A Modular System for Local Differential Privacy

Peter Chang
Department of Computer Science
Harvard University
chang04@college.harvard.edu

Andrew Shackelford
Department of Computer Science
Harvard University
ashackelford@college.harvard.edu

Abstract

Local differential privacy has been touted by many tech companies as a way to truly revolutionize collecting telemetry data. Despite these claims however, local differential privacy has not been widely utilized in practice. In particular, we identify two main problems with current implementations of local differential privacy that obstruct its widespread usage: their limited data utility compared to centralized differential privacy, especially with smaller privacy budget, i.e. smaller ϵ values, and the lack of optimal composition upon multiple queries on the same data. We design and describe a modular, extensible framework for local differential privacy that addresses these problems and demonstrate its viability.

1 Introduction

As concerns over the accumulation of, and the possible abuse of, private data in statistical databases had been growing over the past several decades, Dwork et al. introduced *differential privacy* (DP) as a quantitative measure of the privacy impact on participating individuals [1]. A *centralized* model of differential privacy assumes the existence of a trusted database administrator with direct access to private data. However, this may be problematic for a number of reasons: a trusted administrator may be hard to find and an aggregate database of private information raises the possibility that an untrusted adversary may gain access to it at some future time. Therefore, the *local* model of differential privacy was introduced to avoid the problem of having a database of private information at all: now each user randomizes their own data before sending it to an aggregator [2]. This is formalized as follows:

Definition 1.1. A randomized algorithm $\mathcal{Q}$ is ϵ-local differentially private if for all pairs of user's possible private data x, x', and all adversarial strategies A,

$$\mathbb{P}[A(\mathcal{Q}_i(x)) = \text{YES}] \leq e^{\epsilon} \cdot \mathbb{P}[A(\mathcal{Q}_i(x')) = \text{YES}]$$

Local differential privacy has been described as a "superior model" of privacy due to its lack of need for a trusted database administrator who has direct access to the private data [2]. Despite this advantage, however, the current implementations of local differential privacy are few and far between. Apple [3] uses local differential privacy to detect new emojis, phrases, and some other small telemetry data. However, Tang et al. [4] found concerning amounts of privacy loss over multiple analytics reports. Google [5] uses RAPPOR to collect analytics data from Chrome, yet has not updated the software since 2016, and is currently phasing it out. Lastly, Microsoft Research [6] published a paper and wrote a blog post about collecting telemetry data privately, but has not actually publicized the use of local differential privacy in any of its products.

We identify two main limitations that obstruct the widespread usage of local differential privacy. First, local differential privacy suffers from reduced data utility compared to centralized differential privacy [7]. While local differential privacy may be more attractive to the data providers due to a more robust privacy guarantee, the significantly higher utility guarantee of the centralized model makes it more attractive towards the data analysts, thereby discouraging the use of the local model in practice. Second, unlike the centralized model [8], the effect of composition on the local model of

differential privacy has not been explicitly studied and thereby the current implementations do not optimize the distribution of the privacy budget upon repeated queries on the same data.

As a result of these inconveniences, the only open-source local DP product, RAPPOR, is now being phased out, and there are very few if any open-source local differential privacy frameworks. While there have been a number of frameworks for centralized differential privacy, such as ϵktelo [9], Psi (Ψ) [10], and PinQ [11], there exists no equivalent framework for local differential privacy.

Therefore, we design and describe a modular, extensible framework for local differential privacy that addresses these limitations. Using our framework, any interested party can easily implement local DP and know that their data collection mechanism is private. We implement a small subset of the framework as a proof of concept, allowing for others to extend the framework to their needs. By doing so, we hope that we can increase utilization of differential privacy throughout the industry, and further protect the privacy of users.

2 Related Work

2.1 Google

Google developed and deployed Randomized Aggregatable Privacy-Preserving Ordinal Response (RAPPOR) to collect statistics through Google's Chrome Web browser.

RAPPOR employs two stages of randomized response. First, using a Bloom filter containing the client's private data, it generates a "fake Bloom filter" whose bits are determined via randomized response using the original Bloom filter's bits. This filter is memoized and permanently reused when reporting the client's particular value it represents. Next, a bit array is generated with its bit elements determined via randomized response using the bits of the fake Bloom filter from the previous step. This randomization step is recomputed for each transmission of data.

Note that since the first randomization step is not reversible via averaging over repeated observations, it ensures long-term privacy. On the other hand, the second randomization step, which is reversible via averaging, makes it very difficult for adversaries to uniquely identify a client based on the values of the permanent Bloom filter, and hence ensures short-term privacy.

2.2 Apple

Apple uses local differential privacy in several use cases in order to learn more about its users' habits, specifically estimating the frequencies of elements, without compromising privacy. Its main use cases include determining which emojis are popular, identifying websites that high resource usage or crash often, or determining which custom phrases are popular.

In contrast to Google, Apple does not employ two stages of randomized response. Instead, Apple immediately discards any identifying information, including IP address, as soon as the information is received at the server. Only then is the data ingested, aggregated, and eventually analyzed.

In order to reduce transmission costs, Apple utilizes both the Count Mean Sketch and the Hadamard Count Mean Sketch algorithms to decrease the sizes of the resulting differentially private vectors. Apple also uses the Sequence Fragment Puzzle algorithm in order to detect new phrases without having to loop through all possible combinations.

