduce runtime). It can therefore be applied in any plan, potentially offering automatic improvements. The third innovation is an instance of *recombination*, in which the overall structure of an algorithm's plan stays the same, but some operator instances are substituted for alternatives within their respective classes. We use recombination to improve the MWEM algorithm [14].

## 6.1 Operator inception: new inference operators

The inference methods described in Sec. 4.2.5 require solving large least squares problems, as stated in Eq. (1). Recall that the input to inference is the set of queries in matrix form, denoted by $\mathbf{M}$, and the list of noisy answers $\mathbf{y}$. $\mathbf{M}$ is a $m \times n$ matrix where $n$ is the domain size of the input; $m$ may also be large, possibly larger than $n$, since it is not uncommon for $\mathbf{M}$ to contain a query for each cell of $\mathbf{x}$, in addition to other queries that aggregate elements of $\mathbf{x}$.

The solution to Eq. (1) is given by the solution to the normal equations $\mathbf{M}^T\mathbf{M}\hat{\mathbf{x}} = \mathbf{M}^T\mathbf{y}$. Assuming $\mathbf{M}^T\mathbf{M}$ is invertible, then the solution is unique and can be expressed as $\hat{\mathbf{x}} = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{y}$.

In practice, explicit matrix inversion is usually avoided, in favor of suitable factorizations of $\mathbf{M}$ (e.g., QR or SVD). The time complexity of such direct methods is still generally cubic in the domain size when $m = O(n)$. In practice we have found that the runtime of direct methods is unacceptable when $n$ is greater than about 5000.

An alternative approach to solving Eq. (1) is to use an iterative gradient-based method, which solves the normal equations by repeatedly computing matrix-vector products $\mathbf{M}\mathbf{v}$ and $\mathbf{M}^T\mathbf{v}$ until convergence. The time complexity of these methods is $O(kn^2)$ where $k$ is the number of iterations. In experiments we use a well-known iterative method, LSMR [11]. Empirically, we observe LMSR to converge in far fewer than $n$ iterations when $\mathbf{M}$ is well-conditioned, which is the case as long as the queries are not taken with vastly different noise scales, and thus we expect $k << n$.

The benefit of iterative methods is significant with standard (dense) representations of $\mathbf{M}$, but is even greater if the sparsity of $\mathbf{M}$ (i.e. few non-zero elements) is exploited. Letting $\text{nnz}(\mathbf{M})$ denote the number of non-zero entries in $\mathbf{M}$, the dot products $\mathbf{M}\mathbf{v}$ and $\mathbf{M}^T\mathbf{v}$ for the gradient computation can be evaluated in time $O(\text{nnz}(\mathbf{M}))$.

In $\epsilon$KTELO, query matrices tend to be sparse because the queries are typically being estimated using a noise addition mechanism (e.g. the Laplace mechanism). Noise is calibrated to the sensitivity of $\mathbf{M}$, which is measured as a column norm of the matrix. For example, well-known query sets based on wavelets [32] or binary trees [16] have $\text{nnz}(\mathbf{M}) = O(n \log_2 n)$. And an optimized hierarchical approach [29] found that higher branching factors lead to lower error, in which case $\text{nnz}(\mathbf{M}) < O(n \log_{16} n)$.

As a consequence, sparse matrix representations are unusually effective for accelerating inference, and assuming $\text{nnz}(\mathbf{M}) = O(n \log n)$, the overall time complexity is $\sim O(kn \log n)$. In Sec. 8 we show that using iterative least squares on sparse matrices we can scale inference to domains consisting of millions of cells while staying within modest runtime bounds.

REMARK 2 (IMPACT ON $\epsilon$KTELO). *Algorithms in prior work [16, 28, 29, 32] have used least squares inference on large domains by restricting the selection of queries, namely to those representing a set of hierarchical queries. This allows for inference in time linear in the domain size, avoiding the matrix representation of the queries. The approach above provides a much more general solution to the inference problem. This is critical to the success of* $\epsilon$ktelo*: it allows query selection operators to be freely designed and combined without restrictions on the structure of queries.*

## 6.2 Plan restructuring: workload-based reduction

Next we describe a method for reducing the representation of the $\mathbf{x}$ vector to precisely the elements required to correctly answer a given set of workload queries. This is a new partition selection operator, called workload-based partition selection, which can be used as input to a V-REDUCEBYPARTITION transformation of the input data, making all subsequent operations more efficient. In many cases, the goal of a differentially private algorithm (and the corresponding $\epsilon$KTELO plan) is to answer a workload of queries **M**. In such cases, we prove (Theorem A.3) that, under reasonable assumptions, workload based domain reduction can never hurt accuracy. We empirically show (Sec. 8.2) that it may offer significant improvement in both runtime and error. Thus adding this operator to *any* plan that answers a workload is a "pure win". This is an example of how $\epsilon$KTELO can magnify the impact of innovations to operators, especially when it is possible to prove properties about how they will impact plans.

For a workload **W** of linear queries, described on input vector **x**, it is often possible to define a reduction of **x**, to a smaller $\mathbf{x}'$, and appropriately transform the workload to $\mathbf{W}'$, so that all workload query answers are preserved, i.e. $\mathbf{Wx} = \mathbf{W}'\mathbf{x}'$. Intuitively, such a reduction is possible when a set of elements of **x** is not distinguished by the workload: each linear query in the workload either ignores it, or treats it in precisely the same way. In that case, that portion of the domain need not be represented by multiple cells, but instead by a single cell in a reduced data vector. It is in this sense that the reduction is lossless with respect to the workload. Following this intuition, the domain reduction can be computed from the matrix representation **W** of the workload by finding groups of identical columns: elements of these groups will be merged in **W** to get $\mathbf{W}'$ while the corresponding cells in **x** are summed.

*Example 6.1.* Consider a table with schema Census(age, sex, salary). If the workload consists of queries Q1$(salary \le 100K, sex = M)$ and Q2$(salary > 100K, sex = F)$ the workload only requires a data vector consisting of 2 cells. If the workload consists of all 1-way marginals then no workload-based data reduction is possible.

Note that calculating this partition only requires knowledge of the workload and is therefore done in the unprotected client space (and does not consume the privacy budget). The partition is then input to a V-REDUCEBYPARTITION transformation operator carried out by the protected kernel and its stability is 1. Due to space constraints we refer the reader to the Appendix A.1 for a technical description of this operator (Definition A.1), a proof that this operator does not lose any information for answering queries (Proposition A.2), a clever algorithm that uses randomized hashing for computing the transformation efficiently (Algorithm 2), and a proof that workload based domain reduction can be used in any workload-answering plan without a loss in accuracy (Theorem A.3).

### 6.3 Recombination: improving MWEM

Using $\epsilon$KTELO, we design new variants of the well-known *Multiplicative Weights Exponential Mechanism* (MWEM) [14] algorithm. MWEM repeatedly derives the worst-approximated workload query with respect to its current estimate of the data, then measures the selected query, and uses the multiplicative weights update rule to refine its estimate, often along with any past measurements taken. This repeats a number of times, determined by an input parameter.

When viewed as a plan in $\epsilon$KTELO, a deficiency of MWEM becomes apparent. Its query selection operator selects a *single* query to measure whereas most query selection operators select a set of queries such that the queries in the set measure disjoint partitions of the data. By the parallel composition property of differential privacy, measuring the entire set has the same privacy cost as asking any single query from the set. This means that MWEM could be measuring more than a single query per round (with no additional consumption of the privacy budget). To exploit this opportunity, we designed an augmented query selection operator that adds to the worst-approximated query by attempting to build a binary hierarchical set of queries over the rounds of the algorithm. In round one, it adds any unit length queries that do not intersect with the selected query. In round two, it adds length two queries, and so on.

