`dpsa4fl`: Differential Privacy for Federated Machine Learning with Prio

Olivia Röhrig Maxim Urschumzew

November 7, 2023

Abstract

We present `dpsa4fl`, a framework for differentially private federated learning with secure aggregation using the Prio protocol. It enables training a neural network in a decentralized setting with differential privacy for data owners without disclosing sensitive data to any protocol participant or requiring trust in a single server. Given an appropriate threat model, it also provides resistance to malicious clients attempting data poisoning attacks by leveraging distributed input validation.

1 Introduction

Data privacy is one of the factors motivating *federated learning* (FL). A machine learning model can be trained locally on computers owned by data owners (*clients*), who exchange only intermediate training results (*gradients*), instead of the data itself, with a central server that aggregates them into a shared model [28]. While this relieves the need to collect possibly sensitive data in a central location to perform the training, the shared model and the gradients themselves contain enough information to enable reconstruction of private information [33, 8].

A solution to this problem is using a modified training algorithm that provides *differential privacy* (DP) for the model [1]. The technique adds a small amount of noise to the result of each training step, calibrated in a way that makes high certainty statements about the presence of an individual point in the training dataset unlikely to be inferable from the model.

Adding noise locally would require a large amount of noise and hence large utility loss to achieve privacy, while globally adding noise after the training step would require some party to have access to the non-noised aggregate.

Secure aggregation (SA) is a technique for multiple participants to compute a function on their confidential data that reveals only the computation result without disclosing any other information about the inputs to anyone. This can be leveraged to enable adding the noise after aggregating intermediate results, but keep the individual contributions and the non-noised aggregate secret (*server DP*, [25]).

Contributions. We present `dpsa4fl`, a system which provides server DP with secure aggregation for federated learning. Our privacy guarantees cover the training result, as well as all intermediate aggregation and computation steps. We employ the prio protocol [15] for secure aggregation.

We implement `dpsa4fl` as a set of wrapping layers which make Prio usable from federated learning pipelines. In order for them to be usable for machine learning, we have to add support for gradient vector types, in particular with a verified L2-norm bound. We use `flower` [17] as our target federated learning framework. Concretely, we did the following:

- We designed a "Prio type" for aggregation of gradient vectors with bounded L2-norm for `libprio-rs` [19], a rust implementation of the Prio protocol.
- We added functionality for server differential privacy in `libprio-rs` and `janus` [18], a rust implementation of the server infrastructure required for Prio.
- We developed supporting infrastructure for integrating `janus` into a federated learning pipeline.
- We built bindings to use our system from the `flower` federated learning framework.

All code is available as open source, see the respective repositories[1] for more documentation. In this paper we present the overall architecture as well as a privacy and utility analysis.

Privacy and correctness guarantees. Our system requires two aggregation servers that do not collude with each other or clients, one of which must honestly follow protocol. All other participants, namely the clients and a server coordinating federated model updates, can be malicious. We assume that a client interested in their own data's differential privacy executes the protocol correctly. In this setting, we can guarantee anonymity (no adversary can tell which client submitted which data value) and privacy (no adversary learns anything about any honest clients' data values except the differentially private aggregate) as well as differential privacy for all information exchanged about an honest client between all participants as well as the final training result.

[1]The core library is `dpsa4fl` [36].
Python bindings are provided in `dpsa4fl-bindings.py` [37].
The integration with `flower` is provided in `dpsa4flower` [32].

If we can assure both aggregation servers to honestly follow protocol, even while curious, our system is robust towards data poisoning attacks by ensuring differential privacy even for malicious clients and hence limiting the influence any single client can have on the aggregation result.

Related work. There is a variety of existing work centered around subsets of $\{\mathrm{DP}, \mathrm{FL}, \mathrm{SA}, \mathrm{SA} \text{ with input validation}\}$ with varying architectures and threat models. The most closely related is DPrio [25] by Keeler et al., who propose a different mechanism for adding differential privacy to the Prio protocol. They thus have a very similar threat model and input validation capabilities as our approach. A substantial difference is that in their protocol, the noise to be added for differential privacy is generated by the clients, and the aggregation servers are merely tasked with selecting a subset of clients whose noise they include in the aggregate. This means that their privacy guarantee depends on the fraction of honest clients, while our guarantee only requires one out of two aggregators to be honest.

There is a series of papers [6, 23, 2, 14] based on the original SecAgg [9] protocol by Bonawitz et al. They follow the explicit goal of providing differential privacy specifically for FL tasks. SecAgg is a protocol with a single-server architecture, which means that the threat model, and thus the trade-offs in protocol design, are quite different from the ones used in the present work. Additionally, SecAg currently does not integrate input validation and hence does not protect against malicious clients.

Approaches to secure aggregation with input validation other than Prio are based on *ring learning with errors*, such as RoFL [26] and ACORN [7]. While in RoFL, the input validation results in an 48x increase in communication costs, ACORN is successful in reducing the overhead to 2–10x. Unfortunately, with these being single-server architectures as well, it is not immediately clear how differential privacy could be added in a way that the aggregator server does not see the non-noised sum of gradients. Protection against inference attacks such as [33] hence can only be guaranteed if the server is honest.

An approach which integrates differential privacy but not input validation is FLDP [35] by Stevens et al. They achieve an overhead of 1.7x.

