Privacy-preserving Data Aggregation with Optimal Utility

Fabienne Eigner[1], Aniket Kate[2], Matteo Maffei[1],
Francesca Pampaloni[3], and Ivan Pryvalov[2]

[1] CISPA, Saarland University, Germany
{eigner,maffei}@cs.uni-saarland.de
[2] MMCI, Saarland University, Germany
{aniket,pryvalov}@mmci.uni-saarland.de
[3] Independent researcher
francesca.pampaloni@alumni.imtlucca.it

Abstract

Computing aggregate statistics about user data is of vital importance for a variety of services and systems, but this practice has been shown to seriously undermine the privacy of users. Differential privacy has proved to be an effective tool to sanitize queries over a database, and various cryptographic protocols have been recently proposed to enforce differential privacy in a distributed setting, e.g., statical queries on sensitive data stored on the user's side. The widespread deployment of differential privacy techniques in real-life settings is, however, undermined by several limitations that existing constructions suffer from: they support only a limited class of queries, they pose a trade-off between privacy and utility of the query result, they are affected by the answer pollution problem, or they are inefficient.

This paper presents PrivaDA, a novel design architecture for distributed differential privacy that leverages recent advances in SMPCs on fixed and floating point arithmetics to overcome the previously mentioned limitations. In particular, PrivaDA supports a variety of perturbation mechanisms (e.g., the Laplace, discrete Laplace, and exponential mechanisms) and it constitutes the first generic technique to generate noise in a fully distributed manner while maintaining the optimal utility. Furthermore, PrivaDA does not suffer from the answer pollution problem. We demonstrate the efficiency of PrivaDA with a performance evaluation, and its expressiveness and flexibility by illustrating a variety of application scenarios such as privacy-preserving web analytics.

1 Introduction

Statistics about user data play a significant role in the digital society: they are used daily for improving services, analyzing trends, performing marketing studies, conducting research, and so on. For instance, website owners rely on third-party analytics services to learn statistical information (e.g., gender, age, nationality) about their visitors; electricity suppliers introduced smart meters in order to constantly monitor the user's electricity consumption, which allows them to compute prices based on energy usage trends, to optimize energy distribution, and so on; lastly, service providers often ask their users to evaluate the quality of their services with the goal of publishing the aggregate results.

The acquisition and processing of sensitive information poses serious concerns about the privacy of users. The first problem is how and where user data are aggregated: companies prefer to directly collect and process user data, but this gives them access to a wealth of sensitive information. For instance, web analytics rely on user tracking, allowing aggregators to reconstruct a detailed and precise profile of each individual. The second problem is how to publish aggregate data or statistics in a privacy-preserving manner. For example, researchers demonstrated how precise information about consumer habits can be reconstructed from the electricity consumption information collected by smart meters [47] and how an individual's identity and state of health can be derived from genome wide association studies [52].

Differential Privacy (DP). *Differential privacy* [26] is a popular architecture to define and enforce privacy for statistics on sensitive data. The fundamental idea is that a query on a database is differentially private if the contribution of an individual in the database can only marginally influence the query result. More precisely, the contribution of each single entry to the query result is bounded by a small constant factor, even if all remaining entries are known. A deterministic query can be made differentially private by perturbing the result with a certain amount of noise. The amount of noise depends on the query itself and a variety of perturbation algorithms [28, 44] have been proposed for different queries and datatypes (e.g., numerical and non-numerical data, buckets, histograms, graphs).

Distributed Differential Privacy (DDP). While the original definition of DP focused on a *centralized setting*, in which a database is queried by a curious entity, subsequent work has extended the definition to a *distributed setting* (e.g., [27, 46]), in which mutually distrustful, and potentially compromised, parties collaborate to compute statistics about distributed data. In particular, Dwork et al. [27] were the first to suggest the idea of employing *secure multiparty computation (SMPC)* to aggregate and perturb data in a privacy-preserving distributed manner. In general, in a distributed setting, which will be the focus of this paper, the problem to solve is two-fold: (*i*) how to aggregate data and compute statistics without parties learning each other's data and (*ii*) how to perturb the result to obtain DP even in the presence of malicious parties deviating from the protocol.

State-of-the-Art. Several specialized ad-hoc cryptographic protocols have been proposed recently to solve the problem of data aggregation in a distributed, privacy-preserving manner, which has enabled the enforcement of DDP in some challenging scenarios, such as smart metering [13, 24] and web analytics [7, 20, 21].

These works can be grouped into two categories: *server-based* and *fully dis-*

tributed approaches. The former rely on trusted or *honest-but-curious (HbC)*[1] servers [7, 20, 21, 41] to compute noise perturbation. These servers are assumed not to collude with each other. The latter [6, 19, 49, 51] (see [36] for a comparative survey) propose fully distributed systems to distributively aggregate time-series data, which are directly perturbed by users and then encrypted in such a way that the aggregator is only able to decrypt their noisy sum. Despite the significant progress in this field, the deployment of DDP techniques in real-life systems has so far been undermined by some open challenges, which we discuss below.

Tradeoff between privacy and utility. The existing fully distributed approaches [6, 19, 49, 51] exploit the divisibility properties of certain noise mechanisms and let each party produce a little amount of noise, whose sum yields the noise required to achieve DDP. This solution is affected by a trade-off between privacy and utility, since the amount of noise each user has to add is proportional to the number of tolerated malicious or failing parties: the more malicious parties, the more the noise to be added and, therefore, the less accurate the result. Hence, in order to obtain strong privacy guarantees, each party should assume all others to be malicious, but this leads to an intolerable error ($O(N^2)$, where N is the number of users) as we show in § B. Relying on a lower honesty threshold, however, not only gives lower privacy guarantees but also leads parties to the paradox of having to agree on how many of them are dishonest!

Lack of generality and scalability. Existing solutions are tailored to individual datatypes and perturbation mechanisms [6–8, 19, 21, 36, 41, 49, 51]. Computing different kinds of queries or employing different perturbation mechanisms requires the usage of different protocols, which rely on different cryptographic schemes, communication patterns, and assumptions. The engineering effort and usability penalty are significant and discourage system administrators from deploying such technologies. Furthermore, existing SMPC-based schemes [8] for two parties cannot simply be extended to a multiparty setting.

Inefficiency. Many schemes [6, 19, 49, 51] involve a significant computational effort on the user's side, making them impractical in several scenarios, e.g., for aggregating data stored on mobile devices with limited computation power.

Answer pollution. Fully distributed schemes [6, 19, 49, 51] in particular, suffer from the answer pollution problem: a single party can substantially pollute the aggregate result by adding excessive noise.

Collusion. Server-based systems [7, 21, 41] rely on strong non-collusion assumptions. In case of collusion, both the noise and the individual user's data are disclosed.

For a detailed comparison of the presented works w.r.t. utility, supported queries and perturbation mechanisms, and non-collusion assumptions we refer to § B.

Our Contributions. In this work we present PrivaDA, the first generic architecture for computing differentially private statistics about distributed data. We show how to achieve provable DDP guarantees, while overcoming the previously discussed

[1] An honest-but-curious party follows the protocol, but tries to learn additional information about the private data.

limitations, by leveraging recently proposed SMPC protocols for floating point numbers [9], fixed point numbers [18], and integers [31]. Our construction refines these schemes, originally designed for the HbC setting, so as to make them secure even in the malicious setting.

The overall privacy-preserving data aggregation computation is organized in two phases: the aggregation phase, in which the clients securely compute the aggregate result, and the perturbation phase, in which this result is perturbed so as to achieve DDP. To improve the performance, the SMPC is actually conducted by computation parties, which collect input shares from each client and perform the required computations. For the perturbation phase, the fundamental idea is to let computation parties jointly compute a random seed (i.e., a random variable in $(0, 1)$), which is then used to produce the required noise.

The distinctive features of our approach are:

Generality. PrivaDA supports a variety of perturbation mechanisms, such as noise drawn from the Laplace and the discrete Laplace (symmetric geometric) distribution as well as the exponential mechanism. Consequently, it is well-suited for a variety of application scenarios. We illustrate generality of our architecture by proposing privacy-preserving designs for web analytics, statistics gathering for anonymity networks, and anonymous surveys.

Strong privacy. As long as at least one of the computation parties is honest, malicious parties can only recover the aggregate result. This is a fundamental difference from other approaches (e.g., [7, 20, 21], where colluding parties can immediately read the individual's user data. Furthermore, as long as the majority of the computation parties is not colluding, none learns the seed or the noise (i.e., DDP is achieved).

Optimal utility and resistance to pollution attacks. The result is perturbed with the minimal amount of noise required to achieve DDP, irrespectively of the expected number of dishonest users and computation parties. Hence, our protocol provides optimal utility and resistance to answer pollution. We also provide mechanisms to tackle the orthogonal problem of ensuring that the protocol only accepts client inputs that are contained in a set of valid answers.

Efficiency. We implemented the system and conducted a performance evaluation, demonstrating the practicality of our approach. We find the overheads introduced by our privacy-preserving mechanisms to be acceptable for the data aggregation scenarios. Importantly, the client does not have to perform any expensive computation: she just has to provide each computation party with a share of her data and can then go offline, which makes this approach suitable even for mobile devices. Furthermore, PrivaDA supports a large number of clients without any significant performance penalty. Our implementations provide a clear interface for developers to easily integrate PrivaDA in a privacy-preserving data aggregation scenario.

Outline. The paper is organized as follows: § 2 gives some necessary background information on DP and on SMPCs for arithmetic operations; § 3 presents our architecture and our algorithms for three query sanitization mechanisms; § 4 provides an instantiation of the differentially private algorithms with efficient SMPCs; § 5 analyzes the security of these protocols and § 6 investigates their performance; § 7 illustrates the flexibility of our architecture by showing three use cases for PrivaDA; § 8 concludes and gives directions for future research. More details, additional information on the related work, and proofs are provided in the appendix.

2 Background

We now present the concept of differential privacy and the cryptographic building blocks that PrivaDA builds on.

Differential Privacy (DP). Intuitively, a query is differentially private if it behaves statistically similarly on all databases D, D' differing in one entry, written $D \sim D'$. This means that the presence or absence of each individual database entry does not significantly alter the result of the query. The definition of DP is parameterized by a number ϵ, which *measures* the strength of the privacy guarantee: the smaller ϵ, the smaller the risk to join the database.

Definition 1 (Differential Privacy [26]) *A randomized function $f : \mathcal{D} \to \mathcal{R}$ is ϵ-*differentially private *iff for all databases D, D' such that $D \sim D'$ and every set $S \subseteq \mathcal{R}$, it holds that $Pr[f(D) \in S] \leq e^{\epsilon} \cdot Pr[f(D') \in S]$.*

A deterministic query can be made differentially private by perturbing its result with noise. We describe three popular perturbation mechanisms below. An important insight is that the required amount of noise depends on the query: the more a single entry affects the query result, the stronger the perturbation has to be. This can be expressed using the notion of *sensitivity* of queries, which measures how much a query amplifies the distance between two inputs.

Definition 2 (Sensitivity [28]) *The sensitivity Δf of a query $f : \mathcal{D} \to \mathcal{R}$ is defined as $\Delta f = \max_{\forall D,D'.D\sim D'} |f(D) - f(D')|$.*

Intuitively, queries of low sensitivity map nearby inputs to nearby outputs. For instance, the query *"how many students like the 'Security' lecture?"* has sensitivity 1, since adding or removing one entry affects the result by at most 1.