Adding more measurements to MWEM has an undesirable side effect on runtime, however. Because it measures a much larger number of queries across rounds of the algorithm, and because the runtime of multiplicative weights inference scales with the number of measured queries, the inference step can be considerably slower. Thus, we also use *recombination* to replace it with a version of least-squares with a non-negativity constraint (NNLS) and incorporate a high-confidence estimate of the total which is assumed by the MWEM algorithm.

In total, we consider three MWEM variants: an alternative query selection operator (PLAN #16), an alternative inference operator (PLAN #17), and the addition of both alternative operators (PLAN #18). These are shown in Fig. 2 and evaluated in Sec. 8.

## 7 CASE STUDIES: $\epsilon$KTELO IN ACTION

In this section we consider two practical use-cases. While existing algorithms can be applied to these cases, particular applications often benefit from custom algorithm design. We show below how a plan author can construct novel plans that outperform existing solutions, solely using operators implemented in $\epsilon$KTELO.

### 7.1 Census case-study

The U.S. Census Bureau collects data about U.S. citizens and releases a wide variety of tabulations describing the demographic properties of individuals. We consider a subset of the (publicly released) March 2000 Current Population Survey. The data report on 49,436 heads-of-household describing their income, age (in years), race, marital status, and gender. We divide Income into 5000 uniform ranges from $(0, 750000)$, age in 5 uniform ranges from $(0, 100)$, and there are 7, 4 and 2 possible values for status, race and gender.

We author differentially private plans for answering a workload of queries similar to Census tabulations. This is challenging because the data domain is large and involves multiple dimensions. The workloads we consider are: (a) the Identity workload (or counts on the full domain of 1.4M cells), (b) a workload of all 2-way marginals (age × gender, race × status, and so on), and (c) a workload suggested by U.S. Census Bureau staff: Prefix(Income) which consists of all counting queries of the form (income $\in (0, i_{high})$, age=$a$, marital=$m$, race=$r$, gender=$g$) where $(0, i_{high})$ is an income range, and $a, m, r, g$ may be values from their resp. domains, or $<any>$.

There are few existing algorithms suitable for this task. We were unable to run the DAWA [20] algorithm directly on such a large domain, and, in addition, it was designed for 1d- and 2d- inputs. One of the few algorithms designed to scale to high dimensions is PrivBayes [35]. While not a workload-adaptive algorithm, PrivBayes generates synthetic data which can support the census workloads above. We use PrivBayes as a baseline and we use $\epsilon$KTELO to construct three new plans composed of operators in our library. The proposed plans are: HB-STRIPED (PLAN #14), DAWA-STRIPED (PLAN #13), and

PrivBayesLS (Plan #15). The first two "striped" plans showcase the ability to adapt lower dimensional techniques to a higher dimensional problem avoiding scalability issues. The third plan considers improving on PrivBayes by changing its inference step.

Both Hb-Striped and Dawa-Striped use the same plan structure: first they partition the full domain, then they execute subplans to select measurements for each partition, and lastly, given the measurement answers, they perform inference on the full domain and answer the workload queries. The partitioning of the initial step is done as follows: given a high dimensional dataset with $N$ attributes and an attribute $A$ of that domain, our partitions are parallel "stripes" of that domain for each fixed value of the rest of the $N - 1$ attributes, so that the measurements are essentially the one-dimensional histograms resulting from each stripe. In the case of Hb-Striped, the subplan executed on each partition is the Hb algorithm [29], which builds an optimized hierarchical set of queries, while in the case of the Dawa-Striped the subplan is DAWA algorithm [20]. Note that while the data-independent nature of the Hb subplan means that all the measurements from each stripe are the same, that is not the case with Dawa, which potentially selects different measurement queries for each stripe, depending on the local vector it sees. For our experiments, the attribute chosen was Income, and for Dawa-Striped we set the DAWA parameter $\rho$ to 0.25. Our final plan is a variant of PrivBayes in which we replace the original inference method with least squares, retaining the original PrivBayes query selection and query steps. We call this algorithm PrivBayesLS. Hb-Striped and PrivBayesLS are fully described in Appendix A.3.

We evaluate the error incurred by these plans in Sec. 8.1.1, and show that the best of our plans outperforms the state-of-the-art PrivBayes by at least 10× in terms of error.

## 7.2 Naive Bayes case study

We also demonstrate how $\epsilon$KTELO can be used for constructing a Naive Bayes classifier. To learn a NaiveBayes classifier that predicts a binary label attribute $Y$ using predictor variables $(X_1, \ldots, X_k)$ requires computing 2k+1 1d histograms: a histogram on $Y$, histogram on each $X_i$ conditioned on each value on $Y$. We design $\epsilon$KTELO plans to compute this workload of 2k+1 histograms, and use them to fit the classifier under the Multinomial statistical model [18].

We develop two new plans and compare them to two plans that correspond to algorithms considered in prior work. Workload represents the 2k+1 histograms as a matrix, and uses Vector Laplace to estimate the histogram counts. This corresponds to a technique proposed in the literature [5]. The other baseline is Identity (Plan 1): it estimates all point queries in the contingency table defined by the attributes, adds noise to it, and marginalizes the noisy contingency table to compute the histograms.

The first new plan is WorkloadLS which runs Workload followed by a least squares inference operator, which for this specific workload would make all histograms have consistent totals. Our second plan is called SelectLS (fully described in Appendix A.3) and selects a different algorithm (subplan) for estimating each of the histograms. SelectLS first runs 2k+1 domain reductions to compute 2k+1 vectors, one for each histogram. Then, for each vector, SelectLS uses a conditional statement to select between two subplans: if the domain size of the vector is less than 80, Identity is chosen, else a subplan that runs DAWA partition selection followed by Identity is chosen. We combine the answers from all subplans and use least squares inference jointly on all measurements. The inputs to the inference operator are the noisy answers and the workload of effective queries on the full domain. In Sec. 8.1.2 we show that our new plans not only outperforms existing plans, but also approach the accuracy of the non-private classifier in some cases.

**Table 1: Results on Census data; domain size 1,400,000; scale of error is indicated under each workload.**

| | Workload | | |
|---|---|---|---|
| | Identity | 2-way Marg. | Prefix (Income) |
| Algorithm | (1e−9) | (1e−7) | (1e−7) |
| Identity | 24.18 | 12.04 | 18.97 |
| PrivBayes | 92.61 | 161.90 | 381.97 |
| PrivbayesLS | 6.17 | 20.13 | 58.18 |
| Hb-Striped | 70.31 | 21.91 | 4.13 |
| Dawa-Striped | **3.43** | **1.96** | **2.50** |

# 8 EXPERIMENTAL EVALUATION

Our prototype implementation of $\epsilon$KTELO, including all algorithms and variants used below, consists of 3700 lines of Python code: 22% is the framework itself, while 62% consists of operator implementations. The remaining 16% are definitions of plans used in our experiments.

We evaluate the $\epsilon$KTELO framework in two ways. First, we report the results of using $\epsilon$KTELO in the case studies of Sec. 7. Then we do a focused evaluation of the impact of the three algorithmic innovations proposed in Section 6.

## 8.1 Case studies

*8.1.1 Census data analysis.* We consider the task of computing workloads inspired by Census tabulations (Sec. 7.1) and compare our three new plans PrivBayesLS, Hb-Striped, and DAWA-Striped with the baseline algorithms Identity (plan 1 in Fig. 2) and PrivBayes, our $\epsilon$KTELO reimplementation of a state-of-the-art algorithm for high dimensional data [35].