Finally, with all mentioned work being mostly theoretical papers, the options for actually running real-world FL tasks seem to be few:

- There are building blocks for both secure aggregation [4] (based on SecAgg) and for local or global differential privacy [3] in `flower`. But simply putting them together is not sufficient if the `flower` server is not trusted: In case of global DP, the server has access to the non-noised aggregate. In case of local DP, an adequate amount of privacy implies a very high utility loss.
- The authors of DPrio provide source code[2] for their empirical evaluation in terms of a simulation of FL, but there currently is no integration into

[2] https://github.com/DPrio-PoPETs/dprio

an actual FL framework.

- In a similar vein, the authors of e.g. [23] provide source code[3] which is based on TensorFlow Federated [20]. Unfortunately this framework is currently only usable for simulations as well.

2 Preliminaries

2.1 Federated Learning

Federated learning is a machine learning technique introduced by McMahan et al. [29], aiming to avoid centralized data collection by a single party performing the training. Instead, clients obtain a copy of the global model and train it on their private data locally. Model updates are then collected by a central party (*FL server*) and aggregated into an update for the global model. The updated global model is distributed to the clients again and the procedure is repeated until model performance is satisfactory.

While removing the need for central data collection, the above scheme still requires clients to disclose the model updates they derived from their private data, which contain enough information for reconstruction attacks like the ones detailed in [33, 8]. It seems that differential privacy combined with secure aggregation is the best approach to preserve the clients' privacy in this setting. See [24] for an extensive survey.

2.2 Secure Aggregation and Input Validation

Secure aggregation allows for the aggregation of secret distributed data in a way that reveals only the aggregate result. More concretely, let $x_1, \dots, x_n$ be private values, each located at one of n clients. The goal of SA is to compute a given function $f(x_1, \dots, x_n)$ in a way that every participant learns at most as much about the input values $x_1, \dots, x_n$ as can be deduced mathematically from only knowing the result of f. [15]

A topic related to SA is input validation. In some cases, it might be required that the data submitted by clients fulfills certain properties, such as lying within certain numerical bounds. Since the aggregating servers do not see the submissions in plaintext, they cannot easily check whether this is the case. Solutions usually involve zero-knowledge proof protocols [10], where aggregation servers or clients interact with each other in order to convince themselves of the validity of the data—without revealing it.

2.3 Differential Privacy

Differential privacy is a method of enabling the publication of information derived from a dataset while making it unlikely that information about individual entries of the dataset can be inferred. There are many flavours of differential

[3] https://github.com/google-research/federated/tree/master/distributed_dp

privacy, most of which can be converted into one another. A good introduction for programmers can be found here [30].

We define the variant we will be using throughout the paper, following [11].

Definition 1 (Zero-concentrated Differential Privacy) *A randomized function $M : \mathcal{X} \to Y$ is ρ-zero-concentrated differentially private if for all $X, X' \in \mathcal{X}$ differing in a single entry and all $\alpha \in (1, \infty)$,*

$$D_\alpha(M(X) \| M(X')) \leq \rho \cdot \alpha$$

where $D_\alpha(M(X) \| M(X'))$ is the α-Renyi divergence between the distributions of $M(X)$ and $M(X')$.

The Renyi divergence is a way of measuring dissimilarity between distributions. Intuitively, the output distributions of a ρ-zero-concentrated differentially private function given two input datasets that differ in one entry will be harder to tell apart the smaller ρ. This means that one is unlikely to be able to tell whether an individual entry was an element of the input dataset when observing only the outputs of M.

One way of achieving differential privacy for a deterministic function f is by adding noise drawn from a specific distribution calibrated to the *sensitivity*

$$\Delta f = \max\{\|f(X) - f(X')\|_2\}$$

of the function, where the maximum is taken over all datasets X, X' differing in a single entry. For example, for an integer-valued function $f : \mathcal{X} \to \mathbb{Z}$ the randomized function

$$M(X) = f(X) + n \text{ where } n \sim \mathcal{N}_\mathbb{Z}\left(0, \frac{(\Delta f)^2}{2\rho}\right)$$

is ρ-zero-concentrated differentially private [12, Proposition 1.6], where $\mathcal{N}_\mathbb{Z}(\mu, \sigma^2)$ denotes the discrete Gaussian distribution [12].

Local vs. global differential privacy In a distributed setting, it is relevant to consider at what point in the protocol the noise gets added.

Adding noise client-side (*local* differential privacy) does not require entrusting a central party with the non-noised information and the correct execution of the noising. However, each client needs to add the amount of noise required for their privacy guarantee, while it would be sufficient to add that amount once to the sum of all gradients after aggregation.

The usual relief for this is adding noise on a central server (*global* differential privacy). This requires trust in the noising party, as they get access to the non-noised aggregate and need to do the noising faithfully.

Another approach, which becomes viable in conjunction with some forms of secure aggregation, is to perform noising on the aggregation servers Prior to performing their computation on the encrypted client inputs, so even the aggregators never see the non-noised aggregate gradient (this technique has been coined Server DP [25]).

It is still more noise added than necessary in order to distribute the trust: each server adds the full amount of noise necessary for all clients' guarantees, allowing for all but one aggregator to maliciously deviate from protocol without compromising differential privacy. However, with Server DP the noise scales with the number of aggregation servers (of which only two are required, if the threat model allows it), while with local DP it scales with the number of clients.

