GUPT: Privacy Preserving Data Analysis Made Easy

Prashanth Mohan
UC Berkeley
prmohan@cs.berkeley.edu

Abhradeep Thakurta*
Pennsylvania State University
azg161@cse.psu.edu

Elaine Shi
UC Berkeley
elaines@cs.berkeley.edu

Dawn Song
UC Berkeley
dawnsong@cs.berkeley.edu

David E. Culler
UC Berkeley
culler@cs.berkeley.edu

ABSTRACT

It is often highly valuable for organizations to have their data analyzed by external agents. However, any program that computes on potentially sensitive data risks leaking information through its output. Differential privacy provides a theoretical framework for processing data while protecting the privacy of individual records in a dataset. Unfortunately, it has seen limited adoption because of the loss in output accuracy, the difficulty in making programs differentially private, lack of mechanisms to describe the privacy budget in a programmer's utilitarian terms, and the challenging requirement that data owners and data analysts manually distribute the limited privacy budget between queries.

This paper presents the design and evaluation of a new system, GUPT, that overcomes these challenges. Unlike existing differentially private systems such as PINQ and Airavat, it guarantees differential privacy to programs not developed with privacy in mind, makes no trust assumptions about the analysis program, and is secure to all known classes of side-channel attacks.

GUPT uses a new model of data sensitivity that degrades privacy of data over time. This enables efficient allocation of different levels of privacy for different user applications while guaranteeing an overall constant level of privacy and maximizing the utility of each application. GUPT also introduces techniques that improve the accuracy of output while achieving the same level of privacy. These approaches enable GUPT to easily execute a wide variety of data analysis programs while providing both utility and privacy.

Categories and Subject Descriptors

D.4.6 [**Operating Systems**]: Security and Protection; H.3.5 [**Information storage and retrieval**]: Online Information Services

Keywords

Algorithms, Differential Privacy, Data Mining, Security

*Part of this work was done while visiting UC Berkeley.

SIGMOD '12, May 20–24, 2012, Scottsdale, Arizona, USA.
Copyright 2012 ACM 978-1-4503-1247-9/12/05 ...$10.00.

1. INTRODUCTION

Organizations frequently allow third parties to perform business analytics and provide services using aggregated data. For instance, social networks have enabled many applications to be built on social data providing services such as gaming, car-pooling and online deals. Retailers often share data about their customers' purchasing behavior with product merchants for more effective and targeted advertising but do not want to reveal individual customers. While sharing information can be highly valuable, companies provide limited access to this data because of the risk that an individual's privacy would be violated. Laws such as the *Health Insurance Portability and Accountability Act (HIPAA)* have considered the dangers of privacy loss and stipulate that patient's personally identifiable information should not be shared. Unfortunately, even in datasets with "anonymized" users, there have been cases where user privacy was breached. Examples include the deanonymization of AOL search logs [3], the identification of patients in a Massachusetts hospital by combining the public voters list with the hospital's anonymized discharge list [23] and the identification of the users in an anonymized Netflix prize data using an IMDB movie rating dataset [18].

Often the value in sharing data can be obtained by allowing analysts to run aggregate queries spanning a large number of entities in the dataset while disallowing analysts from being able to record data that pertains to individual entities. For instance, a merchant performing market research to identify their next product would want to analyze customer behavior in retailers' databases. The retailers might be willing to monetize the dataset and share aggregate analytics with the merchant, but would be unwilling to allow the merchant to extract information specific to individual customers in the database. Researchers have invented techniques that ranged from ad-hoc obfuscation of data entries (such as the removal of Personally Identifiable Information) to more sophisticated anonymization mechanisms satisfying privacy definitions like k-anonymity [25] and ℓ-diversity [15]. However, Ganta *et al.* [8] and Kifer [13] showed that practical attacks can be mounted against all these techniques. A recent definition called *differential privacy* [5] formalizes the notion of privacy of an individual in a dataset. Unlike earlier techniques, differentially private mechanisms use statistical bounds to limit the privacy loss of an individual incurred by running statistical queries on datasets. It is designed to perturb the result of a computation in a manner that has little effect on aggregates, yet obscures the data of individual constituents.

While differential privacy has strong theoretical properties, the shortcomings of existing differentially private data analysis systems have limited its adoption. For instance, existing programs cannot be leveraged for private data analysis without modification. The magnitude of the perturbation introduced in the final output is another cause of concern for data analysts. Differential privacy systems operate using an abstract notion of privacy, called the 'privacy budget'. Intuitively a lower privacy budget implies better privacy. However, this unit of privacy does not easily translate into the utility of the program and is thus difficult for data analysts who are not experts in privacy to interpret. Further, analysts would also be required to efficiently distribute this limited privacy budget between multiple queries operating on a dataset. An inefficient distribution of the privacy budget would result in inaccurate data analysis and reduce the number of queries that can be safely performed on the dataset.

We introduce GUPT[1], a platform that allows organizations to allow external aggregate analysis on their datasets while ensuring that data analysis is performed in a differentially private manner. It allows the execution of existing programs with no modifications, eliminating the expensive and demanding task of rewriting programs to be differentially private. GUPT enables data analysts to specify a desired output accuracy rather than work with an abstract privacy budget. Finally, GUPT automatically parallelizes the task across a cluster ensuring scalability for concurrent analytics. We show through experiments on real datasets that GUPT overcomes many shortcomings of existing differential privacy systems without sacrificing accuracy.

1.1 Contributions

We design and develop GUPT, a platform for privacy-preserving data analytics. We introduce a new model for data sensitivity which applies to a large class of datasets where the privacy requirement of data decreases over time. As we will explain in Section 3.3, using this model is appropriate and allows us to overcome significant challenges that are fundamental to differential privacy. This approach enables us to analyze less sensitive data to get reasonable approximations of privacy parameters that can be used for data queries running on the newer data.

GUPT makes the following technical contributions that make differential privacy usable in practice:

1. **Describing privacy budget in terms of accuracy:** Data analysts are accustomed to the idea of working with inaccurate output (as is the case with data sampling in large datasets and many machine learning algorithms have probabilistic output). GUPT uses the aging model of data sensitivity, to allow analysts to describe the abstract 'privacy budget' in terms of expected accuracy of the final output.
2. **Privacy budget distribution:** GUPT automatically allocates a privacy budget to each query in order to match the data analysts' accuracy requirements. Further, the analyst also does not have to distribute the privacy budget between the individual data operations in the program.
3. **Accuracy of output:** GUPT extends a theoretical differential privacy framework called "sample and aggregate" (described in Section 2.1) for practical applicability. This includes using a novel data resampling technique that reduces the error introduced by the framework's data partitioning scheme. Further, the aging model of data sensitivity allows GUPT to select an optimal partition size that reduces the perturbation added for differential privacy.
4. **Prevent side channel attacks:** GUPT defends against side channel attacks such as the privacy budget attacks, state attacks and timing attacks described in [10].

2. BACKGROUND

Differential privacy places privacy research on a firm theoretical foundation. It guarantees that the presence or absence of a particular record in a dataset will not significantly change the output of any computation on a statistical dataset. An adversary thus learns approximately the same information about any individual record, irrespective of its presence or absence in the original dataset.

Definition 1 (ϵ-differential privacy [5]). *A randomized algorithm $\mathcal{A}$ is ϵ-differentially private if for all datasets $T, T' \in \mathcal{D}^n$ differing in at most one data record and for any set of possible outputs $\mathcal{O} \subseteq$ Range$(\mathcal{A})$, $\Pr[\mathcal{A}(T) \in \mathcal{O}] \leq e^{\epsilon} \Pr[\mathcal{A}(T') \in \mathcal{O}]$. Here $\mathcal{D}$ is the domain from which the data records are drawn.*

The privacy parameter ϵ, also called the *privacy budget* [16], is fundamental to differential privacy. Intuitively, a lower value of ϵ implies stronger privacy guarantee and a higher value implies a weaker privacy guarantee while possibly achieving higher accuracy.