2.3 ϵktelo

ϵktelo is a modular framework for centralized differential privacy, developed by researchers at UMass Amherst, Duke, and Colgate University. We chose to look at ϵktelo and other differential privacy frameworks so that we could examine the benefits and drawbacks of current implementations. ϵktelo was particularly insightful for the modularity of its framework. ϵktelo compartmentalizes each of its functions based on the effects they could have on privacy, so users of the framework have to pay as little attention as possible to the privacy implications of their analysis.

Since ϵktelo already has its differentially private functions built-in, any analyst is essentially given privacy "for free" since ϵktelo ensures that any queries or transformations on the data are differentially private. This is another crucial component of ϵktelo – the private-public barrier that ensures that

privacy is not compromised no matter what queries are performed. Lastly, ϵktelo has the concept of "plans" – where ϵktelo uses properties of differential privacy to maximize utility for a given query and to calculate the privacy budget of a complex query.

2.4 Drawbacks of Existing Implementations

While Google's RAPPOR is effective in offering a differential privacy guarantee for both its permanent randomized response and its instantaneous randomized response, one of its biggest drawbacks is that the aggregator has to know the set of possible strings in advance. This stems from the fact that RAPPOR was developed with a Google Chrome-specific goal in mind, namely to collect data regarding users' browser configurations. In other words, RAPPOR does not address the common scenario in which the client-side string does not come from a finite, predetermined set of strings, thus preventing the generalization of its usage.

Apple's deployment of DP attempts to circumvent RAPPOR's problem by utilizing the Sequence Fragment Puzzle method to discover new strings and to add to its set of possible strings. However, the Apple discloses that its DP algorithm uses an ϵ value of 4, which barely offers any protection against an adversary at all, since the probability that the truth is revealed on a single query is equal to $\frac{e^4}{1+e^4} \approx 98.2\%$.

In terms of usability, Apple's deployment is not open source and Google's RAPPOR, while open source, is difficult to use due to its poor documentation and lack of update in the past two years. Furthermore, Apple does not address the effect of composition on their DP releases and while RAPPOR'S two-step bloom filter process offers some protection against composition, the protection is lost over time as the subject's data changes. Finally, since the motivations of both Google and Apple are primarily DP enhancements of their own products, the local DP frameworks they have developed are limited in scope and are difficult to extend to general usage.

ϵktelo is a successful attempt at a modular framework for DP. However, it is limited in scope to centralized DP and is not easily extendable to local DP. For example, its operator classes, namely *Transform, Query, Query Selection, Partition Selection, Inference* offer utility only within the context of centralized DP and have no obvious analogues in local DP.

3 Utility-Optimized Local Differential Privacy

3.1 Definition

Local differential privacy, while a superior model from the perspective of a client who does not want their sensitive information to be stored in a central database, suffers from reduced data utility compared to centralized differential privacy. This is due to the fact that local differential privacy mechanisms regard all personal data as equally sensitive. To address this issue, Murakami and Kawamoto introduced the notion of utility-optimized local differential privacy (ULDP) [7].

Definition 3.1. Given $\mathcal{X}_S, \mathcal{X}_N \subseteq \mathcal{X}$ such that $\mathcal{X}_S \cup \mathcal{X}_N = \mathcal{X}$, $\mathcal{Y}_P, \mathcal{Y}_I \subseteq Y$ such that $\mathcal{Y}_P \cup \mathcal{Y}_i = \mathcal{Y}$, and $\epsilon \geq 0$, an obfuscation mechanism $\mathcal{Q} : \mathcal{X} \to \mathcal{Y}$, which takes personal data $x_i \in \mathcal{X}$ and obfuscates it to $\mathcal{Q}(x_i) \in \mathcal{Y}$ with probability $\mathcal{Q}(y|x)$, is $(\mathcal{X}_S, \mathcal{Y}_P, \epsilon)$-Utility Optimized Local DP if it satisfies the following properties:

1. For any $y \in \mathcal{Y}_I$, there exists an $x \in \mathcal{X}_N$ such that
$$\mathcal{Q}(y|x) > 0 \text{ and } \mathcal{Q}(y|x') = 0 \text{ for any } x' \neq x$$
2. For any $x, x' \in \mathcal{X}$ and any $y \in \mathcal{Y}_P$,
$$\mathcal{Q}(y|x) \leq e^{\epsilon} \cdot \mathcal{Q}(y|x')$$

In other words, the set of personal data $\mathcal{X}$ is expressed as a union of disjoint sets $\mathcal{X}_S, \mathcal{X}_N$, which represent sensitive data and non-sensitive data, respectively, and the mechanism maps sensitive data $x \in \mathcal{X}_S$ to only protected data $y \in \mathcal{Y}_P$. All mappings to Y_P are ϵ-local DP. Some portion of the non-sensitive data $x \in \mathcal{X}_N$ are mapped to invertible data $y \in \mathcal{Y}_I$, and this mapping is not differentially private.