Laplace noise. The most commonly used sanitization mechanism for queries returning a numerical result is the *Laplace mechanism* [28], i.e., the addition of random noise drawn according to a Laplace distribution $\mathsf{Lap}(\lambda)$ to the correct query result. As shown by Dwork et al. [28], this mechanism provides ϵ-DP, if the parameter λ is set to $\frac{\Delta f}{\epsilon}$. The distribution is both parameterized by the sensitivity of the query and the privacy value ϵ.

Discrete Laplace noise. In some scenarios, it is necessary for a query and its sanitization to return an integer result, e.g., to enable future cryptographic operations based on discrete groups on the result. To this end, Ghosh et al. [35] proposed the *discrete Laplace mechanism* (also called geometric mechanism). It is defined by adding a random integer drawn according to the discrete Laplace distribution $\mathsf{DLap}(\lambda)$ [35], also known as the symmetric geometric distribution, to the correct query result. The discrete Laplace mechanism provides ϵ-DP if the parameter λ is set to $e^{-\frac{\epsilon}{\Delta f}}$ [19]. Again, the distribution is both parameterized by the sensitivity of the query and the privacy value ϵ.

Exponential mechanism. There exist many scenarios in which queries return non-numerical results (e.g., strings or trees). For instance, consider the query "*what is your favorite lecture?*". For such queries, the addition of noise either leads to nonsensical results or is not well-defined. To address this issue, McSherry and Talwar [44] proposed the so-called *exponential mechanism*, which considers queries on databases D that are expected to return a query result a of an arbitrary type $\mathcal{R}$. For our purpose we consider the range $\mathcal{R}$ to be finite, e.g., the set of lectures offered by a university. We refer to each $a \in \mathcal{R}$ as a *candidate*. The mechanism assumes the existence of a *utility function* $q : (\mathcal{D} \times \mathcal{R}) \to \mathbb{R}$ that assigns a real valued score to each possible input-output pair (D, a), which measures the quality of the result a w.r.t. input D. The higher such a score, the better (i.e., more exact) the result. The mechanism $\varepsilon_q^\epsilon(D)$ aims at providing the "best" possible result $a \in \mathcal{R}$, while enforcing DP.

Definition 3 (Exponential Mechanism [44]) *For all $q : (\mathcal{D} \times \mathcal{R}) \to \mathbb{R}$ the randomized exponential mechanism $\varepsilon_q^\epsilon(D)$ for $D \in \mathcal{D}$ is defined as $\varepsilon_q^\epsilon(D) :=$ return $a \in \mathcal{R}$ with probability proportional to $e^{\epsilon q(D,a)}$.*

We now formally state the privacy guarantees of the above mechanisms.

Theorem 1 (DP of Mechanisms [19, 28, 44]) *For all queries $f : \mathcal{D} \to \mathbb{R}$, $g : \mathcal{D} \to \mathbb{Z}$, and $q : (\mathcal{D} \times \mathcal{R}) \to \mathbb{R}$ it holds that the queries $f(x) + \mathsf{Lap}(\frac{\Delta f}{\epsilon})$ and $g(x) + \mathsf{DLap}(e^{-\frac{\epsilon}{\Delta f}})$ and the mechanism $\varepsilon_q^{\frac{\epsilon}{2\Delta q}}(D)$ are ϵ-differentially private.*

Alternative approaches to privacy. In this work we focus on the notion of differential privacy, but for completeness, we refer to some recent papers that investigate limitations of this notion [37,43] and propose alternative definitions of privacy for statistical queries [15, 33, 42].

Secure Multiparty Computation (SMPC). SMPC enables a set of parties $\mathcal{P} = \{P_1, P_2, \dots, P_\beta\}$ to jointly compute a function on their private inputs in a privacy-preserving manner [53]. More formally, every party $P_i \in \mathcal{P}$ holds a secret input value x_i, and $P_1, \dots, P_\beta$ agree on some function f that takes β inputs. Their goal is to compute and provide $y = f(x_1, \dots, x_\beta)$ to a recipient while making sure that the following two conditions are satisfied: (i) *Correctness:* the correct value of y is computed; (ii) *Secrecy:* the output y is the only new information that is released to the recipient (see § 5 for a formal definition).

Although the feasibility of SMPC in the computational setting as well as in the information theoretic one is known for more than 25 years, dedicated work to optimize secure realizations of commonly used arithmetic operations has started only in the last few years [9,14,18,23,31]. However, most of these realizations perform limited SMPC arithmetic operations over input elements belonging only to finite fields [23] or integers [31], and they are thus not well-suited or sufficient for DDP mechanisms. In contrast, we build on SMPCs for algorithms on fixed point numbers [18] and a recent work by Aliasgari et al. [9], who presented SMPC arithmetic algorithms over real numbers represented in floating point form. They also went ahead to propose SMPC protocols for complex tasks such as logarithm and exponentiation computations, and conversion of numbers from floating point form to fixed point or integer form and vice-versa.

Our work starts by observing that their logarithm and exponentiation computations SMPC protocols, combined with the SMPC schemes for the basic integer, fixed point, and floating point number operations, pave the way for a practical design of various perturbation mechanisms for DDP in a completely distributed manner. Nevertheless, to be suitable for our design, we have to enhance and implement this large array of protocols to work against a malicious adversary.

Notations. We assume that secret sharing and the basic SMPC operations take place over a field $\mathbb{F}_q$. Let $[x]$ denotes that the value $x \in \mathbb{F}_q$ is secret-shared among $P_1, \ldots, P_\beta$ such that any $\lceil(\beta+1)/2\rceil$ of those can compute x using the reconstruction Rec protocol. In some steps of our protocols, we employ the sharing $[x]_\beta$, which requires participation from all β parties to reconstruct x. The former is called $(\beta, \lceil(\beta+1)/2\rceil)$ secret sharing, while the latter is called (β, β) secret sharing. Note that $[x]+[y]$, $[x]+c$, and $c[x]$ can be computed by each P_i locally using her shares of x and y, while the computation of $[x][y]$ is interactive for $(\beta, \lceil(\beta+1)/2\rceil)$ and impossible for (β, β) secret sharing.

Basic SMPC protocols. The following SMPC protocols [9,18,23,31] are used for our DDP mechanisms.

1. The protocol $[r] \leftarrow \mathsf{RandInt}(k)$ allows the parties to generate shares $[r]$ of a random k-bit value r (i.e., $r \in [0, 2^k)$) without interactive operations [23].

2. The protocols $[a] \leftarrow \mathsf{IntAdd}([a_1],[a_2])$ and $[a] \leftarrow \mathsf{IntScMul}([a_1], \alpha)$ allow for the addition of two shared integers and the multiplication of a shared integer with a scalar respectively, returning a shared integer.

3. The protocols $[b] \leftarrow \mathsf{FPAdd}([b_1],[b_2])$ and $[b] \leftarrow \mathsf{FPScMul}([b_1], \alpha)$ allow for the addition of two shared fixed point numbers and the multiplication of a shared fixed point number with a scalar respectively, returning a shared fixed point number.

4. The protocols FLMul, FLDiv, and FLAdd can be used to multiply, divide, or add two shared floating point numbers respectively. The output is a shared floating point number.

5. The conversion protocols $\mathsf{FL2Int}$ (float-to-int), $\mathsf{Int2FL}$ (int-to-float), $\mathsf{FL2FP}$ (float-to-fixed-point), and $\mathsf{FP2FL}$ (fixed-point-to-float) allow us to convert numbers represented as integers, floating point, or fixed point to another one of these representations.

6. The exponentiation protocol FLExp2 takes a shared floating point number $[r]$ as input and returns the shared floating point number corresponding to $[2^r]$.

7. The logarithm computation FLLog2 takes a shared floating point number $[r]$ and either returns the shared floating point number corresponding to $[\log_2 r]$ or an error (for $r \leq 0$).

8. The protocol FLRound takes a shared floating point value $[r]$ as input and operates on two modes (given as an additional argument). If $\mathsf{mode} = 0$ then the protocol outputs the floor $[\lfloor r \rfloor]$, otherwise, if $\mathsf{mode} = 1$ then the protocol outputs the ceil $[\lceil r \rceil]$. The output is a shared floating point number.

9. The protocol FLLT allows us to compare two shared floating point numbers and returns [1] if the first operand is less than the second, [0] otherwise.

For the exact complexities of the above SMPC protocols, we refer to § A, Table 6. Intuitively, additions and scalar multiplications for both integers and fixed point values are non-interactive and consequently very fast, while for floating point values, the algorithms require communication complexity linear in the size of floating point significant. Floating point additions are more costly than even their multiplication counterparts. As expected, the exponentiation and logarithm algorithms are also the most costly and require communication quadratic in the size of significants. Note that these complexities can be significantly reduced using pre-computation and batched processing [9].The relative efficiency of these SMPC schemes plays a fundamental role in our design.

3 The PrivaDA Architecture

In this section we present the PrivaDA architecture. We first give a general overview of the setup and then present three mechanisms for achieving DP.

The fundamental challenge we had to overcome in order to provide SMPCs for the three noise mechanisms lies in providing SMPC support for drawing random numbers according to various probability distributions. Standard arithmetic SMPC suites we considered only offer support for drawing uniformly random integers. As we will show in § 4, a SMPC for drawing a uniformly random integer can be used to encode the drawing of a uniformly random number between 0 and 1 ($\mathcal{U}_{(0,1]}$). We thus focussed our efforts on finding an encoding that reduces the problem of generating noise according to the distributions employed in three popular sanitization mechanisms to the drawing of a *seed* from $\mathcal{U}_{(0,1]}$.

Setting. We consider a scenario with n users, β computation parties (typically $\beta = 3$, but it can be greater), and an aggregator. Each user P_i has a private input D_i from some domain $\mathcal{D}$. The aggregator would like to compute some aggregate statistics about the users' private inputs, represented by the function $f : \mathcal{D}^n \rightarrow \mathcal{R}$. The range $\mathcal{R}$ of f may be a set of numerical values, but not necessarily. The computation parties are responsible for computing and perturbing the aggregate result, which is eventually

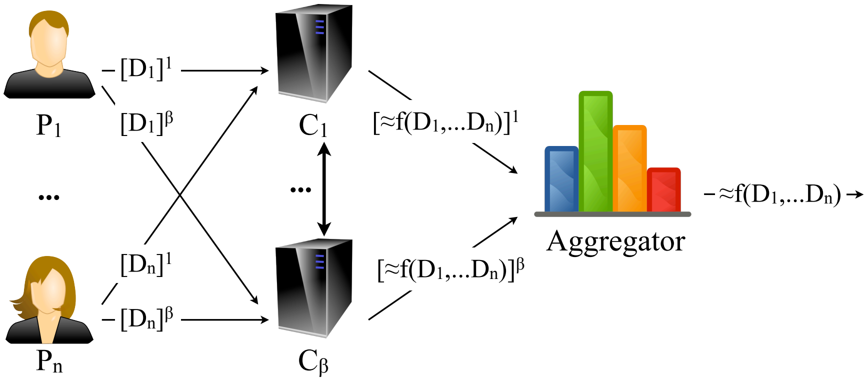

Figure 1: Protocol Flow

returned to the data aggregator. *System assumptions.*The users communicate with the computation parties through secure and authenticated channels. Furthermore, the computation parties are pair-wise connected by secure authenticated channels. The users provide the computation parties with shares of their data and can then go offline. The computation parties instead engage in an interactive protocol. The communication among them can be made asynchronous using a bulletin board, which can be securely and efficiently implemented using standard techniques such as [10, 50].

Privacy goals. The aggregate result should be differentially private and neither the aggregator, nor the computation parties, nor the users should learn any further information about the individual users' data.

