Applications and Implementation of Differential Privacy on Statistical Databases

Jonathan Sumner Evans*, Victoria Girkins†, and Sam Sartor‡
Department of Computer Science, Colorado School of Mines
Golden, Colorado
Email: *jonathanevans@mines.edu, †vgirkins@mines.edu, ‡ssartor@mines.edu,

***Abstract*—Statistical databases have long been know to be vulnerable to inference attacks. We describe the nature of inference attacks and analyze one method of mitigating these attacks called Differential Privacy. We discuss two mechanisms for Differential Privacy. We then explain our implementation of a simple Differential Privacy statistical database and the experiments we performed on the database. We then analyze how well our implementation achieves the objectives of Differential Privacy databases. We conclude by calling on industry to create differential privacy implementations that can be used in general applications.**

I. Introduction

Statistical databases are designed for statistical analysis of the datasets contained within the database. One of the desired properties of statistical databases is that nothing about an individual should be learnable from the database that cannot be learned without access to the database [4]. In other words, gathering information about the records underlying the statistics using allowed queries on the database should be impossible.

Without any protections, well-crafted queries to a statistical database can implicitly reveal private information. This attack is know as an *inference attack*. To prevent these attacks, the key case to consider is that of two databases which differ by only a single row. Consider, for example, a database which associates people with their incomes. The query

```
SELECT SUM(income) FROM table
```

seems harmless at first. But if a new record were added and we ran the query again, the difference between the two results represents the exact income of the new entry. If the attacker knows the identity of the person who was added, the privacy leak becomes even more severe.

Of course, an attacker may not have the luxury of always knowing when a new record will be added to the database, and of timing his queries accordingly. He may, however, achieve the same results by narrowing down his query with conditionals.

The query

```
SELECT SUM(income)
  FROM table
 WHERE zip != 80127
```

may be sufficient in a small enough database, or more conditionals may need to be added to ensure the queries jointly isolate a single row. However, since the danger occurs when information about a single entry is returned, regardless of how the attacker managed it, we will assume for simplicity's sake that the database is queried before and after the addition of a single row.

A. Previous Attempts at Anonymizing Statistical Data

Attempts have been made to privatize data in the past and many are still in use today. However everything from simply removing columns containing personally identifiable information to advanced techniques such as as k-anonymity and l-diversity have been shown to be vulnerable to attack [2]. K-anonymity, for example, does not include any randomization and thus attackers can still make inferences about data sets [1].

B. Differential Privacy

In 2006, Cynthia Dwork, a researcher at Microsoft, proposed a new technique to reduce the risk of a successful inference attack called *Differential Privacy* [6]. The goal of Differential Privacy is to maximize the statistical accuracy of queries on a statistical database while also maximizing the privacy of the individual records. It does this by introducing noise into the dataset. This randomness reduces the likelihood of obtaining meaningful information about the individual records in the database.

The real power of Differential Privacy is when it is used in conjunction with a privacy bound, ε as described

in Section I-B1. The methods for generating random noise are discussed in Section I-B2. Details about how the one of the mechanisms is used to ensure differential privacy are explained in Section I-B3.

1) Epsilon-Differentially Private: One useful way of discussing Differential Privacy is to talk about what it means for a database to be *epsilon-differentially private*. A database falls under this category if it is private as bounded by a chosen value epsilon. As opposed to the mathematical definition, here we give a more intuitive definition as stated by Atockar [2]:

> This [epsilon-differentially private] can be translated as meaning that the risk to one's privacy should not substantially (as bounded by epsilon) increase as a result of participating in a statistical database.

For the sake of rigorousness, the formal mathematical definition of epsilon-differential privacy described by Dwork [4] follows.

Definition 1. *A randomized function $\mathcal{K}$ gives* ***ε-differential privacy*** *if for all data sets D and D' differing on at most one row, and all $S \subseteq Range(\mathcal{K})$,*

$$\Pr[\mathcal{K}(D) \in S] \leq \exp(\varepsilon) \times \Pr[\mathcal{K}(D') \in S] \quad (1)$$

For now, all that concerns us about this equation is that D and D' are the above-mentioned databases differing by a single row, and $\mathcal{K}$ is our mechanism for adding noise drawn from our distribution as described in Section I-B2.