Robustness Differential privacy not only protects from inference attacks, but also limits the influence a single gradients' value can have on the training result. This is a viable defense against *data poisoning attacks* by malicious clients attempting to sabotage or sway training results in their favour [22, 27, 38]. Note that providing differential privacy for clients that don't have an interest in having differential privacy is not trivial in a setting where secure aggregation is used. A usual approach to bounding the sensitivity of many functions is bounding the inputs — something that the noising parties cannot easily do if they don't have access to the plaintext values. Input validation is a relief, but in our case comes with additional trust requirements.

3 Secure aggregation with Prio

The Prio protocol has been designed by Corrigan-Gibbs and Boneh in [15] as a way to securely aggregate client data with two aggregation servers. As long as one server remains honest, the data remains private. One outstanding feature of prio is that the validity of the submitted data can be verified by the aggregators without them having access to the data itself (*input validation*). This however requires *both* servers to be honest.

The original Prio protocol has since been improved, and, under the name of Prio3, integrated into the more general framework of *verifiable distributed aggregation functions* [16] (VDAFs). It is currently being implemented in the rust library `libprio-rs` [19], in tandem with ongoing specification effort [5]. VDAFs are meant to be executed using the *distributed aggregation protocol*, currently being implemented in `janus` [18], with specification ongoing in [21].

For our federated learning system we use Prio3 as implemented in `libprio-rs` and `janus`. Nevertheless, for our explanations in the present paper we mostly refer to the more compact presentation of the original protocol in [15].

3.1 Prio architecture

Prio uses a *secret sharing scheme* [15, Step 1 of scheme on page 3], where each client splits their gradient into a sum of random vectors and sends one summand to each server over an encrypted channel. Each server sums the shares it received and publishes the resulting *aggregate share*, which gets summed with the other servers' aggregate shares to obtain the aggregation result. If at least one of the servers is honest-but-curious, no party gains knowledge of all secret shares (and hence the plaintext value) of any clients' gradient.

3.2 Input validation

The Prio protocol provides *secret-shared non-interactive proofs* [15, Section 4] to allow the servers to jointly validate the input data, without disclosing any clients' submissions. To do so, the required validation criteria have to be encoded as arithmetic circuits. Correct validation requires all servers to follow the validation protocol honestly, but does not disclose any information about honest clients even if all but one server deviate.

For `dpsa4fl`, we use Prio's input validation to ensure that all submitted gradient vectors have L2-norm less than 1.

3.3 Deployment model

The Prio protocol is successful in disentangling the role of the *data analyst* from the role of the *data aggregator*. The party interested in collecting and analyzing data is obviously in the business of extracting valuable information from that data, and thus is incentivized to disrespect individual users' privacy. Prio allows the task of data aggregation to be delegated to third parties, and thus makes it impossible for the data analyst to extract more information from the data than they publicly claim to do. From the client's perspective, splitting the trust between multiple aggregation servers (and thus multiple parties) makes the system especially trustworthy. As long as there is reason to believe that the aggregators do not collude, the data remains private.

The architecture of `janus` allows for a multitude of aggregation tasks to be handled by a single server instance. It thus naturally lends itself to be offered as "Aggregation-as-a-Service". This means that for a use case such as ours, the FL stakeholder does not have to deal with setting up Prio infrastructure. Instead, the whole point of Prio is that the aggregation servers are provided by different parties.

4 The `dpsa4fl` system

The `dpsa4fl` system provides differentially private aggregation for FL in a modular way. It is designed to be an alternative aggregation mechanism which can be plugged into existing FL frameworks. The `dpsa4fl` [36] core library enables the use of the Prio protocol for aggregation of gradient vectors in the context of FL. It is currently specialized to work with the `janus` implementation of Prio. We provide python bindings in the package `dpsa4fl-bindings.py` [37], and an easy-to-use integration with the `flower` framework in the package `dpsa4flower` [32].

4.1 Architecture

A distributed FL system using `dpsa4fl` consists of two parts (figure 1): the original participants as envisioned by the FL framework, i.e., a server and multiple clients, and two Prio aggregation servers executing the Prio protocol. To differentiate the FL server from the aggregation servers we call it the *controller*.

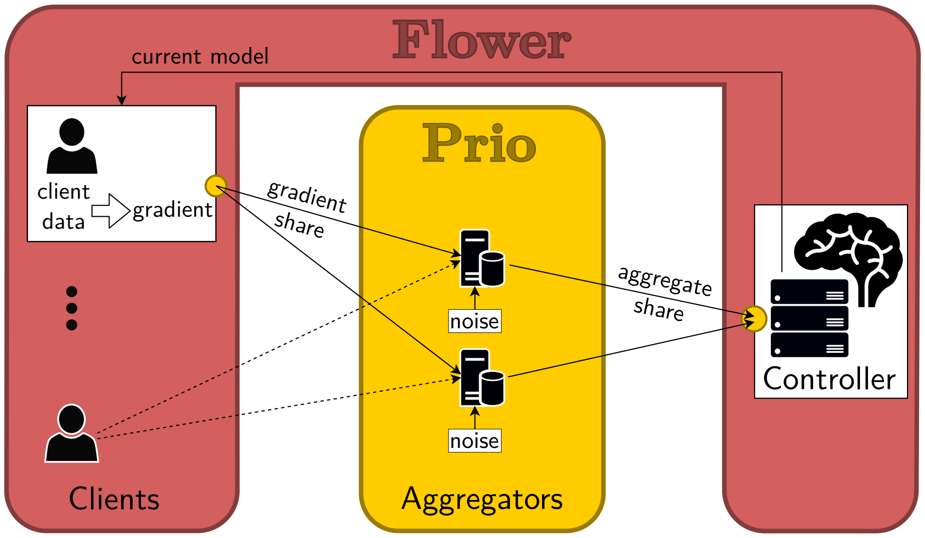

Figure 1: Schematic architecture of an FL system using `dpsa4flower`

The FL framework remains responsible for all functionality of the training process, except for gradient aggregation. This includes broadcasting of the current model, selecting clients for training, and applying the aggregated gradients to the model.