2.1 Sample and Aggregate

Algorithm 1 Sample and Aggregate Algorithm [24]

```
Input: Dataset $T \in \mathbb{R}^n$, length of the dataset $n$, privacy
    parameters $\epsilon$, output range $(\min, \max)$.
Let $\ell = n^{0.4}$
Randomly partition $T$ into $\ell$ disjoint blocks $T_1, \cdots, T_\ell$.
for $i \in \{1, \cdots, \ell\}$ do
    $O_i \leftarrow$ Output of user application on dataset $T_i$.
    If $O_i > \max$, then $O_i \leftarrow \max$.
    If $O_i < \min$, then $O_i \leftarrow \min$.
end for
$A \leftarrow \frac{1}{\ell}\sum_{i=1}^{\ell} O_i + \mathsf{Lap}(\frac{|\max - \min|}{\ell \cdot \epsilon})$
```

GUPT leverages and extends the "sample and aggregate" framework [24, 19] (SAF) to design a practical and usable system which will guarantee differential privacy for arbitrary applications. Given a statistical estimator $\mathcal{P}(T)$, where T is the input dataset , SAF constructs a differentially private statistical estimator $\hat{\mathcal{P}}(T)$ using $\mathcal{P}$ as a black box. Moreover, theoretical analysis guarantees that the output of $\hat{\mathcal{P}}(T)$ converges to that of $\mathcal{P}(T)$ as the size of the dataset T increases.

As the name "sample and aggregate" suggests, the algorithm first partitions the dataset into smaller subsets; *i.e.*, $\ell = n^{0.4}$ blocks (call them $T_1, \cdots, T_\ell$) (see Figure 1). The analytics program $\mathcal{P}$ is applied on each of these datasets T_i and the outputs O_i are recorded. The O_i's are now clamped to within an output range that is either provided by the analyst or inferred using a range estimator function. (Refer to

[1] GUPT is a Sanskrit word meaning 'Secret'.

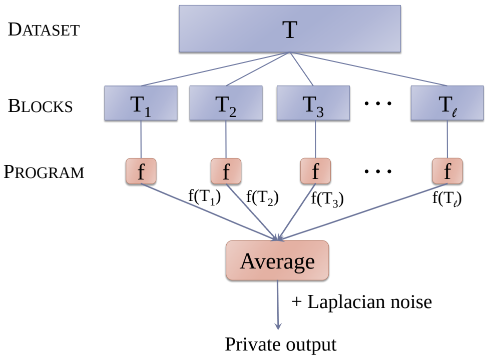

Figure 1: An instantiation of the Sample and Aggregate Framework [24].

Section 4.1 for more details.) Finally, a differentially private average of the O_i's is calculated by adding Laplace noise (scaled according to the output range). This noisy final output is now differentially private. The complete algorithm is provided in Algorithm 1. Note that the choice of number of blocks $\ell = n^{0.4}$ is from [24], used here for completeness. For improved choices of ℓ, see Section 4.3.

GUPT extends the conventional SAF described above in the following ways: i) *Resampling:* GUPT introduces the use of data resampling to improve the experimental accuracy of SAF, without degrading the privacy guarantee; ii) *Optimal block allocation:* GUPT further improves experimental accuracy by finding the better block sizes (as compared to the default choice of $n^{0.6}$) using the *aging of sensitivity* model explained in Section 3.3.

2.2 Related Work

A number of advances in differential privacy have sought to improve the accuracy of very specific types of data queries such as linear counting queries [14], graph queries [12] and histogram analysis [11]. A recent system called PASTE [21] allows queries on time series data where the data is stored on distributed nodes and no trust is laid on the central aggregator. In contract to PASTE, GUPT trusts the aggregator with storing all of the data but provides a flexible system that supports many different types of data analysis programs.

While systems tailored for specific tasks could potentially achieve better output accuracy, GUPT trades this for the generality of the platform. We show through experimental results that GUPT achieves reasonable accuracy for problems like clustering and regression, and can even perform better than the existing customized systems.

Other differential privacy systems such as PINQ [16] and Airavat [22] have also attempted to operate on a wide variety of data queries. PINQ (Privacy INtegrated Queries) proposed programming constructs which enable application developers to write differentially private programs using basic functional building blocks of differential privacy (*e.g.*, *exponential mechanism* [17], *noisy counts* [5] etc.). It *does not* consider the application developer to be an adversary. It further requires the developers to rewrite the application to make use of the PINQ primitives. On the other hand, Airavat was the first system that attempted to run unmodified programs in a differentially private manner. It however required the programs to be written for the Map-Reduce programming paradigm [4]. Further, Airavat only considers the map program to be an "untrusted" computation while the reduce program is "trusted" to be implemented in a differentially private manner. In comparison, GUPT allows for the private analysis of a wider range of unmodified programs. GUPT also introduces techniques that allow data analysts to specify their privacy budget in units of output accuracy. Section 7.3 presents a detailed comparison of GUPT with PINQ, Airavat and the sample and aggregate framework.

Similar to iReduct [28], GUPT introduces techniques that reduce the relative error (in contrast to absolute error). Both systems use a smaller privacy budget for programs that produce larger outputs, as the relative error would be small as compared programs that generate smaller values for the same absolute error. While iReduct optimizes the distribution of privacy budget across multiple queries, GUPT matches the relative error to the privacy budget of individual queries.

3. PROBLEM SETUP

There are three logical parties:

1. The *analyst/programmer*, who wishes to perform aggregate data analytics over sensitive datasets. Our goal is to make GUPT easy to use for an average programmer who is not a privacy expert.
2. The *data owner*, who owns one or more datasets, and would like to allow analysts to perform data analytics over the datasets without compromising the privacy of users in the dataset.
3. The *service provider*, who hosts the GUPT service.

The separation between these parties is logical; in reality, either the data owner or a third-party cloud service provider could host GUPT.

Trust assumptions: We assume that the data owner and the service provider are *trusted*, and that the analyst is *untrusted*. In particular, the programs supplied by the analyst may act maliciously and try to leak information. GUPT defends against such attacks using the security mechanisms proposed in Section 6.

3.1 GUPT Overview

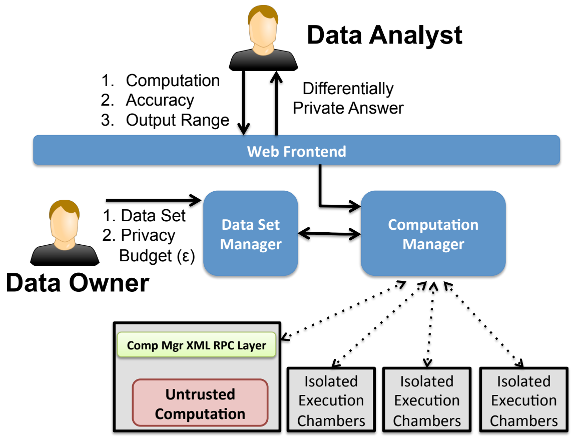

Figure 2: Overview of GUPT's Architecture

The building blocks of GUPT is shown in Figure 2:

- The *dataset manager* is a database that registers instances of the available datasets and maintains the available privacy budget.

- The *computation manager* instantiates computations and seamlessly pipes data from the dataset to the appropriate instances.
- The *isolated execution chambers* isolate and prevent any malicious behavior by the computation instances.

These building blocks allow the principals of the system to easily interact with many parameters being automatically optimized or are left as optional arguments for experts.

Interface with the data owner: (a) A multi-dimensional **dataset** (such as a database table) that for the purpose of our discussion, we assume is a collection of real valued vectors , (b) a total **privacy budget** that can be allocated for computations on the dataset and (c) *[Optional]* **input attribute ranges**, *i.e.* the lower and upper bound for each dimension of the data. A detailed discussion on these bounds is presented in Section 4.1.

For privacy reasons, the input ranges provided should not contain any sensitive information. For example, it is reasonable to expect that a majority of the household's annual income should fall within the range [0; 500,000] dollars, thus it is not considered sensitive. On the other hand if the richest household's income is used as the upper bound private information could be leaked. In this case, a public information source such as the national GDP could be used.

Interface with the analyst: (a) **Data analytics program**, (b) a **reference to the data set** in the dataset manager, (c) either a **privacy budget** or the desired **accuracy** for the final answer and finally (d) an **output range** or a **helper function** for estimating the output range.

