Applications of Secure Multiparty Computation
P. Laud and L. Kamm (Eds.)

doi:10.3233/978-1-61499-532-6-81

Chapter 5
Achieving Optimal Utility for Distributed Differential Privacy Using Secure Multiparty Computation

Fabienne EIGNER [a], Aniket KATE [a], Matteo MAFFEI [a], Francesca PAMPALONI [b], and Ivan PRYVALOV [a]

[a] *CISPA, Saarland University, Germany*
[b] *General Electric, Italy*

Abstract. Computing aggregate statistics about user data is of vital importance for a variety of services and systems, but this practice seriously undermines the privacy of users. Recent research efforts have focused on the development of systems for aggregating and computing statistics about user data in a distributed and privacy-preserving manner. Differential privacy has played a pivotal role in this line of research: the fundamental idea is to perturb the result of the statistics before release, which suffices to hide the contribution of individual users.

In this paper, we survey existing approaches to privacy-preserving data aggregation and compare them with respect to system assumptions, privacy guarantees, utility, employed cryptographic methods, and supported sanitization mechanisms. Furthermore, we explore the usage of secure multiparty computations (SMC) for the development of a general framework for privacy-preserving data aggregation. The feasibility of such an approach is a long-held belief, which we support by providing the first efficient cryptographic realization. In particular, we present PrivaDA, a new and general design framework for distributed differential privacy that leverages recent advances in SMC on fixed and floating point numbers. PrivaDA supports a variety of perturbation mechanisms, e.g. the Laplace, discrete Laplace, and exponential mechanisms. We demonstrate the efficiency of PrivaDA with a performance evaluation and its usefulness by exploring two potential application scenarios, namely, web analytics and lecture evaluations.

Keywords. Differential privacy, secure multiparty computation

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 users' electricity consumption, which allows them to compute prices based on energy usage trends, to opti-

mize energy distribution, and so on; 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 is the user data aggregated: companies find it convenient to collect and process user data directly, but this gives them access to a wealth of sensitive information. For instance, web analytics rely on user tracking, which allows 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 the habits of citizens can be reconstructed from the electricity consumption information collected by smart meters [1] and how the identity and state of health of individuals can be derived from genome-wide association studies [2].

Privacy-Preserving Statistics

The research community has long struggled to understand what privacy means in the context of statistics and, consequently, to devise effective privacy protection techniques.

Differential privacy. Differential privacy (DP) [3] is a popular framework for defining and enforcing 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 [4, 5] 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. [6, 7]), in which mutually distrustful, and potentially compromised, parties collaborate to compute statistics about distributed data. In particular, Dwork et al. [6] were the first to suggest the idea of employing *secure multiparty computation* (SMC) 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 so as to obtain DP even in the presence of malicious parties that deviate from the protocol.

State-of-the-art. Several specialized cryptographic protocols have been proposed recently to solve this problem (e.g. [4, 8–16]), and have paved the way for the enforcement of DP in challenging scenarios, such as smart metering [17,18] and web analytics [11,19, 20]. The different works can be grouped into *fully distributed* approaches [10, 12, 15, 16] (see [13] for a comparison), in which users themselves perform the sanitization mechanism in a distributed manner and *server-based* approaches [11, 14, 19, 20] that rely on few (non-colluding) *honest but curious (HbC)* parties, e.g. an aggregator and a publisher, to compute the noise. For a more detailed comparison we refer to Sec. 2.

Despite the tremendous progress made in this field, the widespread deployment of DP in modern systems still faces some open challenges.

First, many existing approaches to DDP exploit the divisibility properties of certain noise mechanisms and let each party produce a little amount of noise, the sum of which yields the noise required to achieve DP. This solution is affected by a trade-off between privacy and *utility*, as 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 noise has 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 will show in Sec. 2. 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.

Second, several schemes suffer from the answer pollution problem: a single party can substantially pollute the aggregate result by adding excessive noise.

Third, many schemes involve a significant computational effort, which makes them impractical in several scenarios, e.g. for aggregating data stored on mobile devices with limited computation power.

Last, existing solutions are tailored to individual datatypes and perturbation mechanisms. Computing different kinds of queries or employing different perturbation mechanisms requires the use of different protocols that 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.

Our Contributions

In this work, we review and compare existing approaches to distributed privacy-preserving data aggregation. Furthermore, we present PrivaDA, the first generic framework for computing differentially private statistics about distributed data. We show how to achieve provable DDP guarantees, while overcoming the previously discussed limitations, by leveraging recently proposed SMC protocols for floating point numbers [9], fixed point numbers [21], and integers [22]. Our construction refines these schemes, originally designed for the honest but curious setting, so as to make them secure even in the malicious setting. Moreover, considering the recent work by Bendlin et al. [23] and Damgård et al. [24], we make these schemes tolerate any number of faults.

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 performance, SMC is actually conducted by computing parties that collect input shares from each client and perform the required computations. For the perturbation phase, the fundamental idea is to let computing parties jointly compute a random seed (i.e. a random variable in $(0,1)$), which is then used to produce the required noise by encoding.

The distinctive features of our approach are the following.

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.

Strong privacy. As long as at least one of the computation parties is honest, malicious parties cannot learn the aggregate result, the seed or the noise (i.e. DDP is achieved). This is a fundamental difference from other approaches (e.g. [11,19,20], where colluding parties can immediately read the individual user's data.

Optimal utility and resistance to pollution attacks. The result is perturbed with the minimal amount of noise required to achieve DDP, irrespective 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 demonstrated the feasibility of our approach by implementing the system and conducting a performance evaluation. We stress that the client does not have to perform any expensive computation: she just has to provide each computing 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.

Outline. The paper is organized in the following manner. Sec. 1 gives some necessary background information on DP and on SMC for arithmetic operations, and Sec. 2 provides a survey of related work. Sec. 3 presents our framework and our algorithms for two query sanitization mechanisms, while Sec. 4 provides an instantiation of the differentially private algorithms with efficient SMC. Sec. 5 analyzes the security of these protocols, and Sec. 6 investigates their performance. Finally, Sec. 7 illustrates two use cases for our framework.

Unpublished content. The present work extends and revises a paper published at ACSAC 2014 [25]. In this extended version we present some modifications of the original SMC that leverages recent work by Bendlin et al. [23] and Damgård et al. [24] based on [26] to improve performance and to eliminate the honest majority assumption. Furthermore, we have added a detailed survey of related works on DDP, and we have included the proofs of the main security results. Moreover, we have revised the application scenarios and added a new use case.

1. Background

In this section we present the concept of differential privacy and the cryptographic building blocks that PrivaDA builds on.

1.1. Differential Privacy

Differential privacy (DP), originally introduced by Dwork [3], has rapidly become one of the fundamental privacy properties for statistical queries. 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 that measures the strength of the privacy guarantee: the

smaller the parameter ε, the smaller the risk to join the database. We use $\mathcal{D}$ and $\mathcal{R}$ to denote the domain and range of the query respectively.