3.2 Utility-Optimized Randomized Response

A motivating example is as follows. Suppose a survey asks its participants their HIV statuses and we consider only the response HIV^+ as sensitive data. Then, consider the following variant of randomized response. The mechanism returns its outputs with the following probability distribution, where $S^\pm, N^\pm$ refer to sensitive and non-sensitive $\text{HIV}^\pm$ choices, respectively:

$$\mathcal{Q}(\text{HIV}^+|\text{HIV}^+) = 1, \qquad \mathcal{Q}(\text{HIV}^-|\text{HIV}^+) = 0$$
$$\mathcal{Q}(\text{HIV}^+|\text{HIV}^-) = \frac{1}{e^\epsilon}, \qquad \mathcal{Q}(\text{HIV}^-|\text{HIV}^-) = \frac{e^\epsilon - 1}{e^\epsilon}$$

First, note that the mechanism does *not* satisfy ϵ-local DP:

$$\mathbb{P}[\mathcal{Q}(\text{HIV}^-) = \text{HIV}^-] = \frac{e^\epsilon - 1}{e^\epsilon} \not\leq e^\epsilon \cdot \mathbb{P}[\mathcal{Q}(\text{HIV}^+) = \text{HIV}^-] = 0$$

However, it is easy to check that the mechanism *does* satisfy $(\{\text{HIV}^+\}, \{\text{HIV}^+\}, \epsilon)$-ULDP according to Def. 3.1. First, note that $\mathcal{Q}(\text{HIV}^-|\text{HIV}^-) = \frac{e^\epsilon - 1}{e^\epsilon} > 0$ and $\mathcal{Q}(\text{HIV}^-|\text{HIV}^+) = 0$ and thus the first condition is satisfied. Next, we have the following inequalities:

$$\mathcal{Q}(\text{HIV}^+|\text{HIV}^-) = \frac{1}{e^\epsilon} \leq e^\epsilon \cdot \mathcal{Q}(\text{HIV}^+|\text{HIV}^+) = e^\epsilon$$
$$\mathcal{Q}(\text{HIV}^+|\text{HIV}^+) = 1 \leq e^\epsilon \cdot \mathcal{Q}(\text{HIV}^+|\text{HIV}^-) = 1$$

and therefore the second condition is also satisfied.

Therefore, for any output corresponding to sensitive data, i.e. HIV^+, the mechanism satisfies ϵ-local DP, while for any output corresponding to non-sensitive data, i.e. HIV^-, the data is not obfuscated.

A generalization of the motivating example is as follows [7].

Definition 3.2. Let $\mathcal{X}_S \subseteq \mathcal{X}$ and $\epsilon \geq 0$. Let $c_1 = \frac{e^\epsilon}{|\mathcal{X}_S|+e^\epsilon-1}$, $c_2 = \frac{1}{|\mathcal{X}_S|+e^\epsilon-1}$, and $c_3 = \frac{e^\epsilon-1}{|\mathcal{X}_S|+e^\epsilon-1}$. Then the $(\mathcal{X}_S, \epsilon)$-utility-optimized randomized response ($(\mathcal{X}_S, \epsilon)$-URR) is an obfuscation mechanism that maps $x \in \mathcal{X}$ to $y \in \mathcal{Y} = \mathcal{X}$ with probability $\mathcal{Q}_{URR}(y|x)$ defined as follows:

1. if $x \in \mathcal{X}_S$:

$$\mathcal{Q}_{URR}(y|x) = \begin{cases} c_1 & \text{if } y = x \\ c_2 & \text{if } y \in \mathcal{X}_S \backslash \{x\} \\ 0 & \text{if } y \in \mathcal{X}_N \end{cases}$$

2. if $x \in \mathcal{X}_N$:

$$\mathcal{Q}_{URR}(y|x) = \begin{cases} c_2 & \text{if } y \in \mathcal{X}_S \\ c_3 & \text{if } y = x \\ 0 & \text{if } y \in \mathcal{X}_N \backslash \{x\} \end{cases}$$

Murakami and Kawamoto proved that the above mechanism is $(\mathcal{X}_S, \mathcal{X}_S, \epsilon)$-ULDP, and we omit the proof here [7].

Given the obfuscated data $\vec{y}$, one can compute the estimate, $\hat{p}$, of the original probability distribution of the unobfuscated data by solving the following equation:

$$\hat{p}\mathcal{Q} = \vec{y}$$

In other words, the probability distribution can be inferred from the obfuscated data as follows:

$$\hat{p} = \vec{y}\mathcal{Q}^{-1} \tag{1}$$

3.3 Experimental Result and Analysis

We compared the accuracy of the utility-optimized randomized response (URR) mechanism to that of randomized response mechanism (RR). Recall that an ϵ-local DP RR mechanism is as follows.

