# Mathematical Foundations for Distributed Differential Privacy

Ian Masters

A THESIS

in

Mathematics

Presented to the Faculties of the University of Pennsylvania in Partial Fulfillment of the Requirements for the Degree of Master of Arts

2018

Supervisor of Thesis

Graduate Group Chairman

# 1 Acknowledgment

First I would like to thank my advisors on this project, Andreas Haeberlen and Benjamin C. Pierce, for their guidance and generosity with their time. I want to thank the entire Mathematics department at Penn for making my time here an interesting and mind expanding experience. Especially I would like to thank Herman Gluck, whose instruction in my first ever proof based course turned my curiosity about mathematics into a love for the subject. Additionally I would like to thank Wolfgang Ziller and Reshma Tanna for their support and patience with me throughout this process.

I need to thank my friends and classmates who inspired me every day to work harder. I would especially like to thank my close friends and frequent collaborators Anthony Rodriguez and Drew Stone for their many suggestions. Of course I need to thank my family, who have made all of this possible.

# Contents

# 2 Abstract

## Mathematical Foundations for Distributed Differential Privacy

Ian Masters — *imasters@sas.upenn.edu*

**Abstract**

Large scale data analysis is a key tool used on the cutting edge of research in nearly every conceivable field. Demand for personal data has never been higher, and with this demand comes pressure to collect, aggregate, and sell information about individuals at any opportunity. The commodification of data also comes with an ethical responsibility to not harm those whose data is being farmed. Anybody who interacts with society should be concerned about their privacy.

The field of *differential privacy* offers solutions to those who want to see practical insights drawn from personal data without compromising the privacy of the individuals in the data set. Using statistical techniques, differential privacy offers mechanisms for the release of aggregate private data that bound insights about individuals in the data set to an acceptable amount.

The study of distributed privacy looks to maintain the promise of privacy when data and computation are shared among a number of operators. These results are applicable to modern computer systems in a variety of settings; specifically, when no central aggregate program can be trusted. Probability and cryptography underpin the systems at the cutting edge of this field.

In this thesis, I attempt to describe the these mechanisms and discuss the theory behind putting the results to practical use. I look at recent distributed privacy systems and the mathematics that make them possible.

# 3 Introduction

The principal idea underlining differential privacy is that the presence of each individual in the database should not have a meaningful effect on the results of an aggregate query to the database [1]. This is achieved by establishing limits on how an analyst may query a database and by perturbing the results with random noise. Suppose we have two databases $D$ and $D'$ that differ by just one entry, and an aggregate query on these databases $Q$ yielding $m = Q(D)$ and $m' = Q(D')$. If the mechanism that releases the results of the query conforms to differential privacy, then an attacker who has access to both $m$ and $m'$ should be able to learn relatively little about the one entry that differentiates $D$ from $D'$. This notion is defined much more rigorously in the next section

Now I'll consider a practical example to help demonstrate the importance of this definition. Suppose that the graduate school at Penn kept a database that included the hight of each student, and released aggregate data to help those studying whether height correlates in some way with the subject a graduate student chooses to pursue (or any other study that could utilize this data). A common query might be "What is the average hight of a masters student in math?". This result on its own will not reveal insight on any individual students in the math department, but what if an attacker also had the result of the query "What is the average hight of a masters student in math not named Ian Masters?". With these results together, and information on the size of the math department, the attacker could infer my exact height. A differentially private mechanism would add noise to the queries in such a way to make that inference impossible. In fact, differential privacy promises an upper bound on *leakage* given arbitrary prior knowledge [2] from the attacker. This means that the attacker could know the exact height of everyone else in the math department, and should still not be able to infer my hight based on the answer to the first query.

Inferring information in this way is not simply a thought experiment. Narayanan and Shmatikov showed in 2006 that they could de-anonymize the Netflix Prize dataset by cross referencing it with other public databases to reveal "potentially sensitive information" about their users [3]. This example is not alone, there have been many highly publicized instances of exposure from public data that had not been properly anonymized: genome sequencing data in 2004 [4], Aol search data in 2006 [5] and Massachusetts hospital data in 2007 [6]

An additional challenge comes when we want to maintain privacy in the distributed setting: without the the help of a central aggregator that can be trusted to conform to the standards of differential privacy. To accomplish this, we must design systems that can guarantee privacy for their users. To this end, we have developed a new class of cryptographic and probabilistic tools which enable these systems to work.

In this thesis I attempt to summarize the basic foundations of differential privacy and the most common differentially private mechanisms. From this foundation I will look at a number of modern distributed privacy systems and the additional theory required to preserve privacy invariants.

# 4 Theoretical Foundations of Differential Privacy

## 4.1 Basic Definition

Differential Privacy was first explicitly defined in [1] in 2006. We can first define it in the most general setting, then refine our space to the more typical database formats. let $\mathcal{D}$ be a set of possible databases and take $D, D' \in \mathcal{D}$. We say that $D$ and $D'$ are *neighboring* databases if they only differ by the entry of one member. A *mechanism* $M : (D) \to R$ is a function that returns a result from a database. Here $R$ is the set of results.

**Definition 4.1. (Differential Privacy)** A mechanism $M : (D) \to R$ is said to be $(\epsilon, \delta)$-*differentially private* if for all subsets $S \subset R$ and for any two neighboring databases $D, D'$

$$Pr[M(D) \in S] \leq e^{\epsilon} Pr[M(D') \in S] + \delta$$

Essentially, editing a single element of a database should only be able to effect the probability that the result is in a given set by a factor of $e^{\epsilon}$ with some extremely small allowance $\delta$. This is the most general from of the definition, and the term $\epsilon$-*differential privacy* is used when $\delta$ is set to zero.

There are a few things worth noting about this definition. First, since the neighbor relation is symmetric on the space of databases, this definition requires a stronger relationship than might be obvious at first glance; any set of neighboring databases must be a $e^\epsilon$ factor away from each other. Second is that if $\delta$ is set to zero, then equality holds only when the probability is zero. Intuitively, this means the introduction of a user cannot create an outcome that was impossible before that user was introduced.