A key requirement for the analytics program is that it should be able to run on *any* subset of the original dataset. As GUPT executes the application in a black-box fashion, the program may also be provided as a binary executable.

Privacy budget distribution: In order to guarantee ϵ-differential privacy for a dataset T, the privacy budget should be distributed among the various applications (call them $\mathcal{A}_1, \cdots, \mathcal{A}_k$) that operate on T. A *composition lemma* [5] states that if $\mathcal{A}_1, \cdots, \mathcal{A}_k$ guarantee $\epsilon_1, \cdots, \epsilon_k$-differential privacy respectively, then T is ϵ-differential private, where $\epsilon = \sum_{i=1}^{k} \epsilon_i$. Thus judicious allocation of the privacy budget is extremely important. Unlike existing differential privacy solutions, GUPT relieves the analyst and the data owner from that task of distributing this limited privacy budget between multiple data analytics programs.

3.2 Privacy and Utility Guarantees

GUPT guarantees ϵ-differential privacy to the final output. It provides similar utility guarantees as the original sample and aggregate algorithm from [24]. This guarantee applies to the queries satisfying "approximate[2] normality" condition defined by Smith [24], who also observed that a wide class of queries satisfy this normality condition. Some of the examples being various maximum-likelihood estimators and estimators for regression problems. The formal utility guarantees are deferred to Appendix A.

GUPT provides the same level of privacy for queries that are not approximately normal. Reasonable utility could be expected even from queries that are not approximately normal, even though no theoretical guarantees are provided.

3.3 Aging of Sensitivity

In real life datasets, the potential privacy threat for each record is different. A privacy mechanism that considers this can obtain good utility while satisfying the privacy constraints.

We introduce a new model called *aging of sensitivity* of data where older data records are considered to have lower privacy requirements. GUPT uses this model to optimize some parameters of the sample and aggregate framework like *block size* (Section 4.3) and *privacy budget allocation* (Section 5.1). Consider the following motivating example:

Example 1. *Let $T_{70\ yrs}$ and T_{now} be two datasets containing the ages of citizens in a particular region 70 years earlier and at present respectively. It is conceivable that the privacy threat to $T_{70\ yrs}$ is much lower as many of the participating population may have deceased. Although $T_{70\ yrs}$ might not be as useful as T_{now} for learning specific trends about the current population, it can be used to learn some general concepts about T_{now}. For example, a crude estimate of the maximum age present in T_{now} can be obtained from $T_{70\ yrs}$.*

GUPT estimates such general trends in data distribution and uses them to optimize the performance of the system. The optimization results in a significant reduction in error. More precisely, the aged data is used for the following: i) to estimate an optimal block size for use in the sample and aggregate framework, ii) to identify the minimum privacy budget needed to estimate the final result within a given accuracy bound, and iii) to appropriately distribute the privacy budget ϵ across various tasks and queries.

For simplicity of exposition, the particular *aging* model in our analysis is that a constant fraction of the dataset has *completely aged out*, *i.e.* the privacy of the entries in this constant fraction is no more of a concern. In reality, if the aged data is still weakly privacy sensitive, then it is possible to privately estimate these parameters by introducing an appropriate magnitude of noise into these calculations. The weak privacy of aged data allows us to keep the noise low enough such that the estimated parameters are still useful. Existing techniques [1, 26] have attempted to use progressive aggregation of old data in order to reduce its sensitivity. The use of differentially private operations for aggregation can potentially be exploited to generate our training datasets. The use of these complementary approaches offer exciting opportunities that have not been explored in this paper.

It is important to mention that GUPT *does not* require the *aging* model for default functionality. The default parameter choices allow it work well in a generic setting. However, our experiments show that the *aging* model provides an additional improvement in performance.

4. ACCURACY IMPROVEMENTS

The sample and aggregate framework (Algorithm 1) introduces two sources of error:

- *Estimation error:* This arises because the query is evaluated on a smaller data block, rather than the entire dataset. Typically, the larger the block size, the smaller the estimation error.
- *Noise:* Another source of error is due to the Laplace noise introduced to guarantee differential privacy.

[2] By approximately normal statistic we refer to the *generic* asymptotically normal statistic in Definition 2 of [24].

Intuitively, the larger the number of blocks, the lower the sensitivity of the aggregation function – since the aggregation function has sensitivity $\frac{s}{\ell}$, where s denotes the output range of each block, and ℓ denotes the number of blocks. As a result, given a fixed privacy parameter ϵ, with a larger number of blocks, the magnitude of the Laplace noise is lowered.

GUPT uses two strategies (*resampling* and *selecting the optimal block size*) to reduce these types of errors. Before delving into details of the techniques, the following section explains how the output range for a given analysis program is computed. This range is used to decide the amount of noise to be added to the final output.

4.1 Output Range Estimation

The sample and aggregate framework described in Algorithm 1 does not describe a mechanism to obtain the range within which the output can lie. This is needed to estimate the noise that should be added for differential privacy. GUPT implements this requirement by providing the following mechanisms:

1. `GUPT-tight`: The analyst specifies a tight range for the *output.*
2. `GUPT-loose`: The analyst only provides a loose range for the *output.* In this case, the computation is run on each data block and their outputs are recorded. A differentially private percentile estimation algorithm [24] is then applied on the set of outputs to privately compute the 25-th and the 75-th percentile values. These values are used as the range of the output and are supplied to Algorithm 1.
3. `GUPT-helper`: The analyst could also provide a range translation function. If either (a) *input* range is not present or (b) only very loose range for the *input* (e.g., using the national GDP as an upper-bound on annual household income) is available, then GUPT runs the same differentially private percentile estimation algorithm on the inputs to privately compute the 25-th and the 75-th percentile (a.k.a, lower and upper quartiles) of the inputs. This is used as a tight approximation of the input range. The analyst-supplied range translation function is then invoked to convert the "tight" input range into an estimate of the output range.

Our experiments demonstrate that one can get good results for a large class of problems using the noisy lower and upper quartiles as approximations of the output range. If the input dataset is multi-dimensional, the range estimation algorithm is run independently for each dimension. Note that the choice of 25-th and 75-th percentile above is somewhat arbitrary. In fact, one can choose a larger inter-percentile range (*e.g.*, 10-th and 90-th percentile) if there are more data samples. However, this does not affect the asymptotic behavior of the algorithm.

4.2 Resampling

The variance in the final output is due to two sources of randomness: i) partitioning the dataset into blocks and ii) the Laplace noise added. The following resampling technique[3] can be used to reduce the variance due to partitioning the dataset into blocks. Instead of requiring each data entry to reside in exactly one block (as described in the original sample and aggregate framework [24]), each data entry can now reside in multiple blocks. The *resampling factor* γ denotes the number of blocks in which each data entry resides.

If the number of records in the dataset is n, block size is β and each data entry resides in γ blocks, it is easy to see that the number of blocks $\ell = \gamma n/\beta$. To incorporate resampling, we make the following modifications to Algorithm 1. Lines 1 and 2 are modified as follows. Consider $\ell = \gamma n/\beta$ bins of size β each. The i^{th} entry from the dataset T is picked and randomly placed into γ bins that are not full. This process is performed for all the entries in the dataset T. In Line 8, the Laplace noise is changed to $\mathsf{Lap}(\frac{\beta|\max - \min|}{n\cdot\epsilon})$. The rest of Algorithm 1 is left intact.

The main benefit of using resampling is that it reduces the variance due to partitioning the dataset into blocks without increasing the noise needed for the same level of privacy.

CLAIM 1. *With the same privacy level ϵ, resampling with any $\gamma \in \mathbb{Z}^+$, does not increase the Laplace noise being added (for fixed block size β).*

PROOF. Since each record appears in γ blocks, a Laplace noise of magnitude $O(\frac{\gamma s}{\epsilon\ell}) = O(\frac{s\beta}{\epsilon n})$ should be added to preserve ϵ-differential privacy. This means that once the block size is fixed, the noise is independent of the factor γ. □

Intuitively, the benefit from resampling is that the variance due to partitioning of the dataset into blocks is reduced without increasing the Laplace noise (added in Step 8 of Algorithm 1) needed for the same level of privacy with the inclusion of $\gamma > 1$. Consider the following example to get a better understanding of the intuition.

