ϵKTELO: A Framework for Defining Differentially-Private Computations

Dan Zhang
U. of Massachusetts Amherst
dzhang@cs.umass.edu

Ryan McKenna
U. of Massachusetts Amherst
rmckenna@cs.umass.edu

Ios Kotsogiannis
Duke University
iosk@cs.duke.edu

George Bissias
U. of Massachusetts Amherst
gbiss@cs.umass.edu

Michael Hay
Colgate University
mhay@colgate.edu

Ashwin Machanavajjhala
Duke University
ashwin@cs.duke.edu

Gerome Miklau
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, ϵKTELO, 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.

We describe the design and architecture of the ϵKTELO system and show that ϵKTELO is expressive enough to describe many algorithms from the privacy literature. ϵKTELO allows for safer implementations through code reuse and allows both privacy novices and experts to more easily design new algorithms. We demonstrate the use of ϵKTELO by designing new algorithms offering state-of-the-art accuracy and runtime.

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 the collected data—for new applications or modeling their users' behavior—or share it with external partners. Differential privacy [8, 9] 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. We believe 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, 15] (a U.S. Census Bureau data product), RAPPOR [11] (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 [8], 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 [17] 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 such as its size and distribution. 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, a non-trivial task. For instance, techniques used by modern privacy algorithms include optimizing error over 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

This is a minor revision of the work published in SIGMOD June 10–15, 2018, Houston, TX, USA.
https://doi.org/10.1145/3183713.3196921

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Copyright 2008 ACM 0001-0782/08/0X00 ...$5.00.

from noisy, inconsistent and incomplete measurement queries. But different algorithms use different specialized operators for these sub-tasks, 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 [42] showed that many variants of a privacy primitive called the Sparse Vector Technique do not satisfy differential privacy. Differential privacy can also be broken through incorrect implementations of sound algorithms. For example, Mironov [30] showed that standard implementations of basic algorithms like the Laplace Mechanism [8] can violate differential privacy because of their use of floating point arithmetic. Privacy-oriented programming frameworks such as PINQ [10, 29, 32], Fuzz [13], PrivInfer [6] and LightDP [38] help users 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.

Contributions

To address the aforementioned challenges, we propose ϵKTELO, a programming framework and system that aids programmers in developing differentially private programs with high utility. Programmers can use ϵKTELO to solve a core class of statistical tasks that involve answering *linear counting queries*. (This class of queries is defined in Sec. 2.) Tasks supported by ϵKTELO include releasing contingency tables, multi-dimensional histograms, answering OLAP and range queries, and implementing private machine learning algorithms.

In ϵKTELO, differentially private programs are described as *plans* over a high level library of *operators*. Each operator is an abstraction of a key subroutine from a state-of-the-art algorithm. Within ϵKTELO, these operators are organized based on their functionality into a small set of classes: transformation, querying, inference, query selection, and partition selection. These classes, and the operators within each class, are described in more detail in Sec. 3.2.

The design of ϵKTELO has the following characteristics:

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 illustrate the expressiveness of our operators by showing in Sec. 4 that the algorithms from the DPBench benchmark [17] 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. 3.3, the proof of which requires non-trivially extending the formal analysis of a past framework [10]. The privacy guarantee means that plan authors are not burdened with writing proofs for each algorithm they write. Furthermore plan authors do not need to think about how to calibrate the noise that is injected for privacy as this is handled automatically by ϵKTELO.

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 in several ways. First, since an algorithm is expressed as a plan and all plans automatically satisfy differential privacy, the code to be vetted is solely the individual operators. Second, operators need to be vetted only once but may be reused across multiple algorithms. Finally, it is not necessary to vet every operator, but only those that are privacy-critical (as shown in Sec. 3.1, ϵKTELO mandates a clear distinction between privacy-critical and non-private operators). This means that verifying the privacy of an algorithm requires checking fewer lines of code. In Sec. 4, we compare the verification effort to vet the DPBench codebase [2] against the effort required to vet these algorithms when expressed as plans in ϵKTELO.

Transparency In ϵKTELO, since all algorithms are expressed in the same form—a plan, consisting of a sequence of operators where each operator is selected from a class—it is easy to compare algorithms and identify differences. In Sec. 4, we summarize the plan signatures of a number of state-of-the-art algorithms (pictured in Fig. 2). These plan signatures reveal similarities and common idioms in existing algorithms. These are difficult to discover from the research literature or through code inspection.

The modular structure of ϵKTELO creates opportunities for technical innovation. There are broadly two kinds of innovation. The first is performance improvements that can be embedded directly into the framework, for example by improving the implementation of a key operator or designing efficient implementations of essential data structures. The second is technical innovation that arises from *using* ϵKTELO, for example, by using the framework to design new algorithms. In this paper we present innovations of both kinds.

Improved Scalability Many of the operators in ϵKTELO represent data and queries using matrices. Performing matrix operations can become a performance bottleneck. In Sec. 5.1, we present a specialized matrix representation that avoids the expensive materialization of matrix objects. This *implicit matrix* representation did not appear in the first version of the ϵKTELO framework [40] but is described in the extended version [39].