The aggregation servers execute the Prio protocol, thereby computing the sum of the clients' gradients, without the need to disclose individual gradients to anyone. They also verify that all individually submitted vectors have L2-norm less than one. Finally, before revealing the aggregated gradient to the controller, each aggregation server adds noise to their own aggregate share, in order to ensure differential privacy.

The `dpsa4fl` stack provides the glue code for making this all work.

4.2 Integration with an FL framework

We begin by describing how `dpsa4fl` is intended to be used by FL frameworks, using `flower` as the example target framework.

Before starting the training process, the controller notifies both aggregators that it's starting a training session with a given id. The aggregators store session related information under this id. No further `dpsa4fl`-specific setup has to be done. In each round of training, the following sequence of actions happens:

1. `dpsa4fl`: The controller notifies both aggregators that it's starting a training round, and receives a task id under which it can collect the aggregation result.
2. `Flower`: The controller selects available clients for this round and broadcasts its model as well as the task id to them.
3. `Flower`: The clients train their copy of the model on their local training data and compute the overall gradient vector.
4. `Flower`: They use `flower`'s aggregation method to notify the controller that they have succeeded with training, but don't transmit the gradient.
5. `dpsa4fl`: They split and encode the gradient vector according to the Prio protocol and submit each part to an aggregation server under the given task id.
6. `Prio`: The aggregation servers aggregate reports from all clients, verify them, add noise for differential privacy and compute the sum of all gradient vector shares.
7. `dpsa4fl`: The controller collects the aggregate gradient vector via its task id.
8. `Flower`: It applies the gradient to the global model, and begins the next round.

4.3 Configuration

There are two parameters for `dpsa4fl` which users must be aware of: the bit-length b of the fixed-point encoding of the gradient weights and the privacy parameter ρ of zCDP (Definition 1).

For transmission with the Prio protocol, the gradient entries are encoded as b-bit fixed-point numbers. We only support $b \in \{16, 32, 64\}$, this should be enough for all use cases. Choosing a higher bit-length means that there is less utility-loss due to rounding, but also increases message sizes. For most applications it is advisable to use the same bit-length as the respective floating-point representation of their training algorithm.

Choosing the privacy parameter ρ depends on the particular use case and on the amount of privacy which the clients expect to be provided. For N training rounds, our system guarantees $(N \cdot \rho)$-zero-concentrated differential privacy.

5 Threat model and guarantees

The controller and the clients have different goals, so their threat models have to be considered separately. The parties running the aggregation servers should be impartial to the learning process, but of course might deviate from that behaviour. We consider the perspectives of the clients and of the controller below.

Clients. The clients are interested in participating in an FL-scheme without their private information becoming known by other parties. There are two ways this might happen:

1. The submitted gradient vector of a client (for any training round) might become known to other parties. This would allow them to infer properties of the clients' local training set.

2. Access to the trained model makes it possible to make inferences about the combined training set. Combination with other publicly available data can lead to leakage of individual clients' data.

Our system guarantees the following: As long as at least one aggregation server follows the protocol (i.e., is honest-but-curious), the client's data remains private.

Attack (1) is mitigated by the secure aggregation mechanism of Prio. Each aggregation server receives only a share the clients' gradient vectors. It is impossible to reconstruct the original vector from this share alone. All processing on the aggregation server, i.e. the verification, aggregation and noising, happens entirely on its shares. This means that, as long as the aggregation servers don't collude, they don't have access to individual gradients.

Attack (2) is mitigated by the fact that the sum of gradient vectors is never revealed by itself, but only together with the noise added by both aggregation servers. As long as at least one aggregation server adds correctly configured noise, this makes the revealed value differentially private. Differential privacy mitigates inference attacks [34, 31, 13].

In practice this means: clients can be sure that their data is safe as long as they trust one of the aggregation servers to be honest in their execution of the protocol. No trust in the controller is required.

Controller. The controller is interested in training an ML model on the data of the clients. Since it cannot inspect the individually submitted gradients, this introduces the possibility of data poisoning attacks: malicious clients can try to influence the learning process by submitting exaggerated gradient vectors.

The input validation capability of Prio mitigates this sort of attack. As part of the aggregation process, the aggregators verify that each submitted gradient vector has an L2-norm less than 1. This means that while clients can still submit "false" data, they cannot gain disproportionate leverage by doing so. Prio can only guarantee this as long as the aggregation servers do not collude with malicious clients, as in that case it would be easy to craft submissions to fool the other aggregation server into believing that reports are well-formed, even though they are not [15].