The next important definition that is integral to mechanism design is *Sensitivity*. This measures the total impact that a single user can have on the result of a query. This value will be used to calibrate the amount of noise we must add in the mechanism to maintain our privacy invariant. For this definition, we restrict the results of our function to a metric space, as is typical with queries studied in this field. This definition will be refined when we consider specific models.

**Definition 4.2. (Function Sensitivity)** A function $f : \mathcal{D} \rightarrow R$ has *sensitivity*

$$GS(f) = \max_{D, D' \in \mathcal{D}} d(f(D), f(D'))$$

Where $d$ is the distance function in $R$ and we maximize over all possible neighboring databases in $\mathcal{D}$

## 4.2 Post Processing

At this point we can look at the proof of a useful result, resilience to post-processing. Any output of a differentially private mechanism cannot be made less differentially private without additional knowledge of the database. Arron Roth and Cynthia

Dwork provided a proof of this in their 2014 survey of of algorithmic differential privacy [2], which I will follow here.

Here we can also introduce a commonly used representation of a database, as a histogram. We call $\mathcal{X}$ the universe of records and a single database $x$ is a collection of these records. Then we write $x \in \mathbb{N}^{|\mathcal{X}|}$, and $x_i$ represents the number of records of type $i \in \mathcal{X}$ in $x$

**Proposition 4.3.** ***(Resilience to Post Processing)*** *Let $M : \mathbb{N}^{|\mathcal{X}|} \to R$ be a $(\epsilon, \delta)$-differentially private mechanism and $f : R \to R'$ be some randomized mapping. $f \circ M : \mathbb{N}^{|\mathcal{X}|} \to R'$ is $(\epsilon, \delta)$-differentially private.*

*Proof.* Consider any two neighboring databases $x$ and $y$ and choose and event $S \subset R'$. Let $T$ be the pre-image of $S$ in $R$. So we have $Pr[f(M(x)) \in S] = Pr[M(x) \in T]$ then by our definition of differential privacy we have

$$Pr[M(x) \in T] \leq e^{\epsilon} Pr[M(y) \in T] + \delta = e^{\epsilon} Pr[f(M(y)) \in S] + \delta \tag{4.1}$$

Which is the result we are after for a deterministic $f$, and since any randomized function is a convex combination of deterministic functions[2] we are done. □

This result is useful because it restricts the scope of a database operator. Anyone trying to maintain differential privacy guarantees for the users in their database need only be concerned at the release mechanism level.

## 4.3 Laplace Mechanism

Now we have the tools to review the Laplace mechanism, first introduced by Dwork et al. in [1]. Specifically, the Laplace mechanism is used for numerical queries that take the form $Q : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$. A simple example of a numerical query of this form would be a counting query, where $k = 1$: *How many people in this population have blue eyes?* A more complex query might ask for the breakdown in eye color given five options, here $k = 5$. When calibrating our Laplace function, we use the $l_1$-*sensitivity* of the query

**Definition 4.4. ($l_1$-sensitivity)** for a numerical query $Q : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$ the $l_1$-sensitivity is defined as

$$\Delta Q = \max_{\substack{x,y \in \mathbb{N}^{|\mathcal{X}|} \\ \|x-y\|_1 = 1}} \|Q(x) - Q(y)\|_1$$

Where $\|\cdot\|_1$ is *Manhattan Distance*

So by this definition, the sensitivity of both of our simple counting queries from above is one, since introducing a new user to the database can only alter the count of one category by one.

The Laplace mechanism adds random noise via the Laplace distribution, centered at 0

**Definition 4.5. (Laplace Distribution)** The probability density function of the

Laplace Distribution with scale parameter $b$ and centered at 0 is given by

$$Lap(x|0,b) = \frac{1}{2b} exp\Big( - \frac{|x|}{b}\Big)$$

The notation $Lap(b)$ will be used from here out to denote the Laplace distribution with scale parameter $b$ and center at 0. The Laplace mechanism simply uses the Laplace distribution to perturb the values of the output vector of a numerical query.

**Definition 4.6. (Laplace Mechanism)** The Laplace Mechanism takes a Database $x \in \mathbb{N}^{|\mathcal{X}|}$, a numerical query $Q : \mathbb{N}^{|\mathcal{X}|} \to \mathbb{R}^k$ and a privacy parameter $\epsilon$. It is defined as

$$M_L(x, Q(\cdot), \epsilon) = Q(x) + (Y_1, ..., Y_k)$$

Where each $Y_i$ is taken i.i.d. from $Lap(\frac{\Delta Q}{\epsilon})$

Clearly now we would want to ask how private the Laplace mechanism is. This brings us to the first major theorem and proof we will consider. Again, much of the ensuing proof is based on the one found in Roth and Dwork's survey of algorithmic differential privacy [2].

**Theorem 4.7.** *The Laplace mechanism is* $(\epsilon, 0)$*-differentially private*

*Proof.* Consider neighboring databases $x, y \in \mathbb{N}^{|\mathcal{X}|}$ and some numerical query $Q$ : $\mathbb{N}^{|\mathcal{X}|}\mathbb{R}^k$. Let $p_x$ and $p_y$ denote the probability density functions of $M_L(x, Q, \epsilon)$ and $M_L(y, Q, \epsilon)$ respectively. Then at a given point $z \in \mathbb{R}^k$, one of these PDFs would look like

$$p_x(z) = Lap\Big(z|\frac{\Delta Q}{\epsilon}\Big) = \prod_{i=1}^{k} \frac{\epsilon}{2\Delta Q} exp\Big(-\frac{\epsilon|Q_i(x) - z_i|}{\Delta Q}\Big)$$