2) Differential Privacy Mechanisms: There are two primary Differential Privacy Mechanisms: the Exponential Mechanism and the Laplace Mechanism. The Exponential Mechanism should be used on categorical, non-numeric data. For example queries of the form "what is the most common eye color in this room" should be protected using the Exponential Mechanism [2, 7, p. 8]. The Laplace Mechanism should be used with non-categorical, real-value data [5]. Because of the nature of our data, our primary focus for this project is the Laplace Mechanism.

Before we rigorously examine the mathematics, let us review our inputs and our goals:

Inputs A statistical database with sensitive information, an attacker poised with a malicious query, and a desired epsilon, or privacy parameter.

Goals To add enough Laplace-distributed noise to the results of database queries that our database is mathematically considered differentially private.

Mathematically, the Laplace Distribution is defined as follows:

Definition 2. *The* ***Laplace Distribution****, $Lap(\mu, b)$, also known as a symmetric exponential distribution, is the probability distribution with p.d.f.:*

$$P(x|\mu, b) = \frac{1}{2b} \exp\left(-\frac{|x-\mu|}{b}\right) \quad (2)$$

where μ is the location of the center of the distribution and b is the scale parameter.

Figure 1 is a visualization of the p.d.f. for a few values of μ and b.

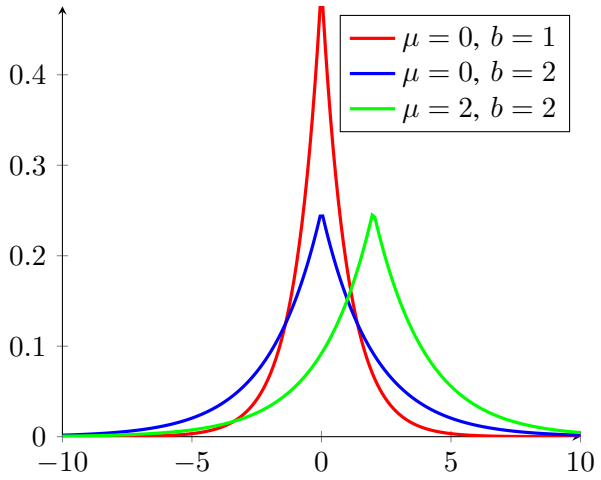

Figure 1. Probability Density Function for the Laplace Distribution

For us, μ is 0 so we can use a variant of the Laplace Distribution called the Zero-Centered Laplace Distribution which has only one parameter, b, the scale.

To draw a value from the Zero-Centered Laplace Distribution efficiently, we use the random variate for the Laplace Distribution which is defined as follows:

Definition 3. *The* ***Laplace Random Variate*** *is*

$$X(u) = \begin{cases} -b \log|u| & \text{if } u > 0 \\ b \log|u| & \text{if } u \leq 0 \end{cases} \quad (3)$$

where u is uniformly distributed in the range $(-1, 1)$.

3) Using the Laplace Differential Privacy Mechanism: In choosing the value for b, it makes sense that we should consider ε, which quantifies the level of privacy we desire. Another factor comes into play, however: the "sensitivity" of the database. This is defined as the

maximum difference between the results of any query run on two databases which differ by only one row, as described above [2]. We shall assume

```
SELECT SUM(income) FROM table
```

is the only query that this database will accept, so the "sensitivity" of our database is clearly the highest income in the table, since this value will be the difference between two queries run on the database before and after the record associated with this income is added.

So, we must add noise while considering both ε and the database's above-defined sensitivity. In fact, in a paper by Dwork [3], it was proven that choosing the "scale" parameter (b) of our Laplace distribution according to this equation

$$b = \Delta f / \varepsilon \tag{4}$$

where Δf is sensitivity, our database will mathematically satisfy epsilon-differential privacy. That is, we should add noise to the data drawn from a 0-centered Laplace distribution with scale parameter $\Delta f / \varepsilon$.