Definition 1 (Differential Privacy [3]) *A randomized function $f : \mathcal{D} \to \mathcal{R}$ is ε-*differentially private *if for all databases $D, D' \in \mathcal{D}$ so that $D \sim D'$ and every set $S \subseteq \mathcal{R}$, it holds that $Pr[f(D) \in S] \leq e^{\varepsilon} \cdot Pr[f(D') \in S]$.*

A deterministic query can be made differentially private by perturbing its results with noise. In the following, we describe three popular perturbation mechanisms. An important insight is that the amount of noise perturbation 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 the *sensitivity* of queries, which measures how much a query amplifies the distance between two inputs.

Definition 2 (Sensitivity [4]) *The sensitivity Δf of a query $f : \mathcal{D} \to \mathcal{R}$ is defined as $\Delta f = \max_{\forall D, D' \in \mathcal{D}. D \sim D'} |f(D) - f(D')|$.*

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, as 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* [4], 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. [4], this mechanism provides ε-DP, if the parameter λ is set to $\frac{\Delta f}{\varepsilon}$. The distribution is both parameterized by the sensitivity of the query and the privacy value ε.

Theorem 1 (DP of the Laplace mechanism [4]) *For all queries $f : \mathcal{D} \to \mathbb{R}$ the query $f(x) + \mathsf{Lap}(\frac{\Delta f}{\varepsilon})$ is ε-differentially private.*

Exponential mechanism. There are 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 [5] proposed the so-called *exponential mechanism*. The mechanism 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) that measures the quality of the result a with respect to input D. The higher such a score, the better (i.e. more exact) the result. The mechanism $\varepsilon_q^{\varepsilon}(D)$ aims at providing the best possible result $a \in \mathcal{R}$, while enforcing DP.

Definition 3 (Exponential mechanism [5]) *For all $q : (\mathcal{D} \times \mathcal{R}) \to \mathbb{R}$ the randomized exponential mechanism $\varepsilon_q^{\varepsilon}(D)$ for $D \in \mathcal{D}$ is defined as $\varepsilon_q^{\varepsilon}(D) :=$ return $a \in \mathcal{R}$ with probability proportional to $e^{\varepsilon q(D,a)}$.*

This definition captures the entire class of differential privacy mechanisms, as proven by McSherry and Talwar [5], who also give an encoding of Laplace noise addition by

choosing an appropriate utility function q. The authors also show that the mechanism $\mathcal{E}_q^{\frac{\varepsilon}{2\Delta q}}(D)$ provides ε-DP.

Theorem 2 (DP of the exponential mechanism [5]) *The mechanism* $\mathcal{E}_q^{\frac{\varepsilon}{2\Delta q}}(D)$ *is* ε*-differentially private.*

1.2. Secure Multiparty Computation

SMC enables a set of parties $\mathcal{P} = \{P_1, P_2, \ldots, P_\beta\}$ to jointly compute a function on their private inputs in a privacy-preserving manner [27]. More formally, every party $P_i \in \mathcal{P}$ holds a secret input value x_i, and $P_1, \ldots, P_\beta$ agree on some function f that takes β inputs. Their goal is to compute and provide $y = f(x_1, \ldots, x_\beta)$ to a recipient while making sure that the following two conditions are satisfied: (i) *correctness:* the correct value of y is computed, and (ii) *secrecy:* the output y is the only new information that is released to the recipient (see Sec. 5 for a formal definition).

Although the feasibility of SMC in the computational setting as well as in the information theoretic one has been 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, 21, 22, 28–30]. Nevertheless, most of these realizations perform limited SMC arithmetic operations over input elements belonging only to finite fields [28] or integers [22]. Thus, they are not well-suited or sufficient for DDP mechanisms. In contrast, we build on SMC for algorithms on fixed point numbers [21] and some recent work by Aliasgari et al. [9], who presented SMC arithmetic algorithms over real numbers represented in floating point form. They also propose SMC protocols for elementary functions such as logarithm and exponentiation, and conversion of numbers from the floating point form to the fixed point form or the integer form and vice-versa. Our work starts by observing that their logarithm and exponentiation SMC protocols, combined with the SMC 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.

We assume that secret sharing and basic SMC operations take place over a field $\mathbb{F}_q$. Let $[x]$ denote that the value $x \in \mathbb{F}_q$ is secret-shared among $P_1, \ldots, P_\beta$ so that participation from all β parties is required to reconstruct x. 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 computation of $[x][y]$ is interactive and performed using Beaver's multiplication triple-based technique [26].

Basic SMC protocols. We use the following SMC protocols [9, 21, 22, 28] 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 [28].
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.

Table 1. Complexity of the employed SMC tasks

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	$13.5\ell+0.5\ell\log\ell+$ $3\log\ell+0.5\ell\log\log\ell+$ $3\log\log\ell+146$	$15\ell^2+90.5\ell+0.5\ell(\log\ell)(\log\log\ell)+$ $3(\log\ell)\log\log\ell+0.5\ell^2\log\ell+11.5\ell\log\ell+$ $4.5\ell k+28k+2\ell\log k+16\log k+128$

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 FL2Int (float-to-int), Int2FL (int-to-float), FL2FP (float-to-fixed-point), and FP2FL (fixed-point-to-float) allow us to convert numbers represented as integers, floating point, or fixed point into 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 ceiling $[\lceil r \rceil]$. 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, and $[0]$ otherwise.

In Table 1, we compare the complexities of the SMC protocols described. The complexity of SMC 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. Additional local computations are not included in the complexity.

Intuitively, additions for both integer and fixed point data types are non-interactive and consequently very fast, while for floating point values, the algorithm is quite involved and is also more costly than floating multiplication. Scalar multiplications for integers are free, however, a scalar multiplication for floating point numbers asks for almost the same amount of computation as in the case of FLMul. As expected, the more complex exponentiation and logarithm algorithms are also the most costly. Note that these complexities can be significantly reduced using pre-computation and batched processing [9]. The relative efficiency of these SMC schemes plays a fundamental role in our design.

2. Literature Review

The problem of privacy-preserving data aggregation has recently received increasing attention in the research community. In this section we survey relevant literature, unifying different terminologies employed in different works to allow for a meaningful comparison. The different works can be grouped into *fully distributed* approaches, in which the users themselves utilize the sanitization mechanism in a distributed manner, and *server-based* approaches that rely on few (non-colluding) parties, e.g. an aggregator and a publisher, to compute the noise. Note that the usage of computing parties that jointly generate the noise means that PrivaDA is not fully distributed, but in contrast to existing server-based solutions, it distributes the trust among multiple parties.

