DPrio: Efficient Differential Privacy with High Utility for Prio

Dana Keeler
Mozilla Corporation
San Francisco, California, USA
dkeeler@mozilla.com

Chelsea Komlo
University of Waterloo
Waterloo, Ontario, Canada
ckomlo@uwaterloo.ca

Emily Lepert
University of Waterloo
Waterloo, Ontario, Canada
elepert@uwaterloo.ca

Shannon Veitch
ETH Zürich
Zürich, Switzerland
shannon.veitch@inf.ethz.ch

Xi He
University of Waterloo
Waterloo, Ontario, Canada
xi.he@uwaterloo.ca

ABSTRACT

Private data collection systems such as Prio ensure data privacy by distributing trust among a set of mutually trusted parties, to allow for aggregate data collection without disclosing any single client's data in the clear. While systems like Prio are undergoing widespread interest and adoption, these systems lack efficient mechanisms to provide differential privacy guarantees. In this work, we present a lightweight method that we call DPrio to augment Prio and related systems with differential privacy assurances while ensuring higher data utility than existing noise generation protocols. We compare our results against four related constructions in the literature, and identify how DPrio achieves improved data utility relative to the assumed number of dishonest clients and servers, with only minimal (and batchable) server communication overhead. We present several case studies and discuss considerations for real-world implementations.

KEYWORDS

differential privacy, multi-party computation

1 INTRODUCTION

Prio [17] is a system which enables the collection of aggregate statistics by a set of servers such that no individual server learns anything about the clients' data, except what they can infer from the aggregate statistic. Prio ensures user privacy assuming all but one of the Prio servers can be corrupted, and in the case of servers corrupting data, simply requires the protocol to abort (i.e, the system is not robust against misbehaving servers). Unlike prior related schemes [23], Prio does not assume clients are acting honestly, and so leverages secret-shared non-interactive proofs (SNIPS) to provide robustness against malicious clients. In part due to its efficiency and pragmatic threat model, Prio has already been deployed in practice at scale, beginning with Mozilla [24] and most recently by the same organization that supports Let's Encrypt [2, 32].

Unfortunately, it is well known that reconstruction attacks can be used to recover individuals' data from aggregate statistics [28]. One method of protecting individual clients' data is with differential privacy (DP) [20], which allows for formal guarantees about the privacy of any user's data within a dataset. By adding noise to aggregate statistics, we can obtain a guaranteed bound on the amount of information disclosed about individual user records in the aggregated query output.

By default, Prio does not ensure differential privacy. The authors recommend a prior design by Dwork et al. [19] which describes a multi-party computation protocol among a set of servers to generate noise. However, in a distributed setting where network calls can fail and servers maintain heavy load, such n-of-n MPC operations are undesirable. Further, this mechanism requires server computation and communication among servers that scales linearly relative to the number of clients, as the operations performed by each client in the "full participation" model must be partitioned among the smaller set of servers. On the other hand, clients that add noise locally by following a local DP mechanism [27] have undesirable data utility tradeoffs. The local DP method also requires that clients honestly add noise to their data and often results in data that is inflexible to a variety of analysis. As such, an ideal DP construction for a practical Prio deployment is one that achieves high utility as in the central DP model (where noise is calculated and added by the trusted data curator directly), with efficiency akin to local DP, while still remaining within the original threat model of Prio.

In this work, we present a lightweight method that we call DPrio to add differential privacy to Prio and related systems while ensuring higher data utility and better efficiency than these prior noise generation designs. In DPrio, all clients submit secret-shared noise and servers perform a minimal MPC protocol to select which client's noise is added to the aggregate total. Happily, data utility in DPrio is nearly identical to that achieved by central DP, while client and server computation and communication remains constant regardless of the number of participating clients or servers. DPrio is a non-interactive protocol among clients, requires minimal interaction among servers, and is differentially private against a small proportion of adversarial clients when at least one server is honest but curious. Notably, DPrio maintains the existing threat model of Prio, in that only one server is assumed to be honest.

We also compare our results against two noise generation protocols that we call Client-DP and Server-DP, modeled after existing notions in the literature [23]. In Client-DP, clients submit a small amount of noise, which sum to a Gaussian value. In Server-DP, servers directly add Gaussian noise to their aggregated sums. DPrio

Proceedings on Privacy Enhancing Technologies 2023(3), 375–390
© 2023 Copyright held by the owner/author(s).
https://doi.org/10.56553/popets-2023-0086

achieves improved data utility over Client-DP and Server-DP with respect to the assumed number of dishonest parties.

In summary, we present the following contributions:

- A careful analysis of DP schemes in the literature and their efficiency when used with Prio.
- A lightweight DP mechanism that builds on top of Prio that we call DPrio. DPrio achieves nearly identical data utility as central DP while its efficiency remains constant relative to the number of clients and servers. DPrio does not require client interaction, but does require minimal server interaction, which can be aggregated within one batched operation.
- We evaluate DPrio relative to existing constructions in the literature as well as Client-DP and Server-DP. In summary, DPrio achieves higher data utility than prior constructions.

2 PRELIMINARIES

This section defines the basic notions of privacy and security. We denote the number of Prio servers as n and the number of clients as N. We use $\mathcal{D}$ to denote the domain for the collection of the records from the clients.

2.1 Differential Privacy

Differential privacy (DP), introduced by Dwork et al. [20], is a privacy notion that enables the calculation of aggregate statistics on users' data in a privacy-preserving manner.

Definition 2.1 (Differential Privacy). *A randomized algorithm $M : \mathcal{D} \mapsto \mathcal{Y}$ is (ϵ, δ)-differential privacy (DP), if for any pair of neighbouring datasets $D, D' \in \mathcal{X}^l$ that differ by* a single record, *and for any $T \subseteq \mathcal{Y}$ we have*

$$\Pr[M(D) \in T] \leq e^{\epsilon} \Pr[M(D') \in T] + \delta. \qquad (1)$$

If $\delta \neq 0$, then we say the mechanism provides approximate differential privacy. Otherwise, when $\delta = 0$, it satisfies pure differential privacy. Intuitively, this definition ensures that a single record only changes the output distribution by at most a factor of e^{ϵ}.

A common method of achieving differential privacy for numerical statistics is the Laplace mechanism [19]. One calculates the *sensitivity*, Δ, of the function f they want to compute, which determines the largest change to the function output (measured in l_1 norm) by changing a single record, and then adds noise sampled from a Laplace distribution with parameter Δ/ϵ.

Theorem 2.1 (Laplace Mechanism [21]). *Given a function $f : \mathcal{D} \mapsto \mathcal{R}^k$, and a data set $D \in \mathcal{D}$, the Laplace Mechanism is defined as $\mathcal{M}_L(D, f(\cdot), \epsilon) = f(D) + (Y_1, ..., Y_k)$ where Y_i are i.i.d. random variables drawn from Lap(Δ/ϵ). This mechanism achieves ϵ-DP.*

Another additive method of achieving differential privacy is to add Gaussian noise sampled from the normal distribution $\mathcal{N}(0, \sigma^2)$. We call this Gaussian mechanism, and the sensitivity of the function Δ in this mechanism uses l_2-norm to measure the maximum change in the query output when changing a record.

Theorem 2.2 (Gaussian Mechanism [21]). *Let $\epsilon \in (0, 1)$ be arbitrary. For $c^2 > 2\ln(1.25/\delta)$, the Gaussian Mechanism with parameter $\sigma \geq c\Delta/\epsilon$ is (ϵ, δ)-differentially private.*

It is possible to formulate the Gaussian Mechanism such that there is no restriction on ϵ [14]. Note that the sum of independent normally distributed random variables is also normal. In particular, if $Y_1 \sim \mathcal{N}(0, \sigma_1^2)$ and $Y_2 \sim \mathcal{N}(0, \sigma_2^2)$ then $Y_1 + Y_2 \sim \mathcal{N}(0, \sigma_1^2 + \sigma_2^2)$. That is, adding many Gaussians yields a Gaussian.

One convenient property of DP is its immunity to post-processing. Intuitively, computing a function on the output of a DP algorithm does not reduce the privacy guarantee on the sensitive input data.

Theorem 2.3 (Post-Processing [21]). *Let $M : \mathcal{D} \mapsto \mathcal{Y}$ be an (ϵ, δ)-differentially private algorithm. Let $f : \mathcal{Y} \mapsto \mathcal{Z}$ be an arbitrary randomized mapping. Then $f \circ M : \mathcal{D} \mapsto \mathcal{Z}$ satsifies (ϵ, δ)-DP.*

For the accuracy of DP mechanisms, we use the mean squared error to measure the amount of noise added by each mechanism.

Definition 2.2. *For some randomized mechanism $\mathcal{M}$ computing a (noisy) query q over a set of data $D \in \mathcal{D}$, we define the error to be*

$$\textit{Error}_{\mathcal{M}} = \mathbb{E}[(q(D) - M(D))^2],$$

where $q(D)$ is the true value to the query and $M(D)$ is the noised value.

This work deals with the setting where parties are computationally bounded which requires the notion of computationally differential privacy or SIM-CDP [41].

Definition 2.3 (SIM-CDP [41]). *An ensemble $\{f_\kappa\}$, $\kappa \in \mathbb{N}$ of randomized functions $f_\kappa : \mathcal{D} \rightarrow \mathcal{R}_\kappa$ provides ϵ_κ-SIM-CDP if there exists an ensemble $\{f_\kappa\}, \kappa \in \mathbb{N}$ of ϵ_κ-differentially-private mechanisms $f_\kappa : \mathcal{D} \rightarrow \mathcal{R}_\kappa$ and a negligible function negl$(\cdot)$, such that for every non-uniform PPT TM A, every polynomial $p(\cdot)$, every sufficiently large $\kappa \in \mathbb{N}$, every data set $D \in \mathcal{D}$ of size at most $p(\kappa)$, and every advice string z_κ of size at most $p(\kappa)$, it holds that,*

$$|\Pr[A_\kappa(f_\kappa(D)) = 1] - \Pr[A_\kappa(F_\kappa(D)) = 1]| \leq negl(\kappa).$$

That is, $f_\kappa(D)$ and $F_\kappa(D)$ are computationally indistinguishable.

2.2 Secure Multi-party Computation

Secure multi-party computation (MPC) [7, 38] allows a set of participants, each holding private data, to jointly compute a function over their data without revealing to one another any information except for the output. A common method of multi-party computation relies on *secret sharing schemes* [10, 18, 48], in which an individual's data is split into shares and divided amongst the participants performing the secure computation. After executing the protocol, participants can reconstruct the output of the function by applying the corresponding reconstruction protocol of the secret sharing scheme. Prio uses such protocols to compute the desired statistics while ensuring privacy of each client's data.

Prio uses affine-aggregatable encodings (AFEs) to efficiently encode data such that it is possible to compute the value of a function given the sum of the encodings. There exist AFEs for sums [23, 36], standard deviations [45], counts [13, 40], and least-squares regressions [35] and Prio adapts these AFEs to compute private statistics.

The AFEs used by Prio also apply *secret-shared non-interactive proofs* (SNIPs) [17] to verify the input of the clients. The interactive version requires an interaction between a client (prover) and multiple servers (verifiers) in which the client attempts to prove to the servers that *Verify*$(x) \rightarrow 1$ without revealing any additional information about x. The requirements of this system are analogous to the properties of a zero-knowledge proof system [30] that is non-interactive; however, Prio is designed for a setting where the role of the verifier is partitioned among several mutually untrusted parties. In particular, SNIPs ensure that the information encoded in the SNIP is both valid (i.e, the information is encoded correctly) and zero-knowledge (i.e, the servers learn nothing beyond whether the SNIP is valid or not), assuming that all servers are honest.

In Prio, the server-to-server computation cost is independent of the complexity of the circuit or the size of the value and is basically the cost of computing *Verify* locally. The client-to-server communication grows linearly with the size of the *Verify* circuit.

3 RELATED WORK

We present related work in central and local differential privacy. We then discuss Dwork-MPC, a multi-party protocol to generate noise, and past work on assumptions of non-collusion.