Attacker model. The data aggregator as well as the users may be corrupted and collude. The SMPC protocols we adopt for the realization of our approach are based on secret sharing: such SMPCs are secure in the malicious setting (i.e., the computation parties may try to deviate from the protocol) but, for certain arithmetic operations, they assume the majority of the computation parties not to collude. We could in principle adopt other kinds of SMPC protocols that do not require this assumption [14, 40, 54], but they are currently less efficient.

Protocol Overview. The protocol proceeds in three steps: (*i*) The users provide the computation parties with shares of their inputs.[2] (*ii*) The computation parties run the SMPC protocol to compute the aggregate statistics and perturb the result. (*iii*) Each computation party gives the aggregator its share of the result, which is reconstructed by the aggregator.

The protocol flow is depicted in Figure 1. In the following we describe three different algorithms to compute queries sanitized with the Laplace, discrete Laplace, and exponential mechanism respectively. For easing the presentation, we consider a class of queries for which $f(D_1, \ldots, D_n) = f(D_1) + \ldots + f(D_n)$. Other arithmetic queries can

[2] Our work focusses on guaranteeing the privacy of the user data. To solve the orthogonal problem of pollution attacks and to prevent malicious users from entering wildly incorrect input shares, we can use ZK range proofs [16] (cf. § 4).

```
In: $d_1, \ldots, d_n$; $\lambda = \frac{\Delta f}{\epsilon}$
Out: $(\sum_{i=1}^{n} d_i) + \mathsf{Lap}(\lambda)$
1: $d = \sum_{i=1}^{n} d_i$
2: $r_x \leftarrow \mathcal{U}_{(0,1]}$;  $r_y \leftarrow \mathcal{U}_{(0,1]}$
3: $r_z = \lambda(\ln\ r_x - \ln\ r_y)$
4: $w = d + r_z$
5: return  w
```

(a) LM

```
In: $d_1, \ldots, d_n$; $\lambda = e^{-\frac{\epsilon}{\Delta f}}$
Out: $(\sum_{i=1}^{n} d_i) + \mathsf{DLap}(\lambda)$
1: $d = \sum_{i=1}^{n} d_i$
2: $r_x \leftarrow \mathcal{U}_{(0,1]}$; $r_y \leftarrow \mathcal{U}_{(0,1]}$
3: $\alpha = \frac{1}{\ln\ \lambda} = -\frac{\Delta f}{\epsilon}$
4: $r_z = \lfloor \alpha(\ln\ r_x) \rfloor - \lfloor \alpha(\ln\ r_y) \rfloor$
5: $w = d + r_z$
6: return  w
```

(b) DLM

```
In: $d_1, \ldots, d_n$;  $a_1, \ldots, a_m$; $\lambda = \frac{\epsilon}{2}$
Out: winning $a_k$
1: $I_0 = 0$
2: for $j = 1$ to $m$ do
3:     $z_j = \sum_{i=1}^{n} d_i(j)$
4:     $\delta_j = e^{\lambda z_j}$
5:     $I_j = \delta_j + I_{j-1}$
6: $r \leftarrow \mathcal{U}_{(0,1]}$;  $r' = rI_m$
7: $k$ = binary_search($r', \leq, I_0, \ldots, I_m$)
8: return  $a_k$
```

(c) EM

Table 1: Algorithms: Sanitization Mechanisms

be implemented in a very similar manner using minor modifications of the presented algorithms, since modern SMPC schemes provide direct support for a large class of arithmetic operations. The algorithms described below do not rely on specific SMPCs: we give one possible efficient instantiation in § 6.

Laplace Mechanism. We now describe an algorithm for calculating the Laplace mechanism (LM) for n inputs. We use the following mathematical results [5, 25] that allow us to reduce the problem of drawing a random number according to the Laplace distribution (Lap) to the problem of drawing a uniformly random number between 0 and 1 ($\mathcal{U}_{(0,1]}$) using the exponential distribution (Exp). It holds that $\mathsf{Lap}(\lambda) = \mathsf{Exp}(\frac{1}{\lambda}) - \mathsf{Exp}(\frac{1}{\lambda})$, where $\mathsf{Exp}(\lambda') = \frac{-\ln\ \mathcal{U}_{(0,1]}}{\lambda'}$. Thus,

$$\mathsf{Lap}(\lambda) \quad = \quad \lambda(\ln\ \mathcal{U}_{(0,1]}) - \lambda(\ln\ \mathcal{U}_{(0,1]}). \tag{1}$$

In particular, we know that $\lambda = \frac{\Delta f}{\epsilon}$ guarantees DP. We thus define our algorithm for the addition of Laplace noise on n inputs as shown in Table 1a. It takes as input (i) the n real numbers $d_1, \ldots, d_n$ owned by $P_1, \ldots, P_n$ respectively, which correspond to locally executing the query f on each P_i's database D_i ($d_i = f(D_i)$) and (ii) the privacy budget parameter λ, set to $\frac{\Delta f}{\epsilon}$ to guarantee ϵ-DP. The algorithm returns the real $w = (\sum_{i=1}^{n} d_i) + \mathsf{Lap}(\lambda)$ that is computed by summing up all d_i (line 1), drawing a random number according to the distribution $\mathsf{Lap}(\lambda)$ (line 2 - 3) using (1), and adding the sum of the results and the noise (line 4).

Privacy of LM. Since the LM algorithm implements $\sum_{i=1}^{n} d_i + \lambda(\ln\ \mathcal{U}_{(0,1]}) - \lambda(\ln\ \mathcal{U}_{(0,1]}) = \sum_{i=1}^{n} d_i + \mathsf{Lap}(\lambda)$, by Theorem 1 it follows that $\mathsf{LM}(d_1, \ldots, d_n, \lambda)$ is ϵ-differentially private for $\lambda = \frac{\Delta f}{\epsilon}$, where $d_i = f(D_i)$.

Discrete Laplace Mechanism. We now give an algorithm to compute the discrete Laplace mechanism (DLM) on n inputs. The construction follows similar principles as the one for the Laplace mechanism that we presented in the previous section.

Additionally, we rely on the following mathematical results [25, 39] to compute the discrete Laplace distribution (DLap) by using the link between DLap, the geometric distribution (Geo), and the exponential distribution (Exp), which can be reduced to the uniform distribution on the interval $(0, 1]$ ($\mathcal{U}_{(0,1]}$) as shown before. It holds that $\mathsf{DLap}(\lambda) = \mathsf{Geo}(1 - \lambda) - \mathsf{Geo}(1 - \lambda)$, where $\mathsf{Geo}(\lambda') = \lfloor \mathsf{Exp}(-\ln\,(1 - \lambda')) \rfloor$. Thus,

$$\mathsf{DLap}(\lambda) = \left\lfloor \frac{1}{\ln\ \lambda} \ln\ \mathcal{U}_{(0,1]} \right\rfloor - \left\lfloor \frac{1}{\ln\ \lambda} \ln\ \mathcal{U}_{(0,1]} \right\rfloor . \qquad (2)$$

In particular, for $\lambda = e^{-\frac{\epsilon}{\Delta f}}$, ϵ-DP is guaranteed. The algorithm to add discrete Laplace noise to n inputs is shown in Table 1b. It takes as input (i) the n integer numbers $d_1, \dots, d_n$ owned by $P_1, \dots, P_n$ respectively, which correspond to locally executing the query f on each P_i's database D_i ($d_i = f(D_i)$) and (ii) the privacy budget parameter λ, which will be set to $e^{-\frac{\epsilon}{\Delta f}}$ to guarantee ϵ-DP. The algorithm returns the integer $w = (\sum_{i=1}^{n} d_i) + \mathsf{DLap}(\lambda)$, which is computed analogously to the Laplace mechanism, using (2).

Privacy of DLM. The DLM algorithm implements $\sum_{i=1}^{n} d_i + \lfloor \frac{1}{\ln\ \lambda} \ln\ \mathcal{U}_{(0,1]} \rfloor - \lfloor \frac{1}{\ln\ \lambda} \ln\ \mathcal{U}_{(0,1]} \rfloor = \sum_{i=1}^{n} d_i +\ \mathsf{DLap}(\lambda)$. By Theorem 1, $\mathsf{DLM}(d_1, \dots, d_n, \lambda)$ is ϵ-differentially private for $\lambda = e^{-\frac{\epsilon}{\Delta f}}$, where $d_i = f(D_i)$.

Exponential Mechanism. Our algorithm to compute the exponential mechanism [44] (EM) for n inputs, is inspired by [8], which is however an ad-hoc protocol that is constrained to a 2-party setting. Here, the challenge lies in computing the exponential mechanism in an n-party setting and make it work in the adversarial setting we consider.

Inputs and outputs. The algorithm to compute the EM on the join of n databases is presented in Table 1c. It outputs the candidate $a \in \mathcal{R}$ (where $|\mathcal{R}| = m \in \mathbb{N}$), which is the result of locally executing the desired query f on the databases $D_1, \dots, D_n$ that are under the control of the participants $P_1, \dots, P_n$ respectively and sanitizing the joint result using the exponential mechanism. The algorithm takes the following inputs: (i) the data sets $d_1, \dots, d_n$ belonging to the participants $P_1, \dots, P_n$ respectively, (ii) the list of candidates $a_1, \dots, a_m$, and (iii) the privacy parameter λ, which will be set to $\frac{\epsilon}{2\Delta q}$ in order to guarantee ϵ-DP. For the sake of simplicity, we assume each data set $d_i \in \mathcal{D}$ to be a histogram that is the result of locally executing $f(D_i)$. Each histogram is a sequence of m natural numbers $z_1, \dots, z_m$ that correspond to the frequency of candidates $a_1, \dots, a_m \in \mathcal{R}$. For instance, for the query $f :=$ "*What is your favorite lecture?*" the sequence of candidates $a_1, \dots, a_5$ might be $\mathsf{Algebra}$, Logic, $\mathsf{Security}$, $\mathsf{Cryptography}$, Java and the individual data set d_2 of student P_2 who prefers the lecture $\mathsf{Security}$ is a histogram of the form $0, 0, 1, 0, 0$. The algorithm outputs the winning candidate a_k drawn according to $\varepsilon_q^{\epsilon}(d_1, \dots, d_m)$.

Utility function. Our approach is general and can support any arithmetic utility function. For the sake of presentation, we focus on the following utility function $q((z_1, \dots, z_m), a_i) = z_i$ for all histograms $d = (z_1, \dots, z_m)$ and candidates $a_1, \dots, a_m$, returning the frequency z_i of candidate a_i stored in d. For instance, in the above example $q(d_2, \mathsf{Security}) = 1$ and $q(d_2, a_i) = 0$ for all candidates a_i, where $i \in \{1, 2, 4, 5\}$.

Since $\Delta q = 1$ it can be omitted from the algorithm. The privacy parameter λ is thus set to $\frac{\epsilon}{2}$.