2.1. Fully Distributed Setting

Dwork et al. [6] propose a protocol for distributed noise generation: their scheme, however, requires interactions among all users. Some recent works [10, 12, 15, 16] (see also [13] for a comparison) propose fully distributed systems to distributively aggregate time-series data, where the latter are directly perturbed by users and then encrypted in such a way that the aggregator is only able to decrypt their aggregation but nothing else. Rastogi and Nath [15] propose to use the additively homomorphic threshold Paillier cryptosystem to encrypt users' data, and Shamir's secret sharing scheme to combine users' decryption shares. In their system, indeed, decrypting the aggregated result requires an extra interaction round between the aggregator and the users, where the latter are required to be online until the end of the decryption. This provokes both an increased communication cost and a long delay, besides the fact that requiring that all users are simultaneously online is a strong assumption that is inappropriate in several practical settings (e.g. mobile). Furthermore, their scheme supports only sum queries (i.e. no other linear combinations are allowed).

The system proposed by Shi et al. [12, 16] does not need the extra round and is valid for more kinds of queries, even if they are always numerical. By exploiting a system to intersect user groups, [12] compensates for user failures with the help of redundant information, which allows the system to be more resistant and to support dynamic joins and leaves. However, the redundant information, besides degrading utility, increases the communication cost (here around $O(\log N)$ and no longer satisfies the *aggregator obliviousness* (AO) property (according to which the aggregator does not learn anything else but the final query result) as intermediate data are now available to him. Furthermore, that work deals with a slightly weaker definition of privacy, namely, computational (ε, δ)-DP.

Ács and Castelluccia [10] propose another scheme for smart metering systems, which is robust against failures and malicious nodes and is built on a very simple modulo addition-based encryption scheme. This system requires each user to receive, and store, several encryption keys that cancel themselves out when aggregated, so that an explicit decryption is not necessary. Both requiring a user's side storage that is linear in the size of the network, and the need to establish several keys for each user, are not easy to realize in practice. More efficient protocols based on somewhat similar ideas have been proposed in the context of smart meters [17, 18].

2.2. Server-Based Setting

Several works rely on *honest but curious* servers that are additionally assumed not to collude with each other [11,14,19,20] to compute noise perturbation: in case of collusion, not only the noise but also the individual user's data is disclosed. In our SMC scheme PrivaDA, colluding parties can neither recover the noise nor directly read the individual user's data.

In their recent work, Li and Cao [31] address the issue of users dynamically joining and leaving the data aggregation protocol, and propose a ring-based interleaved grouping construction that allows for data aggregation with high churn rates. However, their system relies on both a non-collusion assumption between the aggregator and the employed key dealer, and the participation of a threshold of honest users. Since PrivaDA uses computing parties, the aggregation is independent of the users after they forward their data to the servers, circumventing the problem of dynamic user joins and leaves.

2.3. Strengths and Weaknesses of Existing Approaches

Table 2 compares some of the most important works about DDP. Here, N denotes the total number of users, and Δ is used to denote the sensitivity of the respective query (see Sec. 1). The function $\beta(\cdot,\cdot)$ is defined as $\beta(x,y) = \frac{\Gamma(x)\Gamma(y)}{\Gamma(x+y)}$, 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 to guarantee DP, and α is an upper bound on the number of failures (i.e. data that does 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. the use of Gaussian or Gamma distributions to generate the Laplace noise) we refer to [13]. 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 honest but curious or malicious (i.e. allowed to deviate from the protocol).

Utility. As the table demonstrates, a central drawback of all fully distributed models presented above is the poor utility of the result. This is 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 that 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. In PrivaDA, instead, the noise is generated in a distributed fashion starting from a random seed that is jointly computed

by the computing 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), irrespective of the number of computation parties, the fraction of honest entities, or the probability of failures.

Non-collusion. Similarly to [11,14,19], 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 [14] 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 [32]. They propose a method to securely apply it using SMC. 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.

Table 2. Comparison between the existing DDP schemes

ID	Assumptions	Utility (error)	Cryptoscheme & Perturbation Mechanism	Adversary type	Kind of queries
RN'10 [15]	#hon. users $\geq \gamma N$, bidirectional communication	$O(\frac{\Delta}{\varepsilon}(\frac{k}{\gamma N})^2)$, worst case: $O(N^2)$ $k =$ real #hon. users	Paillier scheme, Lap noise	malicious aggr., no failure	sum-statistics for time-series data (counting queries)
SCRCS'11 [16]	#hon. users $\geq \gamma N$	$O(\frac{\Delta}{\varepsilon\gamma})$, w.c.: $O(\sqrt{N})$	Pollard's Rho, diluted* DLap noise	honest-but-curious aggr.	as above
AC'11 [10]	#failures $\leq \alpha N$, several keys to store for user	$O(\frac{\Delta}{\varepsilon}\beta(\frac{1}{2},\frac{1}{1-\alpha})^{-1})$, w.c.: $O(\beta(\frac{1}{2},N)^{-1})$	modulo-addition scheme, Lap noise	malicious aggr., failures	as above
CSS'12 [12]	#hon. users $\geq \gamma N$	$\tilde{O}(\varepsilon^{-1}(\log N)^{1.5})$, w.c: $\tilde{O}(\sqrt{N}(\log N)^{1.5})$	Pollard's Rho , diluted DLap noise	HbC aggr., failures	as above
CRFG'12 [11]	no collusion aggr.-publ., pre-establ. queries, bidirectional communication	$O(\frac{\sqrt{\log N}}{\varepsilon})$	Goldwasser-Micali scheme, binomial noise	HbC aggr., malicious publ.	SQL-style queries (yes/no answers per buckets)
ACHFG'12 [19]	as above	$O(\frac{\Delta}{\varepsilon})$	Paillier scheme, Lap noise	as above	as above
JK'12 [14]	no collusion aggr.-auth.	$O(\frac{\Delta}{\varepsilon})$	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	$O(\frac{\Delta}{\varepsilon})$	SMC, 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.

Over the course of this paper, we demonstrate that the usage of SMC is ideally suited to minimize the trade-off between utility, strong non-collusion assumptions, and privacy guarantees, inherent in existing systems.

2.4. Alternative Approaches to Privacy

Although in this chapter we focus on the notion of DP, for completeness, we refer to some recent papers that investigate the limitations of this notion [33, 34] and propose alternative definitions of privacy for statistical queries [35–37].

2.5. Alternative Secure Multiparty Computation Schemes for Arithmetic Operations

In addition to the SMC schemes employed in this paper, further SMC building blocks that support integer, floating point, and fixed point functions have recently been proposed by Kamm and Willemson [38] and Krips and Willemson [39].

3. The PrivaDA Framework

In this section we present the PrivaDA framework. We first give a general overview of the setup and then present two mechanisms for achieving DP. A third mechanism, the discrete version of the Laplace mechanism, was presented at ACSAC 2014 [25].

3.1. 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 $f : \mathcal{D}^n \to \mathcal{R}$. The range $\mathcal{R}$ of f may be a set of numerical values, but not necessarily. The computing parties are responsible for computing and perturbing the aggregate result, which is eventually returned to the data aggregator. The users communicate with the computing parties through secure and authenticated channels. Furthermore, the computing parties are pair-wise connected by secure authenticated channels. The users provide the computing parties with shares of their data and can then go offline. The computing parties engage in an interactive protocol.

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 SMC protocols we adopt for the realization of our approach are based on secret sharing: such SMCs are secure in the malicious setting (i.e. the computing parties may try to deviate from the protocol).