Central Differential Privacy. In the central DP model, the data curator is assumed to be trusted, and can view all client records. The data curator then directly chooses the appropriate amount of noise and adds it directly to the data. The US Census Bureau used central DP to protect sensitive information in the 2020 census [1]. Programming frameworks like PINQ [39] and Ektelo [51] are popular for leveraging central DP. This model *is* able to guarantee high data utility, a goal we similarly maintain for our constructions. However, it is undesirable when no such a trusted data curator exists.

Local Differential Privacy. In local differential privacy, each client sends their data along with DP noise to the central server. This ensures that the clients' data is differentially private to the central server and the rest of the clients. Apple [49] and Google [27] use this model to gather analytics from millions of users. The benefit of local DP is that each client does not have to trust any other party. The downside is that it introduces noise on the order of the number of clients. This is only valuable in instances with a lot of data and patterns that are still evident despite the noise.

Assumptions of Non-Collusion. To bridge the gap between the low data utility induced by local DP and the high trust assumptions by centralized DP, assumptions of non-collusion have been introduced as a "best of both worlds" option. For example, introducing an intermediate shuffler between clients and the aggregator can provide anonymity and increase the utility [8, 26]. However, methods that rely on another party to perform shuffling still requires that clients locally generate noise and thus cannot achieve utility equivalent to the central model [5]. A different approach involves secure computation [22] or encryption of noise to a separate trusted party [47]. However, these models also introduce undesirably high trust assumptions in single entities, whereas in the Prio model requires distributing trust equally among the n servers.

MPC Protocols for DP. While Prio does not define DP directly, the authors reference an MPC protocol defined by Dwork et al. [19] that we refer to as Dwork-MPC. In Dwork-MPC, noise can be generated by an MPC protocol by servers, achieving (ϵ, δ)-DP. In its base form, Dwork-MPC assumes that at least 1/3 of servers are honest. Eriguchi et al. [25] make improvements to the communication complexity or success probability of the algorithms in Dwork-MPC; however, they assume that all servers are honest but curious.

There exists other work which focuses on implementing specific DP algorithms in an MPC setting. For example, there exist mechanisms for computing a differentially private median [11, 12], sampling biased coins [16], and graph queries [46]. Our work differs from these approaches that design protocols for a particular setting, since we focus on a generalizable, robust, and scalable framework that computes statistics accurately with DP guarantees.

4 SYSTEM GOALS

The goal of this work is to provide a lightweight mechanism on top of an existing Prio architecture to allow for an efficient efficient mechanism to ensure differential privacy (DP), but with high utility. In other words, we aim for the best of both worlds — a data collection mechanism with DP guarantees, without resorting to heavyweight MPC protocols or outputting noisy data with low utility.

4.1 Overview of Prio

We now describe Prio. This system will serve as the base on which we build a differentially private solution. The system is executed in the following steps, illustrated by Fig 1. While Prio does not impose hard constraints on the number of clients and servers, it assumes a small number of servers relative to clients. We provide a range of case studies and their impact on performance in Section 8.

(1) **Upload.** Each client encodes its data in a prescribed manner and splits its private encoded value into one share per server employing an affine-aggregatable function (AFE). The client then constructs a SNIP to prove to the server that the encoding satisfies certain correctness properties. The client forwards the shares of the proof and the encoded data to the corresponding servers.
(2) **Verify and Aggregate.** Upon receiving data from clients, the servers verify the SNIP to ensure that the encoding is well-formed. If the data is well-formed, the servers locally aggregate their shares.
(3) **Publish.** Once enough verified data has been received (e.g., if the protocol requires one million users' data) and locally aggregated, the servers reveal their local aggregations to an analyst who can accumulate all of the data to obtain the final statistic.

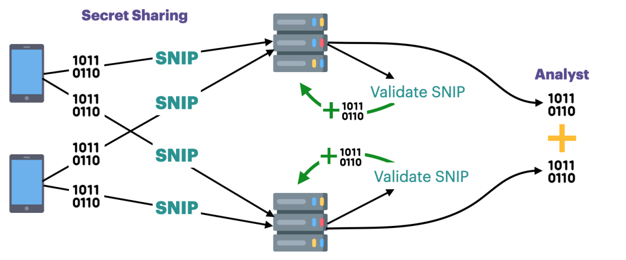

Figure 1: Overview of Prio. Clients send shares to servers who validate the associated SNIP and aggregate the data.

The final statistic does not satisfy DP; however, the system is equipped with several desirable properties. In particular, as long as one server is honest, the Prio servers learn nothing about the clients' data except what they can learn from the output. The system is robust if all servers are honest and is correct in the presence of faulty or malicious clients due to the SNIP proofs. For reference, we provide additional information on AFEs and SNIPs in Appendix A.

We will elaborate on these assumptions and these properties for Prio next, and add new assumptions for achieving DP.

4.2 Threat Model

We follow a similar threat model as employed by Prio, but additionally require an assumption about the bound of honest clients within the system as a whole. We now describe each party that participates in the protocol and their capabilities now.

Data Analyst. The data analyst wishes to learn some *aggregated* statistic about a user base. We consider the data analyst to be honest but curious — in other words, they are expected to follow the protocol of simply receiving aggregate statistics.

Clients. In Prio, clients submit zero-knowledge proofs that their submission is acceptable for the query being performed, to ensure robustness against misbehaving clients. In all of our schemes, we make an assumption on the proportion of honest clients, and adjust our construction accordingly.

Assumption 4.1. *The number of clients N is large, and the honest but curious/malicious adversary can control the noise of only a small number of the clients (m).*

While this assumption may be unacceptable for small-scale deployments, it is certainly pragmatic in deployments of Prio for applications such as modern web browsers.

Servers. Prio provides *privacy* for client's data in the setting where at least one server is honest but curious about the sensitive data. However, Prio is *not* robust against misbehaving servers. In other words, servers are trusted to follow the protocol; if they do not, they can corrupt the correctness of the data that the data analyst receives without detection. Our constructions maintain the same robustness and privacy assumptions. In addition, we assume that the servers do not collude with the client whose randomness is chosen. We also show in Appendix C the optimal protocol when this assumption is removed.

Assumption 4.2. *No collusion between clients and severs.*

While Assumption 4.2 may be perceived as overly strong, this perspective does not necessarily extend to real-world deployments. In the setting that DPrio targets — a large number of honest clients and small number of servers — this assumption becomes quite practical. Major browsers such as Firefox and Chrome which use or plan to use Prio for data collection have millions if not billions of users. While some malicious clients might exist and collude with a server, the likelihood that their noise is chosen is small.

4.3 Limitations on Query Type

Prio uses affine-aggregatable encodings which enable the servers to compute complex statistics by simply computing the sum of the encodings. To maintain the simplicity of the schemes, we only consider noise mechanisms which are additive, the most common of which are the Laplacian and Gaussian mechanisms (Section 2). As a result, the types of queries that our mechanisms support are limited to those which are interpretable after adding noise. Our constructions mainly apply to queries like sums, means, and counts. We do not consider more complex statistics such as those working on categorical data. We note that the existing Prio system also does not consider some complex statistics, including the median. Determining how to integrate DP for more complex statistics is an interesting area of future work.

5 ADAPTING PRIOR WORK TO PRIO

We now explain how prior work can be adapted to the Prio system and the limitations of these solutions.

5.1 Dwork-MPC

In the setting of Prio where clients are assumed to interact only with servers, differential privacy (DP) can be achieved using a variant of Dwork-MPC [19]. In this variant of Dwork-MPC, n servers generate noise from a Gaussian with mean zero and variance $\frac{3}{2}\sigma^2/n$, where σ^2 takes a lower bound from Theorem 2.2. This approach achieves (ϵ, δ)-DP when at least $2/3$ servers are honest. Note that there is an asymmetry between the assumed number of servers that honestly follow the protocol in Prio's threat model and Dwork-MPC, as Prio requires at least one honest server, whereas Dwork-MPC assumes that at least $2/3$ servers are honest. While it is possible to generate noise in a dishonest majority setting, doing so requires performing $O(n)$ rounds as demonstrated in prior literature by Beaver and Goldwasser [6, 29]. As such, we exclude such protocols from our analysis, as the goal of this work is to provide a mechanism that is not much more costly from plain Prio.

Dwork-MPC also proposes an variant to prevent a Byzantine server from adding a significant large amount of noise to its share, that we call Dwork-MPC*. This approach requires the servers to cooperatively generate shares of many random bits that can be transformed shares of a noise drawn from Binomial distribution (close to a Gaussian noise). The number of high-quality random bits for achieving (ϵ, δ)-DP should be at least $k = 64 \log(2/\delta)/\epsilon^2$ [19, §2.1]. All servers must verify that the shared values are in the specified set. As these bits can be chosen adversarially by the server, the shares of these bits must be verified and then combined with high-quality public bits using Verifiable Secret Sharing (VSS) [44] and a deterministic extractor. The main computation costs of the protocol are the multiplications for verifying the shares' membership and the execution of VSS, and hence is proportional to the number of the coins.

Prio requires the setting where each server "represents" a portion of a set of N clients (roughly N/n). Consequently, in Dwork-MPC*, each server must take on the work of N/n clients, and so is responsible for contributing at least $k/N \cdot N/n = k/n$ random bits and at least $k/n \cdot n = k$ number of shares and their corresponding verification. For example, in a system with three servers, and where ($\epsilon = 0.01, \delta = 10^{-6}$), each server would need to generate, secret share, and verify roughly $1.3 \cdot 10^6$ number of random bit and their verification, which adds a significant computation and communication overheads to the servers in Prio.

5.2 Distributed Noise Generation

To align with the assumptions in Prio, we now describe two non-interactive techniques that allow either only clients (Client-DP) or only servers (Server-DP) to generate noise using the Gaussian Mechanism where noise sampled from the normal distribution $\mathcal{N}(0, \sigma^2)$ is added to a value. These mechanisms are modeled after prior work in the literature [23, 31] and can be applied to the Prio system in order to add differential privacy.

5.2.1 Server-DP: Server Noise Generation. We now describe a mechanism to generate noise [31] involving only servers that we call Server-DP. Server-DP uses the Gaussian mechanism and properties of independent normally distributed random variables. Let n denote the number of servers and suppose that the sensitivity, Δ, and privacy parameters (ϵ, δ) have been established beforehand. The protocol used by the servers to add noise is as follows.

(1) Each server samples noise from a normal distribution $\mathcal{N}(0, \sigma^2)$ where $\sigma^2 = \frac{2\ln(1.25/\delta)\Delta^2}{\epsilon^2}$.
(2) When locally summing the shares of data, the servers add the noise they previously sampled, before revealing their final sum.

Privacy. Prio ensures that privacy is achieved if at least one server is honest. For Server-DP, (ϵ, δ)-DP is achieved if at least one server is honest, by Theorem 2.2. This is a generalization of Dwork-MPC on the assumptions on the number of honest server. Even when the adversary (e.g. the data analyst) controls $(n-1)$ servers and $(N-1)$ clients, the input of the honest client is protected by the (ϵ, δ)-DP guarantee.

Utility. Suppose that the output of the mechanism is given to a data analyst who does not collude with any servers. Server-DP uses the Gaussian mechanism which has error σ^2 when noise is sampled from a normal distribution $\mathcal{N}(0, \sigma^2)$. Assume that there are n servers. Then, the expected error of the result received by the analyst for Server-DP is $\textsc{Error}_{\text{Server-DP}} = n(2\ln(1.25/\delta))\Delta^2/\epsilon^2$.

Robustness. Server-DP is robust against adversarial clients when at minimum one server is honest, and so fits within Prio's threat model. Additionally, Server-DP's robustness includes the correctness, soundness, and zero-knowledge properties assured by the use of SNIPs. The malicious behavior by clients is limited to sending incorrect input, but this can be verified by the use of SNIPs.