Now we take the ratio of two PDFs, yielding

$$\frac{p_x(z)}{p_y(z)} = \prod_{i=1}^{k} \left( \frac{exp\big(\frac{\epsilon|Q_i(x)-z_i|}{\Delta Q}\big)}{exp\big(\frac{\epsilon|Q_i(y)-z_i|}{\Delta Q}\big)} \right) \tag{4.2}$$

$$= \prod_{i=1}^{k} exp\Big(\frac{\epsilon(|Q_i(x) - z_i| - |Q_i(y) - z_i|)}{\Delta Q}\Big) \tag{4.3}$$

$$\leq \prod_{i=1}^{k} exp\Big(\frac{\epsilon(|Q_i(x) - Q_i(y)|)}{\Delta Q}\Big) \text{ (by triangle inequality)} \tag{4.4}$$

$$= exp\Big(\frac{\epsilon(\|Q_i(x) - Q_i(y)\|_1)}{\Delta Q}\Big) \tag{4.5}$$

$$\leq exp(\epsilon) \tag{4.6}$$

The final inequality comes from the our definition of sensitivity. $\Delta Q$ maximizes $l_1$-norm distance in the image over all neighboring databases, so $\|Q_i(x) - Q_i(y)\|_1 \leq \Delta Q$ thus their ratio cannot exceed 1. To show $\frac{p_x(z)}{p_y(z)} \geq exp(-\epsilon)$ simply note that the proof works for the reciprocal ratio by symmetry. $\square$

Now that we have shown that the Laplace mechanism preserves differential privacy, it is natural to ask how much error was necessary to provide this guarantee. The trade off between error and privacy is important to the practicality of any mechanism. In her 2008 survey, Dwork called the choice of $\epsilon$ a "social question" [7], but it is still important to quantify error in order to inform the decision.

**Definition 4.8** (Utility)**.** Let $Q$ be a numerical query and $M(Q)$ be a mechanism that returns a noisy estimate, $\tilde{Q} = \tilde{Q}_1, ..., \tilde{Q}_n$. Then we denote the expected error of $M$ at an index $i$ to be $error_i(M) = \mathbb{E}_M[|\tilde{Q}_i - Q_i|]$ where $\mathbb{E}_M$ is the expectation over the randomness of $M$.

With this definition, we can evaluate the utility of the Laplace mechanism, taking from [1].

**Theorem 4.9.** *Let $x \in \mathbb{N}^{|\mathcal{X}|}$ be a database and $Q : \mathbb{N}^{|\mathcal{X}|} \rightarrow \mathbb{R}^n$ be a numerical query. Then for all $i$, $error_i(M(x, Q(\cdot), \epsilon)) = \frac{\Delta Q}{\epsilon}$*

*Proof.* By definition of the Laplace distribution, $\mathbb{E}|Lap(\lambda)| = \lambda$. By Definition 4.6 the mechanism adds noise to each $Q_i$ by sampling from $Lap(\frac{\Delta Q}{\epsilon})$ $\square$

# 5 Distributed Differential Privacy

## 5.1 Background

Proliferation of technology in today's society presents a great opportunity to collect and learn from user data. To accomplish this, data owners must be willing participants in the mining of their data. Many owners are hesitant to participate due to fear over security breaches and a lack of trust in centralized data managers to protect their privacy [8].

The field of distributed privacy looks to build systems that allow for user data analysis while maintaining privacy guarantees in settings where not all parties might be deemed trust worthy. In this section I look into a number of such systems, and the mathematics that make them possible.

Specifically, I will analyze three recent papers that introduce new mathematical tools for distributed privacy. The first, Piro[9], introduces *secret-shared non-interactive proofs* or SNIPs, a method for decentilized verification of data. The second paper, RAPPOR[10], builds on randomized response techniques to provide better differentially private guarantees. Finally, I look at the Fourier Perturbation Algorithm introduced by Rastogi and Nath in their 2010 paper *Differentially Private Aggregation of Distributed Time-series with Transformation and Encryption*[8].

## 5.2 Piro

Pblished in 2017 by Henry Gordan-Gibbs and Dan Boneh of Stanford University, *Prio: Private, Robust, and Scalable Computation of Aggregate Statistics*[9] aims to provide a secure method for computing aggregate statistics over a population of devices using a small set of servers. The aim is to uphold $f$-privacy, defined below. Piro accomplishes this goal by introducing a new tool, *secret-shared non-interactive proofs* ("SNIPs")

**Definition 5.1. ($f$-Privacy)** Let $f$ be an aggregation function that computes over $n$ users, each with a private value $x_i$. Any adversary that controls a proper subset of the system's servers cannot learn anything about any user's value $x_i$, besides what can be learned from the value $f(x_1, ..., x_n)$ itself.

### 5.2.1 SNIPs

The goal of a *secret-shared non-interactive proof* (I will refer to them as "SNIPs" from here out) is to allow multi party validation of a user's data while retaining $f$-privacy. To formalize this, we will say that a user's data takes on the form $x \in \mathbb{F}^L$ where $\mathbb{F}$ is some finite field. The servers need to verify that this data vector is well formed before they can incorporate it into the aggregate computation. The challenge here is that no server can hold the entire vector $x$, as that would violate the privacy guarantee.

To aide this, each party is given a validation predicate $V : \mathbb{F}^L \to \mathbb{F}$, which evaluates to 1 if the data is valid. $V$ is actually represented by an *arithmetic circuit*, which I will describe here and provide a formal definition for in Appendix A. You can think of an *arithmetic circuit* as a directed acyclic graph in which each node takes two inputs, performs a field operation ($+$ or $\cdot$), and outputs the result. In our case, there are $L$ total inputs to the graph and a single output. Our circuit $C$ representing $V$ is over the finite field $\mathbb{F}$, so each input and output "wire" will be an element of $\mathbb{F}$. We choose $\mathbb{F}$ such that it is greater than $2M$, where $M$ is the number of multiplication gates in $C$.

**Secret Generation**

The user provides 2 vectors to each server $i$. The first, $[x]_i$ is a share of its data vector $x$ such that $\sum_i [x]_i = x$ in $\mathbb{F}$. This upholds $f$-privacy since the sum is useless in guessing $x$ if one of the values is missing. The other vector that the user submits is $[h]_i$, considered the "proof string" and is calculated using the arithmetic circuit $C$.