EXAMPLE 2. *Let T be a dataset (with n records) of the ages of a population and max be the maximum age in the dataset. The objective is to find the average age (Av) in this dataset. Let $\hat{Av}$ be the average age of a dataset formed with $n^{0.6}$ uniformly random samples drawn from T with replacement. The expectation of $\hat{Av}$ equals the true average Av. However, the variance of $\hat{Av}$ will not be zero (unless all the entries in T are same). Let $\mathcal{O} = \frac{1}{\psi}\sum_{i=1}^{\psi}\hat{Av}(i)$, where $\hat{Av}(i)$ is the i-th independent computation of $\hat{Av}$ mentioned above and ψ is some constant. Notice that $\mathcal{O}$ has the same expected value as $\hat{Av}$ but the variance has reduced by a factor of ψ. Hence, resampling reduces the variance in the final output $\mathcal{O}$ without introducing bias.*

The above example is a simplified version of the actual resampling process. In the actual resampling process each data block of size $n^{0.6}$ is allowed to have only one copy of each data entry of T. However, even with an inaccurate representation, the above example captures the essence of the underlying phenomenon.

In practice, the resampling factor γ is picked such that it is reasonably large, without increasing the computation overhead significantly. Notice that the increase in accuracy with the increase of γ becomes insignificant beyond a threshold.

4.3 Selecting the Optimal Block Size

In this section, we address the following question: *Given a fixed privacy budget, how do we pick the optimal block size to maximize the accuracy of the private output?*

Observe that increasing the block size β increases the noise magnitude, but reduces the estimation error. Therefore, the question boils down to: *how do we select an optimal block*

[3] A variant of this technique was suggested by Adam Smith.

size that will allow us to balance the estimation error and the noise? The following example elucidates why answering the above question is important.

Example 3. *Consider the same age dataset T used in Example 2. If our goal is to find the average of the entries in T while preserving privacy, then it can be observed that (ignoring resampling) the optimal size of each block is one which attains the optimal balance between the estimation error and noise. If the block size was one, then the expected error will be $O(1/n)$, where n is the size of the dataset. However, if we use the default block size (*i.e., *$n^{0.6}$), the expected error will be $O(1/n^{0.4})$ which is much higher.*

As a result getting the optimal block size based on the specific task helps to reduce the final error to a large extent. The optimal block size varies from problem to problem. For example, in k-means clustering or logistic regression the optimal block size has to be much larger than one.

Let $\ell = n^\alpha$ be the optimal number of blocks, where α is a parameter to be ascertained. Hence, $n^{1-\alpha}$ is the block size. (For the simplicity of exposition we do not consider resampling.) Let $f : \mathbb{R}^{k\times n} \to \mathbb{R}$ be the query which is to be computed on the dataset T. Let the data blocks be represented as $T_1, \cdots, T_\ell$. Let s be the sensitivity of the query, *i.e.,* the absolute value of maximum change that any $f(T_i)$ can have if any one entry of T is changed. With the above parameters in place, the ϵ-differentially private output from the sample and aggregate framework is

$$\hat{f}(T) = \frac{1}{n^\alpha}\sum_{i=1}^{n^\alpha} f(T_i) + \mathsf{Lap}(\frac{s}{\epsilon n^\alpha}) \qquad (1)$$

Assume that the entries of the dataset T are drawn i.i.d, and that there exists a dataset T^{np} (with entries drawn i.i.d. from the same distribution as T) whose privacy we do not care for under the *aging of sensitivity* model. Let n_{np} be the number of entries in T^{np}. We will use the *aged* dataset T^{np} to estimate the optimal block size. Specifically, we partition T^{np} into blocks of size $\beta = n^{1-\alpha}$. The number of blocks ℓ_{np} in T^{np} is therefore $\ell_{\text{np}} = \frac{n_{\text{np}}}{n^{1-\alpha}}$. Notice that ℓ_{np} is a function of α, whose optimal value has to be found. Also note that the minimum value of α must satisfy the following inequality: $n_{\text{np}} \geq n^{1-\alpha}$.

One possible approach for achieving a good value of α is by minimizing the empirical error in the final output. The empirical error in the output of $\hat{f}$ is defined as

$$\underbrace{\left|\frac{1}{\ell_{np}}\sum_{i=1}^{\ell_{np}} f({T_i}^{np}) - f(T^{np})\right|}_{A} + \underbrace{\frac{\sqrt{2}s}{\epsilon n^\alpha}}_{B} \qquad (2)$$

Here A characterizes the estimation error, and B is due to the Laplace noise added. We can minimize Equation 2 w.r.t. α when $\alpha \in [1 - \log n_{\text{np}}/\log n, 1]$. Conventional techniques like *hill climbing* can be used to obtain a local minima.

The α computed above is used to obtain the optimal number of blocks. Since, the computation involves only the non-private database T^{np}, there is no effect on overall privacy.

5. PRIVACY BUDGET MANAGEMENT

In differential privacy, the analyst is expected to specify the privacy goals in terms of an abstract privacy budget ϵ. The analyst performs the data analysis task optimizing it for accuracy goals and the availability of computational resources. These metrics do not directly map onto the abstract privacy budget. It should be noted that even a privacy expert might be unable to map the privacy budget into accuracy goals for arbitrary problems. In this section we describe mechanisms that GUPT use to convert the accuracy goals into a privacy budget and to efficiently distribute a given privacy budget across different analysis tasks.

5.1 Estimating Privacy Budget for Accuracy Goals

In this section we seek to answer the question: *How can* GUPT *pick an appropriate ϵ, given a fixed accuracy goal?* Specifically, we wish to minimize the ϵ parameter to maximally preserve the privacy budget. It is often more intuitive to specify an accuracy goal rather than a privacy parameter ϵ, since accuracy relates to the problem at hand.

Similar to the previous section, we assume the existence of an *aged* dataset T^{np} (drawn from the same distribution as the original dataset T) whose privacy is not a concern.

Consider an analyst who wishes to guarantee an accuracy ρ with probability $1 - \delta$, *i.e.* the output should be within a factor ρ of the true value. We wish to estimate an appropriate ϵ from an aged data set T^{np} of size n_{np}. Let β denote the desired block size. To estimate ϵ, first the permissible standard deviation in the output σ is calculated for a specified accuracy goal ρ and then the following optimization problem is solved. Solve for ϵ, under the following constraints: 1) the expression in Equation 3 equals σ^2, 2) $\alpha = \max\{0, \log(n/\beta)\}$.

$$\underbrace{\frac{1}{n^\alpha}\left(\frac{1}{\ell_{\text{np}}}\sum_{i=1}^{\ell_{\text{np}}}\left(f(T_i^{\text{np}}) - \frac{1}{\ell_{\text{np}}}\sum_{i=1}^{\ell_{\text{np}}} f(T_i^{\text{np}})\right)^2\right)}_{C} + \underbrace{\frac{2s^2}{\epsilon^2 n^{2\alpha}}}_{D} \qquad (3)$$

In Equation 3, C denotes the variance in the estimation error and D denotes the variance in the output due to noise.

To calculate σ from the accuracy goal, we can rely on Chebyshev's inequality: $\Pr[|\hat{f}(T) - \mathbb{E}(f(T_i))| > \phi\sigma] < \frac{1}{\phi^2}$. Furthermore, assuming that the query f is a approximately normal statistic, we have $|\mathbb{E}(f(T_i)) - \text{Truth}| = O(1/\beta)$. Therefore: $\Pr[|\hat{f}(T) - \text{Truth}| > \phi\sigma + O(1/\beta)] < (1/\phi^2)$ To meet the output accuracy goal of ρ with probability $1 - \delta$, we set $\sigma \simeq \sqrt{\delta}|1 - \rho|f(T^{\text{np}})$. Here, we have assumed that the *true* answer is $f(T^{\text{np}})$ and $1/\beta \ll \sigma/\sqrt{\delta}$.

Since in the above calculations we assumed that the true answer is $f(T^{\text{np}})$, an obvious question is "why not output $f(T^{\text{np}})$ as the answer?". It can be shown that in a lot of cases, the private output will be much better than $f(T^{\text{np}})$.