Extensions. If different assumptions on the number of servers that collude are made, different noise can be added. Suppose that all servers are honest, do not collude with one another, and do not leak the noise that they included in the output. Then the noise added to the output is normally distributed with parameters $\sigma^2 = n \cdot \frac{2\ln(1.25/\delta)\Delta^2}{\epsilon^2}$ where n is the number of servers. This ensures $(\epsilon/\sqrt{n}, \delta)$-DP, but comes with tradeoffs with respect to utility. This works well when n is small. For example, Cryptϵ has two servers and they add noise in this way [47]. In short, Server-DP introduces noise that scales linearly relative to the number of honest servers above the assumed threshold.

5.2.2 Client-DP: Client Noise Generation. We now describe a mechanism for clients to add noise [23, 31] that we call Client-DP. In this mechanism, each client generates noise, and then all noise is added to the client's input and then directly included in the aggregate. We describe the protocol in the most general case, where we assume some number of malicious clients equal to m (m could be 0). In summary, all clients perform the following steps:

(1) Sample Gaussian noise from a normal distribution $\mathcal{N}(0, (2\ln(1.25/\delta) \cdot (\Delta/\epsilon)^2)/(N-m))$.
(2) Encode the Gaussian noise according to the same AFE as the data so that Prio servers can aggregate it directly when performing share aggregation.

Suppose there are N clients, the sensitivity is Δ, and the privacy parameters ϵ, δ have been established. We want each client to generate noise from a normal distribution $\mathcal{N}(0, \sigma_i^2)$ such that $\sum_{i=1}^{N} \sigma_i^2 \geq 2\ln(1.25/\delta) \cdot (\Delta/\epsilon)^2$. There are a several different options to do this. First, we could let $\sigma_i = (2\ln(1.25/\delta) \cdot (\Delta/\epsilon)^2)/N$. This is fairly straightforward and ensures (ϵ, δ)-DP if the clients honestly sample noise. An alternative is to assume that some proportion of clients are malicious, say m out of N clients. Then, having clients sample from a normal distribution with $\sigma_i = (2\ln(1.25/\delta) \cdot (\Delta/\epsilon)^2)/(N-m)$ would ensure that the noise provided by honest clients still provides (ϵ, δ)-DP. Any additional noise added by malicious clients can be treated as post-processing (Theorem 2.3), and consequently does not affect the privacy guarantee. This choice depends on the context of the system as well as the number of clients, since a large number of clients means that even in the first case, a single client who incorrectly samples noise will not largely affect the output or the privacy guarantees. The value for m is provided to the client as part of the noise parameters together with ϵ, δ, Δ. As before, clients must submit proofs to show that the affine-aggregatable encoding (AFE) of the noise is correctly generated.

Privacy. Client-DP ensures differential privacy by receiving Gaussian noise from every client. This Gaussian noise is added up, and the aggregate noise ensures DP. Malicious clients can be taken into account by estimating how many exist in the system and increasing the amount of noise honest clients submit to compensate for those clients. This construction guarantees (ϵ, δ)-DP.

Utility. Suppose that a data analyst, not colluding with any servers, receives the output of the mechanism. Client-DP also uses the Gaussian mechanism which has error σ^2 when noise is sampled from a normal distribution $\mathcal{N}(0, \sigma^2)$. If we assume that there are m clients compromised by an honest but curious adversary (e.g, the data analyst), and N clients submit noise values, then we expect the amount of error of the result received by the analyst to be $\textsc{Error}_{\text{Client-DP}} = (N/(N-m))(2\ln(1.25/\delta))\Delta^2/\epsilon^2$.

Robustness. Client-DP's robustness against client misbehavior can be ensured by proving the correctness of client inputs via SNIPs. However, servers will not be able to verify that the noise was correctly sampled from the Gaussian distribution.

Extensions. We may consider extending this approach by applying zero-knowledge proofs as a way to verify that a program to add noise was executed correctly, as is done in VerDP [43]. Unfortunately, this approach is highly inefficient and would introduce unreasonable overhead, even if it were able to be integrated into the structure of the SNIPs used by Prio.

6 DPRIO: DIFFERENTIALLY PRIVATE STATISTICS WITH HIGH UTILITY

We now describe our construction DPrio that incorporates differential privacy (DP) into the Prio system, and provide an overview of the construction in Figure 2. At a high level, instead of many parties adding noise directly as is the case in local DP, Client-DP, or Server-DP (which is efficient but decreases data utility) or servers performing MPC operations to generate noise (which has high utility but low efficiency), we propose the following system. Clients generate noise, but rather than adding noise directly, they secret share their noise (as their data is secret shared) to the Prio servers. The Prio servers then perform an efficient two-round MPC protocol (where the first round can be batched in a pre-processing phase, if desired) to select *a small number* of clients' noise to add. Section 7 shows that this achieves DP, while satisfying high data utility and efficiency for clients and servers.

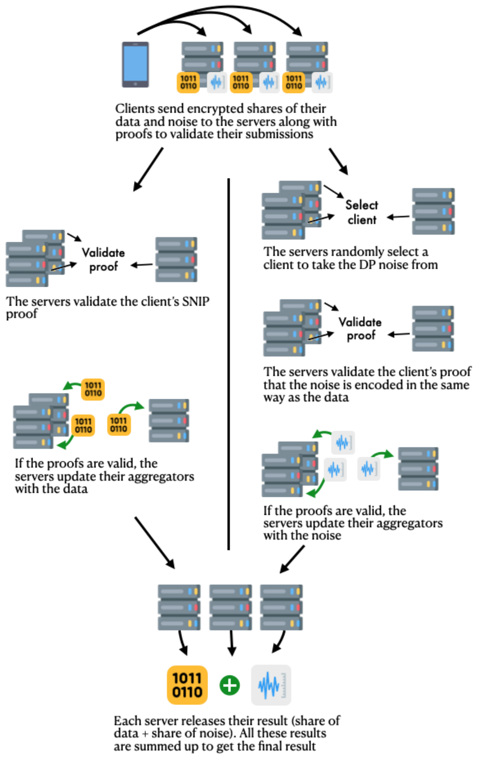
Clients send encrypted shares of their data and noise to the servers along with proofs to validate their submissions
Validate proof
The servers validate the client's SNIP proof
If the proofs are valid, the servers update their aggregators with the data
Select client
The servers randomly select a client to take the DP noise from
Validate proof
The servers validate the client's proof that the noise is encoded in the same way as the data
If the proofs are valid, the servers update their aggregators with the noise
Each server releases their result (share of data + share of noise). All these results are summed up to get the final result

Figure 2: Overview of DPrio. Clients send shares to servers who validate the associated SNIP, select a client's noise, and aggregate the data with the noise.

6.1 Client Noise Generation and Submission

Setup. Initially, a system administrator will determine the set of queries DPrio will support in this instance. It will compute the sensitivity for each of the queries, and decide upon ϵ and δ values.

Protocol. At the time of submitting data to Prio servers, clients additionally perform the following steps:

(1) Sample noise $\rho \xleftarrow{\$} \mathcal{F}$ from a noise distribution $\mathcal{F}$ that leads to a mechanism satisfying (ϵ, δ)-DP.
(2) Encode the noise using the same encoding mechanism as the statistic they are sending to the Prio servers. Note that this also includes a SNIP proving that the noise is validly formatted.
(3) Generate secret shares $s_1, \ldots, s_n$ of the encoded noise using additive secret sharing, such that $\rho = \sum_{i=1}^{n} s_i$. This step is the same as that used by the client to generate secret shares of their statistic to send to the Prio servers.
(4) Submit each s_i along with the secret share of the client's data to the i^{th}Prio server.

Any additive mechanism such as the Gaussian or Laplace mechanism can be used with the protocol. An additive mechanism is one where achieving DP only requires adding noise to the statistic. We now expand on the noise generation and encoding steps.

6.1.1 Noise Generation. To generate noise, all clients begin by sampling a value from a distribution, where the parameter of this distribution is fixed at the time of system setup and is chosen to satisfy the definition of (ϵ, δ)-DP. Each client then encodes this noise and generate a SNIP in a manner identically to that of the statistic at hand, as described in Section 4.1. The client then sends both the statistic SNIP and the noise SNIP to the DPrio servers. The DPrio servers will choose c clients' noise values at random (Section 6.2). Before sending the aggregated sum to the data analyst, the servers will add the selected clients' noise to the sum. The data analyst will only see the noised sum after aggregating all sums.

The tricky step in this protocol is that typical noise distributions are not discrete, so to achieve DP the client would have to send infinite bits of noise. Since this is not computationally feasible, the clients must truncate their noise. This issue can be addressed by using discrete noise generation methods [3, 4, 15, 34, 50]. We adopt the secure noise generation by [50] to generate integer noise in our implementation. A more detailed analysis of the noise truncation can be found in Appendix B. Similar analysis can be applied to other forms of noise, such as discrete Gaussian noise [15].

6.1.2 Noise Encoding. In this model, we simply require clients submit a proof that their noise falls within the same encoding structure as the client's data itself. This proof does not demonstrate that clients have picked the noise from the *correct* distribution. To protect against misbehaving clients that might simply submit zero instead of correctly sampled noise, Prio servers perform an MPC operation to select *which* client's noise to add. Therefore, the security of this construction depends on the assumed bound of misbehaving clients (Section 7.2).

Encoding the noise uses the same affine-aggregatable encoding (AFE), as in Section 2.2, as the data itself, allowing servers to aggregate noise with the data without requiring reconstruction. Once the client has submitted its noise to the servers, it deletes the noise.

While we demonstrate DPrio specifically as a DP mechanism for Prio, the general approach DPrio follows can extend to alternative private data collection designs that employ similar multi-server models for distributing trust when performing data collection.

6.2 Server Noise Selection

We now describe an efficient commit-reveal MPC protocol servers use to select which clients' noise. Let N be the number of clients who have *submitted* noise. We assume that clients' shares of this noise ordered in some known manner and the Prio servers know this ordering (we describe practical ordering options for implementations in Section 9). Assume the existence of some hash function.

(1) Each server selects some random number ρ within the range $[0, \tau]$, where τ is a multiple of N. That is, server i selects some ρ_i. To prevent against a server being able to enumerate all possible inputs and select a specific one, each server also selects some salt value $\psi_i \in [\Psi]$, where Ψ is some value too large to exhaust by brute force. Having a large Ψ prevents a server from waiting to see all the other servers' commitments, enumerating through all possible hashes, and selecting a specific ρ to influence which client is chosen.
(2) Each i^{th} server computes and publishes the hash $\alpha_i = H(\rho_i||\psi_i)$ of their random number to serve as a commitment.
(3) After having received commitments from all other servers, every server "opens" their commitment, i.e., by publishing ρ_i and ψ_i to each other.
(4) All servers verify each others' commitments, by checking $\alpha_i \overset{?}{=} H(\rho_i||\psi_i)$.
(5) If every check is successful, servers derive $\omega = \sum \rho_i \bmod N$. Here, ω corresponds to a particular client's index. The servers then apply this client's noise to their aggregated sums, throwing away the remaining client noise shares.

The initial commitment ensures that *no server may sway the result towards a particular client.* If a server attempts to cheat by revealing an input different to the one they originally chose, the remaining honest servers will discover this by verifying that the output of the hash function matches the original commitment. This is a standard commit-reveal technique to ensure that each party commits to their contribution before learning all other parties' contributions.

Before taking this step, the servers have already received shares of noise from all N clients. Once a particular client is chosen, the servers simply add the corresponding share of noise to their local sum of data sent by the clients. To add noise in this way, the shares that are received by the servers must have some ordering, which we discuss in Section 9. If the protocol would like to add noise values from c clients, without replacement, the servers simply repeat steps (1)-(5) by excluding previously sampled clients.