In practice this means: the controller can be sure that it receives gradients based on real data, as long as it trusts *both* aggregation servers to execute the protocol honestly, and clients do not collude to collectively poison the model.

6 Privacy analysis

Integrating Prio via `dpsa4fl` into an FL framework as described above results in a distributed implementation of the usual differentially private gradient descent [1], with the differences being the integer encoding of the gradient vector to enable secret sharing according to the Prio protocol [15], the aggregation happening on a different machine than the training itself, and the use of discrete Gaussian noise on the integer encoding of the gradient aggregate.

Algorithm 1 Client procedure

```
Input: Training dataset $X$, loss function $\mathcal{L}(\theta; x)$, number of rounds $N$, fixed-point encoding bit length $b$, set of aggregator server addresses S
Output: Model $\theta$ trained on dataset $X$ for $N$ rounds
 1: $\mathbf{round}_{(b,0)}(x) = \mathbf{sign}(x) \cdot \mathbf{floor}_b(|x|)$    ▷ round to $b$ bits towards zero
 2: $\pi(x) = 2^{b-1} \cdot (x+1)$    ▷ project fixed-point $x \in (-1,1)$ to an integer $\leq 2^b$
 3: for $N$ do
 4:     $\theta$ = retrieve current shared model from controller
 5:     $\mathbf{g} = \frac{1}{|X|} \sum_{x \in X} \nabla \mathcal{L}(\theta; x)$    ▷ compute model gradient average
 6:     $\mathbf{g}_{clip} = \mathbf{g}/\mathbf{max}\{1, ||\mathbf{g}||_{L_2}\}$    ▷ clip $\mathbf{g}$ to $L_2$ norm 1
 7:     $\mathbf{g}_{fixed} = \mathbf{map}(\mathbf{round}_{(b,0)}, \mathbf{g}_{clip})$    ▷ round to get $b$-bit fixed-point vector
 8:     $\mathbf{g}_{int} = \mathbf{map}(\pi, \mathbf{g}_{fixed})$    ▷ project to integer vector
 9:     send one secret share of $\mathbf{g}_{int}$ to each of the aggregation servers in S
10: end for
11: $\theta$ = retrieve current shared model from controller
12: return $\theta$
```

The behaviour of client and server are represented by Algorithm 1 and 2. We claim that executing these with non-colluding aggregation servers, one of which is honest but curious, and an arbitrary number of potentially malicious clients, provides $(N \cdot \rho)$-zero-concentrated differential privacy for any honest clients' dataset. If both servers are honest, the protocol also protects against data poisoning attacks by malicious clients.

Privacy We perform the privacy analysis from the point of view of one client executing one iteration of the loop starting at line 3 of Algorithm 1 faithfully. We show that:

1. The function mapping the clients' dataset to the gradient whose shares are transferred to the aggregation servers is 2^b-sensitive.

2. No information about the gradient except its secret shares is revealed to any party prior to noising.

Algorithm 2 Aggregator server procedure

```
Input: Set of client addresses C, set of aggregator server addresses S, privacy
    parameter $\rho$, fixed-point encoding bit length $b$
Output: Aggregate gradient, noised to be $\rho$-zero-concentrated differentially
    private
 1: $\mathbf{verify}_{\texttt{S}}(\mathbf{x})$ = perform Prio protocol with the other aggregators in S to verify
    $\mathbf{x}$ is a share of a fixed-point vector with $L_2$ norm $\leq 1$ with entries projected
    to the integers $\{0, ..., 2^b\}$ using $\pi$
 2: $\mathbf{decrypt}_{\texttt{S}}(\mathbf{x})$ = perform Prio protocol with the other aggregators in S to
    combine their aggregate shares with $\mathbf{x}$ and obtain the aggregation result
 3: $\mathbf{noise}(x) = x + n$ where $n$ is drawn from a discrete Gaussian distribution
    $\mathcal{N}_{\mathbb{Z}}\left(0, \frac{2^{2b}}{2\rho}\right)$
 4: $\pi'(y) = 2^{1-b} \cdot y - |\texttt{C}|$          ▷ project integer $\leq 2^b \cdot |\texttt{C}|$ to float
 5: $\mathbf{G} = 0$
 6: for all clients $\texttt{c} \in \texttt{C}$ do
 7:     $\mathbf{g}$ = retrieve gradient share from c
 8:     $\mathbf{verify}_{\texttt{S}}(\mathbf{g})$
 9:     $\mathbf{G} = \mathbf{G} + \mathbf{g}$
10: end for
11: $\mathbf{G}_{noised} = \mathbf{map}(\mathbf{noise}, \mathbf{G})$          ▷ add noise componentwise
12: $\mathbf{G}_{agg} = \mathbf{decrypt}_{\texttt{S}}(\mathbf{G}_{noised})$          ▷ combine shares to aggregate
13: $\mathbf{G}_{float} = \mathbf{map}(\pi', \mathbf{G}_{agg})$          ▷ map to float vector
14: send $\mathbf{G}_{float}$ to controller
```

3. The noise added on the aggregation server provides ρ-zero-concentrated differential privacy for the aggregation result.

It follows that the execution of Algorithm 1 provides $(N \cdot \rho)$-zero-concentrated differential privacy by the composition property [11, Lemma 1.8 on page 7].