The one further consideration we must make is the possibility of multiple queries to the database. The Laplace distribution is symmetrical, so if unlimited queries were allowed, the attacker could simply run his query many times and take the average to extract sensitive data. So we must limit the number of queries allowed, and fortunately the calculation is linear. Assume we draw all noise from the same distribution (it would be possible to vary it but this unnecessarily complicates things), and that this distribution is Laplace, 0-centered, with a privacy parameter ε_i (that is, with scale parameter $\Delta f / \varepsilon_i$). If we limit learners to k queries, our overall privacy budget is $k\varepsilon_i$. This privacy budget is our actual epsilon; as stated by Atockar, "it reflects the maximum privacy release allowable for the total query session" [2].

So to sum it up practically: if we wish to make our database epsilon-differentially private as bounded by some value epsilon, our database has sensitivity Δf, and we wish for learners to be allowed k queries before being locked out, we should add a random variable to each query, drawn from the 0-centered Laplace distribution with scale parameter $k \times \Delta f / \varepsilon$.

Of course, the lower of a privacy budget we allow, the noisier the data returned by the queries will become [2]. If our queries become useless due to the noise in the data, we may choose to raise the privacy budget—that is, to increase the value of epsilon—or to further limit the number of queries each user is allowed—that is, to lower the value of k. As with many issues in information security, this is a balancing act between security and usability, and we should consider the nature of the database, its contents, and its purpose when choosing where to draw this line.

***

For our final project, we implemented and analyzed a Differential Privacy database which can perform a limited set of aggregate functions on a dataset. We describe our database implementation in Section II. Our experiment is explained in Section III. The results of our experiment are summarized in Section IV and we discuss some of the limitations of our implementation in Section V. In Section VI we describe some of the potential future work in the field of Differential Privacy. We conclude in Section VII and call on industry to create effective, easy-to-use Differential Privacy implementations for inclusion in general industry applications.

II. Database Implementation

Our actual database implementation was written in Python to speed up development time. If we were implementing this in an application that needed high performance, we would use a compiled language such as C or Rust. To help us test our implementation, our database allows the user to enable and disable Differential Privacy. When Differential Privacy is enabled, the user is also able to configure ε and the query limit.

The database reads the data from a CSV file which is specified by the user and then allows the following aggregate, or summary, functions on the data:

- **SUM**: the numerical sum of the values in a column
- **COUNT**: the number of rows
- **MIN**: the minimum numerical value in the column
- **MAX**: the maximum numerical value in the column
- **MEAN**: the average value in the column
- **VARIANCE**: the statistical variance of the column
- **SD**: the standard deviation of the column

We describe how we calculate these statistics and how we find Δf for each of these functions in Section II-A.

Our database implementation allows the user to specify a `WHERE` clause. Currently, we implement this as a directly evaluated Python expression which is executed via the `eval` function in Python. This is a code injection vulnerability, but our goal is to study Differential Privacy, not code injection prevention. Future iterations of our project could wrap this functionality and prevent code injection attacks.

Our implementation also handles edge cases where zero or one row is returned to avoid divide-by-zero errors. For most statistics, in the case when one row is returned, we merely use the value of that row as the sensitivity of the query. For the case that zero rows are returned, we add a fake entry whose value is the mean of the column and then handle it like the one row case. Future iterations of our project could improve the robustness of our edge case handling.

The statistic and Δf are calculated using the functions described in Section II-A. Noise is added according to the Laplace Distribution as described in Section I-B3. This privatized result is then displayed. For testing purposes, we also output the actual statistic, but this can easily be removed in future implementations.

A. Summary Functions

In this section we define each of the summary functions f over the set of values D. We also define the sensitivity Δf for each summary function.

Definition 4. *For any statistic $f(D)$, the **sensitivity of a query**, $\Delta f(D)$, is defined as follows:*

$$\Delta f(D) = \left| f(D) - f(D') \right| \tag{5}$$

where $D' \subset D$, $|D'| = |D| - 1$, and D' maximizes $\Delta f(D)$.

Generally, $\Delta f(D)$ can be found by iterating over all possible D' and finding the maximum $|f(D) - f(D')|$. This brute force algorithm often has $\mathcal{O}(n^2)$ complexity and is very slow on large datasets. To allow for more performant queries, our implementation indvidually defines f and Δf for each of the following summary functions.