We assumed that servers do not collude with clients (Assumption 4.2). A server colluding with a client may learn whether a particular client's noise was selected via the noise selection protocol. In this case, the colluding server could learn the noise value from the client. Under this non-collusion assumption, our commit-reveal protocol could be changed to a protocol which deterministically sets the output (e.g., pick a server and let this server sample a client id and share the id with others, or fix a client id all the time). However, the advantage of this commit-reveal protocol is that it can be easily combined with the shuffle model if we want to remove the non-collusion assumption (Section 7.3.3). Also, its overhead is low, as shown in the Section 8.3. In addition, suppose we relax the non-collusion assumption to cases where servers may collude with a small number of fixed clients. This commit-reveal protocol prevents these servers from swaying the result towards the colluding clients, and thus, it has a stronger guarantee than the alternatives. We discuss alternatives under this collusion in Appendix C.

Unlike prior constructions that employ server cooperation, DPrio achieves significantly higher efficiency and data utility as the MPC operation in DPrio is simply to determine *which* client's noise to add, as opposed to cooperating to generate noise directly.

7 PRIVACY AND SECURITY ANALYSIS

7.1 Differential Privacy

We provide a sketch of a security proof that shows our system satisfies DP, assuming that clients submit well-chosen noise, as described in Section 4.2, and then misbehaving clients in Section 7.3.2.

Clients submit encoded shares of data to Prio servers which ultimately sum to some statistic. They also submit shares of noise encoded similarly such that when the shares are summed, we obtain the statistic with noise sampled from a distribution ensuring (ϵ, δ)-DP. Performing this operation is easily accomplished with the building blocks already provided by Prio; namely, AFEs (such as a linear secret sharing scheme) and SNIPs (we expand further on both in Appendix A). Therefore, after the Prio servers learn the aggregation of their shares locally, the final reconstructed result will be the sum of the desired statistic and the noise to achieve DP. The corresponding mechanism, denoted by $\mathcal{M}$, is simply a perturbation of the aggregate of the client's input data with noise drawn from Gaussian (or Laplace) distribution, that satisfies (ϵ, δ)-DP.

7.1.1 DP against Honest Colluding Servers. We claim that the view and output of *an adversary controlling a single server* is computationally indistinguishable from that of a simulator with access only to the output of the mechanism $\mathcal{M}$ satisfying (ϵ, δ)-DP, and the total size of the database. That is, our protocol satisfies computational differential privacy under SIM-CDP (Definition 2.3).

Let Π denote the protocol executing the mechanism $\mathcal{M}$ and let $\mathcal{M}(D, \epsilon, \delta)$ denote the output of running the protocol on an input database D with tunable parameters ϵ, δ.

THEOREM 7.1. *Let protocol Π correspond to the execution of mechanism $\mathcal{M}$ in DPrio. The view and output of an adversary controlling a server is denoted as $View_\Pi(\mathcal{M}, D, \epsilon, \delta)$, $Output_\Pi(\mathcal{M}, D, \epsilon, \delta)$. There exists a PPT simulator, SIM, such that $\text{SIM}(\mathcal{M}(D, \epsilon, \delta))$ is computationally indistinguishable from $(View_\Pi(\mathcal{M}, D, \epsilon, \delta), Output_\Pi(\mathcal{M}, D, \epsilon, \delta))$.*

The original Prio design [17] demonstrated that there exists an efficient simulator which outputs a transcript of the protocol execution that is indistinguishable from a real transcript and the only information that leaks to the adversary is the value of the function, f, being computed on the clients' private values. The only difference in DPrio is the noise generation by clients and sampling a client at random by Prio servers. From the perspective of an adversary controlling a single server, their view consists only of that which they could see in Prio and a share of some noise, which is private in an information-theoretic sense. Assuming that at least one server is honest, the protocol either outputs a correctly computed (ϵ, δ)-DP result according to the DP mechanism, or an incorrect result which is independent of the true answer. In either case, the adversary learns no more than that which is bounded by the (ϵ, δ)-DP property. Therefore, we obtain the following result.

COROLLARY 7.2. *Protocol Π satisfies computational differential privacy under the SIM-CDP notion.*

While Theorem 7.1 assumes an adversary controls a single server and does not control any of the clients, the proof generalizes to the case where *the adversary* controls up to $(n-1)$ servers, due to the privacy properties of the AFE which requires n out of n shares to reconstruct any input, summarized as follows.

Corollary 7.3. *Theorem 7.1 holds against an adversary controlling all but one of Prio servers.*

As Prio does not consider misbehaving servers, we will not consider misbehaving servers in DPrio as well.

7.1.2 DP against Honest Colluding Clients. Recall that in the DPrio protocol, each client $C_i \in \{C_1, \dots, C_N\}$ samples a noise ρ_i from a distribution that achieves (ϵ, δ)-DP, and then the servers randomly select noise from c clients without replacement from the N clients' noise to perturb the answer. Given a set of clients S, we use $\rho_S = \sum_{i \in S} \rho_i$ to denote the sum of the noise variables contributed by S.

Let m be the number of *honest but curious* clients who collude or are controlled by an adversary. We assume $m \geq 1$ as the adversary can simply include one of the clients who knows its own noise. We want to highlight that this analysis treats the clients' noise differently from their input data. In particular, the colluding clients can only know their own set of noise but not the other clients' noise, but they can know the other client's data from other public sources as in the standard DP analysis. Hence, we consider an adversary who knows $N-1$ clients' data except the last one in addition to m clients' noise. We would like to show that for any colluding m clients $A = \{C_{i_1}, \dots, C_{i_m}\} \subset \{C_1, \dots, C_N\}$, for any neighboring databases D and D' that differ by one client's input data (not a client in A), for any output o, we have the following guarantee:

$$\Pr_{A=\{C_{i_1},\dots,C_{i_m}\}}[o|D] \leq e^{\epsilon} \Pr_{A=\{C_{i_1},\dots,C_{i_m}\}}[o|D'] + \delta. \tag{2}$$

In this protocol, there are two types of randomnesses involved: (i) the noise sampled by the clients; (ii) the set of clients selected by the servers, denoted by $S = \{C_{s_1}, \dots, C_{s_c}\}$. Without loss of generality, we consider the set of compromised m clients is $A = \{C_1, \dots, C_m\}$.

First, we consider a general and simple case when $\underline{c > m \geq 1}$, i.e., the number of noise-providing clients selected by the servers is more than the number of colluding clients. The colluding clients do not know the remaining $(c-m)$ noise values, and hence cannot break the DP guarantee offered by each single noise.

Theorem 7.4. *When $c > m \geq 1$, the protocol against m number of colluding clients satisfies (ϵ, δ)-DP guarantee.*

Proof. Consider a noisy output o and neighbors (D, D'). We use $\rho_{(S-A)}$ to denote the sum of the noise variables contributed by the clients in $S - A$ and $\rho_{(S\cap A)}$ to denote the sum of the noise variables contributed by the clients in $S \cap A$. Given that A knows the noise of the first m clients, we break down the probability of generating the noisy output o into several cases depending on the intersection size between S and A. Note that $S - A$ is non-empty as $c > m$ (there is always some client's noise in S not known by A).

$$\begin{aligned}
&\Pr_{A=\{C_1,\dots,C_m\}}[o|D] \\
= \;& \sum_{i=0}^{m} \Pr[\rho_{(S-A)} = o - q(D) - \rho_{(S\cap A)} \,|\, |A \cap S| = i] \Pr[|A \cap S| = i] \\
\leq \;& \sum_{i=0}^{m} (e^{\epsilon} \Pr[\rho_{(S-A)} = o - q(D') - \rho_{(S\cap A)} \,|\, |A \cap S| = i] + \delta) \\
& \quad \cdot \Pr[|A \cap S| = i] \\
= \;& e^{\epsilon} (\sum_{i=0}^{m} \Pr[\rho_{(S-A)} = o - q(D') - \rho_{(S\cap A)} \,|\, |A \cap S| = i] \\
& \quad \cdot \Pr[|A \cap S| = i]) + \delta \\
= \;& e^{\epsilon} \Pr_{A=\{C_1,\dots,C_m\}}[o|D'] + \delta
\end{aligned} \tag{3}$$

A similar proof applies for the general case $c > m \geq 1$. □

If we know the value of m, then setting $c = m+1$ for the protocol is sufficient to eliminate any adversaries who compromise m clients and to ensure the same level of DP guarantee. If setting $c > m+1$, we can also tighten the privacy parameter of DP for Gaussian noise. For example, if we take the sum of noise from c honest clients who sampled from a Gaussian distribution with parameter σ^2 and m of them are compromised by the adversary, the composed noise of the remaining clients follows a Gaussian distribution with parameter $(c-m) \cdot \sigma^2$. This gives $(\epsilon/\sqrt{c-m}, \delta)$-DP guarantee.

Next, we consider a general case when $\underline{c \leq m}$. We start with a basic case when $c = 1 \leq m$ (a single client's noise is selected by the servers). Let C_s be the client whose noise is selected. Among all the adversaries, the ones who control C_s (including C_s itself as an adversary) are able to distinguish between the true database D and its neighbors D' and hence break the DP guarantee offered by this single noise. In particular, for any set of m colluding clients, there is a probability of $\frac{m}{N}$ that the selected client is one of the colluding clients $C_s \in A$. If this happens, then given a noisy output o, and the true database instance D, then the adversary is certain that one of its noise can produce the exact o. Hence, we have

$$\begin{aligned}
&\Pr_{A=\{C_1,\dots,C_m\}}[o|D] \\
= \;& \Pr[C_s \in A] \Pr[\rho_{C_s} = o - q(D) | C_s \in A] \\
& + \Pr[C_s \notin A] \Pr[\rho_{C_s} = o - q(D) | C_s \notin A] \\
= \;& \frac{m}{N} \cdot 1 + (1 - \frac{m}{N}) \Pr[\rho_{C_s} = o - q(D) | C_s \notin A]
\end{aligned} \tag{4}$$

However, for a true instance's neighbor D', the adversary can test its noise and find that it is very unlikely to produce o from D', i.e.,

$$\begin{aligned}
&\Pr_{A=\{C_1,\dots,C_m\}}[o|D'] \\
= \;& \frac{m}{N} \Pr[\rho_{C_s} = o - q(D') | C_s \in A] \\
& + (1 - \frac{m}{N}) \Pr[\rho_{C_s} = o - q(D') C_s \notin A] \\
\geq \;& \frac{m}{N} \cdot 0 + (1 - \frac{m}{N}) \Pr[\rho_{C_s} = o - q(D') | C_s \notin A]
\end{aligned} \tag{5}$$

Though the second probability term in Eqn. (4) and Eqn. (4), $\Pr[\rho_{C_s} = o - q(D) | C_s \notin A]$ and $\Pr[\rho_{C_s} = o - q(D') | C_s \notin A]$ is bounded by (ϵ, δ), but the difference in the first term cannot be bounded by the initial DP parameters (ϵ, δ). This adds an additional failing probability bounded by $\frac{m}{N}$. In order to reduce this additional failing probability, we will consider protocols that add $c > 1$ clients' noise.

If $c = m > 1$, there exists a set of colluding clients who can break the DP guarantee, by similar reasoning to the basic protocol with $c = 1$. We discuss how to analyze an upper bound for the additional failing probability to achieve the initial (ϵ, δ)-DP guarantee by a single noise. We denote this additional failing probability by δ'. It is possible that the set of colluding clients is exactly the same as the ones selected by the servers $S = A$. In this case, given a noisy output

o, and the true database instance D, the adversary can be certain that its noise can produce the exact o from D. For its neighbors D', the probability is much smaller (can be close to 0). Hence,

$$\begin{aligned}
& \Pr_{A=\{C_1,\ldots,C_m\}}[o|D] \\
= \; & \Pr[S=A]\Pr[\rho_S = o - q(D)|S=A] \\
& + \Pr[C_s \neq A]\Pr[\rho_S = o - q(D)|C_s \neq A] \\
= \; & 1/\binom{N}{c}\cdot 1 + (1 - 1/\binom{N}{c})\Pr[\rho_{C_s} = o - q(D)|C_s \neq A] \\
\leq \; & 1/\binom{N}{c}\cdot 1 + (1 - 1/\binom{N}{c})(e^{\epsilon}\Pr[\rho_{C_s} = o - q(D')|C_s \neq A] + \delta) \\
\leq \; & e^{\epsilon}\Pr_{A=\{C_1,\ldots,C_m\}}[o|D'] + \delta + 1/\binom{N}{c} \qquad (6)
\end{aligned}$$

If $c < m$, the additional failing probability increases, as there is a higher chance to find the set of chosen noise than the case when $c = m$. We have the failing probability $\delta' = \binom{N-c}{m-c}/\binom{N}{c}$. The analysis can be generalized to any neighbors (D, D').