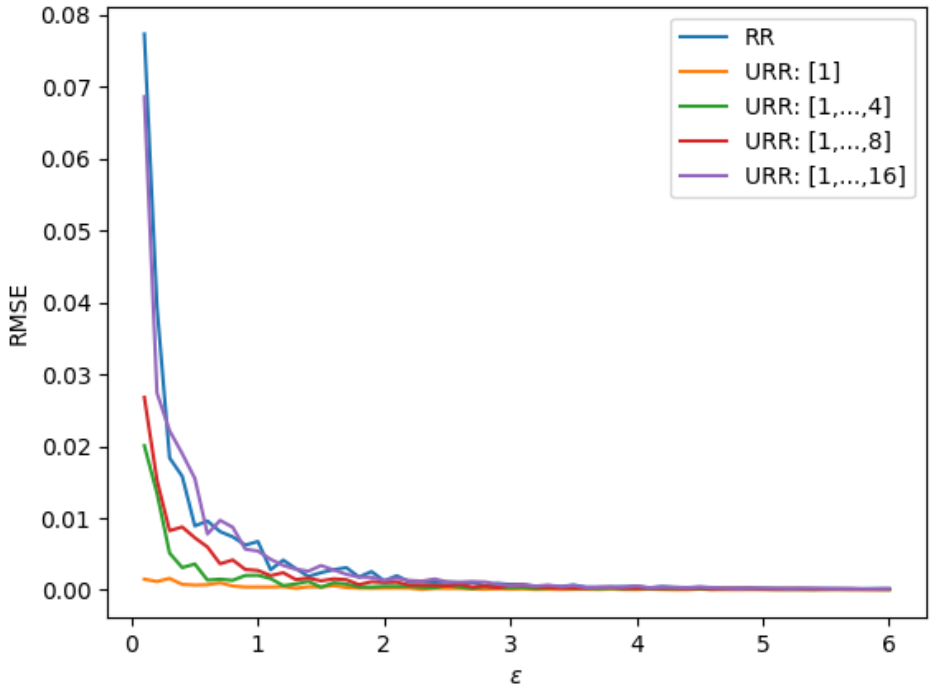

Figure 1: Root Mean Square Error(RMSE) for RR and URR for education level data from the Massachusetts 5% PUMS data. RMSE was computed against the distribution of fraction of individuals in each of the 16 education levels in the raw data. The orange, green, red, and violet curves correspond to RMSE results using URR with education levels $\{1\}, \{1,2,3,4\}, \{1,2,\ldots,8\}, \{1,2,\ldots,16\}$ as the sets of sensitive data, respectively.

Definition 3.3. An ϵ-local DP randomized response is an obfuscation mechanism that maps $x \in \mathcal{X}$ to $y \in \mathcal{Y} = \mathcal{X}$ with probability $\mathcal{Q}_{RR}(y|x)$ defined as follows:

$$\mathcal{Q}_{RR}(y|x) = \begin{cases} \frac{e^\epsilon}{|\mathcal{X}|+e^\epsilon-1} & \text{if } y = x \\ \frac{1}{|\mathcal{X}|+e^\epsilon-1} & \text{if } y \neq x \end{cases}$$

We used the Massachusetts 5% PUMS data for the experiment. More specifically, we considered the education level, which ranges from 1 to 16, of each individual as the private data of interest.

For each mechanism, we computed the estimated fraction of individuals in each education level, $\hat{p}$, using Eq. 1 and computed the root-mean-square error (RMSE) of each mechanism compared to the fractions computed from raw data. We varied the ϵ value in increments of 0.1 from $\epsilon = 0.1$ to $\epsilon = 6$ and plotted the result, as shown in Fig. 1.

For the URR mechanism, we varied the set of sensitive data. As can be seen from the plots below, URR improves most significantly compared to RR when the sensitive data is limited to people with education level 1, i.e. when the sensitive-to-nonsensitive data ratio is at its lowest. This confirms our intuition that limiting DP mechanism to sensitive data limits the corresponding trade-off in data utility.

In Fig. 1, note that the utility gap between the two mechanisms decreases as the set of sensitive data becomes larger. In particular, notice that when the sensitive data is the full dataset itself, i.e. when the entire set of education levels, $\{1,2,\ldots,16\}$, is considered sensitive, URR is equivalent to RR, up to small variance due to the randomness of the algorithms. In other words, RR is a special case of URR.

Our framework will incorporate URR to increase the utility of aggregate data.

4 Composition for Local Differential Privacy

4.1 Motivation

One of the main disadvantages of current frameworks is that they are vague about the effects of composition on their releases. ϵktelo for example does not take into account the effects of multiple plans on the same dataset and the possible privacy risks such plans could pose. RAPPOR *does* use a two-step "Bloom Filter", which they posit adds some protection against composition. However, after multiple trials, the effect of the second bloom filter will be averaged out, and the results will be subject to privacy risk as soon as the user's data ever changes. Lastly, Apple does not claim to protect against composition at all. Instead, they simply discard all identifying information from the data as it comes in. However, if someone were to intercept the data during transmission, this could pose potentially troubling privacy risks.

Our framework aims to bypass all these issues by taking into account the effects of composition. Our framework will prohibit users from sending in their data after their privacy budget has been depleted. Additionally, it will do so invisibly to the user, so as to prevent confusion. Lastly, it will be modular and extensible to allow for different composition theorems including:

- Basic composition
- Advanced composition
- Optimal composition
- Privacy odometers and filters
- Other composition theorems proposed by researchers in the field
- No composition (if the privacy loss dangers are realized and understood by the data analysts)

4.2 Basic Composition Theorem

One of the main differences between local and centralized differential privacy is the idea of how privacy evolves over time. Most uses of centralized differential privacy focus on so-called "one-shot" releases, where many statistics are calculated and released all at once. However, local differential privacy typically involves many releases of data from the same user. Companies interested in collecting user data are often interested in data changes over time; thus gathering repeated releases of data from the same set of users may be of interest. In one of their seminal papers on differential privacy, Dwork et al. proposed a basic composition theorem that provides a strict upper bound on the amount of privacy that is lost on repeated queries, guaranteeing that for a total privacy budget of ϵ for n queries, a per-query privacy budget of $\epsilon_i = \frac{\epsilon}{n}$ will preserve privacy [12].