If the assumption that $1/\beta \ll \sigma/\sqrt{\delta}$ does not hold, then the above technique for selecting privacy budget would produce suboptimal results. This however does not compromise the privacy properties that GUPT wants to maintain, as it explicitly limit the total privacy budget allocated for queries accessing a particular dataset.

5.2 Automatic Privacy Budget Distribution

Differential privacy is an alien concept for most analysts. Further, the proper distribution of the limited privacy budget across multiple computations require significant mathematical expertise. GUPT eliminates the need to manually distribute privacy budget between tasks. The following example will highlight the requirement of an efficient privacy budget distribution rather than distributing equally among various tasks.

EXAMPLE 4. *Consider the same age census dataset T from Example 2. Suppose we want to find the average age and the variance present in the dataset while preserving differential privacy. Assume that the maximum possible human age is max and the minimum age is zero. Assume that the non-private variance is computed as $\frac{1}{n}\sum_{i=1}^{n}(T(i)-Av_{priv})^2$, where Av_{priv} is the private estimate of the average and n is the size of T. If an entry of T is modified, the average Av changes by at most max/n, however the variance can change by at most max^2/n.*

Let ϵ_1 and ϵ_2 be the privacy level expected for average and variance respectively, with the total privacy budget being $\epsilon = \epsilon_1 + \epsilon_2$. Now, if it is assumed that $\epsilon_1 = \epsilon_2$, then the error in the computation of variance will be in the order of max more than in the computation of average. Whereas if privacy budget were distributed as $\epsilon_1 : \epsilon_2 = 1 : max$, then the noise in both the average and variance will roughly be the same.

Given privacy budget of ϵ and we need to use it for computing various queries $f_1, \cdots, f_m$ privately. If the private estimation of query f_i requires ϵ_i privacy budget, then the total privacy budget spent will be $\sum_{i=1}^{m}\epsilon_i$ (by *composition* property of differential privacy [5]). The privacy budget is distributed as follows. Let $\frac{\zeta_i}{\epsilon_i}$ be the standard deviation of the Laplace noise added by GUPT to ensure privacy level ϵ_i. Allocate the privacy budget by setting $\epsilon_i = \frac{\zeta_i}{\sum_{i=1}^{m}\zeta_i}\epsilon$. The rationale behind taking such an approach is that usually the variance in the computation by GUPT is mostly due to the variance in the Laplace noise added. Hence, distributing ϵ across various tasks using the technique discussed above ensures that the variance due to Laplace noise in the private output for each f_i is the same.

6. SYSTEM SECURITY

GUPT is designed as a hosted platform where the analyst is *not* trusted. It is thus important to ensure that the untrusted computation should not be able to access the datasets directly. Additionally, it is important to prevent the computation from exhausting resources or compromising the service provider. To this end, the "computation manager" is split into a server component that interacts with the user and a client component that runs on each node in the cluster. The trusted client is responsible for instantiating the computation in an isolated execution environment. The isolated environment ensures that the computation can only communicate with a trusted forwarding agent which sends the messages to the computation manager.

6.1 Access Control

GUPT uses a mandatory access control framework (MAC) to ensure that (a) communication between different instances of the computation is disallowed and (b) each instance of the computation can only store state (or modify data) within its own scratch space. This is the only component of GUPT that depends upon a platform dependent implementation. On Linux, the LSM framework [27] has enabled many MAC frameworks such as SELinux and AppArmor to be built. GUPT defines a simple AppArmor policy for each instance of the computation, setting its working directory to a temporary scratch space that is emptied upon program termination. AppArmor does not yet allow fine grained control to limit network activity to individual hosts and ports. Thus the "computation manager" is split into a server and client component. The client component of the computation manager allows GUPT to disable all network activity for the untrusted computation and restrict IPC to the client.

We determined an empirical estimate of the overhead introduced by the AppArmor sandbox by executing an implementation of k-means clustering on GUPT $6,000$ times. We found that *the sandboxed version of* GUPT *was only* 0.012 *times slower than the non-sandboxed version (overhead of* 1.26%*)*.

6.2 Protection against side-channel attacks

Haeberlen *et al.* [10] identified three possible side-channel attacks against differentially private systems. They are i) *state attack*, ii) *privacy budget attack*, and iii) *timing attack*. GUPT is not vulnerable to any of these attacks.

State attacks: If the adversarial program can modify some internal state (*e.g.,* change the value of a static variable) when encountered with a specific data record. An adversary can then look at the state to figure out whether the record was present in the dataset. Both PINQ (in it's current implementation) and Airavat are vulnerable to state attacks. However, it is conceivable that operations can be isolated using .NET *AppDomains* in PINQ to isolate data computations. Since GUPT executes the complete analysis program (which may be adversarial) in isolated execution chambers and allows the analyst to access only the final differentially private output, state attacks are automatically protected against.

Privacy budget attack: In this attack, on encountering a particular record, the adversarial program issues additional queries that exhausts the remaining privacy budget. [10] noted that PINQ is vulnerable to this attack. GUPT protects against privacy budget attacks by managing the privacy budget itself, instead of letting the untrusted program perform the budget management.

Timing attacks: In a timing attack, the adversarial program could consume an unreasonably long amount of time to execute (perhaps get into an infinite loop) when encountered with a specific data record. GUPT protects against this attack by setting a predefined bound on the number of cycles for which the data analyst program runs on each data block. If the computation on a particular data block completes before the predefined number of cycles, then GUPT waits for the remaining cycles before producing an output from that block. In case the computation exceeds the predefined number of cycles, the computation is killed and a constant value within the expected output range is produced as the output of the program running on the data block under consideration.

Note that with the scheme above, the runtime of GUPT is independent of the data. Hence, the number of execution cycles does not reveal any information about the dataset. The proof that the final output is still differentially private under this scheme follows directly from the privacy guarantee of the sample and aggregate framework and the fact that a change in one data entry can affect only one data block (ignoring resampling). Thus GUPT is not vulnerable to timing attacks. Both PINQ and Airavat do not protect against timing attacks [10].

7. EVALUATION

For each data analysis program, the program binary and interfaces with the GUPT "computation manager" should be provided. For arbitrary binaries, a lean wrapper program can be used for marshaling data to/from the format of the computation manager.

In this section, we show using results from running common machine learning algorithms (such as k-means clustering and logistic regression on a life sciences dataset) that GUPT does not significantly affect the accuracy of data analysis. Further, we show that GUPT not only relieves the analysts from the burden of distributing a privacy budget between data transformation operations, it also manages to provide superior output accuracy. Finally, we show through benchmarks the scalability of the GUPT architecture and the benefits of using aged data to estimate optimal values of privacy budget and block sizes.

7.1 Case Study: Life Sciences dataset

We evaluate the efficacy of GUPT using the `ds1.10` life sciences dataset taken from `http://komarix.org/ac/ds` as a motivating example for data analysis. This dataset contains the top 10 principal components of chemical/biological compounds with each of the $26,733$ rows representing different compounds. Additionally, the reactivity of the compound is available as an additional component. A k-means clustering experiment enables us to cluster compounds with similar features together and logistic regression builds a linear classifier for the experiment (*e.g.,* predicting carcinogens). It should be noted that these experiments only provide estimates as the final answer, *e.g.,* the cluster centroids in the case of k-means. We show in this section that the perturbation introduced by GUPT only affects the final result marginally.

7.1.1 Output Accuracy

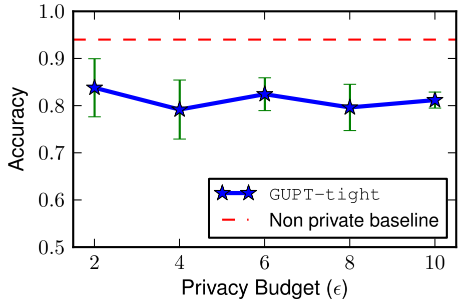

Figure 3: Effect of privacy budget on the accuracy of prediction using Logistic Regression on the life sciences dataset

As mentioned in Section 4, any analysis performed using GUPT has two sources of error – (a) an estimation error, introduced because each instance of the computation works on a smaller subset of the data and (b) Laplace noise that is added in order to guarantee differential privacy. In this section, we show the effect of these errors when running logistic regression and k-means on the life sciences dataset.