Theorem 7.5. *When* $c \leq m$, *the protocol against m number of colluding clients satisfy* $(\epsilon, \delta+\delta')$*-DP guarantee, where* $\delta' = \binom{N-c}{m-c}/\binom{N}{c}$.

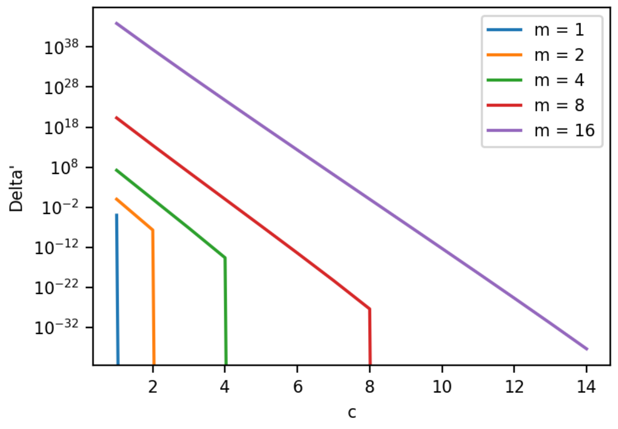

Figure 3: δ' **for** $1 \leq m \leq 16$ **and** $N = 10,000$

Figure 3 shows how δ' changes depending on how big c and m are. δ' decreases exponentially as c gets closer to m and when $c \geq m$, $\delta' = 0$.

The settings in which applying DPrio is appropriate include those where the assumed number of colluding clients, m, is a constant. This is appropriate in large networks with millions of users where it is unlikely that a large proportion of clients would be compromised. Current use cases of Prio fall into this category, such as collecting browser telemetry [24]. In these settings, we achieve a reasonable value for δ'. In our evaluation, we set $c = \log N > m$ (where m is a constant) and hence $\delta' = 0$ by Theorem 7.4. Let's say if m happens to be the same as $c = \log(N)$, then by Theorem 7.5, we have

$$\delta' = 1/\binom{N}{c} = 1/\binom{N}{\log N} \leq 1/N^{\log N}. \qquad (7)$$

The failing probability analysis above assumed the adversary has access to $(N-1)$ clients' data and m clients' noise, but still allows the clients to follow the protocol, the worst case for DP analysis. We analyze a malicious adversary who controls clients to use incorrect data and noise distribution in Sections 7.2 and Section 7.3.

7.1.3 *Comparison with Prio.*

Prio originally provided f-privacy, meaning that for an aggregate function f, an adversary who controls any number of clients and all but one server learns nothing about the honest clients' values x_i, except what they can learn from the value $f(x_1, \ldots, x_n)$ itself. DPrio improves upon this property by ensuring DP. Though the previous protocol presented for DPrio offers DP under the assumption that the adversary controls either up to $(n-1)$ servers or up to m clients, DPrio still offers the same f-privacy in the worst case (i.e., if an adversary controls a Prio server and learn which client's noise is chosen, such as by colluding or directly controlling the client and therefore its choice of noise).

7.2 Robustness against Malicious Clients

Prio is robust against *malicious clients* when all servers are honest. That is, a set of malicious clients cannot influence the final aggregate beyond their ability to choose arbitrary *valid* inputs, due to the use of SNIPs to verify that secret-shared data is in fact with respect to a valid statistic. This guarantee similarly holds true for DPrio and is similarly accomplished by proving the correctness of clients' noise via SNIPs. The key distinction between Prio and DPrio in this regard is that in DPrio, a client submits not just a data value, but also a noise value. Thus, Prio servers can still verify that the input is a valid noise value (i.e., some k-bit integer value). However, Prio servers cannot verify that the noise was correctly sampled with respect to some distribution. This poses some risk since clients can arbitrarily choose very large or very small noise values within the bound of the allowed range to sway the output in their favour. However, given a large number of honest clients, it is unlikely that the servers will choose a malicious client's noise.

The risk of incorrectly sampled noise being chosen is increased when we select noise submitted by more than one client, as suggested in Section 7.1.2. We demonstrated that as the number of noise values chosen from clients, c, increases, it decreases the failing probability of the DP guarantee. However, increasing c also increases the probability that an incorrectly sampled noise value is included in the final result, therefore decreasing the overall robustness of the protocol. In particular, suppose the servers choose c clients' noise without replacement and there are m malicious clients. Assume the worst case scenario where all malicious clients submit incorrectly sampled noise values. Then the probability that at least one incorrectly sampled noise value is included in the result is $1 - \binom{N-m}{c}/\binom{N}{c}$. If m is a small constant relative to N, then this probability is small. Figure 4 shows how this probability changes depending on c and m. It increases linearly as c and m grow larger. Additionally, we note that the risk this poses is not much different than the risk of a malicious client submitting incorrect data. This risk is mitigated by the fact that Prio and DPrio both ensure that data points and noise values conform to a valid data type; neither ensure that the data or noise values are necessarily correct. This vulnerability exists in any system collecting data from clients.

In practice, it is realistic to assume that N is large, and that the adversary controls a small fraction of clients' input. This assumption is reasonable, given that Prio is intended to work at large scale.

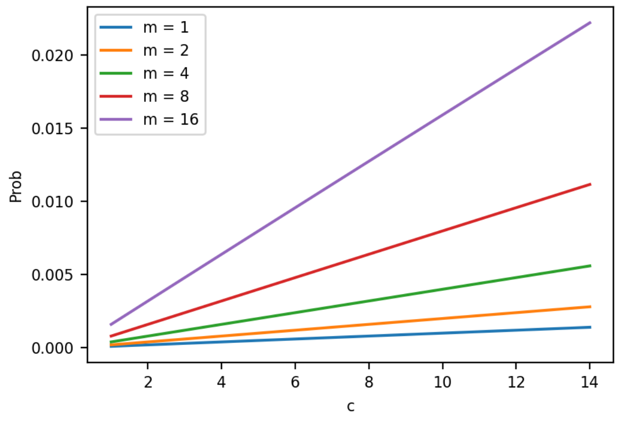

Figure 4: Probability that at least one incorrectly sampled noise is chosen, for $1 \leq m \leq 16$ and $N = 10,000$

Major browsers such as Firefox and Chrome which use or plan to use Prio for data collection have millions if not billions of users. The alternative solution using MPC to sample noise in a distributed setting [19] assumes that at most one third of the servers generating noise are faulty/malicious, where the number of servers may be very small. Our protocol does not rely on such an assumption.

7.3 Comparison of DP Mechanisms

We present a comparison of security and privacy properties in Table 1 between DPrio and related constructions.

7.3.1 Server and Analyst Misbehavior. All protocols, except for the local DP construction require the assumption that servers are honest but curious. There is, however, a difference in the proportion of servers which must act honestly. The Dwork MPC construction requires that at least 2/3 of servers act honestly while the plain Prio construction only assumes at least one server is honest. Further, DPrio, Client-DP, and Server-DP all require that at least one server is honest. All protocols are secure against a Misbehaving Analyst, i.e. any third party which obtains the output cannot learn more than what is revealed from the differentially private output.

7.3.2 Client Misbehavior. Clients can misbehave in multiple ways. They may use incorrect data for the computation, use an incorrect noise distribution, curiously inspect the final/intermediate output to learn information about others' data, or collude with dishonest servers. Each model we evaluate can prevent some forms of misbehavior but not all, with differing implications for data utility.

Assumption on the number of honest clients. DPrio achieves DP so long as the client noise that is selected is honestly generated and included in the aggregate sum by the servers. As such, the security model for DPrio assumes that the client noise that is selected was honestly sampled. . Conversely, Client-DP assumes that the *sum* of client noise that is added is of sufficient quality. Similarly, Server-DP assumes that the noise generated by the designated servers has been honestly generated. As such, the utility of data scales relative to the proportion of assumed honest clients and servers for both Client-DP and Server-DP (as underestimating the number of misbehaving entities results in additional noise, thereby lowering data utility). Conversely, DPrio simply requires that the probability of selecting noise that was generated by a client honestly following the protocol to be sufficiently high.

Clients using an incorrect noise distribution. Clients may misbehave by using an incorrect noise distribution when adding noise. For example, they might choose to use a weaker distribution to impair the DP guarantee, or submit no noise at all. This threat is irrelevant to Dwork-MPC and Server-DP which do not require clients to sample noise. On the other hand, Local DP, Client-DP, and DPrio are more vulnerable to this attack, to varying degrees. In Local DP, an honest client can ensure their data remains private by adding sufficient noise to their own data, thus a lack of additional noise from other parties does not have a catastrophic effect on their privacy; however, this approach significantly reduces utility. In Client-DP, we mitigate this attack by making an assumption on the number of honest clients, as discussed in the previous paragraph. With a sufficient number of honest clients, we can deter the effects of the attack while maintaining better utility than the Local DP model. Finally, in DPrio there is a possibility that a dishonest client's noise is chosen. This would be detrimental to the privacy of the honest clients. Hence, we suggest using DPrio only when the proportion of clients who might misbehave in this way is small.

Clients colluding with other parties. Depending on how many parties the misbehaving clients collude with along with how many servers are included in the misbehavior, different models provide different protections. We drop δ from the discussion for simplicity.

Case 1: $N - 1$ clients and $n - 1$ servers are controlled by an adversary. Dwork-MPC achieves ϵ-DP if no more than $\frac{1}{3}$ of the servers are corrupted. In this case, since n-1 servers are corrupted, the last server still adds some Gaussian noise (sampled with variance $\frac{3\sigma^2}{2n}$), which offers a much weaker DP guarantee of $\frac{2n}{3}\epsilon$-DP. Local DP provides ϵ-DP as each client provides its own protection. Client-DP provides $\frac{N}{1-1/m}\epsilon$-DP and Server-DP provides ϵ-DP. DPrio does not ensure DP as the servers can collude with the clients who contribute noise and remove noise from the answer.

Case 2: Adversary controls $N - 1$ clients and fewer than $\frac{n}{3}$ servers. Dwork-MPC provides ϵ-DP as it assumes that at most $\frac{1}{3}$ servers collude. The other models provide the same protection as the prior case.

Case 3: Adversary controls m clients and $n - 1$ servers. Dwork-MPC only provides $\frac{2n}{3}\epsilon$-DP since the last server still adds Gaussian noise with variance $\frac{3\sigma^2}{2n}$. Local DP, Client-DP, and Server-DP all provide ϵ-DP. DPrio does not offer DP.

Case 4: Adversary controls m clients and 0 servers. Dwork-MPC, Local DP, Server-DP, and DPrio all provide ϵ-DP. Client-DP ensures $\frac{N}{1-1/m}\epsilon$-DP.

Case 5: Adversary controls 0 clients and $n-1$ servers. Dwork-MPC provides $\frac{2n}{3}\epsilon$-DP since since the last server still adds Gaussian noise with variance $\frac{3\sigma^2}{2n}$. Local DP, Client-DP, Server-DP, and DPrio all ensure ϵ-DP.

7.3.3 Collusion Between Clients and Servers. The DPrio design as described does not necessarily protect against the setting where

	Plain Prio	Dwork-MPC (*)	Local DP	Client-DP	Server-DP	DPrio
Robust Against Analyst Misbehavior	Yes	Yes	Yes	Yes	Yes	Yes
Minimum Number of Honest Clients	N/A	N/A	1	$N - m$	N/A	$N - m$
Minimum Number of Honest Servers	1	1/3 of total	0	1	1	1
Differential Privacy	None	Approx.	Pure	Approx.	Approx.	Approx.