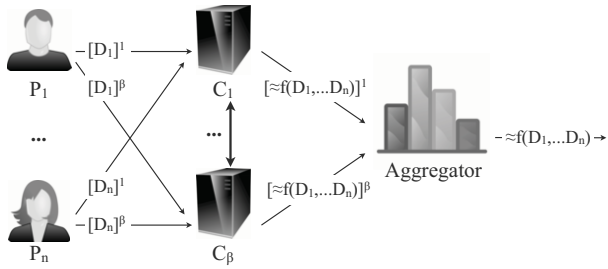

Figure 1. Protocol flow

3.2. Protocol Overview

The protocol proceeds in three steps. First, the users provide the computation parties with shares of their inputs.[1] Secondly, the computing parties run the SMC protocol to compute the aggregate statistics and perturb the result. Finally, each computing party gives the aggregator its share of the result, which is reconstructed by the aggregator. The protocol flow is depicted in Fig. 1. In the following we describe three different algorithms to compute queries sanitized with the Laplace, discrete Laplace, and exponential mechanism, respectively. To ease 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 be implemented in a very similar manner using minor modifications of the presented algorithms, as modern SMC schemes provide direct support to a large class of arithmetic operations. The algorithms described below do not rely on specific SMC protocols: we give one possible efficient instantiation in Sec. 6.

3.3. Laplace Mechanism

We now describe an algorithm for the calculation of the Laplace mechanism (LM) for n inputs. We use the following mathematical results [41, 42], which 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 the distribution $\mathsf{Lap}(\lambda) = \mathsf{Exp}(\frac{1}{\lambda}) - \mathsf{Exp}(\frac{1}{\lambda})$, where $\mathsf{Exp}(\lambda') = \frac{-\ln \mathcal{U}_{(0,1]}}{\lambda'}$. Thus,

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

In particular, we know that $\lambda = \frac{\Delta f}{\varepsilon}$ guarantees DP. We can, thus, define our algorithm for the addition of Laplace noise on n inputs as shown in Alg. 1. It takes as input (i) 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 database D_i ($d_i = f(D_i)$) of each P_i, and (ii) the privacy budget parameter λ, set to $\frac{\Delta f}{\varepsilon}$ to guarantee ε-DP. The algorithm returns the real $w = (\sum_{i=1}^{n} d_i) + \mathsf{Lap}(\lambda)$, which is computed by first computing the sum of all d_i and then drawing a random number according to the distribution $\mathsf{Lap}(\lambda)$ (lines 2 - 3) using (1). It concludes

[1] Note that our work focuses on guaranteeing the privacy of user data. To solve the orthogonal problem of pollution attacks and prevent malicious users from entering wildly incorrect input shares, we can use ZK range proofs [40] (cf. Sec. 4).

Algorithm 1: Laplace mechanism LM

Data: $d_1, \ldots, d_n$; $\lambda = \frac{\Delta f}{\varepsilon}$
Result: $(\sum_{i=1}^{n} d_i) + \mathsf{Lap}(\lambda)$

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

Algorithm 2: Exponential mechanism EM

Data: $d_1, \ldots, d_n$; $a_1, \ldots, a_m$; $\lambda = \frac{\varepsilon}{2}$
Result: winning a_k

1 $I_0 = 0$
2 **for** $j = 1$ *to* m **do**
3 $\quad z_j = \sum_{i=1}^{n} d_i(j)$
4 $\quad \delta_j = e^{\lambda z_j}$
5 $\quad I_j = \delta_j + I_{j-1}$
6 $r \leftarrow \mathcal{U}_{(0,1]}$; $r' = rI_m$
7 $k = \mathsf{binary\_search}(r', \leq, I_0, \ldots, I_m)$
8 **return** a_k

by adding the sum of the results and the noise together (line 4) and returning the result (line 5).

Privacy of LM. As 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 Thm. 1 it follows that $\mathsf{LM}(d_1, \ldots, d_n, \lambda)$ is ε-differentially private for $\lambda = \frac{\Delta f}{\varepsilon}$, where $d_i = f(D_i)$.

3.4. Exponential Mechanism

Concerning the algorithm to compute the exponential mechanism [5] (EM) for n inputs, our approach is inspired by [32], which is, however, constrained to a 2-party setting.

Inputs and outputs. The algorithm to compute the EM on the join of n databases is presented in Alg. 2. 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 $D_1, \ldots, D_n$ that are under the control of the participants $P_1, \ldots, P_n$ respectively and sanitizing the joint result using the exponential mechanism. The algorithm takes the following inputs: (i) the data sets $d_1, \ldots, d_n$ belonging to the participants $P_1, \ldots, P_n$ respectively, (ii) the list of candidates $a_1, \ldots, a_m$, and (iii) the privacy parameter λ. Note that in order to guarantee ε-DP, the parameter λ will be set to $\frac{\varepsilon}{2\Delta q}$. 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 $z_1, \ldots, z_m$ that correspond to the frequency of candidates $a_1, \ldots, a_m \in \mathcal{R}$. For instance, for the query $f :=$ "*What is your favorite lecture?*" the sequence of candidates $a_1, \ldots, a_5$ might be Algebra, Logic, Security, Cryptography, Java and the individual data set d_2 of student P_2 who prefers the lecture Security is a histogram of the form $0, 0, 1, 0, 0$. The algorithm outputs the winning candidate a_k drawn according to $\varepsilon_q^{\varepsilon}(d_1, \ldots, d_m)$.

Utility function. Our approach is general and can support any arithmetic utility function. For the sake of presentation, we focus below on the following utility function $q((z_1, \ldots, z_m), a_i) = z_i$ for all histograms $d = (z_1, \ldots, z_m)$ and candidates $a_1, \ldots, a_m$, which returns 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 $i \in \{1, 2, 4, 5\}$.

Notice that $\Delta q = 1$, so it can be omitted from the algorithm and the privacy parameter λ is thus set to $\frac{\varepsilon}{2}$.

Random variable. Our goal is to compute the exponential mechanism $\varepsilon_q^\varepsilon(D)$ for a discrete range $\mathcal{R}$, where $|\mathcal{R}| = m$. The probability mass [5, 32] for the exponential mechanism is defined as $\Pr\left[\varepsilon_q^\varepsilon(D) = a\right] = \frac{e^{\varepsilon q(D,a)}}{\sum_{j=1}^{m} e^{\varepsilon q(D,a_j)}}$. As pointed out by Alhadidi et al. [32], 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 $r \in (\sum_{k=1}^{j-1} \Pr\left[\varepsilon_q^\varepsilon(D) = a_k\right], \sum_{k=1}^{j} \Pr\left[\varepsilon_q^\varepsilon(D) = a_k\right]]$ corresponds to $a_j \leftarrow \varepsilon_q^\varepsilon(D)$. For instance, assume $\Pr\left[\varepsilon_q^\varepsilon(D) = a_1\right] = 0.3$ and $\Pr\left[\varepsilon_q^\varepsilon(D) = a_2\right] = 0.7$. 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 $a_2 \leftarrow \varepsilon_q^\varepsilon(D)$.