4.3 Advanced Composition Theorem

However, this upper bound can pose limits rather quickly. Even with a large privacy budget of $\epsilon = 5$, with 50 queries our privacy budget will be equal to 0.01, which lacks almost any utility on a per-query basis. We can improve this, however, if we allow a small error probability of δ. Under the standard (ϵ) definition of differential privacy, we ensure that the probability distributions under two neighboring datasets are always within e^{ϵ} of each other. However, if we allow the probability distributions to violate that requirement – thus causing privacy loss – with probability δ, we can often achieve much better utility using the (ϵ, δ) definition of differential privacy. Additionally, if we make δ cryptographically small, we can make the chance of such an event so low such that this tradeoff is acceptable. By adding in δ, we can use Dwork et al.'s advanced composition theorem that provides a much tighter bound on privacy loss over repeated queries [13].

Definition 4.1. For a given privacy loss probability δ, per-query privacy budget ϵ_i, number of queries n, and total privacy budget ϵ, the advanced composition theorem states:

$$\epsilon = \sqrt{2n \ln(\frac{1}{\delta})} \cdot \epsilon_i + n \cdot \epsilon_i \cdot (e^{\epsilon_i} - 1)$$

4.4 Optimal Composition Theorem

This advanced composition theorem, however, is not the best bound on privacy loss that is achievable. While it has been proven that even randomized response has a loss over repeated queries of $\Omega(\sqrt{n \log \frac{1}{\delta}})$, we can still tighten our upper bound on privacy [14]. In their paper "Computing the Optimal Composition of Differential Privacy," Murtagh and Vadhan set about calculating not just the most optimal privacy budget over repeated queries, but how to calculate said budget, given that the problem is **#P**-complete [15]. While we omit the (rather complex) proof, Murtagh and Vadhan found a polynomial-time approximation to the optimal composition theorem that allows determining the correct bound with arbitrary precision.

In these results, we see that the optimal composition theorem provides noticeably better performance than all other theorems, even the closed-form approximations. By utilizing this composition theorem in our framework, we can maximize the utility of our data while still ensuring that we adequately protect privacy.

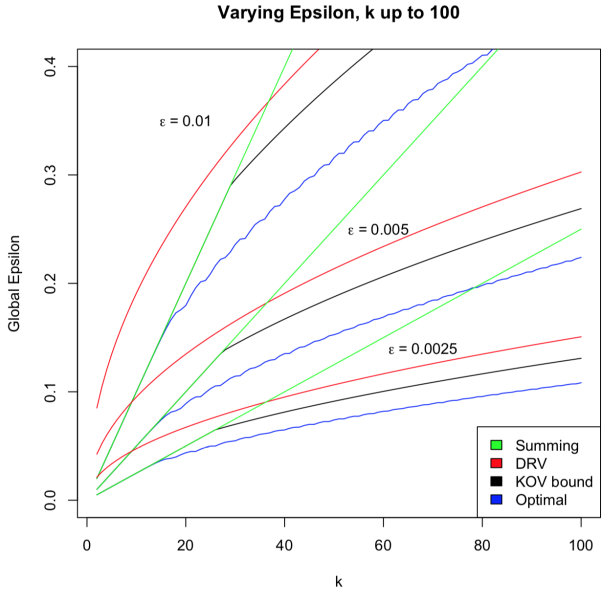

Figure 2: Optimal composition results. In this graph, "Summing" is the basic composition theorem, "DRV" is the advanced composition theorem, "KOV bound" is a closed-form approximation of the optimal composition theorem, and "Optimal" is Murtagh and Vadhan's (arbitrarily precise) approximation of the optimal composition theorem [15].

4.5 Privacy Odometers and Filters

However, all of these composition theorems have one main requirement – the privacy parameters, that is, ϵ_i and δ, must be set beforehand. Changing the parameters in the middle of an experiment violates composition properties, except in the case of basic composition. It is not hard to imagine a scenario in which a data analyst gains more subjects and discovers that he or she needs less utility per-query than previously estimated. In such a case, it would be desirable to allow the data analyst to change the per-query privacy budget in order to allow for more data collection in the future.

In order to address this, we will incorporate Rogers et al.'s privacy "odometers" and "filters" that can be used to ensure that we cut off releases once the privacy budget has been depleted, albeit with a failure probability δ [16]. A privacy "odometer" ensures that we receive a true upper bound on the privacy lost per-query with probability $1 - \delta$, while a privacy "filter" will ensure that we stop releasing responses with probability $1 - \delta$ before our privacy budget is depleted.

Support for these and other composition theorems set our framework apart from the rest. Both Apple and Google hand-wave away the privacy loss ramifications of repeated releases with their

anonymization and dual-layer bloom filters, respectively. However, these justifications do not hold up under intense scrutiny, and ours is the only framework that we are aware of that is mathematically certain to guarantee privacy under repeated releases.

5 A Modular System for Local Differential Privacy

5.1 Motivation

We aim to make our framework modular, cross-platform, and open source. Apple's existing framework is entirely proprietary, while Google's is difficult to use and has little documentation. We aim to expand upon Ektelo's work in the centralized differential privacy space by creating an analogous framework for local differential privacy. By using JavaScript code for our data collection and Python for our data aggregation, our framework will be compatible with virtually every modern device, and support both background data collection as well as user submissions through a web interface that is again cross-platform. By making our framework modular and open-source, we will allow for data analysts to substitute in their own differentially private functions, while keeping the pipeline intact.

Unlike Apple, we aim to make our framework easily auditable. All data collection will be done on-device in a single JavaScript file, thus making it easily inspectable by third-party auditors to ensure that our framework preserves privacy.