Table 1 presents the results for each of the workloads considered. We use scaled, per-query L2 error as our accuracy measure. First, we find that PrivBayes performs worse than Identity on all workloads. Interestingly, it is highly improved by our new plan PrivBayesLS that replaces its inference step with least squares. PrivBayes may be more suitable to input data with higher correlations between the attributes. Second, our striped plans Hb-Striped and Dawa-Striped offer significant improvements in error. Dawa-Striped is the best performer: the data-dependent nature of DAWA exploits uniform regions in the partitioned data vectors. This shows the benefit from $\epsilon$KTELO in allowing algorithm idioms designed for lower-dimensional data to be adapted to high dimensional problems.

*8.1.2 Naive Bayes classification.* We evaluate the performance of the Naive Bayes classifier on *Credit Default* [33], a credit card clients dataset which we use to predict whether a client will default on their payment or not. The data consists of 30k tuples and 24 attributes from which one is the target binary variable "Default" and the rest are the predictive variables. We used the predictive variables $X_3 - X_6$ for a total combined domain size of $17,248$.

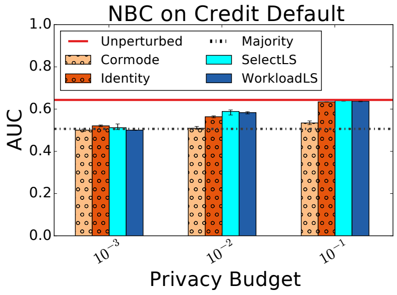

**Figure 3: New ϵKTELO plans WORKLOADLS and SELECTLS result in NaiveBayes classifiers with lower error than plans that correspond to algorithms from prior work, and approach the accuracy of a non-private classifier for various $\epsilon$ values.**

In our experiments we measure the average area under the curve (AUC) of the receiver operating characteristic curve across a 10-fold cross validation test. The AUC measures the probability that a randomly chosen positive instance will be ranked higher than a randomly chosen negative instance. We repeat this process 10 times (for a total of 100 unique testing/training splits) to account for the randomness of the differentially private algorithms and report the $\{25, 50, 75\}$-percentiles of the average AUC. As a baseline we show the majority classifier, which always predicts the majority class of the training data and also show the unperturbed classifier as an upper bound for the utility of our algorithms.

In Fig. 3 we report our findings: each group of bars corresponds to a different $\epsilon$ value and each bar shows the median value of the AUC for an algorithm. For each DP algorithm we also plot the error bars at the 25 and 75 percentiles. The dotted line is plotted at $0.5067$ and shows the AUC of the majority classifier. The continuous red line is the performance of the non-private classifier (Unperturbed). For larger $\epsilon$ values we see that our plans significantly outperform the baseline and reach AUC levels close to the unperturbed. As $\epsilon$ decreases, the quality of the private classifiers degrades and for $\epsilon = 10^{-3}$ the noise added to the empirical distributions drowns the signal and the AUC of the private classifiers reach $0.5$, which is the performance of a random classifier. Our plan WORKLOADLS is essentially the CORMODE algorithm with an extra inference operator, this shows that the addition of an extra operator to a previous solution significantly increases its performance.

## 8.2 Evaluation of algorithmic innovations

Our final experiments demonstrate the improvements resulting from each of the innovations proposed in Section 6.

*8.2.1 Scalability of inference.* Inference is one of the most computation-intensive operators in ϵKTELO  especially for large domains resulting from multidimensional data. Fig. 4 (on a log-log scale) shows the computation time for running our main inference operators (LS and NNLS) as a function of data vector size. Recall that the methods described in Sec. 6.1 provide efficiency improvements by using iterative solution strategies (*iterative* instead of *direct* in the figure) and exploiting sparsity in the measurement matrix (*sparse* as opposed to *dense* in the figure). For this experiment we fix the measured query set to consist of binary hierarchical measurements [16]. Fig. 4 shows that these methods allow inference to scale to data vectors consisting of millions of counts on a single machine in less than a minute. Imposing non-negativity constraints does have a cost in terms of scalability, but is still feasible for large domains.

**Table 2: Runtime (sec) and error improvements resulting from workload-based domain reduction. (W=RandomRange, small ranges. Original domain size: AHP (128,128), DAWA 4096, Identity (256,256), HB 4096)**

| Algorithm | Original Domain | | Reduced Domain | | Factor Improved | |
|---|---|---|---|---|---|---|
| | Error/Runtime | | Error/Runtime | | Error/Runtime | |
| AHP | 1.68e−5 | 777.10 | 1.30e−5 | 145.00 | 1.29 | 5.36 |
| DAWA | 1.06e−5 | 0.23 | 1.07e−5 | 0.25 | 0.99 | 0.92 |
| IDENTITY | 4.74e−5 | 0.66 | 1.64e−5 | 0.90 | 2.89 | 0.73 |
| HB | 3.20e−5 | 0.05 | 2.38e−5 | 0.08 | 1.34 | 0.62 |

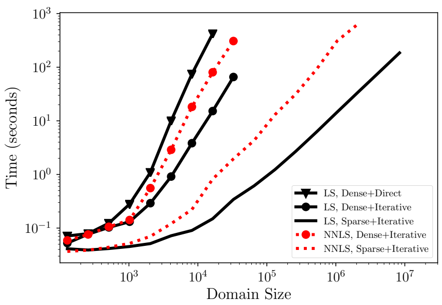

**Figure 4: For a given computation time, the proposed iterative and sparse inference methods permit scaling to data vector sizes as much as $100\times$ larger than previous techniques.**

*8.2.2 Workload-driven data reduction.* Next we evaluate the impact of workload-driven data reduction, as described in Section 6.2. For selected algorithms, Table 2 shows that performing workload-driven data reduction improves error and runtime, almost universally. The biggest improvement in error (a factor of 2.89) is witnessed for the IDENTITY algorithm. Without workload-driven reduction, groups of elements of the domain are estimated independently even though the workload only uses the total of the group. After reduction, the sum of the group of elements is estimated, and will have lower variance than the sum of independent measurements.

The biggest improvement in runtime occurs for the AHP algorithm. This algorithm has an expensive clustering step, performed on each element of the data vector. Workload-driven reduction reduces the cost of this step, since it is performed on a smaller data vector. It also tends to improve error because higher-quality clusters are found on the reduced data representation.

*8.2.3 MWEM: improved query selection & inference.* Lastly we evaluate the three new algorithms described in Sec. 6.3 which were inspired by MWEM [14] and created using operator inception and recombination. These algorithms are data-dependent algorithms so we evaluate them over a diverse collection of 10 datasets taken from DPBench [15]. The results are shown in Table 3. The performance of the first variant, line (b), shows that its alternative query selection operator can significantly improve error: by a factor of 2.8 on average (over various input datasets) and by as much as a factor

**Table 3: For three new algorithms, (b), (c), and (d), the multiplicative factors by which error is improved, presented as (min, mean, max) over datasets. For runtime, the mean is shown, normalized to the runtime of standard MWEM. (1D, n=4096, W=RandomRange(1000), $\epsilon = 0.1$)**

| | MWEM Variants | | ERROR IMPROVEMENT | | | RUNTIME |
|---|---|---|---|---|---|---|
| | Measure Selection | Inference | min | mean | max | mean |
| (a) | worst-approx | MW | 1 | 1 | 1 | 1 |
| (b) | worst-approx + H2 | MW | 1.03 | 2.80 | 7.93 | 354.9 |
| (c) | worst-approx | NNLS, known total | 0.78 | 1.08 | 1.54 | 1.0 |
| (d) | worst-approx + H2 | NNLS, known total | 0.89 | 2.64 | 8.13 | 9.0 |

of 7.9. (Error and runtime measures are normalized to the values for the original MWEM; min/mean/max error values represent variation across datasets.) Unfortunately, this operator substitution has a considerable impact on performance: it slows down by a factor of more than 300. But combining augmented query selection with NNLS inference, line (d), improves performance significantly: it is still slower than the original MWEM algorithm, but by only a factor of 9. Using the original MWEM query selection with NNLS inference, line (c), has largely equivalent error and runtime to the original MWEM. Thus, NNLS inference for this class of algorithms becomes especially useful when the number of measured queries grows, which can significantly improve this class of algorithms.