A vitally important but computationally expensive operator in many plans is inference. Inference is used to combine a collection of noisy measurements into a consistent estimate of the private data. In Sec. 5.2, we introduce a general-purpose, efficient and scalable inference engine, which exploits implicit representations, and subsumes customized inference subroutines from the literature.

Algorithm innovation enabled by ϵKTELO Because ϵKTELO plans are composed of operators, improving existing algorithms and implementing new algorithms becomes much easier: operators can be combined in new ways, for example, by substituting one instance of an operator class for another. In Sec. 6, we present one example of algorithmic innovation. We describe a new algorithm that, when expressed as a plan in ϵKTELO, is similar to the MWEM algorithm [16] but with a few key operators replaced, which can lower error by as much as 8 times.

After providing background in Sec. 2, we describe the system fully in Sec. 3. We illustrate the expressiveness of ϵKTELO plans and the benefits of ϵKTELO in Sec. 4. Scalability innovations provided by ϵKTELO are presented in Sec. 5 while algorithmic innovations enabled by ϵKTELO are described in Sec. 6. We discuss related work and conclude in Secs. 7 and 8.

2. BACKGROUND

The input to ϵKTELO is a database instance of a single-relation schema $T(A_1, A_2, \ldots, A_\ell)$. Multi-relation schemas pose a number of challenges (please see discussion in Sec. 8). Each attribute A_i is assumed to be discrete (or suitably discretized). A *condition for-*

mula, ϕ, 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 ϕ is true. A number of operators in ϵktelo answer linear queries over the table. A linear query is the linear combination of any finite set of condition counts:

Definition 1 (Linear counting query (declarative)). *A linear query q on T is defined by conditions $\phi_1 \dots \phi_k$ and coefficients $c_1 \dots c_k \in \mathbb{R}$ and returns $q(T) = c_1\phi_1(T) + \dots + c_k\phi_k(T)$.*

It is common to consider a vector representation of the database, denoted $\mathbf{x} = [x_1 \dots x_n]$, 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 ϕ evaluates to true for the tuple type i and 0 otherwise.

Definition 2 (Linear counting query (vector)). *For a linear query q defined by $\phi_1 \dots \phi_k$ and $c_1 \dots c_k$, its equivalent vector form is $\vec{q} = [q_1 \dots q_n]$ where $q_i = c_1\phi_1(i) + \dots + c_k\phi_k(i)$. 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 them. We say that T and T' are neighbors if $|T \oplus T'| = 1$.

Definition 3 (Differential Privacy [8]). *A randomized algorithm $\mathcal{A}$ is ϵ-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 [29] and post-processing [9]. Let f be a function on tables that outputs real numbers. The *sensitivity* of the function is defined as: $max_{|T \oplus T'|=1} |f(T) - f(T')|$. The sensitivity of a function evaluated on a table (or vector) resulting from a sequence of transformations can be calculated from the *stability* of each transformation function:

Definition 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. Then the stability of g is:* $\max_{S,S':|S \oplus S'|=1} |g(S) \oplus g(S')|$.

3. EKTELO

In this section, we describe the essential features of ϵktelo: its system architecture, its operator-based programming framework, and its guarantee that every program written in the framework satisfies differential privacy.

3.1 System Architecture

In ϵktelo, the private input data source is encapsulated inside a *protected kernel*. The analyst writes a differentially private program in an unprotected *client* space. Access to the protected data source is mediated by the protected kernel through a set of *operators*. The distinction between the client space and the protected kernel is a fundamental one in ϵktelo. It allows analysts to write *plans* (of differentially private programs) that consist of operator calls to the data source embedded in otherwise arbitrary code (which may include conditionals, loops, recursion, etc.). ϵktelo supports operators 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 acknowledgment that the operation has been performed. The second type is a **Private→Public** operator, which returns information about the private data outside the firewall (e.g., a differentially private measurement) and thus consumes privacy budget. The last type is a **Public** operator, which does not interact with the protected kernel or the protected data source at all and can be executed entirely in client space. We illustrate the operators supported by ϵktelo in Sec. 3.2.

Algorithm 1 ϵktelo CDF Estimator

```
D ← Protected(source_uri, ϵ)                          ▷ Initialize ϵktelo
D ← Where(D, sex == 'M' AND age ∈ [30, 39])           ▷ Transform
D ← Select(salary)                                    ▷ Transform
x ← T-Vectorize(D)                                    ▷ Transform
P ← AHPpartition(x, ϵ/2)                              ▷ Partition Selection
x̄ ← V-ReduceByPartition (x, P)                        ▷ Transform
M ← Identity(|x̄|)                                     ▷ Query Selection
y ← VecLaplace(x̄, M, ϵ/2)                             ▷ Query
x̂ ← NNLS(P, y)                                        ▷ Inference
W_pre ← Prefix(|x|)                                   ▷ Query Selection
return W_pre · x̂                                      ▷ Output
```

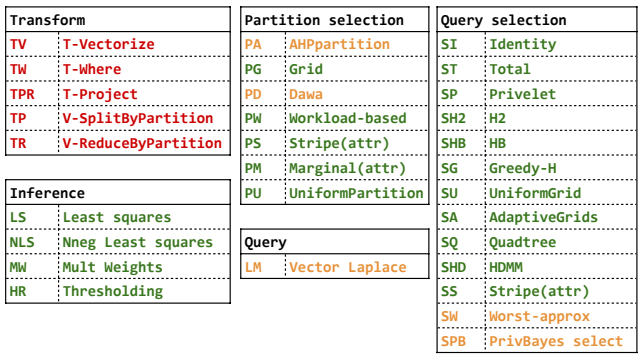
The protected kernel is initialized by specifying a single protected data object—an input table T—and a global privacy budget, ϵ. Note that requests for data transformations may cause the protected kernel to derive additional data sources (whose lineage is tracked to ensure correct privacy semantics).