GUPT can be used to run existing programs with no modifications, thus drastically reducing the overhead of writing privacy preserving programs. Analysts using GUPT are free to use their favorite software packages written in any language. To demonstrate this property, we evaluate black box implementations of logistic regression and k-means clustering on the life sciences dataset.

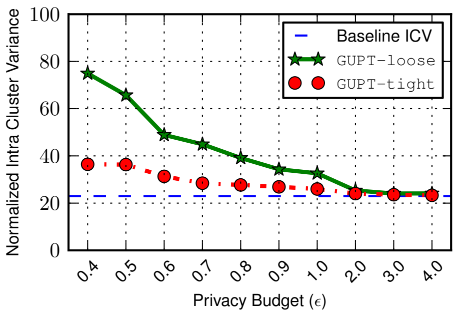

Figure 4: Intra-cluster variance for k-means clustering on the life sciences dataset

Logistic Regression: The logistic regression software package from Microsoft Research (*Orthant-Wise Limited-memory Quasi-Newton Optimizer for L_1-regularized Objectives*) was used to classify the compounds in the dataset as carcinogens and non-carcinogens. Figure 3 shows the accuracy of GUPT for different privacy budgets.

When the package was run on the dataset directly, a baseline accuracy of 94% was obtained. The same package when run using the GUPT framework classified carcinogens with an accuracy between $75 \sim 80\%$. To understand the source of the error, when the non-private algorithm was executed on a data block of size $\frac{n}{n^{0.4}}$ records, the accuracy reduced to 82%. It was thus determined that much of the error stems from the loss of accuracy when the algorithm is run on smaller blocks of the entire dataset reduced. For datasets of increasingly large size, this error is expected to diminish.

k-means Clustering: Figure 4 shows the cluster variance computed from a k-means implementation run on the life sciences dataset. The x-axis is various choices of the privacy budget ϵ, and the y-axis is the normalized Intra-Cluster Variance (ICV) defined as $\frac{1}{n}\sum_{i=1}^{K}\sum_{\vec{x}\in C_i}|\vec{x}-\vec{c}_i|_2^2$, where K denotes the number of clusters, C_i denotes the set of points within the i^{th} cluster, and $\vec{c}_i$ denotes the center of the i^{th} cluster. A standard k-means implementation from the `scipy` python package is used for the experiment.

The k-means implementation was run using GUPT with different configurations for calculating the output range (Section 4.1). For `GUPT-tight`, a tight range for the output is taken to be the exact minimum and the maximum of each attribute (for all 10 attributes). For `GUPT-loose`, a loose output range is fixed as $[\min *2, \max *2]$, where min and max are the actual minimum and maximum for that attribute. Figure 4 shows that with increasing privacy budget ϵ, the amount of Laplace noise added to guarantee differential privacy decreases, thereby reducing the intra-cluster variance, *i.e.* making the answer more accurate. It can also be seen that when GUPT is provided with reasonably tight bounds on the output range (`GUPT-tight`), the output of the k-means experiment is very close to a non-private run of the experiment even for small values of the privacy budget. If only loose bounds are available (`GUPT-loose`), then a larger privacy budget is required for the same output accuracy.

7.1.2 Budget Distribution between Operations

In GUPT, the program is treated as a black box and noise is only added to the output of the entire program.

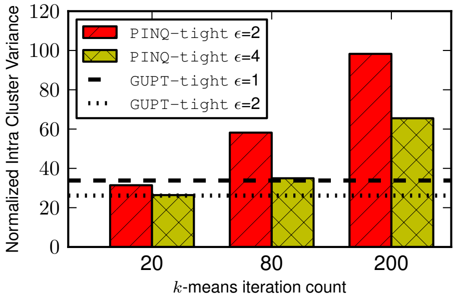

Figure 5: Total perturbation introduced by GUPT does not change with number of operations in the utility function

Thus the number of operations performed in the program itself is irrelevant. A problem with writing specialized differentially private algorithms such as in the case of PINQ is that given a privacy budget ϵ for the task, it is difficult to decide how much ϵ to spend on each query, since it is difficult to determine the number of iterations needed ahead of time. PINQ requires the analyst to pre-specify the number of iterations in order to allocate the privacy budget between iterations. This is often hard to do, since many data analysis algorithms such as PageRank [20] and recursive relation queries [2] require iterative computation until the algorithm reaches convergence. The performance of PINQ thus depends on the ability to accurately predict the number of iterations. If the specified number of iterations is too small, then the algorithm may not converge. On the other hand, if the specified number of iterations is too large, then much more noise than is required will be added which will both slow down the convergence of the algorithm as well as harm its accuracy. Figure 5 shows the effect of PINQ on accuracy when performing k-means clustering on the dataset. In this example, the program output for the dataset converges within a small number of iterations, e.g., $n = 20$. Whereas if a larger number of iterations (e.g., $n = 200$) was conservatively chosen, then PINQ's performance degrades significantly. On the other hand, GUPT produces the same amount of perturbation irrespective of the number of iterations in k-means. Further, it should be noted that PINQ was subjected to a weaker privacy constraint ($\epsilon = 2$ and 4) as compared to GUPT ($\epsilon = 1$ and 2).

7.1.3 *Scalability*

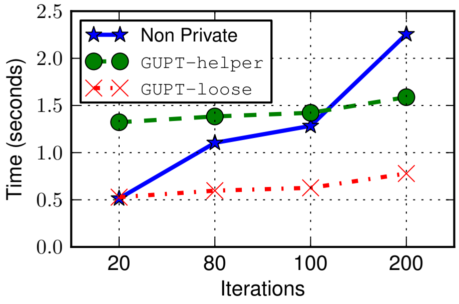

Figure 6: Change in computation time for increased number of iterations in k-means

Using a server with two Intel Xeon 5550 quad-core CPUs and the entire dataset loaded in memory, we compare the execution time of an unmodified (non-private) instance and a GUPT instance of the k-means experiment.

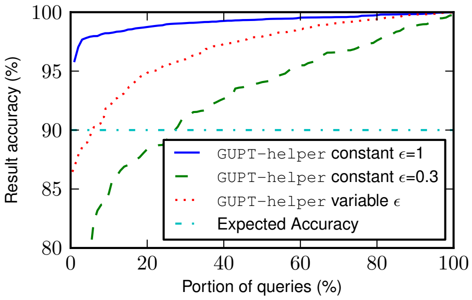

Figure 7: CDF of query accuracy for privacy budget allocation mechanisms

If tight output range (*i.e.*, **GUPT-tight**) is not available, typically, the output range estimation phase of the sample and aggregate framework takes up most of the CPU cycles. When only loose range for the input is available (*i.e.*, **GUPT-helper**), a differentially private percentile estimation is performed on all of the input data. This is a $O(n \ln n)$ operation, n being the number of data records in the original dataset. On the other hand, if even loose range for the output is available (*i.e.*, **GUPT-loose**), then the percentile estimation is performed only on the output of each of the blocks in sample and aggregate framework, which is typically around $n^{0.4}$. This results in significantly reduced run-time overhead. The overhead introduced by GUPT is irrespective of the actual computation time itself. Thus as the computation time increases, the overhead introduced by GUPT diminishes in comparison. Further, there is an additional speed up since each of the computation instances work on a smaller subset of the entire dataset. It should be noted that the reduction in computation time thus achieved could also potentially be achieved by the computational task running without GUPT. Figure 6 shows that the overall completion time of the private versions of the program increases slowly compared to the non-private version as we increase the number of iterations of k-means clustering.

7.2 Using Aged Data

GUPT uses an aged dataset (that is no longer considered privacy sensitive) drawn from a similar distribution as the real dataset. Section 4.3 describes the use of aged data to estimate an optimal block size that reduces the error introduced by data sampling. Section 5.1 describes how data analysts who are not privacy experts can continue to only describe their accuracy goals yet achieve differentially private outputs. Finally, Section 5.2 uses aged data to automatically distribute a privacy budget between different queries on the same data set. In this section, we show experimental results that support the claims made in Sections 4.3 and 5.1.