Recall that $C$ has $M$ multiplication gates, each with two inputs. Call the inputs to the $j$th gate $u_j$ and $v_j$. Then by polynomial interpolation we know we can find two polynomials of degree at most $M-1$ that intersect points $\{(1, v_1), ..., (M, v_M)\}$ and $\{(1, u_1), ..., (M, u_M)\}$ and call these polynomials $f$ and $g$ respectively. Let $h = f \cdot g$ so that for all $1 \leq j \leq M, h(j)$ will represent the output of that multiplication gate. Now if we represent $h$ as a vector of up to $2M-2$ coefficients, we can split the vector into shares the same way we split $x$. This yields a vector $[h]_i$ to be sent to each server, and the end of the first step of the SNIP.

**Secure Verification**

Using the vectors $[x]_i$ and $[h]_i$ the servers can derive all other wire values via affine operations and subsequently construct valid shares of $f$ and $g$, $[f]_i$ and $[g]_i$ respectively[9]. Additionally, if we let $\hat{f}, \hat{g}$ and $\hat{h}$ be the polynomials reconstructed using the shares, $\hat{f} \cdot \hat{g} = \hat{h}$ holds if and only if the shares represent an accurate representation of the wire values in $C$ when summed [9]. This is how the servers will verify that the user has submitted a valid proof string.

Our goal now is to show that $\hat{f} \cdot \hat{g} = \hat{h}$, or equivalently that $\hat{f} \cdot \hat{g} - \hat{h} = 0$ which we show with near certainty in a quick manner using the Schwartz-Zippel Lemma [11, 12]

**Lemma 5.2.** ***(Schwartz-Zippel)*** *let $\mathbb{F}$ be a Field and $f(x)$ be a non-zero polynomial of degree $d$. If $S$ is a finite subset of $\mathbb{F}$ and $r$ is chosen uniformly at random independent from $S$ then $Pr[f(r) = 0] \leq \frac{d}{|S|}$*

Since $\mathbb{F}$ is finite in our case, we can chose $S$ to be the entire field, yielding only a $\frac{2M-2}{|\mathbb{F}|}$ chance we fail to identify that $\hat{f} \cdot \hat{g} \neq \hat{h}$ in the worst case. Since we can choose a very large $|\mathbb{F}|$, this test is very effective.

What we want now is for each server to be able to calculate its share of $\hat{f}(r) \cdot \hat{g}(r) - \hat{h}(r)$, $[\hat{f}(r) \cdot \hat{g}(r) - \hat{h}(r)]_i$ at some randomly chosen point $r$. These values could be published and summed to see if they add to zero. The hard part here being that $[\hat{f}(r) \cdot \hat{g}(r)]_i$ is not easily calculable from $[\hat{f}(r)]_i$ and $[\hat{g}(r)]_i$, and will require communication with the other servers.

**Multi-Party Computation**

To perform this multiplication, we will use Beaver's Multi-Party Computation (MPC) protocol published in 1991 in [13]. I will provide an overview of the protocol here.

The goal of the protocol here is to maintain privacy while allowing each server to calculate a share $[\hat{f}(r) \cdot \hat{g}(r)]_i$ using its existing shares $[\hat{f}(r)]_i$ and $[\hat{g}(r)]_i$. The protocol requires the employment of single-use *multiplication triples*.

**Definition 5.3. (Multiplication Triples)** A *multiplication triple* in a field $\mathbb{F}$ is a tuple $(a, b, c) \in \mathbb{F}^3$ such that $a \cdot b = c$ in $\mathbb{F}$

For one use of the protocol, each server will hold a share of a precomputed multiplication triple, $([a]_i, [b]_i, [c]_i) \in F^3$. Each server then computes two values which will be published.

$$[d]_i = [\hat{f}(r)]_i - [a]_i$$

$$[e]_i = [\hat{g}(r)]_i - [b]_i$$

The servers can now compute values $d = \sum_i [d]_i$ and $e = \sum_i [e]_i$. Using these values, each server is ready to compute its share.

$$[\hat{f}(r) \cdot \hat{g}(r)]_i = \frac{de}{s} + d \cdot [b]_i + e \cdot [a]_i + [c]_i$$

**Claim 5.4.** *The shares, $[\hat{f}(r) \cdot \hat{g}(r)]_i$, that result from the above MPC are valid shares of $\hat{f}(r) \cdot \hat{g}(r)$*

*Proof.* We want to show that the shares produced by the protocol will sum to the value $\hat{f}(r) \cdot \hat{g}(r)$

$$\begin{aligned}
\sum_i [\hat{f}(r) \cdot \hat{g}(r)]_i &= \sum_i \frac{de}{s} + d \cdot [b]_i + e \cdot [a]_i + [c]_i && (5.1)\\
&= d \cdot e + d \cdot b + e \cdot a + c && (5.2)\\
&= (\hat{f}(r) - a) \cdot (\hat{g}(r) - b) + (\hat{f}(r) - a) \cdot b + (\hat{g}(r) - b) \cdot a + c && (5.3)\\
&= (\hat{f}(r) - a) \cdot \hat{g}(r) + (\hat{g}(r) - b) \cdot a + c && (5.4)\\
&= \hat{f}(r) \cdot \hat{g}(r) - a \cdot \hat{g}(r) + a \cdot \hat{g}(r) - a \cdot b + c && (5.5)\\
&= \hat{f}(r) \cdot \hat{g}(r) - c + c && (5.6)\\
&= \hat{f}(r) \cdot \hat{g}(r) && (5.7)
\end{aligned}$$

□

**Final Verification**

After verifying that the user was honest by proving $\hat{f} \cdot \hat{g} = \hat{h}$ using Beaver's MPC protocol, we can now determine if the user data is valid. To do this, each server just publishes its share of the final output wire. If these sum to 1, then we know $V(x) = 1$ thus the input is valid.