We designed ϵktelo to be extensible through the addition of new operators. The effort required depends on the operator type: Public operators can be added at will; Private operators must register their *stability* (Sec. 2); Private→Public operators must be vetted by a privacy engineer to ensure that they satisfy differential privacy (note, that ϵktelo is responsible for appropriately calibrating the privacy budget when such operators are applied to derived data sources).

3.2 Operator Framework

In ϵktelo, 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 ϵktelo also supports plans with iteration (as in plan #7, to be presented in Fig. 2), recursion, and branching. Operators supported by ϵktelo perform a well defined task and typically 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. All the operators supported by ϵktelo are listed in Fig. 1 grouped by operator class and color coded by their type **Private**, **Private→Public** or **Public**. We next describe 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 returns 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 works in multiple steps. The plan uses *transformation* operators on the in-



Transform	
TV	T-Vectorize
TW	T-Where
TPR	T-Project
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)
PU	UniformPartition

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
SHD	HDMM
SS	Stripe(attr)
SW	Worst-approx
SPB	PrivBayes select

Figure 1: The operators currently implemented in ϵktelo. Private operators are red, Private→Public operators are orange, and Public operators are green.

put table D to filter out records that do not correspond to males in their 30's (Line 2), and to select 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. All transformation operators are Private operators as they change the private database and do not return anything outside the protected kernel.

Before adding noise to this histogram, the plan uses a *partition selection* operator, AHPPartition (Line 5). Operators in this class choose a partition $\mathbf{P}$ of the data vector $\mathbf{x}$ (or more generally, a mapping to a lower dimensional space) which is later used to transform $\mathbf{x}$. AHPPartition is a key subroutine in AHP [43], which was shown to have state-of-the-art performance for histogram estimation [17]. AHPPartition uses the data vector to identify a partition of the counts in $\mathbf{x}$ such that counts within a partition group are close. Since AHPPartition uses the private input, it is a Private→Public operator and thus consumes privacy budget (in this case $\epsilon/2$). Fig. 1 shows other examples of partition selection operators that are data independent, and hence are Public operators.

Next the plan uses V-ReduceByPartition (Line 6), another transformation operator, which applies on $\mathbf{x}$ 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 rest of the plan follows the "select-measure-reconstruct" paradigm, an approach exemplified by the matrix mechanism [24, 27] and used in several state-of-the-art algorithms [17]. In this paradigm, in order to answer a workload of queries, the algorithm first *selects* a different set of strategy queries, *measures* them using the Laplace mechanism (with noise calibrated to the sensitivity of the strategy), and lastly *reconstructs* answers to the original workload of queries from the noisy measurements using inference algorithms. The careful selection of a low sensitivity strategy can often lead to much more accurate answers than directly answering the workload.

In our CDF estimation problem, the workload corresponds to a set of prefix queries of the form "# people with salary $< i$". Rather than asking these queries, the plan chooses the strategy of "identity," which corresponds to queries of the form "# people with salary $= i$". In Algorithm 1, this is captured by the Identity operator (Line 7). This operator is a *query selection* operator. Such operators specify a set of measurement queries $\mathbf{M}$, encoded in matrix form to be applied to the data vector. The Identity operator returns the identity matrix, which corresponds to querying all the entries in $\bar{\mathbf{x}}$ (since $\mathbf{M} \cdot \bar{\mathbf{x}} = \bar{\mathbf{x}}$). In general, 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.) Most query selection operators are data independent and thus are Public, while some use the data to select the query strategy, and hence are Private→Public.

Next, the plan performs the measurement step using the Vector Laplace operator, which returns differentially private answers to all the queries in $\mathbf{M}$. First, it automatically calculates the sensitivity of the vectorized queries – which depends on all upstream data transformations – and then adds noise via the standard Laplace mechanism (Line 8). This operator consumes the remainder of the privacy budget. Query operators like Vector Laplace return a noisy measurement from the data, and by definition are differentially private algorithms that expend privacy budget. They are Private→Public.

So far the plan has computed an estimated histogram of partition group counts $\mathbf{y}$, but now it must *reconstruct* the empirical CDF on the original salary domain. From the noisy counts on the reduced domain $\mathbf{y}$, the plan infers non-negative counts in the original vector space of $\mathbf{x}$ by invoking an *inference* operator NNLS (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 entries of $\hat{\mathbf{x}}$ are non-negative. Inference operators never access the protected data and thus can be safely run outside the protected kernel. All inference operators are Public.