*Summary of Findings* The case studies on Census data and Naive Bayes classification show that $\epsilon$KTELO can be used to design novel algorithms from existing building blocks, offering state-of-the-art error rates. The evaluation of the algorithmic innovations described in Sec. 6 show that the new inference operators scales to 100x larger data vectors, and the workload-driven data reduction improves accuracy and runtime, almost universally, so that it can be added to all workload-based plans with little cost and significant potential for gains. Finally, the evaluation shows how recombination can lead to improvements in existing algorithms with little effort from the programmer: the MWEM algorithm can be improved significantly with better query selection and inference operators.

## 9 RELATED WORK

A number of languages and programming frameworks have been proposed to make it easier for users to write private programs [9, 24, 27, 30]. The *Privacy Integrated Queries* (PINQ) platform began this line of work and is an important foundation for $\epsilon$KTELO. We use the fundamentals of PINQ to ensure that plans implemented in $\epsilon$KTELO are differentially private. In particular, we adapt and extend a formal model of a subset of PINQ features, called Featherweight PINQ [9], to show that plans written using $\epsilon$KTELO operators satisfy differential privacy. Our extension adds support for the partition operator, a valuable operator for designing complex plans.

Additionally, there is a growing literature on formal verification tools that prove that an algorithm satisfies differential privacy [4, 12, 34]. For instance, LightDP [34] is a simple imperative language in which differentially private programs can be written. LightDP allows for verification of sophisticated differentially private algorithms with little manual effort. LightDP's goal is orthogonal to that of $\epsilon$KTELO: it simplifies proofs of privacy, while $\epsilon$KTELO's goal is to simplify the design of algorithms that achieve high accuracy. Nevertheless, an interesting future direction would be to implement $\epsilon$KTELO operators in LightDP to simplify both problems of verifying privacy and achieving high utility.

Concurrently with our work, Kellaris et al. [17] observe that algorithms for single-dimensional histogram tasks share subroutines that perform common functions. The authors compare a number of existing algorithms along with new variants formed by combining subroutines, empirically evaluating tradeoffs between accuracy and efficiency. They do not include a well-developed framework for authoring new algorithms and do not extend beyond 1D tasks.

The use of inference in differentially private algorithm design is not new [3, 16, 31], and is used in various guises throughout recent work [2, 6, 19, 20, 23, 26, 32, 37]. Proserpio et al. [26] propose a general-purpose inference engine based on MCMC that leverages properties of its operators to offset the otherwise high time/space costs of this form of inference. Our work is complementary in that we focus on a different kind of inference (based on least squares) in part because it is used, often implicitly, in many published techniques. A deeper investigation of alternative inference strategies is a compelling research direction.

The matrix mechanism [23] formulates an optimization problem that corresponds to query selection in $\epsilon$KTELO. The mechanism then estimates the selected queries and applies least squares inference. This can be seen as a kind of plan optimization, but in a limited plan space which admits only data-independent plans.

Recent work [18] examines the problem of algorithm selection—selecting the best algorithm for a given private dataset and task—and proposes a meta-algorithm, Pythia, capable of choosing among a set of "black box" algorithms. Pythia could be adapted to automatically select operators in $\epsilon$KTELO and Pythia itself could be implemented as an $\epsilon$KTELO plan.

## 10 CONCLUSIONS

We have described the design and implementation of $\epsilon$KTELO: an extensible programming framework and system for defining and executing differentially private algorithms. Many state-of-the-art differentially private algorithms can be specified as plans consisting of sequences of operators, increasing code reuse and facilitating more transparent algorithm comparisons. Algorithms implemented in $\epsilon$KTELO are often faster and return more accurate answers. Using $\epsilon$KTELO, we designed new algorithms that outperform the state of the art in accuracy on some linear query answering tasks.

$\epsilon$KTELO is extensible and, through the addition of a few new operators, we hope to substantially expand the classes of tasks that can be supported. For example, adding non-linear transformations could allow non-linear aggregation queries to be expressed linearly over transformed data. As a result, we could reuse many of the existing operators for linear estimation in order to answer non-linear queries. In addition, we believe even greater scalability could be achieved by the addition of new inference operators that do not require full vectorization of the input data.

**Acknowledgements:** This work was supported by the National Science Foundation under grants 1253327, 1408982, 1409125, 1443014, 1421325, and 1409143; and by DARPA and SPAWAR under contract N66001-15-C-4067. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes not withstanding any copyright notation thereon. The views, opinions, and/or findings expressed are those of the author(s) and should not be interpreted as representing the official views or policies of the Department of Defense or the U.S. Government.

# REFERENCES

[1] 2010. https://onthemap.ces.census.gov/. (2010).

[2] Gergely Ács, Claude Castelluccia, and Rui Chen. 2012. Differentially Private Histogram Publishing through Lossy Compression. In *ICDM*. 1–10.

[3] Boaz Barak, Kamalika Chaudhuri, Cynthia Dwork, Satyen Kale, Frank McSherry, and Kunal Talwar. 2007. Privacy, Accuracy, and Consistency Too: A Holistic Solution to Contingency Table Release. In *PODS*. 273 – 282.

[4] Gilles Barthe, Gian Pietro Farina, Marco Gaboardi, Emilio Jesus Gallego Arias, Andy Gordon, Justin Hsu, and Pierre-Yves Strub. 2016. Differentially Private Bayesian Programming. In *CCS*. 68–79.

[5] Graham Cormode. 2011. Personal Privacy vs Population Privacy: Learning to Attack Anonymization. In *KDD*.

[6] Graham Cormode, Magda Procopiuc, Entong Shen, Divesh Srivastava, and Ting Yu. 2012. Differentially Private Spatial Decompositions. In *ICDE*. 20–31.

[7] Cynthia Dwork, Frank McSherry Kobbi Nissim, and Adam Smith. 2006. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC*. 265–284.

[8] Cynthia Dwork and Aaron Roth. 2014. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science.

[9] Hamid Ebadi and David Sands. 2017. Featherweight PINQ. *JPC* 7, 2 (2017).

[10] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. 2014. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*.

[11] David Chin-Lung Fong and Michael Saunders. 2011. LSMR: An Iterative Algorithm for Sparse Least-Squares Problems. *SIAM J. Sci. Comput.* 33, 5 (Oct. 2011), 2950–2971.

[12] Marco Gaboardi, Andreas Haeberlen, Justin Hsu, Arjun Narayan, and Benjamin C. Pierce. 2013. Linear Dependent Types for Differential Privacy. In *POPL*. 357–370.

[13] Samuel Haney, Ashwin Machanavajjhala, John Abowd, Matthew Graham, Mark Kutzbach, and Lars Vilhuber. 2017. Utility Cost of Formal Privacy for Releasing National Employer-Employee Statistics. In *SIGMOD*.