1) Count: Removing any one row from the dataset decreases the row count by one. Thus, sensitivity of count is always 1.

$$\begin{aligned} \text{count}(D) &= |D| \\ \Delta\text{count}(D) &= 1 \end{aligned} \tag{6}$$

2) Sum: The largest element in a set has the most influence on the set's sum. Thus the sensitivity of the statistic is always the maximum $d \in D$.

$$\begin{aligned} \text{sum}(D) &= \sum D \\ \Delta\text{sum}(D) &= \max(D) \end{aligned} \tag{7}$$

3) Mean: Mean is most changed by the addition or removal of elements with a high variance from the mean. Let d be some $d \in D$ where $|d - \text{mean}(D)|$ is maximal, then

$$\begin{aligned} \text{mean}(D) &= \frac{\sum D}{|D|} \\ \Delta\text{mean}(D) &= \left| \frac{\sum D}{|D|} - \frac{\sum (D) - d}{|D| - 1} \right| \end{aligned} \tag{8}$$

4) Maximum: Let l_n be the n^{th} largest element in D. Since $\max(D)$ is dependent only on l_0, D' must not contain l_0.

$$\begin{aligned} \max(D) &= l_0 \\ \Delta \max(D) &= l_0 - l_1 \end{aligned} \tag{9}$$

5) Minimum: Let s_n be the n^{th} smallest element in D. Since $\min(D)$ is dependent only on s_0, D' must not contain s_0.

$$\begin{aligned} \min(D) &= s_0 \\ \Delta \min(D) &= s_1 - s_0 \end{aligned} \tag{10}$$

6) Variance & Standard Deviation: We use the brute force method for these functions. A more efficient method certainly exists, but we have not yet implemented it.

III. Experiment

To perform our experiment, we generated a large sample dataset suitable for statistical queries (Section III-A). We used the database that we implemented in Section II and crafted a set of queries with which to attack our database (Section III-B).

A. Dataset

To test our database, we created a dataset with 10,000 data points (rows). The columns for each row are listed in Table I on the following page. The dataset was generated using a Python program and is output to a comma separated value (CSV) file. We chose to use a CSV file for simplicity, however future iterations of our project could use a more sophisticated data store.

We chose the generation methods to roughly model the distributions of names, age, income, zip code, and net worth of the US population. For example, the income is distributed log-normal which approximates the distribution of incomes in the United States.

Table I
COLUMNS IN OUR SAMPLE DATASET

Field	Column Name	Generation Method
Name	`name`	the `names` Python library
Age	`age`	uniformly distributed between 18 and 65
Income	`income`	distributed log-normal
Zip Code	`zip`	from a random set of ~40 five-digit numbers
Net-worth	`net_worth`	from age, income, and a random offset

B. Crafted Queries

After generating our dataset, we crafted a set of queries that would to test the effectiveness of our Differential Privacy database implementation. We provide a small sample of queries here to demonstrate the properties of our Differential Privacy database. Because we have access to the underlying data, we were able to craft these queries with inside knowledge. Although our database does not parse SQL queries, we will use SQL here for clarity. Each set of queries targets a certain person's sensitive information. The sets of queries we created were as follows. (Note, `table` is the single table that we created in Section III-A.)

```
SELECT SUM(age)
  FROM table
 WHERE zip = 31643
```

```
SELECT SUM(age)
  FROM table
 WHERE zip = 31643
   AND name != "Abel Woods"
```

Query Set 1: Identifying Abel Woods' Age

```
SELECT MEAN(age)
  FROM table
  WHERE net_worth + 1 >
    (SELECT MAX(net_worth) FROM table)

SELECT MEAN(income)
  FROM table
  WHERE net_worth + 1 >
    (SELECT MAX(net_worth) FROM table)
```

Query Set 2: Identifying the age and income of the ritchest person

```
SELECT COUNT(name)
    FROM table
    WHERE name == "Peter Gist"
```

Query Set 3: Discovering if Peter Gist is in the database

IV. RESULTS

After creating a dataset, implementing a database, and choosing a set of queries to test the database with, we performed our experiment. We performed each of the queries on the dataset with Differential Privacy disabled and enabled and compared the results.