Lastly, the plan constructs the set of queries, $\mathbf{W}_{\mathbf{pre}}$, needed to compute the empirical CDF (a lower triangular $k \times k$ matrix representing prefix sums) by calling the query selection operator Prefix(k) (Line 10), and returns the output (Line 11).

3.3 Privacy Guarantee

In this section, we state the privacy guarantee offered by ϵktelo. Informally, ϵktelo ensures that if the protected kernel is initialized with a source database T and a privacy budget ϵ, then any plan (chosen by the client) satisfies ϵ-differential privacy. ϵktelo ensures privacy by tracking the privacy budget consumed by each operator call. The amount spent depends on the operator (e.g., Public operators consume nothing) and on what operations came before it (e.g., transformations). If at any point, the privacy budget is exhausted, any subsequent call to an operator that requires budget (i.e., a Private→Public operator) will throw an exception.

In the proof of privacy, we model a client's plan as an arbitrary and possibly infinite sequence of operator calls, where each call may be adaptively chosen based on the results of earlier calls. Thus we can think of the plan as a random process, where the randomness comes from the randomness of the operators (though the client code could also be randomized). The length of the plan, measured by the number of operator calls, can vary but we can nevertheless consider the set of (partial) plans of length k for any k. (A shorter plan can be extended with "no op" calls and for a longer plan, we consider the prefix of its first k operators.) This allows us to define a probability distribution over outcomes after k operations where an outcome is a particular length k plan, denoted $p_1 p_2 \dots p_k$, and the outputs received from executing that plan, denoted $o_1 o_2 \dots o_k$.

Theorem 3.1 (Privacy of ϵktelo plans). *Let T, T' be any two neighboring instances. For all $k \in \mathbb{N}^+$,*

$$P(\text{Plan is } p_1 p_2 \dots p_k \text{ with outputs } o_1 o_2 \dots o_k \mid \text{Ektelo init. with } (T, \epsilon))$$
$$\leq e^{\epsilon} P(\text{Plan is } p_1 p_2 \dots p_k \text{ with outputs } o_1 o_2 \dots o_k \mid \text{Ektelo init. with } (T', \epsilon))$$

The proof of Theorem 3.1 extends the proof in [10] to support the V-SplitByPartition operator. While ϵktelo ensures differential privacy, private information could be leaked via side-channel attacks (e.g., timing attacks) [14]. Privacy engineers who design operators are responsible for protecting against such attacks; an analysis of this issue is beyond the scope of this paper.

ID	Cite	Algorithm name	Plan signature
1	Dwork et al. 2006	Identity	SI LM
2	Xiao et al. 2010	Privelet	SP LM LS
3	Hay et al. 2010	Hierarchical (H2)	SH2 LM LS
4	Qardaji et al. 2013	Hierarchical Opt (HB)	SHB LM LS
5	Li et al. 2014	Greedy-H	SG LM LS
6	-	Uniform	ST LM LS
7	Hardt et al. 2012	MWEM	I:(SW LM MW)
8	Zhang et al. 2014	AHP	PA TR SI LM LS
9	Li et al. 2014	DAWA	PD TR SG LM LS
10	Cormode et al. 2012	Quadtree	SQ LM LS
11	Qardaji et al. 2013	UniformGrid	SU LM LS
12	Qardaji et al. 2013	AdaptiveGrid	SU LM LS PU TP[SA LM] LS
13	NEW	MWEM variant b	I:(SW SH2 LM MW)
14	NEW	MWEM variant c	I:(SW LM NLS)
15	NEW	MWEM variant d	I:(SW SH2 LM NLS)

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

4. EKTELO BENEFITS

ϵKTELO provides a number of benefits for the authors of differentially private programs, including code reuse, transparency, expressiveness, and a significant reduction in privacy verification effort.

We illustrate these benefits by reporting on our experience re-implementing state-of-the-art algorithms as ϵKTELO plans. We examined 12 algorithms for answering low-dimensional counting queries that were deemed competitive in a recent benchmark study [17]. The process of re-implementing this seemingly diverse set of algorithms consisted of identifying and isolating key subroutines and translating them into operators.

The prototype implementation of ϵKTELO, including all algorithms used in experiments, consists of 7.9k lines of Python code. The framework itself makes up 25% while operator implementations make up 46% and 15% are tests and examples provided for users. The remaining 14% are definitions of plans used in our experiments, showing that once operators are defined, plan definitions are relatively simple. In fact, plans can be described and compared at a high level by looking at *plan signatures*. Fig. 2 describes the 12 re-implemented algorithms (numbered 1 through 12) using the abbreviations for operators introduced in Fig. 1. We use these plan signatures to highlight the following benefits:

Code reuse Operations that are common to many plans can be implemented once and reused across ϵKTELO programs. For example, once reformulated in ϵKTELO, nearly all the algorithms in Fig. 2 use the VECTOR LAPLACE operator (LM) and least squares inference (LS). This benefits plan authors since it simplifies and accelerates their ability to write new differentially private algorithms without having to reimplement sophisticated and privacy critical operators. In addition, any improvements to these operators will be inherited by all the plans. We show such an example in Sec. 5.2.