[14] Moritz Hardt, Katrina Ligett, and Frank McSherry. 2012. A Simple and Practical Algorithm for Differentially Private Data Release. In *NIPS*.

[15] Michael Hay, Ashwin Machanavajjhala, Gerome Miklau, Yan Chen, and Dan Zhang. 2016. Principled Evaluation of Differentially Private Algorithms using DPBench. In *SIGMOD*.

[16] Michael Hay, Vibhor Rastogi, Gerome Miklau, and Dan Suciu. 2010. Boosting the accuracy of differentially private histograms through consistency. *PVLDB* (2010).

[17] Georgios Kellaris, Stavros Papadopoulos, and Dimitris Papadias. 2015. Differentially Private Histograms for Range-Sum Queries: A Modular Approach. *arXiv* (2015).

[18] Ios Kotsogiannis, Ashwin Machanavajjhala, Michael Hay, and Gerome Miklau. 2017. Pythia: Data Dependent Differentially Private Algorithm Selection. In *SIGMOD*.

[19] Jaewoo Lee, Yue Wang, and Daniel Kifer. 2015. Maximum Likelihood Postprocessing for Differential Privacy Under Consistency Constraints. In *KDD*.

[20] Chao Li, Michael Hay, and Gerome Miklau. 2014. A Data- and Workload-Aware Algorithm for Range Queries Under Differential Privacy. *PVLDB* (2014).

[21] Chao Li, Michael Hay, Vibhor Rastogi, Gerome Miklau, and Andrew McGregor. 2010. Optimizing Linear Counting Queries Under Differential Privacy. In *PODS*. 123–134.

[22] Chao Li and Gerome Miklau. 2012. An Adaptive Mechanism for Accurate Query Answering under Differential Privacy. *PVLDB* 5, 6 (2012), 514–525.

[23] Chao Li, Gerome Miklau, Michael Hay, Andrew McGregor, and Vibhor Rastogi. 2015. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB Journal* (2015), 1–25.

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

[25] Ilya Mironov. 2012. On significance of the least significant bits for differential privacy. In *CCS*.

[26] Davide Proserpio, Sharon Goldberg, and Frank McSherry. 2012. A workflow for differentially-private graph synthesis. In *Workshop on online social networks*.

[27] Davide Proserpio, Sharon Goldberg, and Frank McSherry. 2014. Calibrating Data to Sensitivity in Private Data Analysis: A Platform for Differentially-private Analysis of Weighted Datasets. *Proc. VLDB Endow.* 7, 8 (April 2014), 637–648.

[28] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Differentially private grids for geospatial data. In *ICDE*. IEEE, 757–768.

[29] Wahbeh Qardaji, Weining Yang, and Ninghui Li. 2013. Understanding hierarchical methods for differentially private histograms. *PVLDB* 6, 14 (2013).

[30] Indrajit Roy, Srinath T. V. Setty, Ann Kilzer, Vitaly Shmatikov, and Emmett Witchel. 2010. Airavat: Security and Privacy for MapReduce. In *NSDI*.

[31] Oliver Williams and Frank McSherry. 2010. Probabilistic Inference and Differential Privacy. *NIPS* (2010), 2451–2459.

[32] Xiaokui Xiao, Guozhang Wang, and Johannes Gehrke. 2010. Differential privacy via wavelet transforms. In *ICDE*. 225–236.

[33] I-Cheng Yeh and Che hui Lien. 2009. The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients. *Expert Systems with Applications* (2009), 2473 – 2480.

[34] Danfeng Zhang and Daniel Kifer. 2017. LightDP: Towards Automating Differential Privacy Proofs. In *POPL*. 888–901.

[35] Jun Zhang, Graham Cormode, Cecilia M. Procopiuc, Divesh Srivastava, and Xiaokui Xiao. 2017. PrivBayes: Private data release via Bayesian networks. *TODS* 42 (2017). Issue 4.

[36] Jun Zhang, Xioakui Xiao, and Xing Xie. 2016. PrivTree: A Differentially Private Algorithm for Hierarchical Decompositions. In *SIGMOD*.

[37] Xiaojian Zhang, Rui Chen, Jianliang Xu, Xiaofeng Meng, and Yingtao Xie. 2014. Towards Accurate Histogram Publication under Differential Privacy. In *SDM*.

# A APPENDIX

## A.1 Workload Based Domain Reduction

The new workload-based partition selection operator can be formalized in terms of a linear matrix operator, as follows:

*Definition A.1 (Workload-based partition selection).* Let $\mathbf{w_1}, \dots, \mathbf{w_n}$ denote the columns vectors in $\mathbf{W}$ and let $\mathbf{u_1}, \dots, \mathbf{u_p}$ denote those that are unique. For $h(\mathbf{u}) = \{j \mid \mathbf{w_j} = \mathbf{u}\}$, define the transformation matrix $\mathbf{P} \in \mathbb{R}^{p \times n}$ to have $\mathbf{P}[i,j] = 1$ if $j \in h(\mathbf{u_i})$ and $\mathbf{P}[i,j] = 0$ otherwise. The reverse transformation is the pseudo-inverse $\mathbf{P}^+ \in \mathbb{R}^{n \times p}$.

The matrix $\mathbf{P}$ defines a partition of the data, which can be passed to V-ReduceByPartition to transform the data vector, and $\mathbf{P}^+$ can be used to transform the workload accordingly. When $\mathbf{P}$ is passed to V-ReduceByPartition , the operator produces a new data vector $\mathbf{x}' = \mathbf{Px}$ where $\mathbf{x}'[i]$ is the sum of entries in $\mathbf{x}$ that belong to $i^{th}$ group of $\mathbf{P}$. When viewed as an operation on the workload, $\mathbf{P}^+$ merges duplicate columns by taking the row-wise average for each group. This is formalized as follows:

Proposition A.2 (properties: workload-based reduction). *Given transform matrix* $\mathbf{P}$ *and its pseudo-inverse* $\mathbf{P}^+$*, the following hold:*

- $\mathbf{x}' = \mathbf{Px}$ *is the reduced data vector;*
- $\mathbf{W}' = \mathbf{WP}^+$ *is the workload matrix, represented over* $x'$*;*
- *The transformation is lossless:* $\mathbf{Wx} = \mathbf{W}'\mathbf{x}'$

Proof: First note that $\mathbf{P}^+ = \mathbf{P}^T\mathbf{D}^{-1}$ where $\mathbf{D}$ is the $p \times p$ diagonal matrix with $\mathbf{D}[i,i] = |h(\mathbf{u_i})|$ for $h$ defined in Def. A.1. Since $\mathbf{P}$ has linearly independent rows, $\mathbf{P}^+ = \mathbf{P}^T(\mathbf{PP}^T)^{-1}$ and $\mathbf{PP}^T = \mathbf{D}$ because $h(\mathbf{u_i})$ and $h(\mathbf{u_j})$ are disjoint for $i \neq j$. By definition of $\mathbf{P}$, we see that $\mathbf{x}'[i] = \sum_{j \in h(\mathbf{u_i})} \mathbf{x}[j]$ for $1 \leq i \leq p$. Similarly, the $i^{th}$ column of $\mathbf{W}'$ is given by $\mathbf{w}'_\mathbf{i} = \frac{1}{|h(\mathbf{u_i})|} \sum_{j \in h(\mathbf{u_i})} \mathbf{w_j}$. Since $\mathbf{w_j} = \mathbf{u_i}$ when $j \in h(\mathbf{u_i})$, we have $\mathbf{w}'_\mathbf{i} = \mathbf{u_i}$, which shows that $\mathbf{W}'$ is just $\mathbf{W}$ with the duplicate columns removed. Using these definitions, we show that the transformation is lossless:

$$\mathbf{Wx} = \sum_{i=1}^{n} \mathbf{w_i x}[i] = \sum_{i=1}^{p} \mathbf{u_i} \sum_{j \in h(\mathbf{u_i})} \mathbf{x}[j] = \sum_{i=1}^{p} \mathbf{w}'_\mathbf{i} \mathbf{x}'[i] = \mathbf{W}'\mathbf{x}'$$

The computation of partition $\mathbf{P}$ in Def. A.1 is conceptually straightforward: it simply requires grouping the columns of $\mathbf{W}$. However, explicitly representing the workload as a matrix is sometimes inconvenient or prohibitive, especially for high-dimensional data. Algorithm 2 is an efficient method for finding the column groupings that does not require a complete matrix representation, as long as it is possible to compute the dot product $\mathbf{vW}$. This approach is highly scalable and naturally extends to sparse matrix representations of the workload or other specialized encodings of $\mathbf{W}$ (e.g. if $\mathbf{W}$ consists of range or marginal queries).