1. We want to determine the sensitivity in the dataset X of one iteration of the loop starting at line 3 of Algorithm 1. Assuming the current shared model θ retrieved in line 4 was computed in a way that provides differential privacy for X, its use will not affect the sensitivity. The clipping in line 6 ensures that

$$\|\mathbf{g}_{fixed}\|_2 = \|\overline{\mathbf{round}}_{(b,0)}(\mathbf{g}_{clip})\|_2 \leq \|\mathbf{g}_{clip}\|_2 \leq 1.$$

Denote by $\mathbf{g}'$ the values of the variables in the algorithm obtained by replacing any one entry in X by a different entry, and by $\overline{\pi}$ the componentwise application of the function π. We have

$$\begin{aligned}\|\mathbf{g}_{int} - \mathbf{g}'_{int}\|_2 &= \|\overline{\pi}(\mathbf{g}_{fixed}) - \overline{\pi}(\mathbf{g}'_{fixed})\|_2 \\ &\leq 2^{b-1} \cdot \|\mathbf{g}_{fixed} - \mathbf{g}'_{fixed}\|_2 \\ &\leq 2^{b-1} \cdot (\|\mathbf{g}_{fixed}\|_2 + \|\mathbf{g}'_{fixed}\|_2) \\ &\leq 2^b\end{aligned}$$

One iteration of the computation inside the loop (lines 4 to 8) therefore is 2^b-sensitive in X.

2. Prio secret sharing amounts to the client splitting their secret into summands, and recovering the aggregated shares amounts to summing the sums of secret shares [15, Scheme on page 3]. Doing so does not affect the sensitivity of $\mathbf{g}_{int}$, as the result of the computation is equal to the result obtained by summing all clients' gradients without secret sharing. Share transfer happens over an encrypted channel [15, Step 1 of Scheme on page 3], combining the shares happens after adding noise on the server in line 11. Therefore if at least one of the servers is honest and they don't collude, the only things visible to anyone eavesdropping or participating are single secret shares of the gradient and the noised aggregate.

3. Noise is added prior to combining the aggregate shares to avoid disclosing the non-noised aggregate to a server. Since combining them simply means computing their sum [15, Step 3 of Scheme on page 3], the value of $\mathbf{G}_{agg}$ does not depend on whether the noise is added before or after combining the shares. We can hence view the function mapping one clients' data set X to the non-noised gradient aggregate as a 2^b-sensitive function over the integers, so adding noise drawn from $\mathcal{N}_{\mathbb{Z}}\left(0, \frac{2^{2b}}{2\rho}\right)$ will provide ρ-zero-concentrated differential privacy due to [12, Theorem 14 on page 15, with all $\sigma_j = \frac{2^{2b}}{2\rho}$]. Mapping the integer encoding of the aggregate back to a float vector and using it in further training will not affect privacy due to post-processing invariance. Note that each server adds the entire amount of noise required to obtain the guarantee, so if at least one of them is honest the guarantee holds. This comes at the cost of adding the entire noise on each server, resulting in worse utility.

Robustness In line 8 of Algorithm 2, the servers perform a secret-shared non-interactive proof protocol [15, Section 4] together to verify that the client-side clipping was performed properly. This ensures the sensitivity is as described above, even for gradients submitted by malicious clients, and hence protects from clients attempting to influence the computation result disproportionately. The verification part of the protocol requires all servers to act honestly, so our system offers this protection only under a weaker threat model. The above proof of privacy still holds for all clients that execute the protocol truthfully.

7 Evaluation

7.1 Utility analysis

We want to know how much noise our algorithm adds compared to the standard gradient descent with differential privacy obtained by sampling from the regular normal distribution [1]. There, each iteration of gradient aggregation adds noise drawn from $\mathcal{N}\left(0, \frac{(2C)^2}{2\rho}\right)$ for gradients clipped to norm C ($C = 1$, in our case) and our notion of ρ-zero-concentrated differential privacy.

Consider rounding both gradient entries and noise to obtain b-bit fixed point numbers with one integer bit. Rounding a gradient introduces an error

$$|\mathbf{round}_{(0,b)}(x) - x| \leq \frac{1}{2^{b-1}}$$

to each of the clients' gradient entries in each aggregation step. This should be negligible given that the noise added to each aggregate entry is sampled from a normal distribution with standard deviation of $\sqrt{\frac{2}{\rho}}$, meaning that even for large values of ρ, say $\rho = 2^6$, and small values of b, say $b = 16$, still 99.88% of the noise samples will be an order of magnitude larger than that error.

Next consider applying the projection π to both gradients and noise before applying the noise, and reversing the projection afterwards. This does not alter the magnitude of the noise in the end result, and is equivalent to sampling instead from a scaled rounded Gaussian distribution $\mathcal{N}_{round}\left(0, \frac{2^{2b}}{2\rho}\right)$.

Next consider using the discrete Gaussian instead. [12, Corollary 17] shows that utility is strictly better by a small amount.

Last, consider that our protocol simply is a modification of the regular gradient descent algorithm with differential privacy as described in this section, but the aggreagtion and addition of noise is distributed among multiple servers $\mathtt{S}$. As described in our threat model, in order to require trust in only one of those servers, each server has to add the entire amount of noise, so noise magnitude worsens by a multiple of $|\mathtt{S}|$. The sum of discrete Gaussian variables is distributed closely Gaussian [23, Theorem 11], so utility of our algorithm with ρ-ZCdp is approximately that of the regular gradient descent with $|\mathtt{S}| \cdot \rho$-ZCdp. If all aggregators are trustworthy, the protocol can be adapted so each only adds a fraction of the total noise to improve utility while maintaining the privacy bound.