Algorithm transparency By rewriting algorithms as plans, ϵKTELO makes explicit the typical high-level patterns that reflect design idioms of algorithms in literature. For example, plans 2, 3, 4, 5, 6, 10, and 11 all share a common operator sequence: Query selection, Query (LM), and Inference (LS); they differ only in the specifics of their Query selection method. Moreover, ϵKTELO plans help clarify the distinctive components of complex state-of-the-art algorithms. For instance, AHP and DAWA (plans 8 and 9 in Fig. 2) have the same structure but differ only in two operators: partition selection and query selection.

Reduced privacy verification effort Code reuse also reduces the number of critical operators that must be carefully vetted. To verify privacy, the only operators that require careful vetting are those 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 [23] and AHP [43], a query selection operator used by PrivBayes [41], and a query selection operator used by the MWEM [16] algorithm, which privately derives the worst-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. As a result, fewer lines of code must be verified as correct. For example, to verify the QuadTree algorithm in the DPBench codebase requires checking 163 lines of code. However, with ϵKTELO, this only requires vetting the 30-line VECTOR LAPLACE operator. And, 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.). Considering all 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 ϵKTELO requires checking only 517 lines of code.

5. SCALABILITY INNOVATIONS

In this section we describe a number of innovations that enable key operations in ϵKTELO to scale to larger problem instances.

5.1 Implicit matrix representations

Matrices and operations on matrices are central to the implementation of ϵKTELO operators but can become a performance bottleneck. In an extension [39] to the original version [40] of ϵKTELO, we describe a set of specialized matrix representation techniques, based on the *implicit* definition of matrices, which allows for performance improvements and greater scalability as the size of the data vector grows.

Matrices are used to represent three different objects in ϵKTELO: sets of workload queries, sets of measurement queries, and partitions of the domain. In all cases the matrices contain one column for each element of a corresponding data vector. In the case of both workload and measurement matrices, rows represent linear queries. The number of rows in a workload or measurement matrix is often as large, or larger, than n, the number of elements in the data vector. Partition matrices have at most n rows, but may still be large. For plans operating on large data vectors, where n approaches the size of memory, these matrices, in standard form, are infeasible to represent in memory and operate on.

While distributed computation would be one solution, we find that we can remain with a main memory implementation by using a set of performance enhancements based on two key observations. First, the large matrix objects used in plans tend to possess structure that can be exploited to represent them very concisely. Second, the plan implementations in ϵKTELO use a relatively small set of basic operations on these matrix objects (e.g. matrix-vector product, matrix multiplication, transpose, absolute value). Together, these observations allow matrices to be represented and operated on *implicitly*, which results in significant performance improvements.

As an example of an implicit matrix, recall the Prefix workload, $\mathbf{W}_{\mathbf{pre}}$, an encoding of an empirical CDF, which was used in the example plan (Algorithm 1). In *explicit* form, the prefix workload has n^2 entries and is defined as a lower-triangular matrix containing 1's. Note that a *sparse* representation of $\mathbf{W}_{\mathbf{pre}}$ would store (a list of) only the nonzero elements of this matrix, but the space complexity remains $O(n^2)$. Thus, the time complexity of computing matrix-

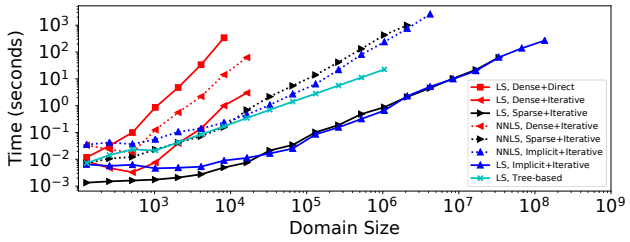


Figure 3: For a given computation time, the proposed iterative and implicit inference methods permit scaling to data vector sizes as much as 1000× larger than previous techniques that use direct approaches and dense matrices.

vector products using an explicit or sparse representation is $O(n^2)$. However, the Prefix matrix can be completely specified by a single parameter, n, which is the only state stored for the implicit version of $\mathbf{W}_{\mathbf{pre}}$. Further, we can evaluate the matrix-vector product $\mathbf{y} = \mathbf{W}_{\mathbf{pre}}\mathbf{x}$ using a simple one-pass algorithm over $\mathbf{x}$. Thus, the *implicit* Prefix workload representation achieves $O(1)$ space complexity and $O(n)$ time complexity for computing matrix-vector products.

While the original version of ϵKTELO exploited sparse matrix representations for some objects, we improve and extend our matrix representations to include implicit matrices in [39]. We describe a set of core implicit matrices, operations to combine them to form new implicit matrices, along with implementations of key operations on matrices, demonstrating significant performance improvements for ϵKTELO plans. One of the most important operations on implicit matrices is inference, which we discuss next.