Random variable. Our goal is to compute the exponential mechanism $\varepsilon_q^\epsilon(D)$ for a discrete range $\mathcal{R}$, where $|\mathcal{R}| = m$. The probability mass [8, 44] for the exponential mechanism is defined as $\Pr\left[\varepsilon_q^\epsilon(D) = a\right] = \frac{e^{\epsilon q(D,a)}}{\sum_{j=1}^{m} e^{\epsilon q(D,a_j)}}$. As pointed out by Alhadidi et al. [8], drawing a random value according to this distribution corresponds to mapping the above defined probability mass onto the interval $(0,1]$ and drawing a random number r in $(0,1]$ to select the interval of the winning candidate. Formally, $r \leftarrow \mathcal{U}_{(0,1]}$ and $r \in (\sum_{k=1}^{j-1} \Pr\left[\varepsilon_q^\epsilon(D) = a_k\right], \sum_{k=1}^{j} \Pr\left[\varepsilon_q^\epsilon(D) = a_k\right]]$ corresponds to $a_j \leftarrow \varepsilon_q^\epsilon(D)$. For instance, assume $\Pr\left[\varepsilon_q^\epsilon(D) = a_1\right] = 0.3$ and $\Pr\left[\varepsilon_q^\epsilon(D) = a_2\right] = 0.7$. We draw a random number r from $(0,1]$ and check whether r is in interval $(0, 0.3]$ or in interval $(0.3, 1]$. In this example, the drawing of $0.86 \leftarrow \mathcal{U}_{(0,1]}$ corresponds to $a_2 \leftarrow \varepsilon_q^\epsilon(D)$.

It is easy to see that by multiplying with $S := \sum_{j=1}^{m} e^{\epsilon q(D,a_j)}$ the check $r \in (\sum_{k=1}^{j-1} \Pr\left[\varepsilon_q^\epsilon(D) = a_k\right], \sum_{k=1}^{j} \Pr\left[\varepsilon_q^\epsilon(D) = a_k\right]]$ is equivalent to $r \cdot S \in (\sum_{k=1}^{j-1} e^{\epsilon q(D,a_k)}, \sum_{k=1}^{j} e^{\epsilon q(D,a_k)}]$, since $\Pr\left[\varepsilon_q^\epsilon(D) = a\right] \cdot S = e^{\epsilon q(D,a)}$. To optimize complexity, our algorithm will compute the exponential mechanism using the latter version, i.e., by drawing a random number $r \leftarrow \mathcal{U}_{(0,1]}$ and then checking $r \cdot S \in (\sum_{k=1}^{j-1} e^{\epsilon q(D,a_k)}, \sum_{k=1}^{j} e^{\epsilon q(D,a_k)}]$ and returning the candidate a_k for which this checks succeeds. Thus, our main effort lies in computing the necessary interval borders $(\sum_{k=1}^{j-1} e^{\epsilon q(D,a_k)}, \sum_{k=1}^{j} e^{\epsilon q(D,a_k)}]$.

Algorithm. Our algorithm consists of the following steps[3]: (i) Initialize the interval border I_0 (line 1). (ii) Compute the joint histogram $d = d_1 + \ldots + d_n$ (line 3) by adding the frequencies for each individual candidate. (iii) Compute interval borders for candidates (line 4 - 5). (iv) Draw a random value r in $(0,1]$ (line 6) and multiply this value by $I_n = \sum_{j=1}^{m} e^{\epsilon q(D,a_j)}$, resulting in the scaled random value r'. (v) Check into which of the intervals $(I_{j-1}, I_j]$ the random value r' falls (line 7) by using binary search that returns k such that $I_{k-1} < r' \leq I_k$. (vi) Return the winning candidate a_k (line 8).

Privacy of EM*.* The EM algorithm implements the join of the individual n histograms, the utility function q as defined above, and the drawing of a random value according to $\varepsilon_q^\lambda(d_1 + \ldots + d_n)$, which is soundly encoded as explained above. Thus, $\mathsf{EM}(d_1, \ldots, d_n, a_1, \ldots, a_m, \lambda)$ computes $\varepsilon_q^\lambda(d_1 + \ldots + d_n)$, where q has sensitivity 1 and by Theorem 1 it follows that $\mathsf{EM}(d_1, \ldots, d_n, a_1, \ldots, a_m, \lambda)$ is ϵ-differentially private for $\lambda = \frac{\epsilon}{2}$, where $d_i = f(D_i)$.

[3] Depending on the instantiation of the SMPCs, the steps might be slightly modified, or type conversions added, to provide the best efficiency. The steps presented in this work are designed so as to get the best efficiency based on SMPC schemes employed in PrivaDA.

4 Instantiation

In this section, we instantiate the three mechanisms described in the previous section. A technical challenge we had to face was to identify, among all possible instantiations (based, e.g., on different type conversions), the most efficient one based on the currently available SMPC schemes (cf. § 2). The protocols we propose to compute the distributed Laplace, the distributed discrete Laplace, and the distributed exponential mechanism are given in Tables 2, 3, and 4 respectively, and are explained below.

Number Representation. For floating point form, each real value u is represented as a quadruple (v, p, z, s), where v is an ℓ-bit significand, p is a k-bit exponent, z is a bit which is set to 1 when the value $u = 0$, s is a sign bit, and $u = (1-2s)\cdot(1-z)\cdot v \cdot 2^p$. Here, the most significant bit of v is always set to 1 and thus $v \in [2^{\ell-1}, 2^{\ell})$. The k-bit signed exponent p is from the range $\mathbb{Z}_{\langle k \rangle} = (-2^{k-1}, 2^{k+1})$. We use γ to denote the bit-length of values in either integer or fixed point representation, and f to denote the bitlength of the fractional part in fixed point values. Every integer value x belongs to $\mathbb{Z}_{\langle \gamma \rangle} = (-2^{\gamma-1}, 2^{\gamma+1})$, while a fixed point number x is represented as $\bar{x}$ such that $\bar{x} \in \mathbb{Z}_{\langle \gamma \rangle}$ and $x = \bar{x}2^{-f}$. Finally, it is required that $k > \max(\lceil \log(\ell + f) \rceil, \lceil \log(\gamma) \rceil)$ and $q > \max(2^{2\ell}, 2^{\gamma}, 2^{k})$. For ease of exposition, we assume that $\gamma = 2\ell$ for integers and fixed point numbers, and that $f = \frac{\gamma}{2}$ for fixed point numbers.

Input Distribution and Output Reconstruction. We assume that prior to the computation the users $P_1, \ldots, P_n$ create β shares of their respective integer or fixed point inputs $d_1, \ldots, d_n$ in the (β, β)-sharing form and distribute them amongst the β computation parties $C_1, \ldots C_\beta$, so that each party C_k holds a share of each input value $[d_i]$, for $k \in \{1, \ldots, \beta\}$ and $i \in \{1, \ldots, n\}$.

Notice that the input values are either integers or fixed point numbers and they are only subject to addition operations. Therefore, for security against $\beta - 1$ (instead of $\lfloor \frac{\beta-1}{2} \rfloor$) compromised parties, we perform (β, β) sharing for input values to obtain $[\cdot]_\beta$. For DP noise generation, we still rely on the honest majority assumption and correspondingly use the usual $(\beta, \lceil \frac{\beta+1}{2} \rceil)$ sharing. After the parties $C_1, \ldots, C_\beta$ jointly computed the shared result $[w]_\beta$ of the sanitization mechanism, the parties collaborate to reconstruct the result w as $w = \mathsf{Rec}([w]_\beta)$.

General Overview. Intuitively, the instantiation for the most part unfolds the mathematical operations used in the algorithms presented in § 3 and replaces them by the corresponding SMPCs for arithmetic operations listed in § 2.

Additions for both integers and fixed point numbers are very fast, while for floating point values, the protocol is costly. We thus choose the n shared data inputs $[d_1], \ldots, [d_n]$ to the mechanisms to be fixed point or integer numbers respectively to lower the cost of adding them together to yield the joint unperturbed query result $[d_1] + \ldots + [d_n]$. [4] We compute the noise values in floating point form as the required logarithm and exponentiation operations are only available for distributed floating point arithmetic. We use the conversion operations $\mathsf{FP2FL}$, $\mathsf{FL2Int}$, $\mathsf{Int2FL}$ whenever neces-

[4] As we pointed out before, this also allows us to rely on (β, β) sharing for input values.

```
In: Shared fixed point form $(\gamma, f)$ inputs $[d_1]_\beta, \ldots, [d_n]_\beta$; $\lambda = \frac{\Delta f}{\epsilon}$
Out: $w = (\sum_{i=1}^{n} d_i) + \mathsf{Lap}(\lambda)$ in the fixed point form
$[d]_\beta = [d_1]_\beta$
for $i = 2$ to $n$ do
    $[d]_\beta = \mathsf{FPAdd}([d]_\beta, [d_i]_\beta)$
$[r_x] = \mathsf{RandInt}(\gamma + 1)$; $[r_y] = \mathsf{RandInt}(\gamma + 1)$
$\langle [v_x], [p_x], 0, 0 \rangle = \mathsf{FP2FL}([r_x], \gamma, f = \gamma, \ell, k)$
$\langle [v_y], [p_y], 0, 0 \rangle = \mathsf{FP2FL}([r_y], \gamma, f = \gamma, \ell, k)$
$\langle [v_{x/y}], [p_{x/y}], 0, 0 \rangle = \mathsf{FLDiv}(\langle [v_x], [p_x], 0, 0 \rangle, \langle [v_y], [p_y], 0, 0 \rangle)$
$\langle [v_{ln}], [p_{ln}], [z_{ln}], [s_{ln}] \rangle = \mathsf{FLLog2}(\langle [v_{x/y}], [p_{x/y}], 0, 0 \rangle)$
$\langle [v_z], [p_z], [z_z], [s_z] \rangle = \mathsf{FLMul}(\frac{\lambda}{\log_2 e}, \langle [v_{ln}], [p_{ln}], [z_{ln}], [s_{ln}] \rangle)$
$[z] = \mathsf{FL2FP}(\langle [v_{z_1}], [p_{z_1}], [z_{z_1}], [s_{z_1}] \rangle, \ell, k, \gamma)$
$[w]_\beta = \mathsf{FPAdd}([d]_\beta, [z])$
return $w = \mathsf{Rec}([w]_\beta)$
```

Table 2: Protocol: Distributed LM

```
In: Shared integer number $(\gamma)$ inputs $[d_1]_\beta, \ldots, [d_n]_\beta$; $\lambda = e^{-\frac{\epsilon}{\Delta f}}$; $\alpha = \frac{1}{\ln \lambda \cdot \log_2 e}$
Out: integer $w = (\sum_{i=1}^{n} d_i) + \mathsf{DLap}(\lambda)$
$[d]_\beta = [d_1]_\beta$
for $i = 2$ to $n$ do
    $[d]_\beta = \mathsf{IntAdd}([d]_\beta, [d_i]_\beta)$
$[r_x] = \mathsf{RandInt}(\gamma + 1)$; $[r_y] = \mathsf{RandInt}(\gamma + 1)$
$\langle [v_x], [p_x], 0, 0 \rangle = \mathsf{FP2FL}([r_x], \gamma, f = \gamma, \ell, k)$
$\langle [v_y], [p_y], 0, 0 \rangle = \mathsf{FP2FL}([r_y], \gamma, f = \gamma, \ell, k)$
$\langle [v_{lnx}], [p_{lnx}], [z_{lnx}], [s_{lnx}] \rangle = \mathsf{FLLog2}(\langle [v_x], [p_x], 0, 0 \rangle)$
$\langle [v_{lny}], [p_{lny}], [z_{lny}], [s_{lny}] \rangle = \mathsf{FLLog2}(\langle [v_y], [p_y], 0, 0 \rangle)$
$\langle [v_{\alpha lnx}], [p_{\alpha lnx}], [z_{\alpha lnx}], [s_{\alpha lnx}] \rangle =$
$\mathsf{FLMul}(\alpha, \langle [v_{lnx}], [p_{lnx}], [z_{lnx}], [s_{lnx}] \rangle)$
$\langle [v_{\alpha lny}], [p_{\alpha lny}], [z_{\alpha lny}], [s_{\alpha lny}] \rangle =$
$\mathsf{FLMul}(\alpha, \langle [v_{lny}], [p_{lny}], [z_{lny}], [s_{lny}] \rangle)$
$\langle [v_{z_1}], [p_{z_1}], [z_{z_1}], [s_{z_1}] \rangle =$
$\mathsf{FLRound}(\langle [v_{\alpha lnx}], [p_{\alpha lnx}], [z_{\alpha lnx}], [s_{\alpha lnx}] \rangle, 0)$
$\langle [v_{z_2}], [p_{z_2}], [z_{z_2}], [s_{z_2}] \rangle =$
$\mathsf{FLRound}(\langle [v_{\alpha lny}], [p_{\alpha lny}], [z_{\alpha lny}], [s_{\alpha lny}] \rangle, 0)$
$[z_1] = \mathsf{FL2Int}(\langle [v_{z_1}], [p_{z_1}], [z_{z_1}], [s_{z_1}] \rangle, \ell, k, \gamma)$
$[z_2] = \mathsf{FL2Int}(\langle [v_{z_2}], [p_{z_2}], [z_{z_2}], [s_{z_2}] \rangle, \ell, k, \gamma)$
$[w]_\beta = \mathsf{IntAdd}([d]_\beta, \mathsf{IntAdd}([z_1], -[z_2]))$
return $w = \mathsf{Rec}([w]_\beta)$
```