**Algorithm 2** An algorithm for workload-based data reduction

```
 1: procedure COMPUTE REDUCTION MATRIX($W$)
 2:     Input: $m \times n$ matrix $\mathbf{W}$
 3:     Output: $p \times n$ matrix $\mathbf{P}$ where $p \leq n$
 4:     set $\mathbf{v}$ = vector of $m$ samples from Uniform$(0, 1)$        ▷ $1 \times m$
 5:     compute $\mathbf{h} = \mathbf{vW}$                                   ▷ $1 \times n$
 6:     let $G = g_1, \ldots, g_p$ be groups of common values in $\mathbf{h}$
 7:     initialize matrix $\mathbf{P}$ with zeros                            ▷ $p \times n$
 8:     for $g_i$ in $G$ do
 9:         set row $i$ of $\mathbf{P}$ to 1 in each position of $g_i$
10:     end for
11:     return $\mathbf{P}$
12: end procedure
```

By grouping the elements of $\mathbf{h}$ (line 6) we recover the column groupings of $\mathbf{W}$ almost surely. Note that this algorithm is correct because if $\mathbf{w_i} = \mathbf{w_j}$ then $h_i = h_j$ and if $\mathbf{w_i} \neq \mathbf{w_j}$ then $P(h_i = h_j) = 0$ since $h_i$ and $h_j$ are continuous random variables.[3]

As noted in Example 6.1, not all workloads allow for reduction (in some cases, the $\mathbf{P}$ matrix computed above is the identity). But others may allow a significant reduction, which improves the efficiency of subsequent operators. Less obvious is that workload-based data reduction would impact accuracy. In fact, many query selection methods from existing work depend implicitly on the representation of the data in vector form, and these approaches may be improved by domain reduction. In Sec. 8.2 we measure the impact of this transform on accuracy and efficiency.

We show next that this reduction does not hurt accuracy: for any selected set of measurement queries, their reduction will provide lower error after transformation. Proof is omitted due to space constraints.

THEOREM A.3. *Given a workload* $\mathbf{W}$ *and data vector* $\mathbf{x}$*, let* $\mathbf{M}$ *be any query matrix that answers* $\mathbf{W}$*. Then if* $\mathbf{q}' = \mathbf{q}\mathbf{P}^+$ *is a reduced query and* $\mathbf{M}' = \mathbf{M}\mathbf{P}^+$ *is the query matrix on the reduced domain,* $Error_{\mathbf{q}'}(\mathbf{M}') \leq Error_{\mathbf{q}}(\mathbf{M})$ *for all* $\mathbf{q} \in \mathbf{W}$.

## A.2 Background: algorithms for linear queries

Here we provide additional background on the algorithms re-implemented in $\epsilon$KTELO, namely plans 1 through 12 in Fig. 2. The algorithms in Fig. 2 are listed roughly in the order in which they were proposed in the literature and reflect the evolution of increasingly complex algorithmic techniques. The simplest plan, IDENTITY (plan 1), is a natural application of the Laplace mechanism for answering linear queries. It simply measures each component of the data vector and since its measurement set is so simple, no inference is necessary.

Plans 2 through 5 reflect the evolution of more sophisticated measurements selection, targeted towards specific workloads. Many of these techniques were originally designed to support range queries (a small subclass of linear queries) over one- or two-dimensional data. PRIVELET uses a Haar wavelet as its measurements, which allows for sensitivity that grows logarithmically with the domain size, yet allows accurate reconstruction of any range query. The HIERARCHICAL (H2) technique uses measurements that form a binary tree over the domain, achieving effects similar to the wavelet measurements. QUADTREE (algorithm 10) is the 2-dimensional realization of the hierarchical structures, which supports 2D range queries. When these more complex measurement selection methods are used, inference becomes important because it allows a single consistent estimate to be derived, and provably reduces error.

The plans above provide measurement selection tuned to range query workloads, but measurement selection is always *static*. The next innovation in the literature consists of *adaptive* measurement selection. HIERARCHICAL OPT (HB) provides an intelligent heuristic for tuning the branching factor of hierarchical measurements to the domain size, often lowering error. GREEDY-H (which was proposed as a subroutine of the DAWA algorithm but can be used by itself) finds an optimal weighting of hierarchical measurements that minimizes error on an input workload.

All of the plans above are *data-independent*, with error rates that remain constant for any input dataset. More recent algorithms are *data-dependent*, displaying different error rates on different inputs, often because the algorithmic techniques are adapting to features of the data to lower error. The simplest data-dependent algorithm is UNIFORM (plan 6) which simply estimates the total number of records in the input and assumes uniformity across the data vector. It is primarily used as a baseline and sanity-check for other data-dependent algorithms. The Multiplicative-Weights Exponential Mechanism (MWEM, plan 7) takes a workload of linear queries as input and runs a sequence of rounds of estimation, measuring one workload query in each round, and using the multiplicative update rule to revise its estimate of the data vector. In each round, the Exponential Mechanism is used to select the workload query that is most poorly approximated using the current data vector estimate. In Fig. 2, the iteration inherent to MWEM is shown with $I : (..)$. Other data-dependent algorithms exploit partitioning, in which components of the data vector are merged and estimated only in their entirety, which uniformity assumption imposed within the regions. The DAWA and AHP algorithms have custom partition selection methods which consume part of the privacy budget to identify approximately uniform partition blocks. UNIFORMGRID and ADAPTIVEGRID focus on 2D data and both end up with partitioned sets of measurements forming a grid over a 2D domain. UNIFORMGRID imposes a static grid, while ADAPTIVEGRID uses an initial round of measurements to adjust the coarseness of the grid, avoiding estimation of small sparse regions.

## A.3 New plans for case studies

Algorithms 3, 4 and 5 fully describe the new plans proposed to support the Census and Naive Bayes use cases.

## A.4 Privacy Proof

This section presents a proof of Theorem 4.3.

*Preliminaries* We introduce some concepts and notation for the proof. (Some notation is adapted from [9].) A configuration, denoted $\mathbb{C} = \langle \mathbf{P}, \mathrm{PK} \rangle$, captures the state of the client, denoted $\mathbf{P}$, and the state of the protected kernel, denoted PK. The client state can be arbitrary, but state updates are assumed to be deterministic. The protected kernel state PK consists of the following components:

- A set of source variables $SV$.

[3] The probability of incorrectly grouping two different columns is approximately $10^{-16}$ with a 64-bit floating point representation, but if needed we can repeat the procedure $k$ times until the probability of failure ($\sim 10^{-16k}$) is vanishingly small.