It is easy to see that by multiplying with $S := \sum_{j=1}^{m} e^{\varepsilon q(D,a_j)}$ the check $r \in (\sum_{k=1}^{j-1} \Pr\left[\varepsilon_q^\varepsilon(D) = a_k\right], \sum_{k=1}^{j} \Pr\left[\varepsilon_q^\varepsilon(D) = a_k\right]]$ is equivalent to $r \cdot S \in (\sum_{k=1}^{j-1} e^{\varepsilon q(D,a_k)}, \sum_{k=1}^{j} e^{\varepsilon q(D,a_k)}]$, as $\Pr\left[\varepsilon_q^\varepsilon(D) = a\right] \cdot S = e^{\varepsilon q(D,a)}$. 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 $r \cdot S \in (\sum_{k=1}^{j-1} e^{\varepsilon q(D,a_k)}, \sum_{k=1}^{j} e^{\varepsilon q(D,a_k)}]$ and returning the candidate a_k for which this check succeeds. Thus, our main effort lies in computing the necessary interval borders $(\sum_{k=1}^{j-1} e^{\varepsilon q(D,a_k)}, \sum_{k=1}^{j} e^{\varepsilon q(D,a_k)}]$.

Algorithm. Our algorithm consists of the following steps.[2] First, initialize the interval border I_0 (line 1). Second, compute the joint histogram $d = d_1 + \ldots + d_n$ (line 3) by adding the frequencies for each individual candidate. Third, compute interval borders for candidates (line 4 - 5). Fourth, draw a random value r in $(0,1]$ (line 6) and multiply this value by $I_n = \sum_{j=1}^{m} e^{\varepsilon q(D,a_j)}$, resulting in the scaled random value r'. Fifth, check in which of the intervals $(I_{j-1}, I_j]$ the random value r' falls (line 7) by using binary search that returns k so that $I_{k-1} < r' \leq I_k$. Finally, return the winning candidate a_k (line 8).

Privacy of EM. The EM algorithm implements the join of n individual 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 Thm. 2 it follows that $\mathsf{EM}(d_1, \ldots, d_n, a_1, \ldots, a_m, \lambda)$ is ε-differentially private for $\lambda = \frac{\varepsilon}{2}$, where $d_i = f(D_i)$.

4. Instantiation

In this section, we instantiate the two mechanisms described in the previous section using the recently proposed SMC arithmetic algorithms over integers, and fixed and floating point numbers [9, 21, 22, 28] that we discussed in Sec. 1.

[2] Note that depending on the instantiation of SMC, the steps might be slightly modified, or type conversions added, to provide the best efficiency.

Algorithm 3: Distributed Laplace mechanism

Data: Shared fixed point form (γ, f) inputs $[d_1], \ldots, [d_n]$; $\lambda = \frac{\Delta f}{\varepsilon}$

Result: $w = (\sum_{i=1}^{n} d_i) + \mathsf{Lap}(\lambda)$ in fixed point form

$[d] = [d_1]$
for $i = 2$ *to* n **do**
  $[d] = \mathsf{FPAdd}([d], [d_i])$
$[r_x] = \mathsf{RandInt}(\gamma+1)$; $[r_y] = \mathsf{RandInt}(\gamma+1)$
$\langle [v_x], [p_x], 0, 0\rangle = \mathsf{FP2FL}([r_x], \gamma, f = \gamma, \ell, k)$
$\langle [v_y], [p_y], 0, 0\rangle = \mathsf{FP2FL}([r_y], \gamma, f = \gamma, \ell, k)$
$\langle [v_{x/y}], [p_{x/y}], 0, 0\rangle = \mathsf{FLDiv}(\langle [v_x], [p_x], 0, 0\rangle, \langle [v_y], [p_y], 0, 0\rangle)$
$\langle [v_{ln}], [p_{ln}], [z_{ln}], [s_{ln}]\rangle = \mathsf{FLLog2}(\langle [v_{x/y}], [p_{x/y}], 0, 0\rangle)$
$\langle [v_z], [p_z], [z_z], [s_z]\rangle = \mathsf{FLMul}(\frac{\lambda}{\log_2 e}, \langle [v_{ln}], [p_{ln}], [z_{ln}], [s_{ln}]\rangle)$
$[z] = \mathsf{FL2FP}(\langle [v_{z_1}], [p_{z_1}], [z_{z_1}], [s_{z_1}]\rangle, \ell, k, \gamma)$
$[w] = \mathsf{FPAdd}([d], [z])$
return $w = \mathsf{Rec}([w])$

4.1. Secure Multiparty Computation for Distributed Mechanisms

Our protocols to compute the distributed Laplace mechanism and the distributed exponential mechanism are given in Alg. 3, and 4 respectively, and they are explained below.

Number representation. We follow the representation in [9] for integers and for real numbers in fixed point and floating point forms. 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 bit-length 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 $\bar{x}$ so 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 computation, participants $P_1, \ldots, P_n$ 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 computing 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 $i \in \{1, \ldots, n\}$.

General overview. For the most part, the instantiation simply unfolds the mathematical operations used in the algorithms presented in Sec. 3 and replaces them by the corresponding SMCs for arithmetic operations that we list in Sec. 1.

Addition for both integers and fixed point numbers is 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 numbers or integers to lower the cost of adding them together to yield the joint unperturbed query result $[d_1]+\ldots+[d_n]$. 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 FP2FL, FL2Int, Int2FL whenever necessary.

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 $\mathcal{U}_{(0,1]}$. As the SMC suite we consider does not include such a function, we encode it using the existing primitive RandInt for the generation of a random integer. For instance, this is done in steps 4 and 5 of Alg. 3. We first generate a shared $(\gamma+1)$-bit integer $[r_x]$ using the random number generator RandInt. We then consider this integer to be the fractional part of the fixed point number, whose integer part is 0 (by choosing $f=\gamma$). Afterwards, the fixed point number is converted to floating point form by using the function 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 FP2FL so that the shared $[0]$ is replaced by the sharing of $[1]$ in step 3 [9, Sec. 5]. We could avoid the modification of 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 SMC protocols for computing exponentiation with base 2 (FLExp2) and logarithm to base 2 (FLLog2). As 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, lines 8 - 9 in Alg. 3 and lines 7 - 8 in Alg. 4 use the above equations to compute logarithm and exponentiation to base e respectively.

Distributed Laplace mechanism. The protocol to compute the distributed Laplace mechanism is shown in Alg. 3. Note that 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 FLLog2 as well as the number of follow-up operations.

Distributed exponential mechanism. The protocol to compute the distributed exponential mechanism using SMC is presented in Alg. 4. Each shared input $[d_i]$ consists of an integer array of the size m, representing the histogram of participant P_i. The instantiation follows the steps of the algorithm presented in Alg. 2 by using the insights and techniques we presented in this section. We straightforwardly implement the binary search to find the winning interval/candidate on lines 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.