Table 3: Protocol: Distributed DLM

sary.

```
In: $[d_1], \dots, [d_n]$; the number $m$ of candidates; $\lambda = \frac{\epsilon}{2}$
Out: $m$-bit $w$, s.t. smallest $i$ for which $w(i) = 1$ denotes winning candidate $a_i$
$I_0 = \langle 0, 0, 1, 0 \rangle$
for $j = 1$ to $m$ do
    $[z_j]_\beta = 0$
    for $i = 1$ to $n$ do
        $[z_j]_\beta = \mathsf{IntAdd}([z_j]_\beta, [d_i(j)]_\beta)$
    $\langle [v_{z_j}], [p_{z_j}], [z_{z_j}], [s_{z_j}] \rangle = \mathsf{Int2FL}([z_j]_\beta, \gamma, \ell)$
    $\langle [v_{z'_j}], [p_{z'_j}], [z_{z'_j}], [s_{z'_j}] \rangle =$
    $\mathsf{FLMul}(\lambda \cdot \log_2 e, \langle [v_{z_j}], [p_{z_j}], [z_{z_j}], [s_{z_j}] \rangle)$
    $\langle [v_{\delta_j}], [p_{\delta_j}], [z_{\delta_j}], [s_{\delta_j}] \rangle =$
    $\mathsf{FLExp2}(\langle [v_{z'_j}], [p_{z'_j}], [z_{z'_j}], [s_{z'_j}] \rangle)$
    $\langle [v_{I_j}], [p_{I_j}], [z_{I_j}], [s_{I_j}] \rangle = \mathsf{FLAdd}(\langle [v_{I_{j-1}}], [p_{I_{j-1}}],$
    $[z_{I_{j-1}}], [s_{I_{j-1}}] \rangle, \langle [v_{\delta_j}], [p_{\delta_j}], [z_{\delta_j}], [s_{\delta_j}] \rangle)$
$[r] = \mathsf{RandInt}(\gamma + 1)$
$\langle [v_r], [p_r], 0, 0 \rangle = \mathsf{FP2FL}([r], \gamma, f = \gamma, \ell, k)$
$\langle [v'_r], [p'_r], [z'_r], [s'_r] \rangle =$
$\mathsf{FLMul}(\langle [v_r], [p_r], 0, 0 \rangle, \langle [v_{I_m}], [p_{I_m}], [z_{I_m}], [s_{I_m}] \rangle)$
$j_{\mathsf{min}} = 1$; $j_{\mathsf{max}} = m$
while $j_{\mathsf{min}} < j_{\mathsf{max}}$ do
    $j_{\mathsf{M}} = \lfloor \frac{j_{\mathsf{min}} + j_{\mathsf{max}}}{2} \rfloor$
    if $\mathsf{FLLT}(\langle [v_{I_{j_\mathsf{M}}}], [p_{I_{j_\mathsf{M}}}], [z_{I_{j_\mathsf{M}}}], [s_{I_{j_\mathsf{M}}}] \rangle, \langle [v'_r], [p'_r], [z'_r], [s'_r] \rangle)$ then
        $j_{\mathsf{min}} = j_{\mathsf{M}} + 1$ else $j_{\mathsf{max}} = j_{\mathsf{M}}$
return $w_{j_{\mathsf{min}}}$
```

Table 4: Protocol: Distributed EM

Random Number Generation. As we have seen in the previous section, our algorithms rely heavily on the generation of a random number in the interval $(0, 1]$ drawn according to the uniform distribution $\mathcal{U}_{(0,1]}$. Unfortunately, the SMPC suite we consider does not include such a function. Hence we devised an SMPC protocol that is based on the idea of encoding such a random number generation using the primitive $\mathsf{RandInt}$ for the generation of a random integer (e.g., cf. steps 4 and 5 in Table 2. We first generate a shared $(\gamma + 1)$-bit integer $[r_x]$ using the SMPC primitive $\mathsf{RandInt}$. We then consider this integer to be the fractional part of fixed point number, whose integer part is 0 (by choosing $f = \gamma$). Afterwards, the fixed point number is converted to floating point by using the function $\mathsf{FP2FL}$ and disregarding the shared sign bit. Notice that strictly speaking, this generates a random number in $[0, 1)$. We can achieve a transition to the expected interval $(0, 1]$ by slightly modifying the conversion primitive $\mathsf{FP2FL}$ such that the shared $[0]$ is replaced by the sharing of $[1]$ in step 3 [9, § 5]. We could avoid the modification of $\mathsf{FP2FL}$ and instead transition into the desired interval by subtracting the random number from 1, but this requires an additional costly addition step.

Exponentiation and Logarithm. The work by Aliasgari et al. [9] provides SMPCs for computing exponentiation with base 2 ($\mathsf{FLExp2}$) and logarithm to base 2 ($\mathsf{FLLog2}$). Since we often require exponentiation and logarithm to a base $b \neq 2$, we use the following mathematical properties $b^a = 2^{a(\log_2 b)}$ and $\log_b x = \frac{\log_2 x}{\log_2 b}$ to compute exponentiation and logarithm for any base b. For instance, steps 8 - 9 in Table 2 and steps 7 - 8 in Table 4 use the above equations to compute logarithm and exponentiation to base e respectively.

Distributed (Discrete) Laplace Mechanism. The protocols to compute the distributed Laplace and distributed discrete Laplace mechanism are shown in Tables 2 and 3 respectively. While the former expects fixed point numbers as inputs, the latter expects integers. Both protocols follow along the same lines, but while the Laplace mechanism can use the simplification $\ln r_x - \ln r_y = \ln \frac{r_x}{r_y}$ and thus reduce the number of necessary logarithm operations $\mathsf{FLLog2}$ as well as the number of follow-up operations, this is not possible for its discrete counterpart due to the floor operations $\mathsf{FLRound}$.

Distributed Exponential Mechanism. The protocol to compute the distributed exponential mechanism using SMPCs is presented in Table 4. Each shared input $[d_i]$ consists of an integer array of size m, representing the histogram of participant P_i. The instantiation follows the steps of the algorithm presented in Table 1c in § 3 by using the insights and techniques we presented in this section. We straightforwardly implement the binary search to find the winning interval/candidate in steps 13-17. Note that we need a slightly simplified version of the FLLT protocol that outputs a value $\{0, 1\}$ that does not need to be shared, thus allowing us to output $w_{j_{\mathsf{min}}}$ immediately without reconstruction, which would require additional interactions. We can also improve performance by running m instances of steps 3-8 in parallel.

Mechanisms in the Malicious Setting. In order to achieve DP against malicious computation parties, we had to strengthen the SMPC protocols so as to make them resistant to computation parties deviating from the protocol [16, 34]. Intuitively, to maintain security, one has to enforce the following TWO additional properties: (i) The protocol-instance observations of honest parties are consistent with each other; (ii) Every party proves that each step of its computation was performed correctly.

Given the real-world impracticality of information-theoretically secure channels and subsequently information-theoretically secure SMPC protocols, in the malicious setting we shift to the computational setting. In particular, we employ a computational verifiable secret sharing scheme (VSS) [12, 22, 48] instead of the basic secret sharing scheme to achieve the first property. For the second property, we introduce zero-knowledge (ZK) proofs such that a party can prove that a correct secret value is shared among the parties [34] and that shared secret values satisfy some mathematical conditions (e.g., they are in a pre-defined range) [16]. We note that these two changes are not sufficient to maintain *liveness*: compromised parties may crash to stop the computation. Although a stronger resiliency condition of $\beta \geq 3t+1$ (instead of $\beta \geq 2t+1$) can ensure the protocol completion, the honest parties can always determine which parties crashed during a computation and replace those before restarting the protocol. Therefore, for our implementation, we stick to $\beta \geq 2t + 1$ parties.

Limitations of Finite-Precision Instantiations. While the theoretical definition of sanitization mechanisms for DP operates on reals $r \in \mathbb{R}$ (or integers $z \in \mathbb{Z}$), the implementations of such mechanisms have to approximate these mathematical abstractions by finite-precision representations due to the physical limitations of actual machines. This mismatch has been shown to give rise to several attacks, as shown by Mironov [45] and Gazeau et al. [32] (cf. Appendix C for more details). The techniques proposed in these works to prevent such attacks rely on arithmetic operations that can be implemented using our arithmetic SMPCs. This allows us to make PrivaDA immune to these attacks. For the sake of simplicity, we omitted this extension from our presentation.

5 Security Analysis

In this section we state the security model, conduct a security analysis in the HbC setting, and discuss how to extend this result to a malicious setting.

We first recall the standard notion of t-secrecy for SMPC, which is formulated as in [9] except for a small modification to accommodate the computation parties. The following definitions refer to computation parties $\mathcal{C} = \{C_1, \ldots, C_\beta\}$ engaging in a protocol Π that computes function $y = f(D)$, where $D = D_1, \ldots, D_n$ and D_i denotes the input of party P_i and $y \in \mathcal{R}$ is the output.

Definition 4 (View) *C_i's view consists of its shares $\{[D]\}_{C_i}$ and its internal random coin tosses r_i, as well as the messages M exchanged with the other parties during the protocol execution induced by the other parties' random coin tosses h: i.e., $\mathsf{VIEW}_{\Pi(D,h)}(C_i) = (\{[D]\}_{C_i}, r_i, M)$. $\mathsf{VIEW}_{\Pi(D)}(C_i)$ denotes the corresponding random function conditioned to the other parties' coin tosses.*

Definition 5 (t-Secrecy) *A protocol Π is t-private in the presence of HbC adversaries if for each coalition $I = \{C_{i_1}, C_{i_2}, \ldots, C_{i_t}\} \subset \mathcal{C}$ of HbC computation parties of size $t < \beta/2$, there exists a probabilistic polynomial time simulator S_I such that $\{S_I(\{[D]\}_I, f(D))\} \equiv \{\mathsf{VIEW}_{\Pi(D,h)}(I), y\}$. Here, $\{[D]\}_I = \bigcup_{C \in I} \{[D]\}_C$, $\equiv$ denotes indistinguishability, $\mathsf{VIEW}_{\Pi(D,h)}(I)$ the combined view of the parties in I, and h the coin tosses of the parties in $\mathcal{C} \backslash I$.*

Let $\mathcal{V}_\Pi$ be the set of all possible views for the protocol Π. We now formally define the notion of DDP for protocols, close in spirit to the one introduced in [29]. Here, two vectors $D, D' \in \mathcal{D}^n$ are said to be *neighbors* if they differ in exactly one coordinate, which corresponds to the scenario in which exactly one user changes her input.