Lastly, our framework will be easily updatable. Each part of the pipeline can be easily substituted for a different complementary function, such as swapping a Hadamard Count Mean Sketch for a Sequence Fragment Puzzle, which while requiring different collection and processing functions, can use the same aggregator and visualization tools. This will allow for quick iteration on ideas and launching of new experiments. Additionally, we will even make the parameters of each experiment easily updatable, so that data analysts can easily increase or decrease the privacy budget as utility requires.

5.2 Framework Outline

See Figure 3 for a block diagram of our framework.

5.2.1 Experiment Creation

Our local differential privacy framework starts with a data analyst wishing to conduct a differentially private experiment. The experiment process begins at our trusted server, where each experiment is created and its parameters initialized. When an experiment is created, it is given a total privacy budget ϵ as well as a per release privacy budget ϵ_i, each chosen by the analyst.

Additionally, the data analyst can choose what type of experiment he or she desires, whether it be (utility-optimized) randomized response, a Hadamard Count Mean Sketch, or some other differentially private method of collecting data. This is where the modularity of our framework comes in – if the desired data collection method is not already available within our framework, the analyst will be able to craft their own method and substitute it in. In the same vein, the analyst will be able to choose their desired composition theorem, whether it be basic, advanced optimal, or some other composition theorem proposed in the future by another researcher. Additionally, it will even allow for our privacy odometers and filters if the analyst desires to change the privacy parameters during the course of the experiment.

5.2.2 Trusted Server

Once these parameters are chosen, the trusted server will craft JavaScript code corresponding to this differentially private mechanism, put it in an HTTP payload along with the ϵ and ϵ_i parameters, and send it to the device. Crucially, once the total privacy budget ϵ is chosen, it cannot be increased or reset, thus preventing the analyst from violating the privacy budget or misleading users.

Obviously, the trusted server is the largest privacy vulnerability of our framework. However, the reach of our trusted server is purposefully limited. The trusted server is only responsible for compiling the local randomizer's JavaScript code, and has no access to the gathered data. Additionally, the code is

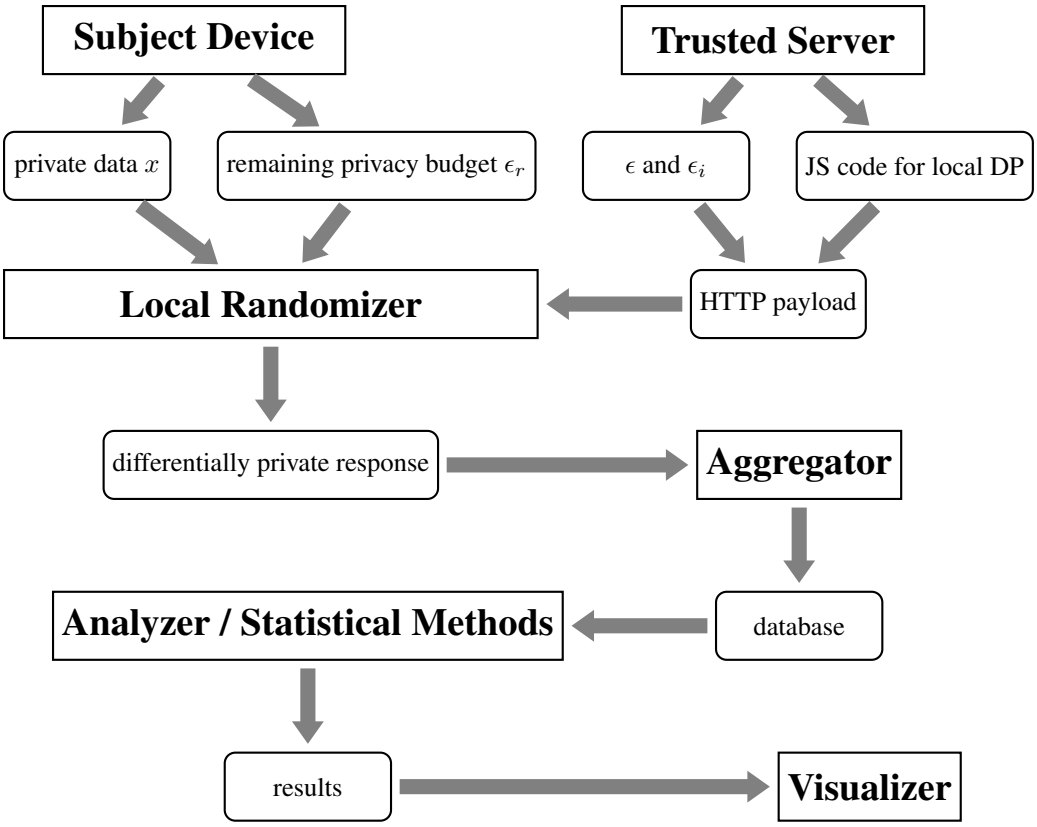

Figure 3: Framework Block Diagram

compiled from pre-approved data collection methods and composition theorems, which can each be easily audited by a third-party.

5.2.3 Local Randomizer

The local randomizer performs the crucial differential privacy steps in our framework. All interactions with the subject's private data x are performed on-device, and are not uploaded to the cloud. Crucially, the entire code for the randomizer is contained in a single JavaScript file, and is easily auditable by third parties.