4.2. Mechanisms in the Malicious Setting

In order to achieve DP against malicious computation parties, we need to strengthen the SMC protocols so as to make them resistant to computing parties that deviate from

Algorithm 4: Distributed exponential mechanism

Data: $[d_1], \ldots, [d_n]$; the number m of candidates; $\lambda = \frac{\varepsilon}{2}$
Result: 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**
  $[z_j] = 0$
  for $i = 1$ *to* n **do**
    $[z_j] = \mathsf{IntAdd}([z_j], [d_i(j)])$
  $\langle [v_{z_j}], [p_{z_j}], [z_{z_j}], [s_{z_j}] \rangle = \mathsf{Int2FL}([z_j], \gamma, \ell)$
  $\langle [v_{z'_j}], [p_{z'_j}], [z_{z'_j}], [s_{z'_j}] \rangle = \mathsf{FLMul}(\lambda \cdot \log_2 e, \langle [v_{z_j}], [p_{z_j}], [z_{z_j}], [s_{z_j}] \rangle)$
  $\langle [v_{\delta_j}], [p_{\delta_j}], [z_{\delta_j}], [s_{\delta_j}] \rangle = \mathsf{FLExp2}(\langle [v_{z'_j}], [p_{z'_j}], [z_{z'_j}], [s_{z'_j}] \rangle)$
  $\langle [v_{I_j}], [p_{I_j}], [z_{I_j}], [s_{I_j}] \rangle =$
  $\mathsf{FLAdd}(\langle [v_{I_{j-1}}], [p_{I_{j-1}}], [z_{I_{j-1}}], [s_{I_{j-1}}] \rangle, \langle [v_{\delta_j}], [p_{\delta_j}], [z_{\delta_j}], [s_{\delta_j}] \rangle)$
$[r] = \mathsf{RandInt}(\gamma + 1)$
$\langle [v_r], [p_r], 0, 0 \rangle = \mathsf{FP2FL}([r], \gamma, f = \gamma, \ell, k)$
$\langle [v'_r], [p'_r], [z'_r], [s'_r] \rangle = \mathsf{FLMul}(\langle [v_r], [p_r], 0, 0 \rangle, \langle [v_{I_m}], [p_{I_m}], [z_{I_m}], [s_{I_m}] \rangle)$
$j_{\mathsf{min}} = 1$; $j_{\mathsf{max}} = m$
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 $w_{j_{\mathsf{min}}}$

the protocol [40, 43]. Intuitively, to maintain secrecy, one has to enforce two additional properties. First, the protocol-instance observations of honest parties must be consistent with each other, and second, every party must prove that each step of its computation was performed correctly.

Given the real-world impracticality of information-theoretically secure channels and subsequently information-theoretically secure SMC protocols, in the malicious setting we shift to the computational setting. In particular, we employ a computational verifiable secret sharing scheme (VSS) [44] instead of the basic secret sharing scheme to achieve the first secrecy property. For the second secrecy property, we introduce zero-knowledge (ZK) proofs so that a party could prove that a correct secret value is shared among the parties [43], and that shared secret values satisfy some mathematical conditions (e.g. they are in a pre-defined range) [40].

4.3. Limitations of Finite-Precision Instantiations

While the theoretical definition of sanitization mechanisms for DP operates on real numbers $r \in \mathbb{R}$ (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. [46]. 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* that truncates large values and rounds the final result so as to achieve DP of the implementation. Gazeau et al. [46] show that, in general, approximation errors of any kind of finite-precision representation of real numbers can lead to the disclosure of secrets. They provide a solution for fixing 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.

We point out that the mitigation techniques proposed in these works rely on arithmetic operations that can be implemented using our arithmetic SMC protocols, thus allowing us to circumvent the known attacks based on finite-precision implementations. 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 honest but curious setting, and discuss how to extend this result to a malicious setting.

We first recall the standard notion of t-secrecy for SMC, which is formulated as in [9] except for a small modification to accommodate the computing parties. The following definitions refer to computing 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) *The view of C_i 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 honest but curious adversaries if for each coalition $I = \{C_{i_1}, C_{i_2}, \ldots, C_{i_t}\} \subset \mathcal{C}$ of honest but curious computing parties of the size $t < \beta/2$, there exists a probabilistic polynomial time simulator S_I so 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, similar to the one introduced in [47]. 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 computing parties $\mathcal{C} = \{C_1, \ldots, C_\beta\}$ achieves ε-DDP with respect to a coalition $I \subset \mathcal{C}$ of honest but curious computing parties of the size t, if the following condition holds: for any neighboring input vectors $D, D' \in \mathcal{D}^n$ and any possible set $\mathcal{S} \subseteq \mathcal{V}_\Pi$, $Pr[\mathsf{VIEW}_{\Pi(D)}(I) \in \mathcal{S}] \leq e^\varepsilon Pr[\mathsf{VIEW}_{\Pi(D')}(I) \in \mathcal{S}]$ holds.*

For the malicious setting, the coalition I of honest but curious parties in Def. 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) [7, 12] relation with a negligible function $\mathrm{negl}(\kappa)$ so that $Pr[\mathsf{VIEW}_{\Pi^{\mathsf{M}}(D)}(I^{\mathsf{M}}) \in \mathcal{S}] \leq e^{\varepsilon} Pr[\mathsf{VIEW}_{\Pi^{\mathsf{M}}(D')}(I^{\mathsf{M}}) \in \mathcal{S}] + \mathrm{negl}(\kappa)$.

We now state our main theorems on ε-DDP in the honest but curious model, and ε-IND-DDP in the malicious model.

Theorem 3 (ε-DDP) *Let $\varepsilon > 0$. In the honest but curious setting, our distributed LM and EM protocols achieve ε-DDP with respect to any honest but curious coalition $I \subset \mathcal{C}$ of the size $t < \beta$.*

Proof sketch. We start our analysis by proving t-secrecy for our protocols and then use this property to prove ε-DDP. The SMC arithmetic protocols over integers, fixed and floating point numbers internally use only two basic SMC primitives over the finite field $\mathbb{F}_q$, namely, the addition and multiplication primitives for shared secret values from $\mathbb{F}_q$. Assuming secure instances of distributed addition and multiplication protocols over $\mathbb{F}_q$ [43] (and secure protocols built on top of them), Aliasgari et al. [9] have proved the correctness and t-secrecy properties of the SMC arithmetic protocols employed in our mechanisms using Canetti's composition theorem [48]. More formally, they suggested that one can build a simulator for their arithmetic SMC protocols by invoking simulators for the corresponding building blocks so that the resulting environment would be 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 SMC protocols and the other building blocks, so that the corresponding environment is indistinguishable from the corresponding real distributed DP protocol execution.