Definition 6 (ϵ-DDP) *We say that the data sanitization procedure implemented by a randomized protocol Π among β computation parties $\mathcal{C} = \{C_1, \ldots, C_\beta\}$ achieves ϵ-DDP w.r.t. a coalition $I \subset \mathcal{C}$ of HbC computation parties of size t, if the following condition holds: for any neighboring input vectors $D, D' \in \mathcal{D}^n$ and any possible set $\mathcal{S} \subseteq \mathcal{V}_\Pi$, $Pr[\mathsf{VIEW}_{\Pi(D)}(I) \in \mathcal{S}] \leq e^\epsilon Pr[\mathsf{VIEW}_{\Pi(D')}(I) \in \mathcal{S}]$ holds.*

For the malicious setting, the coalition I of HbC parties in Definitions 5 and 6 is replaced by an equal-sized coalition I^{M} of malicious computationally-bounded (for a security parameter κ) parties and the protocol Π is replaced by a computational protocol Π^{M} fortified against malicious attackers with the same t-secrecy property. The above DDP relation changes to an indistinguishability-based computational DDP (IND-DDP) [19,46] relation with a negligible function $\mathrm{negl}(\kappa)$ such that $Pr[\mathsf{VIEW}_{\Pi^{\mathsf{M}}(D)}(I^{\mathsf{M}}) \in \mathcal{S}] \leq e^{\epsilon} Pr[\mathsf{VIEW}_{\Pi^{\mathsf{M}}(D')}(I^{\mathsf{M}}) \in \mathcal{S}] + \mathrm{negl}(\kappa)$.

We now state our main theorems on ϵ-DDP in the HbC model, and ϵ-IND-DDP in the malicious model and refer the readers to Appendix D for proof sketches.

Theorem 2 (ϵ-DDP) *Let $\epsilon > 0$. In the HbC setting, our distributed LM, DLM, and EM protocols achieve ϵ-DDP w.r.t. any HbC coalition $I \subset \mathcal{C}$ of size $t < \beta/2$.*

Theorem 3 (ϵ-IND-DDP) *Let $\epsilon > 0$ and κ be a sufficiently large security parameter. In the malicious setting, our distributed LM, DLM, and EM protocols achieve ϵ-IND-DDP w.r.t. any malicious coalition $I^{\mathsf{M}} \subset \mathcal{C}$ of size $t < \beta/2$, under the strong RSA and decisional Diffie-Hellman assumptions for parameter κ.*

6 Performance Analysis

Aliasgari et al. [9] microbenchmarked the performance for most of the required arithmetic SMPC protocols in the HbC setting for three computation parties. However, we could not successfully execute several library functions and their library does not handle the malicious setting. Hence we decided to develop the complete SMPC library for both the HbC and malicious setting from scratch. Here, we present our SMPC implementation for integer, fixed point, and floating point arithmetic and measure the performance costs for the distributed LM, DLM, and EM protocols in the HbC and malicious settings.

Implementation. We implement all SMPC protocols discussed in §2 as well as our DDP mechanisms as a multi-threaded object-oriented C++ code to support any number (≥ 3) of computation parties in the HbC and malicious settings. Our implementation uses the GMP library [1] for all finite field computations, the Relic toolkit [11] for elliptic curve cryptographic (ECC) constructions, and the Boost [3] and OpenSSL [4]libraries for secure communication. Our numeric SMPC libraries can be of independent interest to other distributed computation scenarios, and our complete code base is available online [2].

Experimental Setup. The experiments are performed over an 3.20 GHz (Intel i5) Linux machine with 16 GB RAM, using a 1 Gbps LAN. We run experiments for the 3-party (i.e., $\beta = 3$ and $t = 1$), and 5-party (i.e., $\beta = 5$ and $t = 2$) computation setting. The floating point numbers employed in the experiments have a bit-length of $\ell = 32$ for significands and $k = 9$ for (signed) exponents, which gives a precision of up to 2^{-256}. For integers and fixed point numbers, we use a bit-length of $\gamma = 64$, where $f = 32$ for fixed point numbers. It gives a precision of 2^{-32} for the latter. The experiments use finite fields of size 177 bits for integers, 208 bits for fixed point numbers, and 113 bits for floating point significands. For floating point exponents, as well as sign and zero

Type	**Protocol**	**HbC**		**Malicious**	
		$\beta = 3$, $t = 1$	$\beta = 5$, $t = 2$	$\beta = 3$, $t = 1$	$\beta = 5$, $t = 2$
Float	FLAdd	0.48	0.76	14.6	29.2
	FLMul	0.22	0.28	3.35	7.54
	FLScMul	0.20	0.28	3.35	7.50
	FLDiv	0.54	0.64	4.58	10.2
	FLLT	0.16	0.23	2.82	6.22
	FLRound	0.64	0.85	11.4	23.4
Convert	FP2FL	0.83	1.21	25.7	50.9
	Int2FL	0.85	1.22	25.7	50.9
	FL2Int	1.35	1.91	26.3	54.3
	FL2FP	1.40	1.96	26.8	55.3
Log	FLLog2	12.0	17.0	274	566
Exp	FLExp2	7.12	9.66	120	265

Table 5: Performance of a single 3-party and 5-party SMPC operations measured in sec

bits, significantly smaller fields suffice. In contrast to [9], we do not employ batching (which improves average computations times) since our distributed mechanisms call the individual arithmetic SMPC functions only a few times. To determine an average performance, we run the experiments ten times for both parameter sets. In Table 5, we show our results for all required SMPC functionalities in the HbC and malicious settings; in particular, we include the computation time for single 3-party and 5-party arithmetic SMPC operations measured in seconds.

Cost Analysis (HbC setting). As expected, the logarithm and exponentiation SMPC are the most expensive operations, and they will drive our distributed mechanism cost analysis. Our protocols also use Rec, IntAdd, FPAdd, RandInt SMPC, but we do not include them in Table 5 as they are local operations that can be performed significantly faster than the rest of the protocols.

Next, we determine the average performance costs for our distributed LM, DLM, and EM protocols for ($\beta = 3, t = 1$) computation parties and $100,000$ users. The distributed LM protocol has a computation cost of 15.5 sec, while the distributed DLM protocol requires around 31.3 sec. The better efficiency of the LM mechanism is due to the fact that we halved the number of costly logarithm operations FLLog2 and necessary follow-up operations by using the property $\ln r_x - \ln r_y = \ln \frac{r_x}{r_y}$, which is not possible for its discrete counterpart due to the necessary floor operations FLRound. The computation cost of the distributed EM protocol linearly depends on the number $m = |\mathcal{R}|$ of result candidates. For instance, for $m = 5$, the cost of computation is 42.3 sec.

For larger numbers of computation parties β, one can extrapolate the performance from our analysis for ($\beta = 3, t = 1$) and ($\beta = 5, t = 2$). Even for $\beta \approx 100$, we expect the distributed LM and DLM protocols to take about a few hundred seconds in the HbC setting. We also compared our experimental results with [9]. We could not reproduce their results, possibly due to the introduced memory management and correctness verifications.

Cost Analysis (Malicious Setting). As expected, the computations times for the SMPC operations secure against an active adversary are significantly higher (around 15-20 times) than those of the operations secure against an HbC adversary. The average performance costs for our distributed LM, DLM, and EM protocols for ($\beta = 3$, $t = 1$) computation parties and $100,000$ users in the malicious setting are as follows: The distributed LM protocol has an average computation cost of 344 sec, while the distributed DLM protocol requires 477 sec. The cost of the distributed EM protocol, for $m = 5$ result candidates, is 652 sec.

We stress that these operations are performed by computation parties, and that there are no critical timing restrictions on DDP computations in most real-life scenarios, such as web analytics. Nevertheless, we expect 1 order of magnitude performance gain in the HbC as well as the malicious setting by employing high performance computing servers. Furthermore, since users have to simply forward their shared values to the computation parties, which is an inexpensive operation (< 1 msec in the HbC setting and a couple of milliseconds in the malicious setting), we believe that these numbers demonstrate the practicality of PrivaDA even in a setting where clients are equipped with computationally limited devices, such as smartphones.

7 Application Scenarios

We showcase the flexibility of our architecture by briefly discussing how PrivaDA can be used to improve the state-of-the-art in three different application scenarios.

Web Analytics. Web analytics consist in the measurement, collection, analysis, and reporting of Internet data about users visiting a website. For instance, data can include user demographics, browsing behavior, and information about the clients' systems. This information is important for publishers, because it enables them to optimize their site content according to the users' interests, for advertisers, because it allows them to target a selected population, and many other parties, which we will refer to as *analysts*.

State-of-the-Art. In order to obtain aggregated user information, today, websites commonly use third party web analytics services, called *aggregators*, which however track individual users' browsing behavior across the web, thereby violating their privacy. Newer systems, e.g., a series of non-tracking web analytics systems [7, 20, 21] recently proposed by Chen et al., provide users with DP guarantees but rely on strong non-collusion assumptions. Should a collusion happen, not only the noise but also the individual user's data would be disclosed.

Protocol design in PrivaDA. The computation parties are operated by third-parties, which are possibly paid by the aggregator. In order to avoid multiple responses by each user without relying on a public key infrastructure, which is unrealistic in this setting, we add an initial step to the protocol. The publisher signs and gives each visiting user a different token, along with one or more queries and an associated expiry

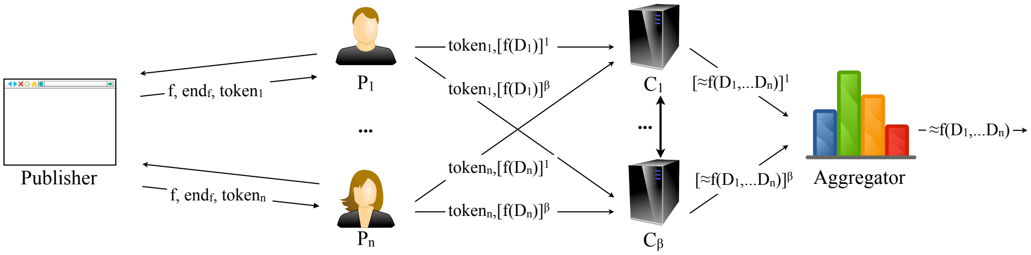

Figure 2: Privacy-preserving Web Analytics: Protocol Flow

time (within which the result has to be computed). The user sends the tokens to the computation parties, together with their answer shares, so that the computation parties are able to detect duplicates and to discard them before the aggregation. The users have just to submit their shares and can then go offline. Finally, the support for a variety of perturbation mechanisms enables the execution of different kinds of analytical queries. The protocol is depicted in Figure 2.

Anonymous Surveys. A further application scenario consists of anonymous surveys. In this setting, it is often reasonable to tolerate a little result perturbation in favor of strong privacy guarantees for the participating users.

State-of-the-Art. ANONIZE [38] is a recently proposed large-scale anonymous survey system. The authors exemplify it on an anonymous course evaluation service, in which students grade the courses they attend. However, ANONIZE does not address the problem that the survey result itself might still leak a lot of information about the individual user, which differential privacy aims at preventing.