Once the randomizer is downloaded to the device, it starts by initializing the total privacy budget and storing it locally on-device. Once this budget is stored on-device in JavaScript's "local storage," it cannot be cleared or augmented from the server without changes to the JavaScript code itself, which would be caught by third party auditors. However, our framework does allow for the server to change the per-response privacy budget ϵ_i, in the event that it is discovered that more or less utility per-query is required. However, this functionality is dependent on the composition theorem used. For the basic composition theorem, privacy is kept intact since the ϵ_i's simply add up. However, with other composition theorems, advanced techniques such as Rogers et al.'s [16] privacy "odometers" and "filters" can be used to ensure that we cut off releases once the privacy budget has been depleted, albeit with a failure probability δ.

5.2.4 Data Gathering

After the randomizer is downloaded and the privacy budget initialized, the JavaScript code is ready to accept the private data from the subject. This function can be called directly by the subject's device in the background, or can be attached to a web interface to be presented to the user if the desired data is survey-based rather than usage-based. Crucially, since the entire process involves only JavaScript code (and some HTML/CSS, if desired), this framework is cross-platform and will work on virtually

every modern device, no matter whether it is a supercomputer or a lightweight IoT sensor. If detailed system data is desired, the device can simply call our framework's JavaScript functions from within its low-level system implementation. The privacy characteristics of our framework will still hold, since all data still must go through our differentially private release functions.

The function, when called, takes the subject's private data x and first determines if there is enough privacy budget ϵ_r remaining to perform the analysis. If not, it will refuse to submit the subject's data and exit out. If there is enough privacy budget, it will subtract the correct amount and update the privacy budget ϵ_r (based on the composition theorem being used), perform the differentially private release (whether it be randomized response, utility-optimized randomized response, or some other method), and upload the results to the server. If Additionally, it will keep track of the past privacy parameters utilized if a privacy odometer or filter is being used.

Unfortunately, JavaScript's built-in `Math.random()`, while a good pseudo-random number generator, is still deterministic given its seed [17]. This could open up our framework to attacks if an adversary is able to observe many repeated releases of the same data and infer the generator's seed. As a result, we will instead implement our randomness using W3C's *Web Cryptography API* [18]. This API will allow us to obtain cryptographically strong random values easily with the following code:

```
function secureMathRandom() {
    return window.crypto.getRandomValues(new Uint32Array(1))[0] / (2**32 - 1);
}
```

5.2.5 Aggregator

Now on an untrusted server, our framework will collect each response and aggregate them, keeping track of the response, which experiment the response was obtained from, as well as its ϵ_i value. When storing the data, it will keep only these values and nothing else. For demonstration purposes, we also keep a random ID string associated with each response so that each participant can recognize their data in the results, but while this does not compromise privacy it is also not strictly necessary.

5.2.6 Analyzer / Statistical Methods

Once all the data has been aggregated, it is passed to an analyzer that can perform different statistical methods on the data. For a simple randomized response, for example, it could simply calculate the estimated population mean with the correct scaling factor. For something more complicated like a Hadamard Count Mean Sketch, it could perform the necessary dictionary sampling to calculate the true counts of each phrase. The aggregator is also extensible to inference tools like differentially private stochastic gradient descent, which would allow us to train a model on the data. Since all the data is aggregated at a different stage in the pipeline, the modular nature of our framework allows us to switch up the analysis tools without having to re-write our pipeline from scratch.

5.2.7 Visualizer

Once our data is analyzed and turned into actionable results, these results are passed into a visualizer that helps us understand what the data means. This could be as simple as a confidence interval graph for our population mean, or as complicated as a model of a gradient surface.

5.3 Demo

For demonstration purposes, we chose to implement a simple randomized response mechanism accessible via a barebones Flask website. The code for this demonstration is available on GitHub. To run it locally on your machine, simply `cd` into the `Website` directory, set the environment by running `export FLASK_APP=application.py`, and execute `flask run`.

Alternatively, we have also hosted the website publicly using PythonAnywhere, and it will be available for the near future at http://cs208.pythonanywhere.com.

Our website has two main user-facing functions, as well as a debug function for demonstration purposes.

As a single user, you can submit your data to the experiment by clicking on the "Submit Data" option in the navbar, choosing the experiment you wish to submit data to, and then clicking the "True" or "False" option. The submit page also displays the total privacy budget, the privacy budget per response, as well as the user's privacy budget remaining. Upon attempting to submit data, the page will redirect to either a success or failure page, depending on whether there was enough privacy budget left.

For demonstration purposes, we have added functionality to allow the submission of multiple data each with a certain probability of being True/False, as a tool to show how the calculated population mean will converge on the actual population mean as the number of responses increases. To use this function, click on the "Submit Multiple Data" option in the navbar, choose your dataset, desired population mean (on a $[0, 1]$ scale) and number of responses, and then click "Perform Randomized Response". Since this functionality is really only for debug purposes and was tacked on at the end of our project, each submission sends an individual POST request, which can take a good deal of time for large numbers of responses. As a result, we display an alert when the responses finish submitting. Additionally, we hard-coded the ϵ_i value to 1 simply because it made adding this functionality to our website easier – in the future we would improve this, or realistically remove the functionality altogether since it is only for demonstration purposes.

To view the data, click on the "View Data" option in the navbar. Then, after choosing the dataset you'd like to view data from, the website will display the calculated population mean, as well as each individual response's ID and value. This allows each person to see that their response was indeed collected for demonstration purposes.