**Algorithm 3** HB-STRIPED

```
D ← PROTECTED(source_uri)                        ▷ Init
x ← T-VECTORIZE(D)                               ▷ Transform
R ← StripeReduction($x$, Att)                    ▷ Partition Selection
$x_R$ ← V-SPLITBYPARTITION (x, R)
M ← ∅
y ← ∅
for $x' \in x_R$ do
    M ← M∪ HB(x′)                                ▷ Query Selection
    y ← y∪ VECLAPLACE(x′, M, $\epsilon$)          ▷ Query
end for
$\hat{x}$ ← LS(M, y)
return $\hat{x}$                                 ▷ Output
```

**Algorithm 4** PRIVBAYESLS

```
D ← PROTECTED(source_uri)                        ▷ Init
x ← T-VECTORIZE(D)                               ▷ Transform
M ← PBSELECT(x, $\epsilon_2$)                     ▷ Query Selection
y ← VECLAPLACE(x, M, $\epsilon_3$)                ▷ Query
$\hat{x}$ ← LS(M, y)                             ▷ Inference
return W · $\hat{x}$                             ▷ Output
```

**Algorithm 5** SELECTLS

```
D ← PROTECTED(source_uri)                        ▷ Init
x ← T-VECTORIZE(D)                               ▷ Transform
R ← MARGREDUCTION($x$, Att)                      ▷ Partition Selection
M ← ∅, y ← ∅
for $i = 1 : k$ do                               ▷ Iterate over Dimensions
    x′ ← V-REDUCEBYPARTITION (x, $R_i$)
    if $\text{DomainSize}_i > 80$ then
        R′ ← RDAWA (x′, $\epsilon_1$/k)           ▷ Partition Selection
        $x'_R$ ← V-REDUCEBYPARTITION (x, R′)
        M ← GREEDYH($x'_R$)                      ▷ Query Selection
        y ← y∪ VECLAPLACE($x'_R$, M, $\epsilon_2$/k)  ▷ Query
    else
        M ← M∪ IDENTITY(x′)                      ▷ Query Selection
        y ← y∪ VECLAPLACE(x′, $M$, $\epsilon$/k)  ▷ Query
    end if
    x ← V-REDUCEBYPARTITION (x′, $R_i$)          ▷ Domain Expansion
end for
$\hat{x}$ ← LS(M, y)
return $\hat{x}$                                 ▷ Output
```

- A data source environment $E$ maps each source variable $sv \in SV$ to an actual data source $S$, as in $E(sv) = S$. (Recall that sources can be tables or vectors. This dummy variable is used in Algorithm 6.)
- A transformation graph where nodes are the elements of $SV$ and there is an edge from $sv$ to $sv'$ if $sv'$ was derived via transformation from $sv$. (Note: a partition transformation introduces a special dummy data source variable whose parent is the source variable being partitioned and whose children are the variables associated with each partition.)
- A stability tracker $St$ maps each source variable $sv \in SV$ to a non-negative number: $St(sv)$ represents the stability factor of the transformations that created the data source $sv$, or 1 if $sv$ is the initial source.
- A budget consumption tracker $B$ that maps each source variable $sv \in SV$ to a non-negative number: $B(sv)$ represents the total budget consumption made by queries to $sv$ or *to any source derived from* $sv$.
- Query history for each source variable. $\mathcal{Q}(sv)$ captures information about the state of queries asked about $sv$ or any of its descendants. Specifically, it consists of a set of tuples $(q, s, \sigma, v)$ where the meaning of the tuple is that query $q$ was executed on data source $s$ (which is $sv$ or one of its descendants) with $\sigma$ noise, the result was $v$.
- The global privacy budget, denoted $\epsilon_{tot}$.

We can define the similarity of two configurations $\mathbb{C}$ and $\mathbb{C}'$ as follows. (Notation: we use $X'$ to refer to component $X$ of configuration $\mathbb{C}'$.) We say that $\mathbb{C} \sim \mathbb{C}'$ iff $\mathbf{P} = \mathbf{P}'$ and $\mathrm{PK}' \sim \mathrm{PK}'$ where $\mathrm{PK} \sim \mathrm{PK}'$ iff $SV = SV'$ and the transformation graphs are identical and for each $sv \in SV$ the following conditions hold:

- $St(sv) = St'(sv)$, $B(sv) = B'(sv)$, $\mathcal{Q}(sv) = \mathcal{Q}'(sv)$, and $\epsilon_{tot} = \epsilon'_{tot}$.
- $|E(sv) \oplus E'(sv)| \leq St(sv) = St'(sv)$ where $|x \oplus y|$ is measured as symmetric difference when the sources $x$ and $y$ are tables and $L_1$ distance for vectors; see Definition 3.4.)

When the protected kernel is initialized, as in $\mathbf{Init}(T, \epsilon_{tot})$, it sets global budget to $\epsilon_{tot}$, creates new source variable $sv_{root}$, sets $E(sv_{root}) = T$, sets $St(sv_{root}) = 1$, and $B(sv_{root}) = 0$, and adds $sv_{root}$ to the transformation graph.

*Lemma on Budget Management* When a query request is issued to the protected kernel, the protected kernel uses Algorithm 6 to check whether the query can be answered given the available privacy budget.

We introduce a lemma that bounds the difference probability between query answers. Let $P(q(E(s), \sigma) = v)$ denote the probability that query operator $q$ when applied to data source $E(s)$ with noise $\sigma$ returns answer $v$.

LEMMA A.4. *Let $\mathbb{C} \sim \mathbb{C}'$. For any $sv \in SV$ with non-empty $\mathcal{Q}(sv)$, the following holds:*

$$\prod_{(q,s,\sigma,v) \in \mathcal{Q}(sv)} P(q(E(s), \sigma) = v) \tag{3}$$
$$\leq \exp(B(sv) \times |E(sv) \oplus E'(sv)|) \times \prod_{(q,s,\sigma,v) \in \mathcal{Q}'(sv)} P(q(E'(s), \sigma) = v)$$

PROOF OF LEMMA A.4. Proof by induction on a reverse topological order of the transformation graph.

**Base case**: Consider a single $sv$ at the end of the topological order (therefore it has no children). If $\mathcal{Q}(sv)$ is empty, it holds trivially. Assume non-empty. Consider any $(q, s, \sigma, v) \in \mathcal{Q}(sv)$. Since $sv$ has no children, then $s = sv$. Furthermore, because the only budget requests that apply to $sv$ are from direct queries, we have (according to Algorithm 6), $B(sv) = \sum_{(q,s,\sigma,v) \in \mathcal{Q}(sv)} \sigma$. Since we assume that any query operator satisfies $\epsilon$-differential privacy with respect to its source input, we have $P(q(E(s), \sigma) = v) \leq P(q(E'(s), \sigma) = v) \times \exp(\sigma \times |E(s) \oplus E'(s)|)$. Substituting $sv$ for $s$ and taking the product over all terms in $\mathcal{Q}(sv)$, we get Eq. (3).

**Inductive case**: Assume Eq. (3) holds for all nodes later in the topological order. Therefore it holds for any child $c$ of $sv$. We can combine the inequalities for each child into the following inequality over all children,

$$\prod_{c\in\text{children}(sv)} \prod_{(q,s,\sigma,v)\in\mathcal{Q}(c)} P(q(E(s),\sigma)=v)$$

$$\le \prod_{c\in\text{children}(sv)} \exp(B(c)\times|E(c)\oplus E'(c)|)\times \prod_{(q,s,\sigma,v)\in\mathcal{Q}(c)} P(q(E'(s),\sigma)=v)$$

$$= \exp\left(\sum_{c\in\text{children}(sv)} B(c)\times|E(c)\oplus E'(c)|\right)$$

$$\times \prod_{c\in\text{children}(sv)} \prod_{(q,s,\sigma,v)\in\mathcal{Q}(c)} P(q(E'(s),\sigma)=v)$$

There are two cases, depending what type of table variable $sv$ is.