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

Theorem 4 (ε-IND-DDP) *Let $\varepsilon > 0$ and κ be a sufficiently large security parameter. In the malicious setting, our distributed LM and EM protocols achieve ε-IND-DDP with respect to any malicious coalition $I^{\mathsf{M}} \subset \mathcal{C}$ of the size $t < \beta$, under the strong RSA and decisional Diffie-Hellman assumptions for parameter κ.*

Proof sketch. As the computational verifiable secret sharing (VSS) scheme we use [44] enjoys the perfect secrecy property, the t-secrecy analysis for the SMC protocols in the malicious setting remains almost the same as in the honest but curious setting. Nevertheless, an active adversary can target the secure communication channels between 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 the t-secrecy of SMC protocols with negligible probability (in κ).

The correctness of the computational SMC 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 [44], zero-knowledge range proofs require the strong RSA assumption [40], and finally, the ZK proofs for secure multiplication require the discrete logarithm assumption [43].

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

6. Performance Analysis

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

6.1. Implementation

We implement all SMC protocols discussed in Sec. 1 as well as our DDP mechanisms as multi-threaded object-oriented C++ code to support any number (≥ 3) of computing parties in the honest but curious and malicious settings. Our implementation uses the GMP library [49] for all finite field computations, the Relic toolkit [50] for elliptic curve cryptographic constructions, and the Boost [51] and OpenSSL libraries for secure communication. Our numeric SMC libraries can be of independent interest to other distributed computation scenarios, and our complete code base is available online [52].

6.2. Experimental Setup

The experiments are performed using a 3.20 GHz (Intel i5) Linux machine with 16 GB RAM, and 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 the size of 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 bits, significantly smaller fields suffice. In contrast to [9], we do not employ batching (which improves average computation times) as our distributed mechanisms call the individual arithmetic SMC functions only a few times. To determine an average performance, we run the experiments ten times for both parameter sets. In Table 3, we show our results for all required SMC functionalities in the

Table 3. Performance of a single 3-party and 5-party SMC operations measured in seconds

Type	Protocol	HbC		Malicious	
		$\beta = 3$, $t = 1$	$\beta = 5$, $t = 2$	$\beta = 3$, $t = 1$	$\beta = 5$, $t = 2$
Float	FLAdd	0.75	1.00	7.91	16.3
	FLMul	0.24	0.28	1.79	3.30
	FLScMul	0.24	0.29	1.80	3.29
	FLDiv	0.90	1.00	3.22	5.90
	FLLT	0.19	0.22	1.46	2.76
	FLRound	0.75	0.89	5.80	11.67
Convert	FP2FL	1.42	1.21	12.4	25.9
	Int2FL	1.07	1.44	12.4	26.0
	FL2Int	1.65	2.01	13.4	26.9
	FL2FP	1.67	2.08	13.6	27.5
Log	FLLog2	16.56	21.44	147	296
Exp	FLExp2	8.58	10.22	63.6	120

honest but curious and malicious settings. In particular, we include the computation time for single 3-party and 5-party arithmetic SMC operations measured in seconds. Note that as we employ Beaver's triple technique for multiplications [26], we have an offline (background) phase of generating those triples [23, 24], and a fast online phase for every multiplication. We only consider the online phase computation times here.

6.3. Cost Analysis (Honest but Curious Setting)

As expected, the SMC protocols for logarithm and exponentiation are the most expensive operations, and they will drive our distributed mechanism cost analysis. Our protocols also use Rec, IntAdd, FPAdd, RandInt, but we did not include them in Table 3 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 and EM protocols for ($\beta = 3$, $t = 1$) computing parties and 100,000 users. The distributed LM protocol has a computation cost of 22.5 sec. The good 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 $\ln r_x - \ln r_y = \ln \frac{r_x}{r_y}$. 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 44.6 sec.

For larger numbers of computing parties β, one can extrapolate the performance from our analysis. Even for $\beta \approx 100$, we expect the distributed LM protocol to take about a few hundred seconds in the honest but curious 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.

6.4. Cost Analysis (Malicious Setting)

As expected, the computations times for the SMC operations secure against an active adversary are around an order of magnitude higher than those of the operations secure

against an honest but curious adversary. The average performance costs for our distributed LM and EM protocols for ($\beta = 3$, $t = 1$) computing parties and 100,000 users in the malicious setting are the following. The distributed LM protocol has an average computation cost of 187 sec. The cost of the distributed EM protocol, for $m = 5$ result candidates, is 316 sec. Here, we observe that shifting to multiplication using Beaver's triple-based method [26] reduced the computation cost by a factor of two as compared to our earlier result [25].

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 one order of magnitude performance gain in the honest but curious setting as well as the malicious setting by employing high-performance computing servers. Furthermore, as users have to simply forward their shared values to the computing parties, which is an inexpensive operation (< 1 msec in the honest but curious setting and a few 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 show the flexibility of our architecture by briefly discussing how PrivaDA can be used to improve the state of the art in two different application scenarios.

7.1. Web Analytics

Web analytics consist of 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, who we will refer to as *analysts*.

State of the art. In order to obtain aggregated user information, websites commonly use third-party web analytics services called *aggregators* that track users' browsing behavior across the web, thereby violating their privacy. Newer systems, e.g. a series of non-tracking web analytics systems [11, 19] 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 computing parties are operated by third parties that 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 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 would be able to detect duplicates and discard them before the aggregation. The users only have to submit their shares and can then go offline. Finally, the support for a variety of perturbation

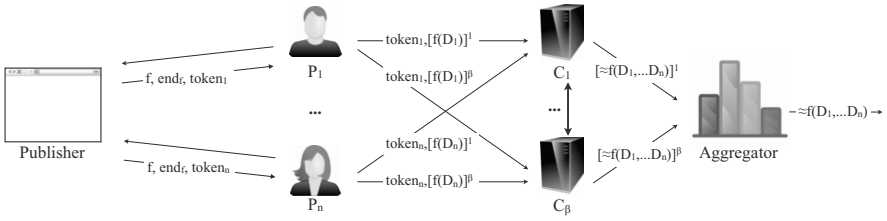

Figure 2. Protocol flow of privacy-preserving web analytics

mechanisms enables the execution of different kinds of analytical queries, for instance, our distributed exponential mechanism can be used to compute the average nationality and age group of visitors. The protocol is depicted in Fig. 2.

7.2. Anonymous Surveys

Next, let us consider 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 [53] 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 organizations that are mutually distrustful, but are all interested in the results of the evaluation, such as the student association, or the university administration. The average grade is computed using the distributed Laplace mechanism.

References

[1] Andrés Molina-Markham, Prashant Shenoy, Kevin Fu, Emmanuel Cecchet, and David Irwin. Private Memoirs of a Smart Meter. In *BuildSys'10*, pages 61–66, 2010.

[2] Rui Wang, Yong Fuga Li, XiaoFeng Wang, Haixu Tang, and Xiaoyong Zhou. Learning Your Identity and Disease from Research Papers: Information Leaks in Genome Wide Association Study. In *CCS'09*, pages 534–544, 2009.

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

[4] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating Noise to Sensitivity in Private Data Analysis. In *TCC'06*, pages 265–284, 2006.

[5] Frank McSherry and Kunal Talwar. Mechanism Design via Differential Privacy. In *FOCS'07*, pages 94–103, 2007.