7.2 Communication complexity

The communication costs of transmitting the clients' gradients in a differentially private way compared to transmitting plaintext data is an important metric for evaluating aggregation protocols for FL. Since gradients are typically large, their transmission accounts for a large portion of the time complexity of the whole learning procedure.

			Clear	ACORN-robust	RoFL	RoFL	`dpsa4fl`
Client comm.			ℓ	$\ell + \log^2(n)$	$\ell + n$	$\ell + n$	ℓ
n	ℓ	γ					
10^2	10^4	0.05	0.04MB	0.365MB	2MB	2.9MB	3MB
10^3	2^{18}	0.05	1.05MB	1.35MB	51MB	75MB	85MB
10^3	2^{18}	0.33	1.05MB	14MB	51MB	75MB	85MB
10^3	2^{18}	1	1.05MB		51MB	75MB	85MB
Norm bound			✗	L_∞	L_∞	L_2	L_2
Server DP				✗	✗	✗	✓

Figure 2: Empirical comparison of communication costs with RoFL and ACORN-robust. Values are for transmission of ℓ-length, 32bit fixed-point gradient vectors, n clients, and robustness against a fraction of γ malicious clients. The servers are assumed to be honest but curious.

In the Prio protocol, the message size is mostly determined by the size of the encoded gradient together with the size of the encoded proof of its well-formedness. Such a message must be sent only once from each client. Other messages sent as part of the Prio protocol are negligible.

In our current implementation of `dpsa4fl` we use a naive encoding of gradient entries as field elements. This results in a rather large blow-up of message size. Currently, a single 16-bit fixed point number is encoded by 16 128-bit field elements, which means that our messages are at least 128 times the size of plaintext gradients. There are both trivial and more complex methods to improve on this: the field size of 128-bit can be reduced to 64-bit while still supporting gradients with up to $2^{34} \approx 17 \cdot 10^9$ entries. Furthermore, the encoding can be compactified, such as not to require a seperate field element for each transmitted bit. Since this will increase the size of the correctness proof, a balance which optimizes message size has to be found.

Current state of the art methods for SA with input validation which provide comparable guarantees as Prio are RoFL [26] and ACORN-robust [7]. Both can validate norm bounds on the submitted gradient vectors and exclude invalid submissions from the aggregate. While RoFL provides both L_2 and L_∞ bounds, ACORN-robust only supports L_∞. Additionally, ACORN-robust has a parameter γ which describes the fraction of invalid submissions against which it is robust. Lower values for γ achieve smaller message sizes, but provide less robustness.

As figure 2 shows, our unoptimized encoding is quite close to the L_2 variant of RoFL. Additionally, `dpsa4fl` does not require any client-client communications. Thus our message complexity only depends on the size of the gradient vector, making it preferable for applications with a large number of clients. Also, neither RoFL nor ACORN have explicit provisions for DP, which means that while both global or local DP could be used with their aggregation methods, server DP with its much better privacy-utility tradeoff is not available.

8 Conclusion

The previous section concerning message complexity illustrates that while `dpsa4fl` is almost on par with existing methods, message sizes are more than one order of magnitude higher compared with plaintext gradients. This shows that `dpsa4fl` is not yet ready for real-world deployment, but the present state is promising nevertheless. The `libprio-rs` and `janus` libraries were comparably easy to extend, and the threat model provided by Prio seems to be strong and yet pragmatic enough for it to gain widespread adoption. Integration with the `flower` framework opens possibilities for convenient use in existing applications. Ideas for approaches to minimize message sizes exist, and are left for future work.

Acknowledgements

This project[4] was funded through the *NGI Assure Fund*[5], a fund established by *NLnet*[6] with financial support from the European Commission's *Next Generation Internet*[7] program, under the aegis of DG Communications Networks, Content and Technology under grant agreement No 957073.

References

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H. Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*. ACM, oct 2016. doi: 10.1145/2976749.2978318. URL https://doi.org/10.1145%2F2976749.2978318.

[2] Naman Agarwal, Peter Kairouz, and Ziyu Liu. The skellam mechanism for differentially private federated learning, 2021.

[3] Flower authors. Differential privacy wrapper classes in flower. https://flower.dev/docs/framework/explanation-differential-privacy.html, 2023.

[4] Flower authors. Secure aggregation protocols in flower. https://flower.dev/docs/framework/contributor-ref-secure-aggregation-protocols.html, 2023.

[4] https://nlnet.nl/project/dist-mech-learn
[5] https://nlnet.nl/assure
[6] https://nlnet.nl
[7] https://ngi.eu

[5] R. Barnes, C. Patton, and P. Schoppmann. Verifiable distributed aggregation functions. Internet-Draft draft-irtf-cfrg-vdaf-03, 2023. URL https://datatracker.ietf.org/doc/draft-irtf-cfrg-vdaf/07/. work in Progress.

[6] James Bell, K. A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. Cryptology ePrint Archive, Paper 2020/704, 2020. URL https://eprint.iacr.org/2020/704. https://eprint.iacr.org/2020/704.