## 5.3 RAPPOR

*RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response* was published in 2014 by Úlfar Erlingsson, Vasyl Pihur and Aleksandra Korolova. The paper builds on *randomized response* techniques to provide users differential privacy guarantees in the distributed setting. I will first provide a system overview, then I will narrow my focus to the differentially-private steps in the system and their proofs.

### 5.3.1 System Overview

The system aims to allow users to share a value $v$ with the aggregating server while limiting the server's ability to figure out the value. To accomplish this, the client first obfuscates their own value through the RAPPOR algorithm to generate a $k$ bit array that will be a noisy representation of $v$. The algorithm takes in a number of parameters: $k$ is an integer and determines the size of the bit array, $h$ determines the number of hash functions used in the initial step and $f, p$ and $q$ are the probability parameters used in the randomized response.

The first step of the algorithm makes use of a Bloom filter which is a tool first devised by Burton H. Bloom in 1970 in [14]. At a high level, a Bloom filter hashes values into a bit array to store them in a set, and trades off some certainty for space. Bloom filters typically allow owners of the data structure to query the set for a specific value and can return a false positive with some probability, but never a false negative. Use of the bloom filter in RAPPOR adds to the uncertainty in the bit array provided to the server. In the first step, the user hashes value $v$ onto the Bloom filter $B$ of length $k$ using $h$ hash functions.

The next step is called the *Permanent Randomized Response*. We create a new bit array $B'$ from $B$ by performing a randomized response on each bit of $B$.

$$B'_i = \begin{cases} 1, & \text{with probability } f/2 \\ 0, & \text{with probability } f/2 \\ B_i, & \text{otherwise} \end{cases}$$

Now this $B'$ is a perturbed estimator for $B$. This step is called permanent randomized response because $B'$ is stored by the user and used for all future reports for the value $v$ from the user.

Lastly, the user generates the final bit string to report using *Instantaneous Randomized Response*. The user makes a new bit string $S$ of size $k$ with the default value of 0 in each entry, then changes the bit to 1 with probabilities

$$Pr[S_i = 1] = \begin{cases} q, & \text{when } B'_i = 1 \\ p, & \text{when } B'_i = 0 \end{cases}$$

$S$ is then sent to the server.

On the server side, the paper provides a number of methods for high-utility decoding of the aggregate collected by users reporting through the RAPPOR system. These methods include Bayesian estimation, Lasso regression fitting and least squares regression [10]. We leave these up to the reader to investigate as we are more concerned with the differential-privacy guarantees of the system.

### 5.3.2 Privacy Guarantees

Here we will discuss and prove the differential privacy guarantees provided by the RAPPOR system. As shown above, the algorithm has two randomized response steps, both of which provide differential privacy. The permanent randomized response step protects the user by permanently saving a differentially private estimate for the user's value. By Proposition 4.3, if this value is used to satisfy all further queries from the user, then privacy is satisfied. The instantaneous randomized response is to prevent the user from getting identified by their perturbed Bloom filter ($B'$) over multiple requests. If the value is only going to be submitted once, then only one step of randomized response is necessary.

**Permanent Randomized Response**

**Theorem 5.5.** *Permanent randomized response satisfies $\epsilon$-differential privacy for $\epsilon = 2h \cdot ln\left(\frac{1-f/2}{f/2}\right)$*

*Proof.* To show that permanent randomized response is differentially private, we need to show that the result of the process does not change by a factor of more than $exp(\epsilon)$ if we begin with two different bloom filters. Specifically, we want to show that the ratio $RR = \frac{Pr[B' \in R | B = B_1]}{Pr[B' \in R | B = B_2]}$ is bounded by $exp(\epsilon)$ for any set of possible results $R$.

First we look the relationship between $B$ and $B'$. We do not need to include the probability of deriving $B$ from $v$ since hashing to the bloom filter is a deterministic process. We now look at $Pr[B'|B]$ on a bitwise basis, using the probabilities given in the protocol.

$$Pr[b_i' = 1 | b_i = 1] = 1 - \frac{f}{2} \; ; \; Pr[b_i' = 1 | b_i = 0] = \frac{f}{2}$$

Since we know that $B$ was derived by hashing a value with $h$ hash functions, we know that the sequence contains exactly $h$ ones and $k - h$ zeros. We can W.L.O.G assume that the the first $h$ bits are set to one and the remaining bits are set to zero $b = \{b_1 = 1, ..., b_h = 1, b_{h+1} = 0, ..., b_k = 0\}$. Now we can represent the probability of any result from the permanent randomized response:

$$Pr[B' = b' | B = b] = \prod_{i=1}^{h} \left(\frac{f}{2}\right)^{b_i'} \left(1 - \frac{f}{2}\right)^{1-b_i'} \times \prod_{i=h+1}^{k} \left(1 - \frac{f}{2}\right)^{b_i'} \left(\frac{f}{2}\right)^{1-b_i'}$$

And now we can analyze the ratio mentioned at the beginning of the proof.

$$RR = \frac{Pr[B' \in R | B = B_1]}{Pr[B' \in R | B = B_2]} \tag{5.8}$$

$$= \frac{\sum_{b' \in R} Pr[B' = b' | B = B_1]}{\sum_{b' \in R} Pr[B' = b' | B = B_2]} \tag{5.9}$$

$$\leq \max_{b' \in R} \frac{Pr[B' = b' | B = B_1]}{Pr[B' = b' | B = B_2]} \tag{5.10}$$

$$= \left(\frac{f}{2}\right)^{2(b'_1 + \ldots + b'_h - b'_{h-1} - \ldots - b'_{2h})} \times \left(1 - \frac{f}{2}\right)^{2(b'_{h+1} + \ldots + b'_{2h} - b'_1 - \ldots - b'_h)} \tag{5.11}$$