A. Query Set 1

With Differential Privacy disabled, our database implementation output for the first query set is as follows.

```
use dp (y/n): n
database csv: bigdata.csv
==================
summarize field: age
summary function: sum
where: zip == 31643
10412
==================
summarize field: age
summary function: sum
where: zip == 31643 and name != "Abel Woods"
10379
```

From this set of queries, we can determine the total age of the people in ZIP code 31643 is 10412. However, we can also infer that Abel Woods is $10412 - 10379 = 33$ years old. This is a violation of privacy.

With Differential Privacy enabled, the output is as follows.

```
use dp (y/n): y
epsilon: 5
query limit: 2
database csv: bigdata.csv
==================
summarize field: age
summary function: sum
where: zip == 31643
10409
==================
summarize field: age
summary function: sum
where: zip == 31643 and name != "Abel Woods"
10435
```

From this set of queries, we determine that the total age of the people in ZIP code 31643 is 10409, a 0.2% error. If we attempt to use these queries to find the age of Abel Woods we will get $10409 - 10445 = 49$, a 48% error.

We can see that with Differential Privacy enabled, Abel Woods' privacy is protected.

B. Query Set 2

With Differential Privacy disabled, our database implementation output for the first query set is as follows.

```
use dp (y/n): n
database csv: bigdata.csv
==================
summarize field: net_worth
summary function: max
where: True
5246937
==================
summarize field: income
summary function: mean
where: net_worth > 5246936
304322
==================
summarize field: age
summary function: mean
where: net_worth > 5246936
65
```

From these queries, we can determine that the richest person in the database has a net worth of $5,246,937. Using this person's net worth, we can filter our queries to only this person and can then determine the income ($304,233) and age of this person (65). This is the true age and income of the richest person and thus is obviously a breach of privacy.

With Differential Privacy enabled, the output is as follows.

```
use dp (y/n): y
epsilon: 3
query limit: 3
database csv: bigdata.csv
==================
summarize field: net_worth
summary function: max
where: True
# stat=5246937 b=1332431
3900194
==================
summarize field: income
summary function: mean
where: net_worth > 3900194
# stat=275988 b=28335
240135
==================
summarize field: age
summary function: mean
where: net_worth > 3900194
# stat=63 b=2
62
```

Here, the maximum net worth is protected. There is a massive discrepancy between the returned maximum net worth and the actual max net worth. This is because maximums are very sensitive; they reflect only the most extreme data points. As a result, a large amount of randomness must be added to protect them. In this instance, the Δf value is over $1.3 million resulting in a 25% error.

The income and age are fairly accurate, but this is only due to the fact that the lower net worth caused us to match two different individuals, both of whom were wealthy and in their sixty's. However, the richest person's privacy is still safe.

C. Query Set 3

With Differential Privacy disabled, our database implementation for the third query set is as follows.

```
use dp (y/n): n
database csv: bigdata.csv
==================
summarize field: name
summary function: count
where: name == "Peter Gist"
1
==================
summarize field: name
summary function: count
where: name == "Foo Bar"
0
```

From these queries, we can determine that Peter Gist is in the database while Foo Bar is not.

With Differential Privacy Enabled, the output is as follows.

```
use dp (y/n): y
epsilon: 4
query limit: 4
database csv: bigdata.csv
==================
summarize field: name
summary function: count
where: name == "Peter Gist"
# stat=1 b=1
-3.10
==================
summarize field: name
summary function: count
where: name == "Peter Gist"
# stat=1 b=1
0.63
==================
summarize field: name
summary function: count
where: name == "Foo Bar"
# stat=0 b=1
-2.23
==================
summarize field: name
```

```
summary function: count
where: name == "Foo Bar"
# stat=0 b=1
1.15
```

From these queries, we can infer nothing about whether or not Peter Gist and Foo Bar exist in the database.