7.2.1 *Privacy Budget Estimation*

To illustrate the ease with which GUPT can be used by data analysts, we evaluate the efficiency of GUPT by executing queries that are not provided with a privacy budget. We use a census income dataset from the UCI machine learning repository [7] which consists of 32561 entries. The age data from this dataset is used to calculate the average age. A reasonably loose range of $[0, 150]$ was enforced

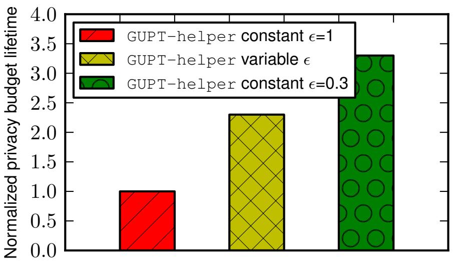

Figure 8: Increased lifetime of total privacy budget using privacy budget allocation mechanism

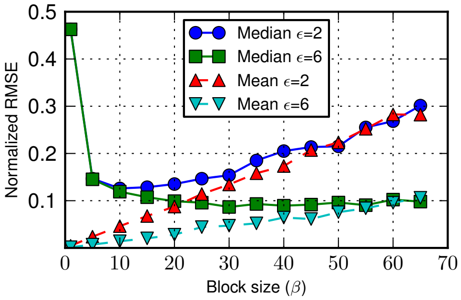

Figure 9: Change in error for different block sizes

on the output whose true average age is 38.5816. Initially, the experiment was run with a constant privacy budgets of $\epsilon = 1$ and $\epsilon = 0.3$. GUPT allows the analyst to provide looser constraints such as "90% result accuracy for 90% of the results" and allocates only as much privacy budget as is required to meet these properties. In this experiment, the 10% of the dataset was assumed to be completely privacy insensitive and was used to estimate ϵ given a pre-determined block size. Figure 7 shows the CDF of the output accuracy both for constant privacy budget values as well as for the accuracy requirement. Interestingly, not only does the figure show that the accuracy guarantees are met by GUPT, but also it shows that if the analyst was to define the privacy budget manually (as in the case of $\epsilon = 1$ or $\epsilon = 0.3$), then either too much or too little privacy budget is used. The privacy budget estimation technique thus has the additional advantage that the lifetime of the total privacy budget for a dataset will be extended. Figure 8 shows that if we were to run the average age query with the above constraints over and over again, GUPT will be able to run 2.3 times more queries than using a constant privacy budget of $\epsilon = 1$.

7.2.2 *Optimal Block Size Estimation*

Section 4.3 shows that the estimation error decreases with an increase in data block size, whereas the noise decreases with an increased number of blocks. The optimal trade off point between the block size and number of data blocks would be different for different queries executed on the dataset. To illustrate the tradeoff, we show results from queries executed on an internet advertisement dataset also from the UCI machine learning repository [7]. Figure 9 shows the normalized root mean square error (from the true value) in estimating the mean and median aspect ratio of advertisements shown on Internet pages with privacy budgets ϵ of 2 and 6. In the case of the "mean" query, since the averaging operation is already performed by the sample and aggregate framework, smaller data blocks would reduce the noise added to the output and thus provide more accurate results. As expected, we see that the ideal block size would be one.

For the "median" query, it is expected that increasing the block size would generate more accurate inputs to the averaging function. Figure 9 shows that when the "median" query is executed with $\epsilon = 2$, the error is minimal for a block size of 10. With increasing block sizes, the noise added to compensate for the reduction in number of blocks would have a dominating effect. On the other hand, when executing the same query with $\epsilon = 6$, the error continues to drop for increased block sizes, as the estimation error dominates the Laplace noise (owing to the increased privacy budget). It is thus clear that GUPT can significantly reduce the total error by estimating the optimal block size for the sample and aggregate framework.

	GUPT	**PINQ**	**Airavat**
Works with unmodified programs	Yes	No	No
Allows expressive programs	Yes	Yes	No
Automated privacy budget allocation	Yes	No	No
Protection against privacy budget attack	Yes	No	Yes
Protection against state attack	Yes	No	No
Protection against timing attack	Yes	No	No

Table 1: Comparison of GUPT, PINQ and Airavat

7.3 Qualitative Analysis

In this section, GUPT is contrasted with both PINQ and Airavat on various fronts (see Table 1 for a summary). We also list the significant changes introduced by GUPT in order to mold the sample and aggregate framework (SAF) [24] into a practically useful one.

Unmodified programs: Because PINQ [16] is an API that provides a set of low-level data manipulation primitives, applications will need to be re-written to perform all operations using these primitives. On the other hand, Airavat [22] implements the Map-Reduce programming paradigm [4] and requires that the analyst splits the user's data analysis program into an "untrusted" map program and a reduce aggregator that is "trusted" to be differentially private.

In contrast, GUPT treats the complete application program as a black box and as a result the entire application program is deemed untrusted.

Expressiveness of the program: PINQ provides a limited set of primitives for data operations. However, if the required primitives are not already available, then a privacy unaware analyst would be unable to ensure privacy for the output. Airavat also severely restricts the expressiveness of the programs that can run in it's framework: a) the "untrusted" map program is completely isolated for each data element and cannot save any global state and b) it restricts the number of key-value pairs generated from the mapper. Many machine learning algorithms (such as clustering and classification) require global state and complex aggregation functions. This would be infeasible in Airavat without placing much of the logic in the "trusted" reducer program.

GUPT places no restriction on the application program, and thus does not degrade the expressiveness of the program.

Privacy budget distribution: As was shown in Section

7.1.2, PINQ requires the analyst to allocate a privacy budget for each operation on the data. An inefficient distribution of the budget either add too much noise or use up too much of the budget. Like GUPT, Airavat spends a constant privacy budget on the entire program. However, neither Airavat nor PINQ provides any support for distributing an aggregate privacy budget across multiple data analysis programs.

Using the *aging of sensitivity* model, GUPT provides an mechanism for efficiently distributing an aggregate privacy budget across multiple data analysis programs.

Side Channel Attacks: Current implementation of PINQ lays the onus of protecting against side channel attacks on the program developer. As was noted in [10], although Airavat protects against privacy budget attacks, it remains vulnerable to state attacks. GUPT defends against both state attacks and privacy budget attacks (see Section 6.2).

Other differences: GUPT extends SAF to use a novel data resampling mechanism to reduce the variance in the output induced via data sub-sampling. Using the *aging of sensitivity* model, GUPT overcomes a fundamental issue in differential privacy not considered previously (to the best of our knowledge): *for an arbitrary data analysis application, how do we describe an abstract privacy budget in terms of utility?* The model also allows us to further reduce the error in SAF by estimating a reasonable block size.

8. DISCUSSION

Approximately normal computation: GUPT does not provide any theoretical guarantees on the accuracy of the outputs, if the applied computation is not approximately normal, or if the data entries are not independently and identically distributed (i.i.d). However, GUPT still guarantees that the differential privacy of individual data records is always maintained. In practice, GUPT provides reasonably accurate results for a lot of queries that do not satisfy approximate normality.

For some real world datasets, such as sensing and streaming data that have temporal correlation and do not satisfy the i.i.d property, the current implementation of GUPT fails to provide reasonable output. In our future work we intend to design GUPT for catering to these kind of datasets.

Ordering of multiple outputs: When data analysis program yields multi-dimensional outputs, the program may not be able to guarantee the same ordering of outputs for different data blocks. In this case, we have to sort the outputs according to some canonical form before applying the averaging operation. For example, in the case of k-means clustering, each block can yield different ordering of the k cluster centers. In our experiments, the cluster centers were sorted according to their first coordinate.

8.1 Limitations

Foreknowledge of output dimension: GUPT assumes that the output dimensions are known in advance. This may however not always be true in practice. For example, Support Vector Machines (SVM) output an indefinite number of support vectors. Unless the dimension is fixed in advance, private information may leak through the dimensionality itself. Another problem is that if the output dimensions for different blocks are different, then performing an average of the outputs will not be meaningful. A partial solution is to clamp or pad the output dimension to a predefined number. We note that this limitation is not unique to GUPT. For example, Airavat [22] also suffers from the same problem in the sense that their mappers have to output a fixed number of `(key, value)` pairs.