And sensitivity is at a maximum when we set $b'_1 = \ldots = b'_h = 0$ and $b'_{h+1} = \ldots = b'_{2h} = 1$. Plugging these values in above yields $RR = \left(\frac{1-f/2}{f/2}\right)^{2h}$ and thus $\epsilon = 2h \cdot ln(\frac{(1-f/2)}{f/2})$ holds. $\square$

**Instantaneous Randomized Response**

The proof of differential privacy for instantaneous randomized response is similar to the above proof, but still worth seeing as it sheds light on why both steps are included. Before we start the proof, we want to calculate two values that will make notation easier. Let $S$ be an array resulting from the RAPPOR algorithm with $B$ as the initial Bloom filter and $B'$ as the intermediate.

$$q^* = Pr[S_i = 1 | b_i = 1] = \frac{1}{2} f(p + q) + (1 - f)q \tag{5.12}$$

$$p^* = Pr[S_i = 1 | b_i = 0] = \frac{1}{2} f(p + q) + (1 - f)p \tag{5.13}$$

The calculation for these comes directly from the protocol. We are now ready to state our theorem.

**Theorem 5.6.** *Instantaneous randomized response satisfies $\epsilon$-differential privacy for $\epsilon = h \cdot ln(\frac{q^*(1-p^*)}{p^*(1-q^*)})$*

*Proof.* Similar to the previous proof, we want to find a bound for $RR = \frac{Pr[S \in R|B=B_1]}{Pr[S \in R|B=B_2]}$ where $R$ is the set of results and $B_1$ and $B_2$ are two different initial Bloom filters.

$$RR = \frac{Pr[S \in R|B = B_1]}{Pr[S \in R|B = B_2]} \tag{5.14}$$
$$= \frac{\sum_{s \in R} Pr[S = s|B = B_1]}{\sum_{s \in R} Pr[S = s|B = B_2]} \tag{5.15}$$
$$\leq \max_{s \in R} \frac{Pr[S = s|B = B_1]}{Pr[S = s|B = B_2]} \tag{5.16}$$
$$= \left(\frac{q^*(1-p^*)}{p^*(1-q^*)}\right)^h \tag{5.17}$$

giving us $\epsilon = h \cdot ln(\frac{q^*(1-p^*)}{p^*(1-q^*)}$ □

## 5.4 Time Series Aggregation

In this section I will discuss the new tools proposed by Vibhor Rastogi and Suman Nath in their 2010 paper *Differentially Private Aggregation of Distributed Time-series with Transformation and Encryption.* The paper attempts to address the specific challenges in aggregating useful statistics from time series data in the distributed setting.

Most prior work had focused on providing differential privacy for relational data with little correlation across tuples. On the other hand, time-series data can exhibit heavy correlation between tuples with timestamps in close proximity. Additionally, data quality from differentially private mechanisms on relational data tends to degrade rapidly with the number of queries; this makes the existing mechanisms impractical for time-series data in which a long sequence of queries is to be answered.

Additional challenges arise due to the distributed setting. Without a central aggregator, users must perturb the data themselves. If each user adds noise independently, then noise will scale with the number of users which is often impractical.

The paper introduces two major tools to combat these issues. The Fourier Perturbation Algorithm ($\mathrm{FPA}_k$) use a Discrete Fourier Transform to reduce the amount of noise needed to respond to a high number of queries. The Distributed Laplace Perturbation Algorithm (DLPA) adds noise via the Laplace Mechanism in a distributed manner and scales well with a large number of users.

### 5.4.1 Fourier Perturbation Algorithm

The idea behind the Fourier Perturbation Algorithm ($\mathrm{FPA}_k$) is to reduce the noise needed to answer a series of queries $\mathbf{Q} = \{Q_1, ..., Q_n\}$ while maintaining differential-privacy. Each $Q_i$ is a *snapshot query* that returns a single real number from an input database $I$. For example, $\mathbf{Q}$ may ask for the average height of all users each month for a given time window. To reduce noise, we use the Discrete Fourier Transform to create a smaller vector that we can use to estimate $\mathbf{Q}$. We can then perturb this vector with less noise and estimate the original query and maintain differential privacy.

**Discrete Fourier Transform**

FPA$_k$ utilizes the Discrete Fourier Transform (DFT), though any orthonormal transform would do. At its most general, DFT is a linear transform that takes in an $n$ dimensional sequence of complex numbers $X$ and produces another $n$ dimensional sequence of complex numbers. The $j^{th}$ element of the the result sequence is given by

$$DFT(X)_j = \sum_{l=1}^{n} exp(\frac{2\pi i}{n} jl) X_l$$

Where $i = \sqrt{-1}$. Additionally, the inverse of DFT will be denoted $IDFT$ and is given by

$$IDFT(X)_j = \frac{1}{n} \sum_{l=1}^{n} exp(\frac{2\pi i}{n} jl) X_l$$

Additionally, $IDFT(DFT(X)) = X$ [8].

For our algorithm, we want to truncate the the resulting vector to just the first $k$ values. We call this function $DFT^k$ and it gives us a vector of length $k$ that we can use to approximate the original vector, $F^k = DFT^k(X)$. We can approximate $X$ again by padding $F^k$ to length $n$ with zeros and using the Inverse Discrete Fourier Transform on the result. The quality of the approximation, $X' = IDFT(PAD^n(DFT^k(X)))$, depends on the difference between $n$ and $k$. This error will be taken into account later when we analyze the accuracy of the algorithm.