6 Conclusion

In this paper, we examined the limitations of current implementations of local DP. Namely, we first studied a variant of local DP called *utility-optimized local DP* that increases the data utility in the case where the private data can be separated into sensitive and non-sensitive data. Then, we examined the different composition theorems that allow us to extract more utility from multiple releases of an individual's data. Once we had these methods that allowed us to extract more utility from local DP, we crafted a framework to allow for easy implementation of these and other methods. This framework is modular and easily expandable, giving data analysts privacy "for free" through the implementation of trusted methods. We also created a demo of this framework as a proof-of-concept, showing how we could easily implement local differential privacy with this framework and expand upon it in the future.

In the future, we'd like to expand our framework by adding more data gathering, analyzing, and visualization methods. There are many different types of data collected using local differential privacy, and we'd like to support them all. Additionally, our current demo framework is not as modular as we'd like it to be. In the future, we'd like to re-write it to be truly modular and open-source it with appropriate documentation, so that any data analyst can add their own features in an easy and extensible manner. By making the creation of experiments modular and allowing the selection of already-proven methods, we can allow data analysts to perform experiments without having to worry about their subject's privacy. With our framework, we have made privacy a true guarantee while at the same time increasing utility, thus making data analysis easier, more useful, and more private for the benefit of everyone.

Acknowledgments

We would like to thank Professor Salil Vadhan, Professor James Honaker, and Jayshree Sarathy for their assistance and feedback, as well as the students of CS208 for a semester of wonderfully engaging and intellectually stimulating discussions.

References

[1] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. *Calibrating Noise to Sensitivity in Private Data Analysis*. TCC, March 2006.
https://dl.acm.org/citation.cfm?id=2180305

[2] Cynthia Dwork, Aaron Roth. *The Algorithmic Foundations of Differential Privacy*. Foundations and Trends in Theoretical Computer Science, Auguest 2014.
https://dl.acm.org/citation.cfm?id=2693053

[3] *Learning with Privacy at Scale*. Apple Machine Learning Journal, December 2018.
https://machinelearning.apple.com/2017/12/06/learning-with-privacy-at-scale.html

[4] Jun Tang, Aleksandra Korolova, Xiaolong Bai, Xueqiang Wang, and Xiaofeng Wang. *Privacy Loss in Apple's Implementation of Differential Privacy on MacOS 10.12*. arXiv, September 8, 2017.
https://arxiv.org/abs/1709.02753

[5] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. *RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response*. Google Research, November 2014.
https://research.google.com/pubs/archive/42852.pdf

[6] Bolin Ding, Jana Kulkarni, and Sergey Yekhanin. *Collecting Telemetry Data Privately*. Microsoft Research, December 8, 2017.
https://www.microsoft.com/en-us/research/blog/collecting-telemetry-data-privately/

[7] Takao Murakami and Yusuke Kawamoto. *Utility-Optimized Local Differential Privacy Mechanisms for Distribution Estimation*. February 2019.
https://arxiv.org/abs/1807.11317

[8] Peter Kairouz, Sewoong Oh, Pramod Viswanath. *The Composition Theorem for Differential Privacy*. IEEE Transactions on Information Theory, June 2017.
https://ieeexplore.ieee.org/document/7883827

[9] Dan Zhang, Ryan McKenna, Ios Kotsogiannis, Michael Hay, Ashwin Machanavajjhala, and Gerome Miklau. *EKTELO: A Framework for Defining Differentially-Private Computations*. SIGMOD, June 2018.
https://dl.acm.org/citation.cfm?id=3196921

[10] Marco Gaboardi, James Honaker, Gary King, Jack Murtagh, Kobbi Nissim, Jonathan Ullman, and Salil Vadhan. *PSI (Ψ): a Private Data Sharing Interface*. Privacy Tools Project, August 7, 2018.
https://arxiv.org/abs/1609.04340

[11] Frank McSherry. *Privacy Integrated Queries: An Extensible Platform for Privacy-Preserving Data Analysis*. Microsoft Research, September 2010.
https://www.microsoft.com/en-us/research/publication/privacy-integrated-queries-2/

[12] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. *Our Data, Ourselves: Privacy via Distributed Noise Generation*. Microsoft Research, May 2006.
https://www.iacr.org/archive/eurocrypt2006/40040493/40040493.pdf

[13] Cynthia Dwork, Guy N. Rothblum, Salil Vadhan. *Boosting and Differential Privacy*. 2010 IEEE 51st Annual Symposium on Foundations of Computer Science, October 2010.
https://ieeexplore.ieee.org/document/5670947

[14] Stanley L. Warner. *Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias*. Journal of the American Statistical Association, April 10, 2012.
https://www.tandfonline.com/doi/abs/10.1080/01621459.1965.10480775

[15] Jack Murtagh, Salil Vadhan. *The Complexity of Computing the Optimal Composition of Differential Privacy*. Theory of Computing, June 2018.
http://theoryofcomputing.org/articles/v014a008/

[16] Ryan Rogers, Aaron Roth, Jonathan Ullman, and Salil Vadhan. *Privacy Odometers and Filters: Pay-as-you-Go Composition*. arXiv, June 2, 2016.
https://arxiv.org/abs/1605.08294

[17] Yang Guo. *There's `Math.random()`, and then there's `Math.random()`*. V8 Dev Blog, December 17, 2015.
https://v8.dev/blog/math-random

[18] Mozilla Developer Network Web Docs. *`Crypto.getRandomValues()`*.
https://developer.mozilla.org/en-US/docs/Web/API/Crypto/getRandomValues