Table 1: Comparison of Privacy and Security Properties. N/A means the threat is not applicable in that model. Pure refers to ϵ-DP, and Approx. refers to (ϵ, δ)-DP. HBC means Honest but Curious. We assume at least one honest server and the number of dishonest clients are below some bound. N is the number of clients, m of which are assumed to be dishonest.

	Plain Prio	Dwork-MPC*	Local DP	Client-DP	Server-DP/Dwork-MPC	DPrio
Client Computation	N/A	N/A	$O(1)$	$O(1)$	N/A	$O(1)$
Server Computation	$O(1)$	$\Theta(n)$	$O(1)$	$O(1)$	$O(1)$	$O(1)$
Client-Server Bandwidth	$O(nN)$	$O(nN)$	$O(nN)$	$O(nN)$	$O(nN)$	$O(nN)$
Server-Server Bandwidth	0	$O(nN)$	0	0	0	$O(1)$
Server-Server Rounds	0	2	0	0	0	2
Server MPC Overhead	N/A	$O(n)$	0	0	$O(n)$	$O(c)$
Data Utility	N/A	Central	Local	Central (small m)	Central (small n)	Central (small c)

Table 2: Comparison of Efficiency and Utility Properties. Server-server rounds represents the number of round trips required between servers. n represents the number of servers, and N represents the number of clients. $O(1)$ represents constant overhead.

malicious clients collude with a single server. Suppose a malicious client's noise is chosen. The malicious client and server can work together to determine the true result of the statistic once it is published, by subtracting the client's noise. A simple defense against this attack is to employ an intermediary shuffler between the clients and servers, to shuffle the submitted noise before sending it to the DPrio servers. Doing so ensures that DPrio servers cannot link noise received from the shuffler to noise submitted by DPrio clients. This technique is commonly considered in the literature as the shuffle model [9, 26]. While introducing additional overhead of a separate shuffler entity, employing the shuffle model in combination with DPrio mitigates potential collusion between clients and servers while maintaining the utility of the central model.

Alternatively, the servers can perform an oblivious shuffling protocol upon receiving the data [37], removing the requirement of an independent shuffler role entirely. Laur et al. [37] provide an algorithm of oblivious shuffling with communication complexity $O(2^n N \log N)$ where n is the number of servers and N is the number of data points, a practical option for many real-world use cases of Prio. When a large number of servers are required, the authors describe a protocol with a constant number of rounds relative to servers, achieving total communication complexity of $O(nN^2)$ over n rounds. Alternatively, Movahedi et al. [42] suggest a multi-party oblivious shuffling algorithm with communication complexity $\tilde{O}(1)$ over $\log n$ rounds that is secure in a malicious setting of up to 1/3 corrupted servers, the same model as in Dwork-MPC.

8 EVALUATION

We now evaluate and compare our constructions by their utility guarantees and their performance overheads.

8.1 Accuracy

We use the mean squared error in the final noisy answer outputted by the servers to measure the accuracy of each approach. First, the error for Server-DP (Section 5.2.1) is $\textsc{Error}_{\text{Server-DP}} = n(2\ln(1.25/\delta))\Delta^2/\epsilon^2$, where n is the number of servers. This error only depends on the number of servers. When n is small, the error of Server-DP is close to the accuracy of central DP (independent of the data size/the number of clients N). This error term is independent of the number of honest clients.

Next, for Client-DP (Section 5.2.2), the error was computed as $\textsc{Error}_{\text{Client-DP}} = (N/(N-m))(2\ln(1.25/\delta))\Delta^2/\epsilon^2$, where $(N - m)$ is the number of honest clients and N the number of clients submitting noise. This approach assumes that the number of honest colluding clients is no more than m (these clients still follow the protocol and submit proper noise). When m is small, then this error term is close to the accuracy of central DP.

For DPrio, the error in the final query answer is $\textsc{Error}_{DPrio} = 2c\Delta^2/\epsilon^2$, where c is the number of clients' noise sampled by the servers. As shown in Section 7.1.2, when $c > m$, DPrio satisfies DP guarantee, where m is the number of of honest colluding clients (Theorem 7.4). When m is small (independent of the number of clients or data size), DPrio sets $c = m + 1$ and hence its error is also independent of the data size and has accuracy close to central DP.

The three approaches for Prio have error terms close to central DP, which are all much better than that offered by the local model. We summarize the results in Table 2, Among them, Client-DP has the smallest absolute error term. However, Client-DP has the poorest robustness against malicious clients who submit a bad noise. As the servers aggregate every noise from the client, even if only one client is malicious and submit a bad noise, this noise will be aggregated to the final noisy answer and incur a large error for Client-DP. On the other hand, DPrio samples a very small number of noise from the clients, hence it has a small chance to pick a bad client's noise. As shown in Section 7.2, the probability that a bad noise is sampled in the final query answer is small when c and m are small. Therefore, DPrio has a much better robustness than Client-DP. Last, Server-DP does not add any client-generated noise, and hence the accuracy guarantee is not affected at all. Therefore, when the number of malicious clients is greater than 1 but smaller than than the number of servers $n > m > 1$, DPrio is the best option; when $m > n$, Server-DP is preferred; when $m = 0$, Client-DP is preferred, based on their corresponding utility guarantee. We generalize the use of these approaches for Prio with DP in Appendix C.

8.2 Efficiency

We present an efficiency comparison in Table 2 between DPrio, Client-DP, Server-DP, and related constructions we seek to improve upon. The table summarizes computational costs on the part of clients and servers, client-server communication costs, server-server communication costs, computational costs of MPC protocols, and the number of rounds of MPC protocols, if any. It also summarizes the utility of aggregated data if a DP mechanism is in place. A single party can sample noise in constant time ($O(1)$). For DPrio, we assume that input sizes to the hash function are fixed, so a single call to the hash function costs $O(1)$.

Although DPrio requires some computation on the part of the clients and servers, it does not require significant computation costs. Notably, DPrio requires less computation costs on the part of the servers than the Dwork MPC method.

8.3 Implementation and Case Studies

8.3.1 Implementation. To empirically assess DPrio, we implement a library and utility program that processes data from simulated clients using either Prio or DPrio. This code makes use of *libprio-rs*, a Rust implementation of Prio by Divvi Up of ISRG [33], and consists of less than 700 lines of Rust. We additionally extend *libprio-rs* to be able to interpret client data as binary integers rather than discrete bits during aggregation. This extension requires only 55 additional lines of code. Any project using *libprio-rs* can make use of the provided library to implement DPrio with minimal changes.

This library implements the approximate Laplace method described by [50] to generate Laplace noise and truncate the noise generated by clients. We use $b + 1$ bits to store the noise, where $b = \lceil \log_2 ((r6\ln(10))/\lambda) \rceil$ so that the truncation probability for the magnitude of the noise greater than 2^b is smaller than 10^{-6}, where r is the resolution parameter and λ is the noise parameter $\frac{r\epsilon}{\Delta+r}$. The details can be found in Appendix B. As noise can be negative, for each sampled noise, we add 2^b and encode the shifted value with $b + 1$ bits. This ensures that only shares of positive values are sent to servers to be operated upon. After aggregation, $c2^b$ is subtracted from the sum to find the actual noisy aggregated value, where c is the number of clients' noise selected.

8.3.2 Setup. We explore a set of case studies to determine the relative performance of DPrio and Prio. We test the hypothesis that DPrio incurs limited server processing overhead compared to Prio. All code is available at https://github.com/DPrio-PoPETs/dprio.

These case studies simulate two servers computing a population count over a set of clients. Because we focus on server processing overhead in these studies, to avoid network communication delays impacting the results, both servers coexist in the same process. When simulating Prio, the dimension of the data is 1 bit; each client is either in the population being counted or it is not, as represented by its data. In other words, each client submits a data point of 0 or 1. The query sums the data of the clients. In one of these studies, we change ϵ and hence the number of bits for the noise in DPrio. *libprio-rs* does not distinguish noise from data, so the data must be of the same dimension as the noise when simulating DPrio.

8.3.3 Results. The first case study varies ϵ. We measure the time it takes the implementation to process input from 10,000 clients under Prio and DPrio while varying epsilon among the values 0.025, 0.05, 0.1, 0.2, 0.4, and 0.8. The number of clients' noise values selected is $\lceil log_2(N) \rceil = 14$. Table 3 lists the average results of 50 runs. As ϵ increases, the overhead of DPrio decreases as the number of bits for storing noise decreases. Similarly, the average error (the absolute difference between the calculated value and the actual population count) decreases as ϵ increases. Overall, the server processing overhead incurred by DPrio is indeed minimal at no more than 5.65% for these parameters.

The second case study varies the number of clients. We measure the time it takes to process input from 1,000, 10,000, 100,000, or 1,000,000 clients. For these studies, ϵ is fixed at 0.1 and the number of clients' noise values selected is $\lceil log_2(N) \rceil$. As illustrated by Table 4, the overhead of DPrio remains roughly constant as the number of clients increases. The average error slightly increases, as the number of clients' noise selected is based on the population, which increases. The server processing overhead incurred by DPrio is minimal at no more than 6.25% for these parameters.

The third case study varies the number of clients' noise selected. We measure the time it takes the implementation to process input from 10,000 clients while varying the number of clients' noise selected from 1 through 16 in a geometric sequence with a common ratio of 2. ϵ is fixed at 0.1. As illustrated by Table 5, the overhead of DPrio remains roughly constant as the number of clients' noise increases. The average error increases, however, as more noises are selected. The server processing overhead incurred by DPrio is minimal at no more than 5.45% for these parameters.

In all of the case studies, each client takes around 2.1 times as much processing time to encode the data submitted to the servers for DPrio as with Prio. This is due to the increased dimension of the data in DPrio and the fact that both data and noise has to be encoded and submitted, rather than just the data in Prio. However, the amount of work each individual client does is minuscule compared to the work done by the servers, so the additional work necessary for DPrio is not expected to be prohibitive.

ϵ	Prio Server Processing Time (ms)	DPrio Server Processing Time (ms)	Overhead (%)	Error
0.025	1793.9	1895.2	5.65%	180.8
0.05	1795.3	1895.7	5.59%	80.0
0.1	1794.2	1891.1	5.40%	38.6
0.2	1793.2	1888.0	5.29%	19.7
0.4	1790.3	1834.7	2.48%	10.6
0.8	1798.5	1841.5	2.39%	5.0

Table 3: Average simulation of server processing time with varying ϵ. Each simulation has a fixed population size of 10,000 clients, each submitting one bit of data, and $\lceil log_2(N) \rceil = 14$ client noises were selected.

Population Size	Client Noises Selected	Prio Server Processing Time (ms)	DPrio Server Processing Time (ms)	Overhead (%)	Error
1,000	10	177.9	189.0	6.25%	39.4
10,000	14	1790.9	1887.7	5.41%	40.8
100,000	17	17930.0	18899.9	5.41%	46.6
1,000,000	20	179737.8	189179.1	5.25%	55.4

Table 4: Average simulation of server processing time with varying client population size, each client submitting one bit of data. Each simulation has a fixed ϵ of 0.1. $\lceil log_2(N) \rceil$ client noises were selected.

Client Noises Selected	Prio Server Processing Time (ms)	DPrio Server Processing Time (ms)	Overhead (%)	Error
1	1802.6	1898.0	5.29%	11.3
2	1804.1	1899.8	5.31%	12.0
4	1793.2	1887.5	5.26%	20.4
8	1793.3	1889.3	5.35%	35.1
16	1803.5	1901.7	5.45%	41.3

Table 5: Average simulation of server processing time with varying client noises elected. Each simulation has a fixed ϵ of 0.1 and a fixed population size of 10,000, each client submitting one bit of data.

9 ADDITIONAL CONSIDERATIONS

An Optimization for Large Client Sets. DPrio relies upon the probability that enough honest clients submit noise such that servers have a high probability of selecting one honest client at random (or a small set thereof). When the number of clients is large, however, it might be sufficient for only a subset of clients to submit noise. In this setting, clients can use a local probabilistic function (such as flipping a coin) to determine if they should submit noise.