V. Limitations

Our Differential Privacy implementation is very limited. Some of the limitations include:

- Our implementation does not parse actual SQL queries. However, we have designed our program in a way that makes it easy to add a parsing engine on top of our program.
- Our implementation supports a very limited set of SQL operations. For example, it does not support `join` statements. Future iterations of this project could implement more of the SQL standard.
- Our implementation can only process numerical data and is limited to the Laplace Mechanism. Adding alternative Differential Privacy mechanism such as the Exponential Mechanism would greatly improve the practicality of our Differential Privacy implementation.
- Our implementation is not heavily optimized. Currently, some queries have $\mathcal{O}(n^2)$ algorithmic complexity. In further iterations of our implementation, we could improve the performance of our system using caching methods and by optimizing the algorithms we are using.
- Our code base is not particularly extensible. Future implementations could improve the software design to make it adhere to good software engineering principles.
- We did not do any research regarding the proper handling of these edge cases where one or zero rows match the query. Our implementation seems to protect the privacy of individuals in these cases as shown by Query Set 3. However, we have no research to back up the validity of our approach, and thus there may be some vulnerabilities in our implementation.

Although our Differential Privacy implementation is not complete, our experiment proved that it was extremely effective at protecting the privacy of the individual records in the database.

VI. Future Work

Differential Privacy is a relatively new method of protecting data in a statistical database. As such, there are not many implementations of it outside of academia and large companies like Microsoft and Apple. If differential privacy was adopted by a larger percentage of the industry, more statistical databases with sensitive information would be resistant to inference attacks.

Software engineers have a great opportunity to create implementations of differential Privacy suitable for use in general industry applications. Such an implementation could be a library which other programmers could include in their project using dependency managers such as `pip`, `npm`, or `cargo`. The creation of these general-purpose libraries could help make differential Privacy more accessible and cost effective for industry professionals. This would help increase adoption of Differential Privacy and result in an overall improvement of user's privacy in the industry.

VII. Conclusion

Before starting on this project, our understanding of Differential Privacy was very limited. We expected that we would need to parse each query and handle a large number of cases to ensure the privacy of the data. After significant research, we found implementing of Differential Privacy would not require significant query parsing or edge-case handling. We were surprised by the mathematical rigor of Differential Privacy.

Even from our limited experience implementing a simple Differential Privacy solution, we were able to demonstrate its effectiveness at protecting users' privacy. Despite its usefulness, Differential Privacy has not seen significant adoption in industry outside of large companies. Thus, we take this opportunity to call on industry to create Differential Privacy libraries that are suitable for use in general industry applications. We believe that if such libraries are created, the privacy of user data throughout the industry will be vastly increased.

References

[1] Charu Aggarwal. "On k-Anonymity and the Curse of Dimensionality." In: *VLDB 2005 - Proceedings of 31st International Conference on Very Large Data Bases*. Vol. 2. Jan. 2005, pp. 901–909.

[2] Atockar. "Differential Privacy: The Basics". In: (2014). URL: https://research.neustar.biz/2014/09/08/differential-privacy-the-basics/.

[3] Cynthia Dwork. "A Firm Foundation for Private Data Analysis". In: *Communications of the ACM* (Jan. 2011). URL: https://www.microsoft.com/en-us/research/publication/a-firm-foundation-for-private-data-analysis/.

[4] Cynthia Dwork. “Differential Privacy”. In: *33rd International Colloquium on Automata, Languages and Programming, part II (ICALP 2006)*. Vol. 4052. Springer Verlag, 2006. ISBN: 3-540-35907-9. URL: https://www.microsoft.com/en-us/research/publication/differential-privacy/.

[5] Quan Geng and Pramod Viswanath. “The optimal mechanism in differential privacy”. In: *2014 IEEE International Symposium on Information Theory* (2014). DOI: 10.1109/isit.2014.6875258. URL: https://arxiv.org/pdf/1212.1186.pdf.

[6] Michael Hilton. *Differential Privacy: A Historical Survey*. URL: http://www.cs.uky.edu/~jzhang/CS689/PPDM-differential.pdf.

[7] Aaron Roth. “Lecture 3: The Exponential Mechanism”. Sept. 2011. URL: http://www.cis.upenn.edu/~aaroth/courses/slides/Lecture3.pdf.