Protocol design in PrivaDA. As compared to ANONIZE, the usage of PrivaDA yields differential privacy guarantees, besides avoiding the need to design and implement a complicated ad-hoc protocol. We exemplify the usage of PrivaDA for anonymous surveys on the previously mentioned course evaluation service. Before submitting a grade for a certain course, students have to authenticate to prove their enrollment in that class. We envision a public key infrastructure maintained by the university or an anonymous credential system used by the professor to grant her students access credentials. The computation parties will be implemented by mutually distrustful organizations, yet all interested in the results of the evaluation, such as the student association, the university administration, and so on.

Traffic Statistics for Anonymous Communication Networks (ACNs). Given their anonymous nature, it is hard to collect egress traffic statistics from ACNs, such as Tor, without violating the privacy of users. Such statistics are interesting to both designers and researchers, which might for instance want to know how much of the network traffic is made up by people trying to circumvent censorship.

State-of-the-Art. Elahi et al. recently proposed PrivEx [30], a system for collecting differentially private statistics on ACNs traffic in predefined slots of time (epochs). Their work provides two ad-hoc protocols that rely on secret sharing and distributed decryption respectively. Nevertheless, to tolerate even an HbC adversary PrivEx has to compromise on the utility or the epoch duration.

Protocol design in PrivaDA. We can easily apply PrivaDA to the problem of collecting anonymous traffic statistics: we simply let the ACN egress nodes, which relay traffic between the ACN and the destination websites, count the accesses to the different destinations that they relayed. After a fixed epoch, they then share their individual counts among mutually distrustful computation parties (e.g., privacy organizations, research centers, and service providers), which jointly compute the overall egress traffic in a privacy-preserving manner with optimal utility.

8 Conclusion and Future Work

Although it is a long-held belief that SMPCs may be used to generically design differentially private data aggregation protocols, such an approach has not been undertaken so far due to the inefficiency of generic constructions. In this work we demonstrated the viability of such an approach, by designing an SMPC architecture that constitutes not only a generic, but also a practical building block for designing a variety of privacy-preserving data aggregation protocols. In particular, the computational effort on the client side is negligible, which makes PrivaDA suitable even for computationally limited devices, such as smartphones. In contrast to previous works, PrivaDA supports a variety of perturbation mechanisms, offers strong privacy guarantees as well as optimal utility, and is resistant to answer pollution attacks. Furthermore, PrivaDA can support a large number of clients without any significant performance penalty.

For the security of certain arithmetic operations, the SMPC schemes we use assume that the majority of the computation parties are not colluding. This assumption is present in any secret sharing-based SMPC scheme[5]. There exist SMPCs based on other techniques (homomorphic encryption, oblivious transfer, etc.) that do not assume an honest majority (e.g., [40, 54]), but that are currently less efficient. Nevertheless, since PrivaDA is parameterized over the underlying arithmetic SMPCs, it can take immediate advantage of the rapid progress in this research field.

As a future work, we indeed intend to investigate the usage of alternative SMPC schemes and to explore the integration of more sanitization mechanisms. To foster further progress in this field, we made the implementation of PrivaDA publicly available [2]: to the best of our knowledge, this is the first publicly available SMPC implementation that supports a variety of arithmetic operations in the malicious setting.

References

[1] GMP: The GNU Multiple Precision Arithmetic Library. `http://gmplib.org`.

[5] However, stronger assumptions are common in the DP literature and deemed appropriate in many realistic scenarios.

[2] Our Distributed Differential Privacy Library. `https://sites.google.com/site/arithmeticsmpc/`.

[3] The Boost C++ Libraries. `http://www.boost.org`.

[4] The OpenSSL Project. `http://www.openssl.org`.

[5] M. Abramowitz and I. A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables.* Dover, 1964.

[6] G. Ács and C. Castelluccia. I have a DREAM! (DiffeRentially privatE smArt Metering). In *IH'11*, pages 118–132, 2011.

[7] I. E. Akkus, R. Chen, M. Hardt, P. Francis, and J. Gehrke. Non-tracking Web Analytics. In *CCS'12*, pages 687–698, 2012.

[8] D. Alhadidi, N. Mohammed, B. C. M. Fung, and M. Debbabi. Secure Distributed Framework for Achieving ϵ-Differential Privacy. In *PETS'12*, pages 120–139, 2012.

[9] M. Aliasgari, M. Blanton, Y. Zhang, and A. Steele. Secure Computation on Floating Point Numbers. In *NDSS'13*, 2013.

[10] Y. Amir, C. Nita-Rotaru, J. R. Stanton, and G. Tsudik. Secure Spread: An Integrated Architecture for Secure Group Communication. *TDSC*, 2(3):248–261, 2005.

[11] D. F. Aranha and C. P. L. Gouvêa. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.

[12] M. Backes, A. Kate, and A. Patra. Computational Verifiable Secret Sharing Revisited. In *ASIACRYPT'11*, pages 590–609, 2011.

[13] G. Barthe, G. Danezis, B. Grégoire, C. Kunz, and S. Zanella-Béguelin. Verified Computational Differential Privacy with Applications to Smart Metering. In *CSF'13*, pages 287–301, 2013.

[14] A. Ben-David, N. Nisan, and B. Pinkas. FairplayMP: A System for Secure Multiparty Computation. In *CCS'08*, pages 257–266, 2008.

[15] R. Bhaskar, A. Bhowmick, V. Goyal, S. Laxman, and A. Thakurta. Noiseless Database Privacy. In *ASIACRYPT'11*, pages 215–232, 2011.

[16] F. Boudot. Efficient Proofs that a Committed Number Lies in an Interval. In *EUROCRYPT'00*, pages 431–444, 2000.

[17] R. Canetti. Security and Composition of Multiparty Cryptographic Protocols. *Journal of Cryptology*, 13(1):143–202, 2000.

[18] O. Catrina and A. Saxena. Secure Computation With Fixed-Point Numbers. In *FC'10*, pages 35–50, 2010.

[19] T.-H. H. Chan, E. Shi, and D. Song. Privacy-Preserving Stream Aggregation with Fault Tolerance. In *FC'12*, pages 200–214, 2012.

[20] R. Chen, I. E. Akkus, and P. Francis. SplitX: High-Performance Private Analytics. In *SIGCOMM'13*, 2013. to appear.

[21] R. Chen, A. Reznichenko, P. Francis, and J. Gehrke. Towards Statistical Queries over Distributed Private User Data. In *NSDI'12*, pages 13–13, 2012.

[22] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable Secret Sharing and Achieving Simultaneity in the Presence of Faults. In *Proc. 26th IEEE Symposium on Foundations of Computer Science (FOCS)*, pages 383–395, 1985.

[23] R. Cramer, I. Damgård, and Y. Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *TCC'05*, pages 342–362, 2005.

[24] G. Danezis, M. Kohlweiss, and A. Rial. Differentially Private Billing with Rebates. In *IH'11*, pages 148–162, 2011.

[25] L. Devroye. Non-Uniform Random Variate Generation, 1986.

[26] C. Dwork. Differential Privacy. In *ICALP'06*, pages 1–12, 2006.

[27] C. Dwork, K. Kenthapadi, F. McSherry, I. Mironov, and M. Naor. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In *EUROCRYPT'06*, pages 486–503, 2006.

[28] C. Dwork, F. McSherry, K. Nissim, and A. Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC'06*, pages 265–284, 2006.

[29] F. Eigner and M. Maffei. Differential Privacy by Typing in Security Protocols. In *CSF'13*, 2013.

[30] T. Elahi, G. Danezis, and I. Goldberg. PrivEx: Private Collection of Traffic Statistics for Anonymous Communication Networks. Technical Report CACR 2014-08, 2014.

[31] S. L. From and T. Jakobsen. Secure Multi-Party Computation on Integers. Master's thesis, University of Aarhus, Denmark, 2006.

[32] I. Gazeau, D. Miller, and C. Palamidessi. Preserving differential privacy under finite-precision semantics. In *QAPL'13*, pages 1–18, 2013.

[33] J. Gehrke, E. Lui, and R. Pass. Towards Privacy for Social Networks: A Zero-Knowledge Based Definition of Privacy. In *TCC'11*, pages 432–449, 2011.

[34] R. Gennaro, M. O. Rabin, and T. Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In *PODC'98*, pages 101–111, 1998.

[35] A. Ghosh, T. Roughgarden, and M. Sundararajan. Universally Utility-Maximizing Privacy Mechanisms. In *STOC'09*, pages 351–360, 2009.

[36] S. Goryczka, L. Xiong, and V. Sunderam. Secure Multiparty Aggregation with Differential Privacy: A Comparative Study. In *EDBT/ICDT'13*, pages 155–163, 2013.

[37] A. Haeberlen, B. C. Pierce, and A. Narayan. Differential Privacy under Fire. In *USENIX'11*, 2011.

[38] S. Hohenberger, S. Myers, R. Pass, and abhi shelat. ANONIZE: A Large-Scale Anonymous Survey System. In *S&P'14*, 2014.

[39] S. Inusah and T. J. Kozubowski. A Discrete Analogue of the Laplace Distribution. *JSPI*, 136(3):1090–1102, 2006.

[40] Y. Ishai, M. Prabhakaran, and A. Sahai. Secure Arithmetic Computation with No Honest Majority. In *TCC'09*, pages 294–314, 2009.

[41] M. Jawurek and F. Kerschbaum. Fault-Tolerant Privacy-Preserving Statistics. In *PETS'12*, pages 221–238, 2012.

[42] S. P. Kasiviswanathan and A. Smith. A Note on Differential Privacy: Defining Resistance to Arbitrary Side Information. Report 2008/144, 2008.

[43] D. Kifer and A. Machanavajjhala. No Free Lunch in Data Privacy. In *SIGMOD'11*, pages 193–204, 2011.

[44] F. McSherry and K. Talwar. Mechanism Design via Differential Privacy. In *FOCS'07*, pages 94–103, 2007.

[45] I. Mironov. On Significance of the Least Significant Bits for Differential Privacy. In *CCS'12*, pages 650–661, 2012.

[46] I. Mironov, O. Pandey, O. Reingold, and S. P. Vadhan. Computational Differential Privacy. In *Crypto'09*, pages 126–142, 2009.

[47] A. Molina-Markham, P. Shenoy, K. Fu, E. Cecchet, and D. Irwin. Private Memoirs of a Smart Meter. In *BuildSys'10*, pages 61–66, 2010.

[48] T. P. Pedersen. Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing. In *Crypto'91*, pages 129–140, 1991.

[49] V. Rastogi and S. Nath. Differentially Private Aggregation of Distributed Time-Series with Transformation and Encryption. In *SIGMOD'10*, pages 735–746, 2010.

[50] M. K. Reiter. Secure Agreement Protocols: Reliable and Atomic Group Multicast in Rampart. In *CCS'94*, pages 68–80, 1994.

[51] E. Shi, T.-H. H. Chan, E. G. Rieffel, R. Chow, and D. Song. Privacy-Preserving Aggregation of Time-Series Data. In *NDSS'11*, 2011.

[52] R. Wang, Y. F. Li, X. Wang, H. Tang, and X. Zhou. Learning Your Identity and Disease from Research Papers: Information Leaks in Genome Wide Association Study. In *CCS'09*, pages 534–544, 2009.

[53] A. C.-C. Yao. Protocols for Secure Computations (Extended Abstract). In *FOCS'82*, pages 160–164, 1982.

[54] C.-H. Yu, S. S. Chow, K.-M. Chung, and F.-H. Liu. Efficient Secure Two-Party Exponentiation. In *Topics in Cryptology, CT-RSA 2011*, pages 17–32. 2011.