Deterministically Selecting Client Noise. DPrio assumes servers have some deterministic selection mechanism. While some implementations may submit client data along with their identifiers, others may wish to allow clients to remain anonymous. In this setting, clients can simply hash their noise locally (along with a randomly chosen salt) before generating secret shares, and send this as a one-time identifier to servers to use for selection. If there is not an exact match between the server's random value ω and a client identifier, a deterministic fuzzy matching mechanism can instead be used until a match is found.

Extending Beyond Additive Mechanisms. Our construction and associated privacy property applies to the setting where the servers wish to compute an integer sum, integer mean, or frequency count. Of course, the system can be adapted to have the clients sample noise from different distributions. However, applying more complex mechanisms, such as the exponential mechanism [11], which require more than simply summing the noise with the statistic may be more difficult to integrate. We leave this for future work.

10 CONCLUSION

The use of private statistics is no longer of purely academic interest; systems like Prio are currently being deployed and used by some of the world's largest organizations. However, simply aggregating all client statistics can still allow for denomination attacks; in such settings, differential privacy provides a formal guarantee for users' privacy. In this work, we define a lightweight mechanism that we call DPrio to add differential privacy to existing Prio systems, with high data utility. DPrio defines an efficient MPC protocol that achieves the same utility as centralized differential privacy, and whose computational overhead to servers remains constant regardless of the number of participating clients. Further, the assumption about the number of honest clients remains low, unlike existing distributed noise generation mechanisms along the lines of Client-DP and Server-DP. While DPrio does require server interaction, such interactions can be batched, so at minimum requires no more communication overhead than plain Prio.

ACKNOWLEDGMENTS

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors.

REFERENCES

[1] Census bureau sets key parameters to protect privacy in 2020 census results, Oct 2021.
[2] Josh Aas and Tim Geoghegan. Introducing ISRG Prio Services for Privacy Respecting Metrics. https://www.abetterinternet.org/post/introducing-prio-services/. Accessed 2021-04-08.
[3] Naman Agarwal, Peter Kairouz, and Ziyu Liu. The skellam mechanism for differentially private federated learning. *NeurIPS*, 2021.
[4] Naman Agarwal, Ananda Theertha Suresh, Felix Yu, Sanjiv Kumar, and H. Brendan McMahan. Cpsgd: Communication-efficient and differentially-private distributed sgd. In *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, page 7575–7586, Red Hook, NY, USA, 2018. Curran Associates Inc.
[5] Borja Balle, James Bell, Adrià Gascón, and Kobbi Nissim. The privacy blanket of the shuffle model. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology — CRYPTO 2019*, pages 638–667. Springer International Publishing, 2019.
[6] Donald Beaver and Shafi Goldwasser. Multiparty computation with faulty majority (extended announcement). In *30th Annual Symposium on Foundations of Computer Science, Research Triangle Park, North Carolina, USA, 30 October - 1 November 1989*, pages 468–473. IEEE Computer Society, 1989.
[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation. In *ACM Symposium on Theory of Computing*, page 1–10, 1988.
[8] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnés, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. *CoRR*, abs/1710.00901, 2017.
[9] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. Prochlo: Strong privacy for analytics in the crowd. In *26th Symposium on Operating Systems Principles*, SOSP '17, pages 441–459, 2017.
[10] George Robert Blakley. Safeguarding cryptographic keys. In *International Workshop on Managing Requirements Knowledge (MARK)*, pages 313–318, 1979.
[11] Jonas Böhler and Florian Kerschbaum. Secure multi-party computation of differentially private median. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2147–2164, 2020.
[12] Jonas Böhler and Florian Kerschbaum. Secure sublinear time differentially private median computation. In *Network and Distributed Systems Security Symposium (NDSS)*, 2020.
[13] Anne Broadbent and Alain Tapp. Information-theoretic security without an honest majority. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 410–426. Springer, 2007.
[14] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. *CoRR*, abs/1605.02065, 2016.
[15] Clément L. Canonne, Gautam Kamath, and Thomas Steinke. The discrete gaussian for differential privacy, 2020.
[16] Jeffrey Champion, Abhi Shelat, and Jonathan Ullman. Securely sampling biased coins with applications to differential privacy. In *2019 ACM SIGSAC Conference on Computer and Communications Security*, pages 603–614, 2019.
[17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 259–282, USA, 2017. USENIX Association.
[18] Ronald Cramer, Ivan Damgård, and Ueli Maurer. General secure multi-party computation from any linear secret-sharing scheme. In *Advances in Cryptology – EUROCRYPT 2000*, pages 316–334, 2000.
[19] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. Our data, ourselves: Privacy via distributed noise generation. In Serge Vaudenay, editor, *Advances in Cryptology - EUROCRYPT 2006*, pages 486–503, Berlin, Heidelberg, 2006. Springer Berlin Heidelberg.
[20] Cynthia Dwork, Frank McSherry, Kobbi Nissim, and Adam Smith. Calibrating noise to sensitivity in private data analysis. In *Theory of cryptography conference*, pages 265–284. Springer, 2006.
[21] Cynthia Dwork, Aaron Roth, et al. The algorithmic foundations of differential privacy. *Foundations and Trends in Theoretical Computer Science*, 9(3-4):211–407, 2014.
[22] Fabienne Eigner, Aniket Kate, Matteo Maffei, Francesca Pampaloni, and Ivan Pryvalov. Differentially private data aggregation with optimal utility. In *Proceedings of the 30th Annual Computer Security Applications Conference*, ACSAC '14, pages 316–325, New York, NY, USA, 2014. Association for Computing Machinery.
[23] Tariq Elahi, George Danezis, and Ian Goldberg. Privex: Private collection of traffic statistics for anonymous communication networks. In Gail-Joon Ahn, Moti Yung, and Ninghui Li, editors, *ACM SIGSAC Conference on Computer and Communications Security, Scottsdale, AZ, USA, November 3-7, 2014*, pages 1068–1079. ACM, 2014.
[24] Steven Englehardt. Next steps in privacy-preserving Telemetry with Prio. blog.mozilla.org/security/2019/06/06/next-steps-in-privacy-preserving\-telemetry-with-prio/. Accessed 2021-04-08.
[25] Reo Eriguchi, Atsunori Ichikawa, Noboru Kunihiro, and Koji Nuida. Efficient noise generation to achieve differential privacy with applications to secure multiparty computation. In *International Conference on Financial Cryptography and Data Security*, pages 271–290. Springer, 2021.
[26] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 2468–2479. SIAM, 2019.
[27] Úlfar Erlingsson, Vasyl Pihur, and Aleksandra Korolova. Rappor: Randomized aggregatable privacy-preserving ordinal response. In *2014 ACM SIGSAC conference on computer and communications security*, pages 1054–1067, 2014.
[28] Simson Garfinkel, John M Abowd, and Christian Martindale. Understanding database reconstruction attacks on public data. *Communications of the ACM*, 62(3):46–53, 2019.
[29] Shafi Goldwasser and Leonid A. Levin. Fair computation of general functions in presence of immoral majority. In Alfred Menezes and Scott A. Vanstone, editors, *CRYPTO '90, 10th Annual International Cryptology Conference, Santa Barbara, California, USA, August 11-15, 1990*, volume 537 of *LNCS*, pages 77–93. Springer, 1990.
[30] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The knowledge complexity of interactive proof systems. *SIAM Journal on computing*, 18(1):186–208, 1989.
[31] Slawomir Goryczka, Li Xiong, and Vaidy S. Sunderam. Secure multiparty aggregation with differential privacy: a comparative study. In Giovanna Guerrini, editor, *Joint 2013 EDBT/ICDT Conferences, EDBT/ICDT '13, Genoa, Italy, March 22, 2013*, pages 155–163. ACM, 2013.
[32] Internet Security Research Group. Let's Encrypt. https://letsencrypt.org/. Accessed 2021-04-08.
[33] ISRG. libprio-rs. https://github.com/divviup/libprio-rs. Accessed 2022-04-30.
[34] Peter Kairouz, Ziyu Liu, and Thomas Steinke. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In Marina Meila and Tong Zhang, editors, *International Conference on Machine Learning, ICML 2021, 18-24 July 2021, Virtual Event*, volume 139 of *Proceedings of Machine Learning Research*, pages 5201–5212. PMLR, 2021.
[35] Alann F. Karr, Xiaodong Lin, Ashish P. Sanil, and Jerome P. Reiter. Secure regression on distributed databases. *Journal of Computational and Graphical Statistics*, 14(2):263–279, 2005.
[36] Klaus Kursawe, George Danezis, and Markulf Kohlweiss. Privacy-friendly aggregation for the smart-grid. In *International symposium on privacy enhancing technologies symposium*, pages 175–191. Springer, 2011.
[37] Sven Laur, Jan Willemson, and Bingsheng Zhang. Round-efficient oblivious database manipulation. In *International Conference on Information Security*, pages 262–277. Springer, 2011.
[38] Ueli Maurer. Secure multi-party computation made simple. *Discrete Applied Mathematics*, 154(2):370–381, 2006.
[39] Frank McSherry. Privacy integrated queries: An extensible platform for privacy-preserving data analysis. *Communications of the ACM*, 53(9):89–97, Sep 2010.
[40] Luca Melis, George Danezis, and Emiliano De Cristofaro. Efficient private statistics with succinct sketches. In *Network and Distributed Systems Security Symposium (NDSS)*, 2016.
[41] Ilya Mironov, Omkant Pandey, Omer Reingold, and Salil Vadhan. Computational differential privacy. In *Annual International Cryptology Conference*, pages 126–142. Springer, 2009.
[42] Mahnush Movahedi, Jared Saia, and Mahdi Zamani. Secure multi-party shuffling. In *International Colloquium on Structural Information and Communication Complexity*, pages 459–473. Springer, 2015.
[43] Arjun Narayan, Ariel Feldman, Antonis Papadimitriou, and Andreas Haeberlen. Verifiable differential privacy. In *Proceedings of the Tenth European Conference on Computer Systems*, pages 1–14, 2015.
[44] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Proceedings of the 11th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '91, page 129–140, Berlin, Heidelberg, 1991. Springer-Verlag.
[45] Raluca Ada Popa, Andrew J Blumberg, Hari Balakrishnan, and Frank H Li. Privacy and accountability for location-based aggregate statistics. In *18th ACM conference on Computer and communications security*, pages 653–666, 2011.
[46] Edo Roth, Karan Newatia, Yiping Ma, Ke Zhong, Sebastian Angel, and Andreas Haeberlen. Mycelium: Large-scale distributed graph queries with differential privacy. In *ACM SIGOPS 28th Symposium on Operating Systems Principles (SOSP)*, pages 327–343, 2021.
[47] Amrita Roy Chowdhury, Chenghong Wang, Xi He, Ashwin Machanavajjhala, and Somesh Jha. Cryptϵ: Crypto-assisted differential privacy on untrusted servers. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 603–619, New York, NY, USA, 2020. Association for Computing Machinery.
[48] Adi Shamir. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

[49] Differential Privacy Team. Learning with privacy at scale. Apple, December 2017.
[50] Google Privacy Team. Secure noise generation. Google, June 2020.
[51] Dan Zhang, Ryan McKenna, Ios Kotsogiannis, Michael Hay, Ashwin Machanavajjhala, and Gerome Miklau. Ektelo. *2018 International Conference on Management of Data*, May 2018.

A ADDITIONAL BACKGROUND INFORMATION

We now provide additional background information on encoding private information as required by Prio and by extension DPrio.