[7] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. Acorn: Input validation for secure aggregation. Cryptology ePrint Archive, Paper 2022/1461, 2022. URL https://eprint.iacr.org/2022/1461.

[8] Franziska Boenisch, Adam Dziedzic, R. Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. When the curious abandon honesty: Federated learning is not private. *ArXiv*, abs/2112.02918, 2021.

[9] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 1175–1191, 2017.

[10] Dan Boneh, Elette Boyle, Henry Corrigan-Gibbs, Niv Gilboa, and Yuval Ishai. Zero-knowledge proofs on secret-shared data via fully linear pcps. Cryptology ePrint Archive, Paper 2019/188, 2019. URL https://eprint.iacr.org/2019/188. https://eprint.iacr.org/2019/188.

[11] Mark Bun and Thomas Steinke. Concentrated differential privacy: Simplifications, extensions, and lower bounds. *CoRR*, abs/1605.02065, 2016. URL http://arxiv.org/abs/1605.02065.

[12] Clément L. Canonne, Gautam Kamath, and Thomas Steinke. The discrete gaussian for differential privacy. *CoRR*, abs/2004.00010, 2020. URL https://arxiv.org/abs/2004.00010.

[13] Junjie Chen, Wendy Wang, and Xinghua Shi. Differential privacy protection against membership inference attack on machine learning for genomic data, 08 2020.

[14] Wei-Ning Chen, Christopher A Choquette Choo, Peter Kairouz, and Ananda Theertha Suresh. The fundamental price of secure aggregation in differentially private federated learning. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 3056–3089. PMLR, 17–23 Jul 2022. URL https://proceedings.mlr.press/v162/chen22c.html.

[15] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 259–282, USA, 2017. USENIX Association. ISBN 9781931971379.

[16] Hannah Davis, Christopher Patton, Mike Rosulek, and Phillipp Schoppmann. Verifiable distributed aggregation functions. Cryptology ePrint Archive, Paper 2023/130, 2023. URL https://eprint.iacr.org/2023/130. https://eprint.iacr.org/2023/130.

[17] Flower developers. Flower: A friendly federated learning framework. https://github.com/adap/flower, 2023.

[18] Janus developers. Experimental implementation of the distributed aggregation protocol (dap) specification. https://github.com/divviup/janus, 2023.

[19] Prio developers. Implementation of prio in rust. https://github.com/divviup/libprio-rs, 2023.

[20] Tensorflow federated developers. Tensorflow federated. https://github.com/tensorflow/federated, 2023.

[21] T. Geoghegan, C. Patton, E. Rescorla, and C.A. Wood. Distributed aggregation protocol for privacy preserving measurement. Internet-Draft draft-ietf-ppm-dap-07, 2023. URL https://datatracker.ietf.org/doc/draft-ietf-ppm-dap/07/. work in Progress.

[22] Sanghyun Hong, Varun Chandrasekaran, Yiğitcan Kaya, Tudor Dumitraş, and Nicolas Papernot. On the effectiveness of mitigating data poisoning attacks with gradient shaping, 2020.

[23] Peter Kairouz, Ziyu Liu, and Thomas Steinke. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning*, 2021.

[24] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, et al. Advances and open problems in federated learning, 2021.

[25] Dana Keeler, Chelsea Komlo, Emily Lepert, Shannon Veitch, and Xi He. Dprio: Efficient differential privacy with high utility for prio. *Proceedings on Privacy Enhancing Technologies*, 2023:375–390, 07 2023. doi: 10.56553/popets-2023-0086.

[26] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. Rofl: Robustness of secure federated learning, 2023.

[27] Yuzhe Ma, Xiaojin Zhu, and Justin Hsu. Data poisoning against differentially-private learners: Attacks and defenses, 2019.

[28] H. B. McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *International Conference on Artificial Intelligence and Statistics*, 2016.

[29] H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data, 2023.

[30] Joseph P. Near and Chiké Abuah. *Programming Differential Privacy*, volume 1. 2021. URL https://uvm-plaid.github.io/programming-dp/.

[31] Md.Atiqur Rahman, Tanzila Rahman, Robert Laganière, and Noman Mohammed. Membership inference attack against differentially private deep learning model. *Trans. Data Priv.*, 11:61–79, 2018. URL https://api.semanticscholar.org/CorpusID:13699042.

[32] Olivia Röhrig and Maxim Urschumzew. Server and client to use the flower framework for differentially private federated learning with secure aggregation. https://github.com/dpsa4fl/dpsa4flower, 2023.

[33] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 3–18, 2017. doi: 10.1109/SP.2017.41.

[34] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. Membership inference attacks against machine learning models, 2017.

[35] Timothy Stevens, Christian Skalka, Christelle Vincent, John Ring, Samuel Clark, and Joseph P. Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. *ArXiv*, abs/2112.06872, 2021.

[36] Maxim Urschumzew and Olivia Röhrig. Differentially private, secure aggregation for federated learning. https://github.com/dpsa4fl/dpsa4fl, 2023.

[37] Maxim Urschumzew and Olivia Röhrig. Python bindings for the dpsa4fl library. https://github.com/dpsa4fl/dpsa4fl-bindings.py, 2023.

[38] Chulin Xie, Yunhui Long, Pin-Yu Chen, and Bo Li. Uncovering the connection between differential privacy and certified robustness of federated learning against poisoning attacks, 2022.