**Adding Perturbation**

The algorithm maintains differential-privacy and accuracy by adding noise to $F^k$

instead of directly to $\mathbf{Q}(I)$. Recall Definition 4.6 the Laplace Mechanism, we will use it to add noise to $F^k$.

$$\tilde{F}^k = M_L(F^k, f, \epsilon)$$

Where $f$ is the identity function. Finally we can get our differentially-private estimate for $\mathbf{Q}(I)$

$$\tilde{\mathbf{Q}}(I) = IDFT(PAD^n(\tilde{F}^k))$$

This concludes the algorithm, and we can write $\tilde{\mathbf{Q}}(I) = FPA_k(\mathbf{Q}(I))$

**Claim 5.7.** *$FPA_k$ is $\epsilon$-differentially private*

*Proof.* By Theorem 4.7 we know that $\tilde{F}^k$ is $\epsilon$-differentially private. $\tilde{\mathbf{Q}}(I)$ is obtained using only $\tilde{F}^k$, so by Proposition 4.3 $FPA_k$ is $\epsilon$-differentially private. □

By definition, we added noise to $F^k$ using a the Laplace distribution with parameter $b = \frac{\Delta F^k}{\epsilon}$ where $\Delta F^k$ is really the sensitivity of the composition $F^k = DFT^k(\mathbf{Q}(I))$. While discussing accuracy it will be useful to find a bound of $\Delta F^k$ in terms of the sensitivity of $\mathbf{Q}$, so we take the following theorem and proof from [8].

**Theorem 5.8.** *The $L_1$ sensitivity of $F^k$, $\Delta F^k$, is at most $\sqrt{k}$ times the $L_2$ sensitivity of $\mathbf{Q}$, $\Delta_2 \mathbf{Q}$.*

*Proof.* First we show $\Delta_2 F^k \leq \Delta_2 \mathbf{Q}$. DFT is an orthonormal transformation, so the $L_2$ norm of $DFT(\mathbf{Q})$ is equal to that of $\mathbf{Q}$ for any input $I$. Since $F^k$ only takes the first $k$ elements and ignores the contributions of the last $n-k$ Fourier coefficients, the inequality holds. We also know that $\Delta_1 F^K \leq \sqrt{k}\Delta_2 F^k$ by a standard property of the $L_1$ and $L_2$ norm. Thus we have $\Delta_1 F^K \leq \sqrt{k}\Delta_2 F^k \leq \sqrt{k}\Delta_2 \mathbf{Q}$, which proves the result. □

Then based on Theorem 5.8 we can set our Laplace parameter to $b = \frac{\sqrt{k}\Delta_2 \mathbf{Q}}{\epsilon}$ and $\epsilon$-differential privacy will hold. I will abuse notation slightly and use $FPA_k(\mathbf{Q}, b)$ to denote the Fourier Perturbation Algorithm in which we add noise via the Laplace Mechanism with parameter $b$ instead of the parameter as defined in Definition 4.6.

**Error**

FPA$_k$ introduces two kinds of error: the perturbation error from the Laplace mechanism and the reconstruction error that comes from recreating the estimate for a query of length $n$ from Fourier Coefficients that have been truncated to length $k$ and padded with zeros. We have already given a formal definition for mechanism error in Definition 4.8, so here we must define reconstruction error.

**Definition 5.9** (Reconstruction Error)**.** Let $X$ be a sequence of length $n$ and let $X' = IDFT(PAD^n(DFT^k(X)))$ be the sequence of length $n$ reconstructed using the first $k$ Fourier Coefficients from $X$. Then we denote the reconstruction error at index $i$ as $RE_i^K(X) = |X_i' - X_i|$

Before we analyze the error of FPA$_k$ we will make two assumptions. First we assume W.L.O.G. that the sensitivity of each query in the sequence $\mathbf{Q}$ is 1. We can make this assumption without losing generality because we can always normalize each query to have sensitivity 1 by dividing by the actual sensitivity. The other assumption is that $\mathbf{Q}$ is an *irreducible query*, or $\Delta\mathbf{Q} = \sum_{i=1}^{n} \Delta Q_i$. FPA$_k$ shows the most improvement over the basic Laplace Mechanism on irreducible queries[8]. Based on these assumptions we can calculate $\Delta\mathbf{Q} = n$. Now we can look at the theorem and proof provided in [8].

**Theorem 5.10.** *Fix* $b = \sqrt{k}\Delta_2\mathbf{Q}/\epsilon$ *so* $FPA_k(\mathbf{Q}, b)$ *is* $\epsilon$*-differentially private. Then for all* $i$, $error_i(FPA_k) = k/\epsilon + RE_i^k(\mathbf{Q}(I))$

*Proof.* let $\tilde{\mathbf{Q}} = \tilde{Q}_1, ..., \tilde{Q}_n$ be the perturbed result of $FPA_k(\mathbf{Q}, b)$. Using our initial assumptions, we know that $\Delta_2\mathbf{Q} = \sqrt{n}$. We can calculate the variance of an individual element of our perturbed sequence.

$$Var(\tilde{Q}_i) = \sum_{j=1}^{k} \frac{Var(\tilde{F}_j^k)}{n^2} \tag{5.18}$$

$$= \frac{kb^2}{n^2} \tag{5.19}$$

$$= \frac{k^2(\Delta_2\mathbf{Q})^2}{n^2\epsilon^2} \tag{5.20}$$

$$= \frac{k^2n^2}{n^2\epsilon^2} \tag{5.21}$$

$$= \frac{k^2}{\epsilon^2} \tag{5.22}$$

We can now use this value to calculate the error. For simplicity, we denote $\mu_i = \mathbb{E}[\tilde{Q}_i]$.

$$error_i(FPA_k) = \mathbb{E}|\tilde{Q}_i - Q_i| \tag{5.23}$$

$$\leq \mathbb{E}|\mu_i - Q_i| + \mathbb{E}|\mu_i - \tilde{Q}_i| \tag{5.24}$$

$$= RE_i^k(\mathbf{Q}(\mathbf{I})) + \mathbb{E}|\mu_i - \tilde{Q}_i| \tag{5.25}$$

$$\leq RE_i^k(\mathbf{Q}(\mathbf{I})) + \sqrt{\mathbb{E}|\mu_i - \tilde{Q}_i|_2^2} \tag{5.26}$$

$$= RE_i^k(\mathbf{Q}(\mathbf{I})) + \sqrt{Var(Q_i)} \tag{5.27}$$

$$= RE_i^k(\mathbf{Q}(\mathbf{I})) + \frac{k}{\epsilon} \tag{5.28}$$

Where the inequality at (4.16) is given by Jensen's Inequality [1]. □

### 5.4.2 System Overview

The remaining parts of the decentralized system would take up too much space to cover here in depth, but I will provide a brief overview here. The major challenge of adding noise via the Laplace mechanism in the decentralized setting is that each user's share of the total noise is not enough to guarantee differential privacy for the individual data. If a user were to perturb their data enough to safely send it to the aggregator, then the total noise would render the data useless.