5.2 Scalable and general inference

Inference is a fundamental operator that can improve error with no cost to privacy and, accordingly, we see that it appears in almost every algorithm re-implemented in ϵKTELO (as shown in Fig. 2). But inference can be a costly operation. Recall that the input to inference is a measurement matrix, denoted by $\mathbf{M}$, containing m queries defined over a data vector of size n, along with the list of noisy measurement answers $\mathbf{y}$. The most common inference method in existing algorithms is based on minimizing squared error:

DEFINITION 5 (ORDINARY LEAST SQUARES (LS)).

$$\hat{\mathbf{x}} = \arg\min_{x \in \mathbb{R}^n} \|\mathbf{M}\mathbf{x} - \mathbf{y}\|_2 \quad (1)$$

The least squares solution (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 $\hat{\mathbf{x}} = (\mathbf{M}^T\mathbf{M})^{-1}\mathbf{M}^T\mathbf{y}$. Matrix inversion is often avoided in favor of matrix factorizations of $\mathbf{M}$, but these methods, which we will refer to as *direct*, have time complexity cubic in the domain size, making it unacceptable when n is greater than about 5000.

Algorithms in prior work [18, 33, 34, 37] have performed least squares inference on large domains by restricting the selection of queries, namely to those representing a set of hierarchical queries. For this special case, inference can be performed in time linear in the domain size, avoiding the explicit matrix representation of the queries. We avoid this approach in ϵKTELO because it means that a custom inference method may be required for each query selection operation, and because it limits the measurement sets that can be used. In addition, hierarchical methods only work for least squares but not other inference methods, such as least squares with non-negativity constraints (NNLS).

An alternative to the *direct* implementation of least squares inference uses 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 $O(kn^2)$ for dense matrix representations where k is the number of iterations. We use a well-known iterative method, LSMR [12], and empirically we observe LSMR to converge in far fewer than n iterations when $\mathbf{M}$ is well-conditioned, and thus we expect $k << n$.

Table 1: 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

The benefits of iterative inference methods are amplified when the underlying matrix representation is implicit. Letting $Time(\mathbf{M})$ denote the time complexity of evaluating a matrix-vector product with $\mathbf{M}$, the time complexity of least squares inference is $O(k \cdot Time(\mathbf{M}))$ where k is the number of iterations. For implicit matrices, $Time(\mathbf{M})$ is often $O(n)$, resulting in a very favorable $O(kn)$ time complexity for inference. Iterative approaches, using implicit matrices, are also well-suited to the other inference methods in ϵKTELO: least squares with non-negativity constraints (NNLS) and multiplicative weights (MW).

We compare the performance of the above approaches for a fixed measured query set consisting of binary hierarchical measurements [18]. Fig. 3 shows that using sparse matrices and iterative methods allows inference to scale to data vectors consisting of millions of counts on a single machine in less than a minute. The use of implicit matrices permits additional scale-up for both LEASTSQUARES and NNLS. We also compare against the custom inference method introduced by Hay et al., denoted 'Tree-based' in the figure. It is an algorithm that is logically equivalent to LEASTSQUARES but specialized for hierarchically structured measurements. The general-purpose LEASTSQUARES implementation is able to scale to much larger domains.

Importantly, the combination of general implicit matrix construction techniques with iterative inference results in flexible inference capabilities for plan authors. With relative freedom, authors can construct measurement matrices, or combine measurements from different parts of a plan, and apply a single generic inference operator, which will run efficiently.

6. ALGORITHMIC INNOVATIONS

The operator-based model of ϵKTELO enables novel improvements to algorithm design through (i) *operator inception*, in which a new operator is proposed for an operator class; (ii) *recombination*, where different operator instances are substituted for those in an existing plan to form a new plan; and (iii) *plan restructuring*, in which a plan is systematically restructured by applying a general design principle or heuristic rule. In the original version of this paper [40], we provide detailed examples of each of these innovation types. Below we provide a single example, improving the well-known MWEM algorithm, through *operator inception* and *recombination*.

The *Multiplicative Weights Exponential Mechanism* (MWEM) [16] algorithm computes answers to a given workload of linear queries. MWEM operates in a sequence of rounds, determined by an input parameter. In each round it selects the worst-approximated workload query with respect to its current estimate of the

data, measures the selected query, and then uses the multiplicative weights update rule to refine its estimate of the data.

When viewed as a plan in ϵKTELO, a deficiency of MWEM becomes apparent. Its query selection operator selects a *single* query to measure in each round, whereas most query selection operators select a set of queries, each measuring 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 a new query selection operator that adds to the worst-approximated query by attempting to build a binary hierarchical set of queries over the sequence of 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 the runtime of multiplicative weights inference scales with the number of measured queries, inference 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 MWEM.

Using ϵKTELO, it was easy to describe three MWEM variants, which are shown in Fig. 2: an alternative query selection operator (PLAN #13) which augments selected measurements with hierarchical queries, an alternative inference operator (NNLS) (PLAN #14), and the addition of both alternative operators (PLAN #15).