A.1 Affine-Aggregatable Encodings (AFEs)

Given an input $x_1, \dots, x_n$, an *affine-aggregatable encoding* allows for "splitting" each input $x_i, i \in \{1, \dots, n\}$, such that the functions f, f' exist where

$$f'(x_{1j}, \dots, x_{nj}) \longrightarrow \hat{x_j}, \text{ for some } j \in \{1, \dots, \ell\}, \text{ and}$$
$$f(x_1, \dots, x_n) = f(\hat{x_1}, \dots, \hat{x_\ell})$$

Further, an AFE preserves *privacy* if f' reveals nothing about $x_1, \dots, x_n$ beyond the evaluation of f' on its inputs. Towards this end, AFEs define the following:

- *Encode*$(x_i) \longrightarrow \{x_{i1}, \dots, x_{i\ell}\}$: Accepts a data item $x_i, i \in \{1, \dots, n\}$ and outputs a sharing of x_i into ℓ parts.
- *Verify*$(x_{i1}, \dots, x_{i\ell}) \longrightarrow \{0, 1\}$: Accepts ℓ parts and verifies they are a valid encoding of x_i.
- *Decode*$(\hat{x_1}, \dots, \hat{x_\ell})$: Accepts aggregations of parts and computes the output, which is equal to $f(x_1, \dots, x_n)$.

One simple example of an AFE is additive secret sharing, where *Encode*(x_i) simply outputs the set $\{x_{i1}, \dots, x_{i\ell}\}$ such that $x_i = \sum_{j=1}^{\ell} x_{ij}$.

A.2 Secret-Shared Non-Interactive Proofs (SNIPs)

Clients employ an AFE to partition their data value x_i into shares $\{x_{i1}, \dots, x_{i\ell}\}$, and then upload shares to ℓ Prio servers, such that each server receives one share. Additionally, clients upload a SNIP, which is a secret-shared non-interactive zero-knowledge proof of knowledge demonstrating the data value x_i is valid with respect to the corresponding statistic type, given only the shares $\{x_{i1}, \dots, x_{i\ell}\}$, the SNIPs, and a MPC operation that each Prio server participates in. For example, if the statistic allows only binary inputs, the set of SNIPs would demonstrate that x_i is either zero or one, without disclosing which is the case.

To perform verification, Prio servers use their SNIP as well as their share x_{ij} to collectively perform *Verify* via a multi-party computation (MPC) operation. The output from each server participating in the *Verify* protocol is either 0 or 1, indicating whether *Verify* completed successfully, and nothing else.

B NOISE TRUNCATION

DPrio uses the approximate Laplace method described by the secure noise generation algorithm [50] to generate Laplace noise and truncate the noise generated by clients. Approximate Laplace first sets r, the resolution parameter which is the smallest power of two to be greater than $\frac{\Delta/\epsilon}{2^{40}}$ in its java implementation [1]. Given a function f that maps a database to a real number, $f_r(D)$ refers to $f(D)$ rounded to the nearest multiple of r. Then it samples an integer i with probability proportional to $e^{-|i|r\epsilon/\Delta_r}$, where $\Delta_r = \max_{D,D'} \|f_r(D) - f_r(D')\| \le \Delta + r$. Last, the algorithm returns $f_r(D) + ir$ as the noisy output. This process satisfies ϵ-differential privacy.

If we use b bits to store the magnitude of the noise, then the probability for the noise $|ir|$ greater than 2^b can be charged to the δ guarantee of the mechanism. We will show the minimum choice of b for this truncation probability to be smaller than 10^{-6}.

The sampling of integer i is drawn from a geometric distribution that is mirrored at 0. The non-negative part of the distribution's PDF matches the PDF of a geometric distribution (of parameter $p = 1 - e^{-\lambda}$, where $\lambda = \frac{r\epsilon}{\Delta_r}$) that is shifted to the left by 1 and scaled accordingly. Let $2^{b'} = \frac{2^b}{r}$. The probability for the scaled non-negative part ir to be truncated at 2^b is

$$\begin{aligned}
\delta = \Pr[ir \ge 2^b] &= \Pr[i \ge 2^{b'}] \\
&= 1 - \Pr[i < 2^{b'}] \\
&= 1 - \Pr[i \le 2^{b'}] + \Pr[i = 2^{b'}] \\
&= 1 - (1 - (1-p)^{2^{b'}}) + (1-p)^{2^{b'}-1}p \\
&= (1-p)^{2^{b'}-1} \\
&= e^{-\lambda(2^{b'}-1)} \le e^{-\lambda(2^{b'})}
\end{aligned}$$

We need $\lambda > 6\ln(10)2^{-b'} = 6\ln(10)2^{-b}r$ such that $\delta \le 10^{-6}$ (in a similar fashion as the implementation of [50]). Given chosen values for ϵ, Δ (and hence λ and r that depend on ϵ and Δ), to maintain the truncation probability of $\delta \le 10^{-6}$ it suffices to choose the smallest possible b such that

$$b > \left\lceil \log_2 \left(\frac{r6\ln(10)}{\lambda} \right) \right\rceil$$

DPrio uses b to truncate the value ir at 2^b.

C GENERALIZING DPRIO

DPrio is optimal for a scenario with a large number of clients, very few of which are malicious or controlled by the adversary. We can generalize our construction to consider various adversarial structures. To do so, we present an optimization problem which, given privacy parameters ϵ, δ, a utility constraint, and an adversarial structure, returns an optimal setup to achieve (ϵ, δ)-DP in the most efficient manner. We generalize DPrio by combining it with Server-DP and Client-DP. That is, there are three points at which a party can add noise to the function being computed:

- Clients add Gaussian noise using the mechanism described by Client-DP, each with parameter σ_1;
- Clients and servers cooperate to choose Gaussian noise sampled by c clients ($c \ge 1$), as in DPrio, with parameter σ_2;
- Servers sample Gaussian noise, as in Server-DP, each with parameter σ_3.

[1]https://github.com/google/differential-privacy/blob/main/java/main/com/google/privacy/differentialprivacy/LaplaceNoise.java

Let x denote the number of clients controlled by the adversary and y denote the number of servers controlled by the adversary. Depending on x and y, we can choose the optimal combination of the above three options such that the overall guarantees of all the noise satisfies (ϵ, δ)-DP to such an adversary. The composed noise in the case of an adversary who controls x clients and y servers is

$$\sigma_1^2 \cdot (\text{\# of honest clients}) + \sigma_2^2 \cdot c \cdot \mathbb{I} + \sigma_3^2 \cdot (\text{\# of honest servers})$$

where $\mathbb{I}$ denotes an indicator function which is equal to 1 if x or y is non-zero but not both, i.e. the adversary does not control a combination of clients and servers. Otherwise, $\mathbb{I} = 0$. That is, the noise is

$$\sigma_1^2 \cdot (N - x) + \sigma_2^2 \cdot c \cdot \mathbb{I}[(x \neq 0) \oplus (y \neq 0)] + \sigma_3^2 \cdot (n - y).$$

Continuing in line with Section 7.1.2, we assume an upper bound, m, on the number of clients who collude or are controlled by the adversary. Here, this value is encompassed by x. To achieve (ϵ, δ)-DP, the total noise should be greater than $f^2\Delta^2/\epsilon$, where $f^2 > 2\ln(1.25/\delta)$.

To a data analyst who does not collude with anyone (the best case), the total error in the noisy aggregate by this construction is then

$$\sigma_1^2 \cdot N + \sigma_2^2 \cdot c + \sigma_3^2 \cdot n.$$

Minimizing this best case error function for a given worst case privacy constraint (ϵ, δ) is formalized as follows:

$$\begin{aligned}\min_{\sigma_1,\sigma_2,\sigma_3} \; & \sigma_1^2 \cdot N + \sigma_2^2 \cdot c + \sigma_3^2 \cdot n \\ \text{s.t. } & \sigma_1^2 \cdot (N - x) + \sigma_2^2 \cdot c \cdot \mathbb{I}[(x \neq 0) \oplus (y \neq 0)] \\ & + \sigma_3^2 \cdot (n - y) \geq 2\ln(1.25/\delta)\Delta^2/\epsilon^2\end{aligned}$$

for some given parameters N, n, x, y, which determine the adversarial structure. In this generalized setting, we can adapt our construction to adversaries which collude, submit malformed data, or submit malformed noise. We do not consider the possibility of Byzantine failures on the part of servers. Of course, if a client fails to submit data, their information will simply not be included in the result.

Example C.1. *Suppose $x = 0$ and $y = n - 1$, i.e., the adversary controls all but one honest server and no clients. Then, we need to minimize $\sigma_1^2 \cdot N + \sigma_2^2 \cdot c + \sigma_3^2 \cdot n$ subject to*

$$\sigma_1^2 \cdot N + \sigma_2^2 \cdot c + \sigma_3^2 \geq 2\ln(1.25/\delta)\Delta^2/\epsilon^2$$

Solutions which minimize this function are valid according to the optimization function; however, we can narrow down our options by making some observations about the adversarial structure. Given that most of the servers are assumed to be in control of the adversary, we do not want to instruct potentially compromised servers to sample and add their own noise to the result. So, we let $\sigma_3 = 0$. On the other hand, if all clients are honest, we can use either Client-DP or DPrio (with $c = 1$) to include an optimal amount of noise in the result. In this setting, DPrio is more robust against client failures, i.e., if a single client fails (e.g. adds zero noise), Client-DP does not ensure DP, whereas DPrio still does with a high probability. Hence, the optimal choice in this setting is to let $\sigma_1 = 0$ as well and revert to the DPrio solution.

Example C.2. *Suppose $x = N - 1$ and $y = 0$, i.e., the adversary controls all but one client and no servers. Then, approaching the protocol with the Client-DP solution would clearly be non-optimal, and hence we set $\sigma_1 = 0$. If the adversary is semi-honest, then DPrio or Server-DP or some combined of the two are acceptable. If the adversary may actively change the clients' noise, the probability that an honest client's noise is chosen in DPrio by the servers would be very small, unless c was disproportionately large. So, in this case it is optimal to let $\sigma_1 = \sigma_2 = 0$, and revert to the Server-DP solution.*

Example C.3. *Suppose $x \approx N/3$ and $y \approx n/3$. Then we optimize $\sigma_1^2 \cdot N + \sigma_2^2 \cdot c + \sigma_3^2 \cdot n$ subject to*

$$\sigma_1^2 \cdot 2N/3 + \sigma_3^2 \cdot 2n/3 \geq 2\ln(1.25/\delta)\Delta^2/\epsilon^2$$

In this scenario we do not want to use DPrio because there is a 1/3 chance that a dishonest client's noise is selected each time a noise value is chosen and the adversary can determine the value of the noise. This would violate the DP requirement. Hence, we have $\sigma_2 = 0$. The remaining options are to add noise via Client-DP, Server-DP, or some combination of the two.

Depending on the value of N and n, it may be optimal to have only servers or only clients add noise if there is a large number of malicious parties that may submit incorrect noise. On the other hand, we could have clients sample some noise, perform range checks to ensure that the noise falls within a reasonable range, and then have servers sample their own noise to add to the result, ultimately distributing the responsibility between the two types of honest parties. Adding noise in such a way ensures that even in the face of collusion between some clients and servers, the adversary cannot determine the true output by excluding their own noise. Any solution that minimizes the objective function above according to the privacy constraint would provide optimal utility.

Example C.4. *Suppose $x \geq 1$ is a small constant and $y = n - 1$. Then, we should not use Client-DP so we let $\sigma_1 = 0$. We want to minimize $\sigma_2^2 \cdot c + \sigma_3^2 \cdot n$ subject to*

$$\sigma_2^2 \cdot c + \sigma_3^2 \geq 2\ln(1.25/\delta)\Delta^2/\epsilon^2.$$

In this example, if x is small then DPrio is optimal because it is still robust, as discussed in Section 7.2. The value of c also depends on our assumption on x. We require that $c \geq x + 1$ to ensure a DP guarantee in DPrio. If $c < n$ then DPrio is preferred. Otherwise, when $c \geq n$, Server-DP is the better approach. For greater robustness, one might also consider a combination of the two approaches, although this would decrease the accuracy of the output.

Depending on the given constraints and the adversarial model, we can choose a combination of noise addition approaches. The fact that Client-DP, Server-DP, and DPrio are each compatible with one another makes it simple to combine the approaches and cater to different needs. The flexibility of these constructions and the ability to adapt to different scenarios makes this work unique.