A Basic Arithmetic SMPC Protocols

Type	**Protocol**	**Rounds**	**Interactive Operations**
Rand. Generation	RandInt	0	0
Reconstruction	Rec	1	1
Addition	IntAdd	0	0
	FPAdd	0	0
	FLAdd	$\log\ell + \log\log\ell + 27$	$14\ell + (\log\log\ell)\log\ell + (\ell+9)\log\ell + 9k + 4\log k + 37$
Multiplication	FLMul	11	$8\ell + 10$
Division	FLDiv	$2\log\ell + 7$	$2\log\ell(\ell+2) + 3\ell + 8$
Scalar Multiplication	IntScMul	0	0
	FPScMul	0	0
	FLScMul	10	$8\ell + 7$
Comparison	FLLT	6	$4\ell + 5k + 4\log k + 13$
Conversion	Int2FL	$\log\ell + 13$	$\log\ell(2\ell-3) - 11$
	FP2FL	$\log\ell + 13$	$\log\ell(2\ell-3) - 10$
	FL2Int	$3\log\log\ell + 53$	$27\ell + 3(\log\log\ell)\log\ell + 18\log\ell + 20k + 19$
	FL2FP	$3\log\log\ell + 53$	$27\ell + 3(\log\log\ell)\log\ell + 18\log\ell + 24k + 17$
Rounding	FLRound	$\log\log\ell + 30$	$15\ell + (\log\log\ell)\log\ell + 15\log\ell + 8k + 10$
Exponentiation	FLExp2	$12\log\ell + \log\log\ell + 27$	$8\ell^2 + 9\ell + \ell\log\ell + (\log\ell)\log\log\ell + 12k + 9$
Logarithm	FLLog2	$13.5\ell + 0.5\ell\log\ell + 3\log\ell + 0.5\ell\log\log\ell + 3\log\log\ell + 146$	$15\ell^2 + 90.5\ell + 0.5\ell(\log\ell)(\log\log\ell) + 3(\log\ell)\log\log\ell + 0.5\ell^2\log\ell + 11.5\ell\log\ell + 4.5\ell k + 28k + 2\ell\log k + 16\log k + 128$

Table 6: Complexity of the employed SMPC tasks

Table 6 compares the complexities of the SMPC protocols introduced in § 2. The complexity of SMPC protocols is generally measured in terms of two parameters: interactive operations and rounds. An interactive operation involves every party sending a message to every other party, while round complexity measures the number of sequential invocations of interactive operations. The additional local computations are not included in the complexity. Note that the overhead to handle exceptions such as overflow, underflow, invalid operation, division by zero is not included.

ID	Assumptions	Utility (error)	Crypto-scheme & Perturbation Mechanism	Adversary type	Kind of queries
RN'10 [49]	#hon. users $\geq \gamma N$, bidirectional communication	$O(\frac{\Delta}{\epsilon}(\frac{k}{\gamma N})^2)$, w.c.: $O(N^2)$ $k =$ real #hon. users	Paillier scheme, Lap noise	malicious aggr., no failure	sum-statistics for time-series data (counting queries)
SCRCS'11 [51]	#hon. users $\geq \gamma N$	$O(\frac{\Delta}{\epsilon\gamma})$, w.c.: $O(\sqrt{N})$	Pollard's Rho, diluted* DLap noise	HbC aggr.	as above
AC'11 [6]	#failures $\leq \alpha N$, several keys to store for user	$O(\frac{\Delta}{\epsilon}\beta(\frac{1}{2}, \frac{1}{1-\alpha})^{-1})$, w.c.: $O(\beta(\frac{1}{2}, N)^{-1})$	modulo-addition scheme, Lap noise	malicious aggr., failures	as above
CSS'12 [19]	#hon. users $\geq \gamma N$	$\tilde{O}(\epsilon^{-1}(\log N)^{1.5})$, w.c: $\tilde{O}(\sqrt{N}(\log N)^{1.5})$	Pollard's Rho , diluted DLap noise	HbC aggr., failures	as above
CRFG'12 [21]	no collusion aggr.-publ., pre-establ. queries, bidirectional communication	$O(\frac{\sqrt{\log N}}{\epsilon})$	Goldwasser-Micali scheme, binomial noise	HbC aggr., malicious publ.	SQL-style queries (yes/no answers per buckets)
ACHFG'12 [7]	as above	$O(\frac{\Delta}{\epsilon})$	Paillier scheme, Lap noise	as above	as above
JK'12 [41]	no collusion aggr.-auth.	$O(\frac{\Delta}{\epsilon})$	Paillier scheme , Shamir's secret sharing , DLap noise	malicious aggr., HbC auth., failures	linear queries for time-series data
PrivaDA	hon. majority between computation parties	$O(\frac{\Delta}{\epsilon})$	SMPC, Lap, DLap noise, Expon. Mech.	malicious aggr.	multiple kinds of queries

* *Diluted* DLap noise: according to a certain probability p it follows the DLap distribution, otherwise it is set to 0.

Table 7: Comparison between the existing DDP schemes

B Detailed Comparison with Related Work

Table 7 compares some of the most important works about DDP with ours. Here, N denotes the total number of users; Δ is used to denote the sensitivity of the respective query (see § 2); the function $\beta(\cdot, \cdot)$ is defined $\beta(x, y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$, where $\Gamma(x) = \int_0^{+\infty} x^{t-1}e^{-x}dx$; γ is a lower bound on the fraction of honest users that we require as to guarantee DP; and α is an upper bound on the number of failures (i.e., data that do not reach the aggregator) the system can accept. For the specifics of how the noise for sanitization is generated for the individual approaches (i.e., use of Gaussian or Gamma distributions to generate the Laplace noise) we refer to [36]. We note that all papers in the table assume some compromised users (or computation parties), that is, users that follow the protocol correctly but may collude with the aggregator, passing him some information like the noise they have added or their data. Furthermore, the table specifies whether third parties, such as data aggregators (aggr.) or website publishers (publ.), are HbC or malicious (i.e., allowed to deviate from the protocol).

Utility. As the table demonstrates, a central drawback of all fully distributed models we compare is the poor utility of the result, due to the fact that the amount of noise each user has to add in order to satisfy privacy guarantees depends on other users' behaviors (i.e., the fraction of possibly malicious users and the probability of failure specified by γ, α, which are supposed to be known in advance and that must not be exceeded so as to achieve DP). The more users are falsely assumed to be malicious (i.e., small γ, large k) the lower the final accuracy in the worst case (w.c.). In PrivaDA, instead, the noise is generated in a distributed fashion starting from a random seed, which is jointly computed by the computation parties: differently from the fully distributed models, the final amount of noise obtained is exactly the one required to achieve DP (i.e., the utility is optimal), irrespectively of the number of computation parties, the fraction of honest entities, or the probability of failures.

Non-Collusion. Similarly to [7,21,41], PrivaDA relies on a non-collusion assumption, but contrary to those approaches we distribute the trust not only amongst two, but multiple parties (for which it suffices to assume an honest majority). In [41] an extension to the distributed case is proposed but the authors do not specify a method to distributively generate the noise. We note that we use mutually distrustful computation parties to mitigate the computational effort from the users, but that we could in principle let the users directly execute the perturbation phase if external parties were to be avoided.

Supported Queries. Another drawback, common to all previous models, is the restriction to specific queries and perturbation mechanisms. Most of the models described above, indeed, consider only counting queries, where the function is limited to weighted sums or even only supports sums, and use the Laplace or discrete Laplace mechanism to perturb data. The exponential mechanism, allowing perturbation in case of non numerical queries, is studied in [8]. They propose a method to securely apply it using SMPC. However, the system they propose is valid only for a two-party setting, differently from ours, that instead targets a multiparty scenario. By contrast, PrivaDA does support all three of the above mechanisms, providing a uniform framework to answer different kinds of queries in a differentially private manner.

C Limitations of Finite-precision Instantiations

While the theoretical definition of sanitization mechanisms for DP operates on reals $r \in \mathbb{R}$ (or integers $z \in \mathbb{Z}$), the implementations of such mechanisms have to approximate these mathematical abstractions by finite-precision representations due to the physical limitations of actual machines. This mismatch has been shown to give rise to several attacks, as pointed out by Mironov [45] and Gazeau et al. [32]. Mironov [45] shows that the irregularities of floating point implementations result in porous Laplace distributions, thus undermining the privacy guarantees of floating point implementations of this sanitization mechanism. He proposes the *snapping mechanism*, which truncates large values and rounds the final result so as to achieve DP of the implementation. Gazeau et al. [32] show that, in general, approximation errors of any kind of finite-precision representation of reals can lead to the disclosure of secrets. They provide a solution to fix such privacy breaches for a large class of sanitization mechanisms. The solution is based on the concept of *closeness* and uses rounding and truncation to guarantee a limited (but acceptable) variant of DP.

D Postponed Proof Sketches

D.1 Proof Sketch for Theorem 2

We start our analysis by proving t-secrecy for our protocols and then use this property to prove ϵ-DDP. The SMPC arithmetic protocols over integers, fixed and floating point numbers internally use only two basic SMPC primitives over finite field $\mathbb{F}_q$, namely, the addition and multiplication primitives for shared secret values from $\mathbb{F}_q$. Due to the linearity of the secret sharing protocol, addition can be performed locally for $n > t$; however, multiplication requires one interaction among all parties for $n \geq 2t+1$. Assuming secure instances of distributed addition and multiplication protocols over $\mathbb{F}_q$ [34] (and secure protocols built on top of them), Aliasgari et al. [9] have proved the correctness and t-secrecy properties of the SMPC arithmetic protocols employed in our mechanisms using Canetti's composition theorem [17]. More formally, they suggested that one can build a simulator for their arithmetic SMPC protocols by invoking simulators for the corresponding building blocks such that the resulting environment is indistinguishable from the real protocol execution of participants.

The proof of t-secrecy for our protocols follows along the same lines, building a simulator for each of the distributed DP mechanisms using the simulators for the underlying floating point arithmetic SMPC protocols and the other building blocks, such that the corresponding environment is indistinguishable from the corresponding real distributed DP protocol execution.

The correctness and t-secrecy properties of our SMPC protocols allow us to lift the DP analysis for the LM, DLM, and EM algorithms from § 3 to the corresponding SMPC protocols. In particular, the correctness property ensures that the result is perturbed as specified by the LM, DLM, and EM algorithms, while the t-secrecy of the SMPC arithmetic protocols ensures that no information about user inputs and the noise is available to the adversary controlling the t compromised computation parties.

D.2 Proof Sketch for Theorem 3

Since the computational verifiable secret sharing (VSS) scheme we use [48] enjoys the perfect secrecy property, the t-secrecy analysis for the SMPC protocols in the malicious setting remains almost the same as in the HbC setting. Nevertheless, the active adversary can target the secure communication channels between the honest parties, whose security relies on the decisional Diffie-Hellman assumption (or another stronger Diffie-Hellman variant). However, an active adversary can only break channel secrecy and consequently t-secrecy of SMPC protocols with a negligible probability (in κ).

The correctness of the computational SMPC protocols is also maintained in the malicious setting up to a negligible probability in the security parameter κ: For a computational VSS scheme, correctness requires the discrete logarithm assumption [48]; zero-knowledge (ZK) range proofs require the strong RSA assumption [16]; and finally, the ZK proofs for the secure multiplication require the discrete logarithm assumption [34].

As a result, using the correctness and t-secrecy properties of the computational SMPC schemes we can lift the DP analysis for the LM, DLM, and EM algorithms from § 3 to the corresponding SMPC-based protocol by only introducing an additive negligible factor corresponding to the event that one of the above assumption is broken.