[6] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our Data, Ourselves: Privacy Via Distributed Noise Generation. In *EUROCRYPT'06*, pages 486–503, 2006.
[7] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil P. Vadhan. Computational Differential Privacy. In *Crypto'09*, pages 126–142, 2009.
[8] Moritz Hardt and Guy N. Rothblum. A Multiplicative Weights Mechanism for Privacy-Preserving Data Analysis. In *FOCS'10*, pages 61–70, 2010.
[9] Mehrdad Aliasgari, Marina Blanton, Yihua Zhang, and Aaron Steele. Secure Computation on Floating Point Numbers. In *NDSS'13*, 2013.
[10] Gergely Ács and Claude Castelluccia. I have a DREAM! (DiffeRentially privatE smArt Metering). In *IH'11*, pages 118–132, 2011.
[11] Ruichuan Chen, Alexey Reznichenko, Paul Francis, and Johannes Gehrke. Towards Statistical Queries over Distributed Private User Data. In *NSDI'12*, pages 13–13, 2012.
[12] T.-H. Hubert Chan, Elaine Shi, and Dawn Song. Privacy-Preserving Stream Aggregation with Fault Tolerance. In *FC'12*, pages 200–214, 2012.
[13] Slawomir Goryczka, Li Xiong, and Vaidy Sunderam. Secure Multiparty Aggregation with Differential Privacy: A Comparative Study. In *EDBT/ICDT'13*, pages 155–163, 2013.
[14] Marek Jawurek and Florian Kerschbaum. Fault-Tolerant Privacy- Preserving Statistics. In *PETS'12*, pages 221–238, 2012.
[15] Vibhor Rastogi and Suman Nath. Differentially Private Aggregation of Distributed Time-Series with Transformation and Encryption. In *SIGMOD'10*, pages 735–746, 2010.
[16] Elaine Shi, T.-H. Hubert Chan, Eleanor G. Rieffel, Richard Chow, and Dawn Song. Privacy-Preserving Aggregation of Time-Series Data. In *NDSS'11*, 2011.
[17] George Danezis, Markulf Kohlweiss, and Alfredo Rial. Differentially Private Billing with Rebates. In *IH'11*, pages 148–162, 2011.
[18] Gilles Barthe, George Danezis, Benjaming Grégoire, César Kunz, and Santiago Zanella-Béguelin. Verified Computational Differential Privacy with Applications to Smart Metering. In *CSF'13*, pages 287–301, 2013.
[19] Istemi Ekin Akkus, Ruichuan Chen, Michaela Hardt, Paul Francis, and Johannes Gehrke. Non-tracking Web Analytics. In *CCS'12*, pages 687–698, 2012.
[20] Ruichuan Chen, Istemi Ekin Akkus, and Paul Francis. SplitX: High-Performance Private Analytics. In *SIGCOMM'13*, 2013. to appear.
[21] Octavian Catrina and Amitabh Saxena. Secure Computation With Fixed-Point Numbers. In *FC'10*, pages 35–50, 2010.
[22] Strange L. From and Thomas Jakobsen. Secure Multi-Party Computation on Integers. Master's thesis, University of Aarhus, 2006.
[23] Rikke Bendlin, Ivan Damgård, Claudio Orlandi, and Sarah Zakarias. Semi-homomorphic encryption and multiparty computation. In *EUROCRYPT'11*, pages 169–188, 2011.
[24] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *Crypto'12*, pages 643–662, 2012.
[25] Fabienne Eigner, Aniket Kate, Matteo Maffei, Francesca Pampaloni, and Ivan Pryvalov. Differentially Private Data Aggregation with Optimal Utility. In *ACSAC'14*, 2014.
[26] Donald Beaver. Efficient multiparty protocols using circuit randomization. In *Crypto'91*, pages 420–432, 1991.
[27] Andrew Chi-Chih Yao. Protocols for Secure Computations (Extended Abstract). In *FOCS'82*, pages 160–164, 1982.
[28] Ronald Cramer, Ivan Damgård, and Yuval Ishai. Share Conversion, Pseudorandom Secret-Sharing and Applications to Secure Computation. In *TCC'05*, pages 342–362, 2005.
[29] Assaf Ben-David, Noam Nisan, and Benny Pinkas. FairplayMP: A System for Secure Multi-party Computation. In *CCS'08*, pages 257–266, 2008.
[30] Dan Bogdanov, Margus Niitsoo, Tomas Toft, and Jan Willemson. High-performance secure multi-party computation for data mining applications. *Int. J. Inf. Sec.*, 11(6):403–418, 2012.
[31] Qinghua Li and Guohong Cao. Efficient Privacy-Preserving Stream Aggregation in Mobile Sensing with Low Aggregation Error. In *PETS'13*, pages 60–81, 2013.
[32] Dima Alhadidi, Noman Mohammed, Benjamin C. M. Fung, and Mourad Debbabi. Secure Distributed Framework for Achieving ε-Differential Privacy. In *PETS'12*, pages 120–139, 2012.
[33] Daniel Kifer and Ashwin Machanavajjhala. No Free Lunch in Data Privacy. In *SIGMOD'11*, pages

193–204, 2011.

[34] Andreas Haeberlen, Benjamin C. Pierce, and Arjun Narayan. Differential Privacy under Fire. In *USENIX'11*, 2011.

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

[36] Raghav Bhaskar, Abhishek Bhowmick, Vipul Goyal, Srivatsan Laxman, and Abhradeep Thakurta. Noiseless Database Privacy. In *ASIACRYPT'11*, pages 215–232, 2011.

[37] Johannes Gehrke, Edward Lui, and Rafael Pass. Towards Privacy for Social Networks: A Zero-Knowledge Based Definition of Privacy. In *TCC'11*, pages 432–449, 2011.

[38] Liina Kamm and Jan Willemson. Secure Floating Point Arithmetic and Private Satellite Collision Analysis. *IJIS*, pages 1–18, 2014.

[39] Toomas Krips and Jan Willemson. Hybrid Model of Fixed and Floating Point Numbers in Secure Multiparty Computations. In *ISC'14*, pages 179–197, 2014.

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

[41] Milton Abramowitz and Irene A. Stegun. *Handbook of Mathematical Functions with Formulas, Graphs, and Mathematical Tables*. Dover, 1964.

[42] Luc Devroye. Non-Uniform Random Variate Generation, 1986.

[43] Rosario Gennaro, Michael O. Rabin, and Tal Rabin. Simplified VSS and Fact-Track Multiparty Computations with Applications to Threshold Cryptography. In *PODC'98*, pages 101–111, 1998.

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

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

[46] Ivan Gazeau, Dale Miller, and Catuscia Palamidessi. Preserving differential privacy under finite-precision semantics. In *QAPL'13*, pages 1–18, 2013.

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

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

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

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

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

[52] PrivaDA Project Page: Full Version + Our SMPC Library. `http://crypsys.mmci.uni-saarland.de/projects/ArithmeticSMPC`.

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