First, consider the case when $sv$ is *not* a special partition variable. We know by transformation stability that $|E(c)\oplus E'(c)| \le s\times|E(sv)\oplus E'(sv)|$ where $s$ is the stability factor for the transformation. In addition, $\sum_c B(c)\times s \le B(sv)$ because, according to Algorithm 6, every time a request of $\sigma$ is made to child $c$, a request of $s\times\sigma$ is made to $sv$. Therefore,

$$\sum_{c\in\text{children}(sv)} B(c)\times|E(c)\oplus E'(c)| \le \sum_{c\in\text{children}(sv)} B(c)\times s\times|E(sv)\oplus E'(sv)|$$
$$\le B(sv)\times|E(sv)\oplus E'(sv)|$$

Furthermore, observe that each term in $(q,s,\sigma,v)\in\mathcal{Q}(c)$ also appears in $\mathcal{Q}(sv)$. In addition, $\mathcal{Q}(sv)$ includes any queries on $sv$ directly (and we know from an argument similar to the base case that Eq. (3) holds for these queries). Therefore Eq. (3) holds on $sv$.

Now, consider the case where $sv$ is the special partition variable. Let $m = \max_c B(c)$. We get the following

$$\sum_{c\in\text{children}(sv)} B(c)\times|E(c)\oplus E'(c)| \le \sum_{c\in\text{children}(sv)} m\times|E(c)\oplus E'(c)|$$
$$= m\times\sum_{c\in\text{children}(sv)} |E(c)\oplus E'(c)| = m\times|E(sv)\oplus E'(sv)|$$
$$= B(sv)\times|E(sv)\oplus E'(sv)|$$

The second to last line follows from the fact that $sv$ is partition transformation. The last line follows from how $B(sv)$ is updated according Algorithm 6. □

*Main Proof* We use $\mathbb{C}_0(T,\epsilon_{tot},P_0)$ to denote the initial configuration in which the protected kernel has been initialized with **Init**$(T,\epsilon_{tot})$ and the client state is initialized to $P_0$. We use the notation $\mathbb{C}_0(T,\epsilon_{tot},P_0)\overset{t}{\Rightarrow}_p\mathbb{C}$ to mean that starting in $\mathbb{C}_0$ after $t$ operations, the probability of being in configuration $\mathbb{C}$ is $p$.

Theorem A.5. *If* $T\sim_1 T'$ *and* $\mathbb{C}_0(T,\epsilon_{tot},P_0)\overset{t}{\Rightarrow}_p\mathbb{C}$ *such that* $B(sv_{root})=\epsilon$ *in* $\mathbb{C}$, *then* $\epsilon\le\epsilon_{tot}$ *and there exists* $\mathbb{C}'$ *such that* $\mathbb{C}_0(T',\epsilon_{tot},P_0)\overset{t}{\Rightarrow}_q\mathbb{C}'$ *where* $\mathbb{C}\sim\mathbb{C}'$ *and* $p\le q\cdot\exp(\epsilon)$.

Theorem 4.3 follows as a corollary from Theorem A.5.

Proof of Theorem A.5. Proof by induction on $t$.

**Base case**: $t=0$. This implies that $p=q=1$, $\epsilon=0$, and $\mathbb{C}=\mathbb{C}_0(T,\epsilon_{tot},P_0)$ and $\mathbb{C}'=\mathbb{C}_0(T',\epsilon_{tot},P_0)$. It follows that $\mathbb{C}\sim\mathbb{C}'$ because we are given that $T\sim_1 T'$ and the rest of the claim follows.

**Inductive case**: Assume the claim holds for $t$, we will show it holds for $t+1$. Let $\mathbb{C}_1$ be any configuration such that $\mathbb{C}_0(T,\epsilon_{tot},P_0)\overset{t}{\Rightarrow}_{p_1}\mathbb{C}_1$ where in $\mathbb{C}_1$, we have $B(sv_{root})=\epsilon_1$.

The inductive hypothesis tells us that $\epsilon_1\le\epsilon_{tot}$ and that there exists a $\mathbb{C}'_1$ such that $\mathbb{C}_0(T',\epsilon_{tot},P_0)\overset{t}{\Rightarrow}_{q_1}\mathbb{C}'_1$ and $\mathbb{C}_1\sim\mathbb{C}'_1$ and $p_1\le q_1\times\exp(\epsilon_1)$.

Because $\mathbb{C}_1\sim\mathbb{C}'_1$, it follows that the client is in the same state and so the next operation request from the client will be the same in $\mathbb{C}_1$ and $\mathbb{C}'_1$. The proof requires a case analysis based on the nature of the operator. We omit analysis of transformation operators or operators that are purely on the client side as those cases are straightforward: essentially we must show that the appropriate bookkeeping is performed by the protected kernel. We focus on the case where the operator is a query operator.

For a query operator, there are two cases: (a) running out of budget, and (b) executing a query. For the first case, by the inductive hypothesis $\mathbb{C}_1\sim\mathbb{C}'_1$ and therefore if executing Algorithm 6 yields False on the protected kernel state in $\mathbb{C}_1$, it will also do so on the protected kernel state in $\mathbb{C}'_1$. For the second case, suppose query $q$ is executed on source $sv$ with noise $\sigma$ and answer $v$ is obtained. The protected kernel adds the correpsonding entry to the query history $\mathcal{Q}$. Let $\mathbb{C}$ denote the resulting state. Let $\mathbb{C}'$ correspond to extending $\mathbb{C}'_1$ in a similar way. Thus $\mathbb{C}\sim\mathbb{C}'$.

It remains to show two things. First, letting $B(sv_{root})=\epsilon$, we must show that $\epsilon\le\epsilon_{tot}$. This follows from Algorithm 6 which does not permit $B(sv_{root})$ to exceed $\epsilon_{tot}$. Second, we must bound the probabilities. Suppose that the probability of this query answer in $\mathbb{C}$ is $p_2$ and the probability of this answer on $\mathbb{C}'$ is $q_2$. It remains to show that $p_1\cdot p_2\le\exp(\epsilon)\cdot q_1\cdot q_2$. For this we rely on Lemma A.4 applied to $sv_{root}$ with the observations that the product of probabilities bounded in Lemma A.4 corresponds to the probabilities in $p_1\cdot p_2$ that do not trivially equal 1 and that $|E(sv_{root})\oplus E'(sv_{root})|=1$. □

**Algorithm 6** An algorithm for budget requests

```
procedure Request($sv$, $\sigma$)
    if $sv$ is the root then
        If $B(sv)+\sigma>\epsilon_{tot}$, return False. Otherwise $B(sv)$ += $\sigma$ and return True.
    else if $sv$ is a partition variable then
        Let $sv_{child}$ be the child from which the request came..
        Let $r = \max\{B(sv_{child})+\sigma-B(sv), 0\}$
        Let $ans$ = Request(parent($sv$), $r$).
        If $ans$ = False, return False. Otherwise, $B(sv)$ += $r$ and return True.
    else
        $ans$ = Request($parent(sv)$, $s\cdot\sigma$)        ▷ $s$ is stability factor of $sv$ wrt its parent
        if $ans$ = False, return False.
        $B(sv)$ += $\sigma$. Return True.
    end if
end procedure
```