The solution in this system is an algorithm they call Distributed Laplace Perturbation Algorithm or DLPA. The algorithm utilizes threshold homomorphic encryption, which allows each user to send their data with the appropriate share of noise to the aggregator. The aggregator can then use the *homomorphic addition* property of the encryption to compute an encrypted aggregate sum. This sum can then be decrypted as shares by the users and the results safely sent to the aggregator for the

[1] Here I use the proof provided in the appendix of [8], however I believe I may have found an error in the proof and currently have an open inquiry with the authors regarding the proof

final combination into a result.

When combining $\text{FPA}_k$ and DLPA, the system takes advantage of the linearity of $DFT$. This allows users to calculate their own $F^k$ and combine them while adding noise via $k$ iterations of DLPA. The aggregator then has a perturbed $\tilde{F}^k$ which it can use to provide a differentially private estimate for $\mathbf{Q}$.

# 6 Related Work

Differential privacy has become a very active field with many publications in recent years. The seminal paper on differential privacy by Dwork et al. in 2006 is [1]. Dwork produced a great survey of results in differential privacy in 2008 [7]. The most complete survey of the algorithmic foundations of differential privacy is the 2014 textbook on the subject by Dwork and Roth[2].

There have been numerous distributed differential privacy papers published recently in addition to the ones I chose to cover here. The papers tend to focus more on the systems side than the math or algorithmic foundations. These papers include more systems that privately aggregate user data such as PDDP[15] and Prochlo[16]. Other papers focus on distributed computation over multiple administrative domains with incomplete information like Opaque[17] and DStress[18]. Additionally, DJoin focuses on making and combining queries over distributed databases[19].

# References


[1] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In Shai Halevi and Tal Rabin, editors, *Theory of Cryptography*, pages 265–284, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.

[2] Cynthia Dwork and Aaron Roth. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(34):211–407, 2014.

[3] Arvind Narayanan and Vitaly Shmatikov. How to break anonymity of the netflix prize dataset. *CoRR*, abs/cs/0610105, 2006.

[4] Bradley Malin and Latanya Sweeney. How (not) to protect genomic data privacy in a distributed network: Using trail re-identification to evaluate and design anonymity protection systems. *J. of Biomedical Informatics*, 37(3):179–192, jun 2004.

[5] Saul Hansel. Aol removes search data on group of web users, August 2006.

[6] Latanya Sweeney. Weaving technology and policy together to maintain confidentiality. *The Journal of Law, Medicine & Ethics*, 25(23):98–110.

[7] Cynthia Dwork. Differential privacy: A survey of results. In Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, editors, *Theory and Applications of Models of Computation*, pages 1–19. Springer Berlin Heidelberg, 2008.

[8] Vibhor Rastogi and Suman Nath. Differentially private aggregation of distributed time-series with transformation and encryption. In *Proceedings of the 2010 ACM SIGMOD International Conference on Management of Data*, SIGMOD '10, pages 735–746, New York, NY, USA, 2010. ACM.

[9] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. *CoRR*, abs/1703.06255, 2017.

[10] Úlfar Erlingsson, Aleksandra Korolova, and Vasyl Pihur. RAPPOR: randomized aggregatable privacy-preserving ordinal response. *CoRR*, abs/1407.6981, 2014.

[11] Richard Zippel. Probabilistic algorithms for sparse polynomials. In Edward W. Ng, editor, *Symbolic and Algebraic Computation*, pages 216–226, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

[12] J. T. Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM*, 27(4):701–717, October 1980.

[13] Donald Beaver. Efficient multiparty protocols using circuit randomization. In Joan Feigenbaum, editor, *Advances in Cryptology — CRYPTO '91*, pages 420–432, Berlin, Heidelberg, 1992. Springer Berlin Heidelberg.

[14] Burton H. Bloom. Space/time trade-offs in hash coding with allowable errors. *Commun. ACM*, 13(7):422–426, July 1970.

[15] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johannes Gehrke. Towards statistical queries over distributed private user data. In *Proceedings of the 9th USENIX Conference on Networked Systems Design and Implementation*, NSDI'12, pages 13–13, Berkeley, CA, USA, 2012. USENIX Association.

[16] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Usharree Kode, Julien Tinnés, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. *CoRR*, abs/1710.00901, 2017.

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

[18] Antonis Papadimitriou, Arjun Narayan, and Andreas Haeberlen. DStress: Efficient differentially private computations on distributed data. In *Proceedings of EuroSys 2017*, April 2017.

[19] Arjun Narayan and Andreas Haeberlen. DJoin: differentially private join queries over distributed databases. In *Proceedings of the 10th USENIX Symposium on Operating Systems Design and Implementation (OSDI'12)*, October 2012.

# 7 Appendix

## 7.1 A Arithmetic Circuits

**Definition 7.1.** An arithmetic circuit $C$ over a finite field $\mathbb{F}$ takes a sequence $x = x_1, ..., x_L \in \mathbb{F}^L$ and produces a single field element. The circuit is represented as a directed acyclic graph in which each vertex represents either an *input*, *output*, or a *gate*.

There are $L$ input vertices, one for each element in the input sequence. Gate vertices each have in-degree two and out-degree one and have an associated field operation $+$ or $\times$. There is a single output vertex.

In order to compute the output $C(x)$, we simply walk through the graph and assign values to the outgoing edges of each gate by performing the field operation. In this way, we will eventually end up assigning a value to the in-edge of the output vertex, which is out final value. This gives us the mapping $C : \mathbb{F}^L \to \mathbb{F}$.