Table 1 shows the experimental results over a collection of 10 datasets taken from [17]. The performance of the first variant on line (b) shows that the new query selection operator can significantly improve error: by a factor of 2.8 on average and by as much as a factor of 7.9. As explained above, it also has a considerable impact on performance because inference must operate on a larger set of queries: the slow down is a factor of more than 300. Line (c) shows that using the original MWEM query selection with NNLS inference has largely equivalent error and runtime to the original MWEM. However, combining augmented query selection with NNLS inference, shown on line (d), reduces runtime significantly while maintaining good accuracy: it is still slower than the original MWEM algorithm, but by only a factor of 9. The performance gains of NNLS inference over MW appear to be most pronounced when the number of measured queries is large. Overall, the algorithm variant (d) would likely be favored by users, who are typically willing to sacrifice efficiency for greater accuracy.

7. RELATED WORK

A number of languages and programming frameworks have been proposed to make it easier for users to write private programs [10, 29,32,35]. The *Privacy Integrated Queries* (PINQ) platform began this line of work and is an important foundation for ϵKTELO. We use the fundamentals of PINQ to ensure that plans implemented in ϵKTELO are differentially private. In particular, we adapt and extend a formal model of a subset of PINQ features, called Featherweight PINQ [10], to show that plans written using ϵKTELO 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, 6, 13,38]. For instance, LightDP [38] is a simple imperative language in which differentially private programs can be written, allowing verification with little manual effort. LightDP's goal is orthogonal to that of ϵKTELO: it simplifies proofs of privacy, while ϵKTELO's goal is to simplify the design of algorithms that achieve high accuracy.

Concurrently with our work, Kellaris et al. [19] observe that algorithms for single-dimensional histogram tasks share subroutines that perform common functions.

The use of inference appears in many differentially private algorithms [3, 5, 7, 18, 22, 23, 25, 31, 36, 37, 43]. Proserpio et al. [31] 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 algorithms.

A full treatment of automated plan optimization is an important future goal for ϵKTELO, however ϵKTELO could directly incorporate limited forms of automation already proposed in the literature. The matrix mechanism [25,27] formulates an optimization problem that corresponds to automated query selection in ϵKTELO. Other recent work [20, 26] considers the problem of data-dependent algorithm selection. These methods could be adapted to automatically select from a set of predefined plans in ϵKTELO.

8. CONCLUSIONS AND FUTURE WORK

We have described the design and implementation of ϵKTELO: an extensible programming framework and system for defining differentially private algorithms. Many state-of-the-art differentially private algorithms can be specified as ϵKTELO plans, consisting of sequences of operators, increasing code reuse and facilitating more transparent algorithm comparisons. Algorithms implemented in ϵKTELO are often faster and return more accurate answers.

ϵKTELO is extensible and we hope to expand the classes of tasks that can be supported. First, ϵKTELO is currently focused on statistical queries on a single table. Supporting more expressive aggregate queries, for example those expressible as SQL queries over multi-relational schemas, will require a number of extensions, including more complex transformations, advanced stability analysis, and improved inference [21].

Second, ϵKTELO currently relies on materializing the data vector in memory. Larger data vectors often occur with high-dimensional data, and while parallel computation is one possible solution, private algorithms may perform better when they are structured as a collection of private measurements over lower-dimensional projections of the original data. The ϵKTELO architecture is well-suited to perform transformation, selection, and measurement in such plans, and could adopt recently proposed methods [28] to perform "global" inference without materializing the full data vector.

Third, ϵKTELO provides a promising foundation for automatic optimization. Much like a relational optimizer, we envision adding a component that can explore a plan space implied by the collection of operators implemented in ϵKTELO. However, optimization here has dual objectives (accuracy and efficiency) and, in addition, it may be important to accommodate user-defined accuracy metrics. Further, the accuracy of some plans depends on the input data and may incur a privacy cost if it is used naively by the system during optimization.

Acknowledgments: Work supported by the National Science Foundation under grants 1253327, 1408982, 1409125, 1443014, 1421325, and 1409143; and by DARPA under contract N66001-15-C-4067.

9. REFERENCES

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

[2] https://github.com/dpcomp-org/dpcomp_core, 2016.

[3] G. Ács, C. Castelluccia, and R. Chen. Differentially private histogram publishing through lossy compression. In *ICDM*, pages 1–10, 2012.

[4] A. Albarghouthi and J. Hsu. Synthesizing coupling proofs of differential privacy. *Proc. ACM Program. Lang.*, (POPL), Dec. 2017.

[5] B. Barak, K. Chaudhuri, C. Dwork, S. Kale, F. McSherry, and K. Talwar. Privacy, accuracy, and consistency too: A holistic solution to contingency table release. In *PODS*, pages 273 – 282, 2007.

[6] G. Barthe, G. P. Farina, M. Gaboardi, E. J. G. Arias, A. Gordon, J. Hsu, and P.-Y. Strub. Differentially private bayesian programming. In *CCS*, pages 68–79, 2016.

[7] G. Cormode, M. Procopiuc, E. Shen, D. Srivastava, and T. Yu. Differentially private spatial decompositions. In *ICDE*, pages 20–31, 2012.

[8] C. Dwork, F. M. K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, pages 265–284, 2006.

[9] C. Dwork and A. Roth. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science, 2014.

[10] H. Ebadi and D. Sands. Featherweight pinq. *JPC*, 7(2), 2017.

[11] Ú. Erlingsson, V. Pihur, and A. Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *CCS*, 2014.