Inherits limitations of differential privacy: GUPT inherits some of the issues that are common to many differential privacy mechanisms. For high dimensional data outputs, the privacy budget needs to be split across the multiple outputs. The privacy budget also needs to be split across multiple data mining tasks and users. Given a fixed total privacy budget, the more tasks and queries we divide the budget over, the more noise there is – and at some point, the magnitude of noise may cause the data analysis to become unusable. In Section 5.2, we outlined a potential method for managing privacy budgets. Alternative approaches such as *auctioning of privacy budget* [9] can also be used.

While differential privacy guarantees the privacy of individual records in the dataset, many real world application will want to enforce higher level privacy concepts such as user-level privacy [6]. In cases where multiple records may contain information about the same user, user-level privacy needs to be accounted for accordingly.

9. CONCLUSION AND FUTURE WORK

GUPT makes privacy-preserving data analytics easy for privacy non-experts. The analyst can upload arbitrary data mining programs and GUPT guarantees the privacy of the outputs. We propose novel improvements to the sample and aggregate framework to enhance the usability and accuracy of the data analysis. Through a new model that reduces the sensitivity of the data over time, GUPT is able to represent privacy budgets as accuracy guarantees on the final output. Although this model is not strictly required for the default functioning of GUPT, it improves both usability and accuracy. Through the efficient distribution of the limited privacy budget between data queries, GUPT is able to ensure that more queries that meet both their accuracy and privacy goals can be executed on the dataset. Through experiments on real-world datasets, we demonstrate that GUPT can achieve reasonable accuracy in private data analysis.

Acknowledgements

We would like to thank Adam Smith, Daniel Kifer, Frank McSherry, Ganesh Ananthanarayanan, David Zats, Piyush Srivastava and the anonymous reviewers of SIGMOD for their insightful comments that have helped improve this paper. This project was funded by Nokia, Siemens, Intel through the ISTC for Secure Computing, NSF awards (CPS-0932209, CPS-0931843, BCS-0941553 and CCF-0747294) and the Air Force Office of Scientific Research under MURI Grants (22178970-4170 and FA9550-08-1-0352).

10. REFERENCES

[1] N. Anciaux, L. Bouganim, H. H. van, P. Pucheral, and P. M. Apers. Data degradation: Making private data less sensitive over time. In *CIKM*, 2008.

[2] F. Bancilhon and R. Ramakrishnan. An amateur's introduction to recursive query processing strategies. In *SIGMOD*, 1986.

[3] M. Barbaro and T. Zeller. A face is exposed for aol searcher no. 4417749. *The New York Times*, Aug. 2006.

[4] J. Dean and S. Ghemawat. Mapreduce: Simplified data processing on large clusters. *Operating Systems Design and Implementation*, October 2004.

[5] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating noise to sensitivity in private data analysis. In *TCC*, 2006.

[6] C. Dwork, M. Naor, T. Pitassi, and G. N. Rothblum. Differential privacy under continual observation. In *STOC*, 2010.

[7] A. Frank and A. Asuncion. UCI machine learning repository, 2010.

[8] S. R. Ganta, S. P. Kasiviswanathan, and A. Smith. Composition attacks and auxiliary information in data privacy. In *KDD*, 2008.

[9] A. Ghosh and A. Roth. Selling privacy at auction. `http://arxiv.org/abs/1011.1375`.

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

[11] M. Hay, V. Rastogi, G. Miklau, and D. Suciu. Boosting the accuracy of differentially private histograms through consistency. *Proc. VLDB Endow.*, 3:1021–1032, September 2010.

[12] V. Karwa, S. Raskhodnikova, A. Smith, and G. Yaroslavtsev. Private analysis of graph structure. In *VLDB*, 2011.

[13] D. Kifer. Attacks on privacy and definetti's theorem. In *SIGMOD*, 2009.

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

[15] A. Machanavajjhala, J. Gehrke, D. Kifer, and M. Venkitasubramaniam. l-diversity: Privacy beyond k-anonymity. In *ICDE*, 2006.

[16] F. McSherry. Privacy integrated queries: an extensible platform for privacy-preserving data analysis. In *SIGMOD*, 2009.

[17] F. McSherry and K. Talwar. Mechanism design via differential privacy. In *FOCS*, 2007.

[18] A. Narayanan and V. Shmatikov. Robust de-anonymization of large sparse datasets. In *IEEE Symposium on Security and Privacy*, 2008.

[19] K. Nissim, S. Raskhodnikova, and A. Smith. Smooth sensitivity and sampling in private data analysis. In *STOC*, 2007.

[20] L. Page, S. Brin, R. Motwani, and T. Winograd. The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab, 1999.

[21] V. Rastogi and S. Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *SIGMOD*, 2010.

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

[23] A. Serjantov and G. Danezis. Towards an information theoretic metric for anonymity. In *PET*, 2002.

[24] A. Smith. Privacy-preserving statistical estimation with optimal convergence rates. In *STOC*, 2011.

[25] L. Sweeney. k-anonymity: A model for protecting privacy. *International Journal on Uncertainty, Fuzziness and Knowledge-based Systems*, 2002.

[26] H. H. van, M. Fokkinga, and N. Anciaux. A framework to balance privacy and data usability using data degradation. In *CSE*, 2009.

[27] C. Wright, C. Cowan, S. Smalley, J. Morris, and G. Kroah-Hartman. Linux security modules: General security support for the linux kernel. In *USENIX Security*, 2002.

[28] X. Xiao, G. Bender, M. Hay, and J. Gehrke. ireduct: differential privacy with reduced relative errors. In *SIGMOD*, 2011.

APPENDIX

A. THEORETICAL GUARANTEES

We combine the privacy guarantees of the different pieces used in the system to present a final privacy guarantee. We provide three different privacy guarantees based on how the output range is being estimated.

THEOREM 1 (PRIVACY GUARANTEE FOR GUPT). *Let $T \in \mathbb{R}^{k\times n}$ be a dataset and $f : \mathbb{R}^{k\times n} \to \mathbb{R}^p$ be the query.* GUPT *is ϵ-differentially private if the following holds:*

1. **GUPT uses `GUPT-helper` with loose range for the input:** *Execute percentile estimation algorithm defined in [24] for each of the k input dimensions with privacy parameter $\epsilon/(2k)$ and then run the sample and aggregate framework (SAF) with privacy parameter $\epsilon/(2p)$ for each output dimension.*
2. **GUPT uses `GUPT-tight`:** *Run SAF with privacy parameter ϵ/p for each output dimension.*
3. **GUPT uses `GUPT-loose`:** *For each output dimension, run the percentile estimation algorithm defined in [24] with privacy parameter $\epsilon/(2p)$ and then run SAF with privacy parameter $\epsilon/(2p)$.*

The proof of this theorem directly follows from the privacy guarantees for each of the module of GUPT and the composition theorem of [5]. In terms of utility, we claim the following about GUPT.

THEOREM 2 (UTILITY GUARANTEE FOR GUPT). *Let $f : \mathbb{R}^{k\times n} \to \mathbb{R}^p$ be a generic asymptotically normal statistic (see Definition 2 in [24]). Let T be a dataset of size n drawn i.i.d. from some underlying distribution F. Let $\hat{f}(T)$ be the statistic computed by* GUPT.

1. *If* GUPT *uses* `GUPT-tight`*, then $\hat{f}(T)$ converges to $f(T)$ in distribution.*
2. *Let f be a statistic which differ by at most γ (under some distance metric d) on two datasets T and $\tilde{T}$, where $\tilde{T}$ is obtained by clamping the dataset T on each dimension by the* 75*-th percentile and the* 25*-th percentile for that dimension. If* GUPT *uses* `GUPT-helper` *with loose input range, then we have $d(\hat{f}(\tilde{T}), f(T)) \leq \gamma$ as $n \to \infty$.*
3. *If* GUPT *uses* `GUPT-loose`*, then $\hat{f}(T)$ converges in distribution to $f(T)$ as $n \to \infty$ as long as k, $\frac{1}{\epsilon}$ and $\log(|\max - \min|)$ are bounded from above by sufficiently small polynomials in n, where $|\max - \min|$ is the loose output range provided.*

The proof follows using a similar analysis used in [24].