[12] D. C.-L. Fong and M. Saunders. LSMR: An iterative algorithm for sparse least-squares problems. *SIAM J. Sci. Comput.*, 33(5):2950–2971, Oct. 2011.

[13] M. Gaboardi, A. Haeberlen, J. Hsu, A. Narayan, and B. C. Pierce. Linear dependent types for differential privacy. In *POPL*, pages 357–370, 2013.

[14] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential privacy under fire. In *USENIX Conference on Security*, 2011.

[15] S. Haney, A. Machanavajjhala, J. Abowd, M. Graham, M. Kutzbach, and L. Vilhuber. Utility cost of formal privacy for releasing national employer-employee statistics. In *SIGMOD*, 2017.

[16] M. Hardt, K. Ligett, and F. McSherry. A simple and practical algorithm for differentially private data release. In *NIPS*, 2012.

[17] M. Hay, A. Machanavajjhala, G. Miklau, Y. Chen, and D. Zhang. Principled evaluation of differentially private algorithms using dpbench. In *SIGMOD*, 2016.

[18] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *PVLDB*, 2010.

[19] G. Kellaris, S. Papadopoulos, and D. Papadias. Differentially private histograms for range-sum queries: A modular approach. *arXiv*, 2015.

[20] I. Kotsogiannis, A. Machanavajjhala, M. Hay, and G. Miklau. Pythia: Data dependent differentially private algorithm selection. In *SIGMOD*, 2017.

[21] I. Kotsogiannis, Y. Tao, A. Machanavajjhala, G. Miklau, and M. Hay. Architecting a differentially private SQL engine. In *Conf. on Innovative Data Systems Research (CIDR)*, 2019.

[22] J. Lee, Y. Wang, and D. Kifer. Maximum likelihood postprocessing for differential privacy under consistency constraints. In *KDD*, 2015.

[23] C. Li, M. Hay, and G. Miklau. A data- and workload-aware algorithm for range queries under differential privacy. *PVLDB*, 2014.

[24] C. Li, M. Hay, V. Rastogi, G. Miklau, and A. McGregor. Optimizing linear counting queries under differential privacy. In *PODS*, pages 123–134, 2010.

[25] C. Li, G. Miklau, M. Hay, A. McGregor, and V. Rastogi. The matrix mechanism: optimizing linear counting queries under differential privacy. *The VLDB Journal*, pages 1–25, 2015.

[26] J. Liu and K. Talwar. Private selection from private candidates. *CoRR*, abs/1811.07971, 2018.

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

[28] R. McKenna, D. Sheldon, and G. Miklau. Graphical-model based estimation and inference for differential privacy. In *ICML*, 2019.

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

[30] I. Mironov. On significance of the least significant bits for differential privacy. In *CCS*, 2012.

[31] D. Proserpio, S. Goldberg, and F. McSherry. A workflow for differentially-private graph synthesis. In *Workshop on online social networks*, 2012.

[32] D. Proserpio, S. Goldberg, and F. McSherry. Calibrating data to sensitivity in private data analysis: A platform for differentially-private analysis of weighted datasets. *Proc. VLDB Endow.*, 7(8):637–648, Apr. 2014.

[33] W. Qardaji, W. Yang, and N. Li. Differentially private grids for geospatial data. In *ICDE*, pages 757–768. IEEE, 2013.

[34] W. Qardaji, W. Yang, and N. Li. Understanding hierarchical methods for differentially private histograms. *PVLDB*, 2013.

[35] I. Roy, S. T. V. Setty, A. Kilzer, V. Shmatikov, and E. Witchel. Airavat: Security and privacy for mapreduce. In *NSDI*, 2010.

[36] O. Williams and F. McSherry. Probabilistic Inference and Differential Privacy. *NIPS*, pages 2451–2459, 2010.

[37] X. Xiao, G. Wang, and J. Gehrke. Differential privacy via wavelet transforms. In *ICDE*, pages 225–236, 2010.

[38] D. Zhang and D. Kifer. Lightdp: Towards automating differential privacy proofs. In *POPL*, pages 888–901, 2017.

[39] D. Zhang, R. McKenna, I. Kotsogiannis, G. Bissias, M. Hay, A. Machanavajjhala, and G. Miklau. Ektelo: A framework for defining differentially-private computations. https://arxiv.org/abs/1808.03555v3.

[40] D. Zhang, R. McKenna, I. Kotsogiannis, M. Hay, A. Machanavajjhala, and G. Miklau. Ektelo: A framework for defining differentially-private computations. In *ACM Conference on Management of Data (SIGMOD)*, pages 115–130, 2018.

[41] J. Zhang, G. Cormode, C. M. Procopiuc, D. Srivastava, and X. Xiao. PrivBayes: Private data release via Bayesian networks. *TODS*, 42, 2017.

[42] J. Zhang, X. Xiao, and X. Xie. Privtree: A differentially private algorithm for hierarchical decompositions. In *SIGMOD*, 2016.

[43] X. Zhang, R. Chen, J. Xu, X. Meng, and Y. Xie. Towards accurate histogram publication under differential privacy. In *SDM*, 2014.