Secure Stateful Aggregation: A Practical Protocol with Applications in Differentially-Private Federated Learning

Marshall Ball
New York University

James Bell-Clark
Google

Adria Gascon
Google

Peter Kairouz
Google

Sewoong Oh
University of Washington
Google

Zhiye Xie
NYU Shanghai

Abstract

Recent advances in differentially private federated learning (DPFL) algorithms have found that using *correlated* noise across the rounds of federated learning (DP-FTRL) yields provably and empirically better accuracy than using independent noise (DP-SGD). While DP-SGD is well-suited to federated learning with a single untrusted central server using lightweight secure aggregation protocols, secure aggregation is not conducive to implementing modern DP-FTRL techniques without assuming a trusted central server. DP-FTRL based approaches have already seen widespread deployment in industry, albeit with a trusted central curator who provides and applies the correlated noise.

To realize a fully private, single untrusted server DP-FTRL federated learning protocol, we introduce *secure stateful aggregation*: a simple append-only data structure that allows for the private storage of aggregate values and reading linear functions of the aggregates. Assuming Ring Learning with Errors, we provide a lightweight and scalable realization of this protocol for high-dimensional data in a new security/resource model, *Federated MPC*: where a powerful persistent server interacts with weak, ephemeral clients. We observe that secure stateful aggregation suffices for realizing DP-FTRL-based private federated learning: improving DPFL utility guarantees over the state of the art while maintaining privacy with an untrusted central party. Our approach has minimal overhead relative to existing techniques which do not yield comparable utility. The secure stateful aggregation primitive and the federated MPC paradigm may be of interest for other practical applications.

1 Introduction

The widespread use of deep learning on user-generated data, that is often sensitive, has made privacy-preserving techniques increasingly important. One prominent framework that has emerged for conducting privacy-preserving machine learning is differentially-private federated learning (DPFL).

Differentially-private federated learning framework. While the exact details of differentially-private federated learning protocols may vary widely, many such systems [McMahan et al.(2018)] follow a similar architecture:

- A single central party plays the role of conductor for the learning process, grouping users into cohorts and facilitating communication. We refer to this persistent and powerful party as *the server*, but it need not be localized onto a single device.
- Lightweight, ephemeral client devices (such as phones) are grouped into "cohorts". We refer to such these parties as *clients*.

When a client's cohort's time comes, the device downloads the current model state and uses the on-device data to compute a local update (in gradient descent, this involves computing local gradients). These local updates are then aggregated across the entire cohort with some noise. The server then uses this noisy aggregate to update the model.

With this communication architecture and resource allocation in mind, we introduce a new practical model for designing secure multi-party computation at scale: *secure federated multiparty computation (FMPC)*. This model/paradigm can be seen as a hybrid of two emerging trends in secure multiparty computation: combining ephemeral, stateless participants (Fluid MPC [Choudhuri et al.(2021)], YOSO [Gentry et al.(2021)]) with a powerful persistent central (untrusted) party (Gulliver MPC [Alon et al.(2024)]).

Learning with independent noise (differentially-private stochastic gradient descent) and secure aggregation. Early successes in DPFL were due to a learning framework known as *differentially-private stochastic gradient descent (DP-SGD)* [Song et al.(2013), Bassily et al.(2014), Abadi et al.(2016), Bonawitz et al.(2019)]. DP-SGD learners perform stochastic gradient descent, but at each step, they clip gradients to bound their influence and add independent Gaussian noise to the gradient update to preserve privacy *throughout* the learning process. These algorithms exhibited favorable privacy/utility tradeoffs and were easy to adapt to the federated learning architecture using lightweight protocols for *secure aggregation* [Bonawitz et al.(2017), Bell et al.(2020), Bell et al.(2023a), Ma et al.(2023), Karthikeyan and Polychroniadou(2024)].

A secure aggregation protocol enables a server to learn a sum of vectors and nothing else. Practical secure aggregation protocols are characterized by the ability to scale with very high dimensional data and massive numbers of participants (in contrast to generic secure multiparty computation): they should have very limited interaction (and a simple, sparse communication pattern), almost no communication overhead for high dimensional inputs (relative to the privacy-free baseline of sending inputs in the clear), low computational complexity, and robustness to client dropouts.

Given a secure aggregation protocol, one can turn DP-SGD into a federated learning protocol via the following:

1. The server distributes the (differentially-private) state of the model to the current client cohort.
2. Clients locally compute a gradient update add some locally sampled noise[1] and securely aggregate the result across the entire cohort.
3. The server then uses the aggregated gradient to update the model. Because the output of the aggregation is differentially-private (due to the local noise contributions), the new model also preserves differential privacy.

A number of examples of secure aggregation protocols exist in the literature [Bonawitz et al.(2017), Bell et al.(2023b), Ma et al.(2023), Li et al.(2023)] and indeed been proposed for deploying DP-SGD-based federated learning at scale.

The use of such techniques has been refined in a series of papers [Chen et al.(2022a), Chen et al.(2022b), Kairouz et al.(2022), Agarwal et al.(2021)] and this work has seen production deployment [Hartmann and Kairouz(2023)].

Unfortunately, due to the less than optimal[2] privacy/utility tradeoff of DP-SGD, such federated learning procedures can yield underwhelming accuracy guarantees. [Abadi et al.(2016), Tramer and Boneh(2020), Kairouz et al.(2021), Xu et al.(2023), Choquette-Choo et al.(2024), Choquette-Choo et al.(2023)]

A new paradigm for private learning: correlated noise (differentially-private follow-the-regularized-leader). In recent years, a new paradigm for private learning has emerged. At a very high level, it has been observed both provably and empirically that by adding *correlated noise* in the training steps, the utility

[1]The local noise contributions need not be wide enough to provide privacy on their own: they need only provide privacy in aggregate. This is critical to yielding useful utility/accuracy guarantees at scale.

[2]DP-SGD utility can be improved using privacy amplification via sampling or shuffling, but these techniques are infeasible in the federated learning setting where data arrives in an arbitrary order.

can be dramatically improved while preserving the same level of privacy *throughout* the training process. In particular, in such mechanisms, the noise added at different training steps is *not* independently sampled.

This new family of algorithms, differentially-private follow-the-regularized-leader (DP-FTRL), is similar to DP-SGD but instead of adding independent noise η_i to the gradient in round i, instead adds $\langle \boldsymbol{\lambda}^i, \boldsymbol{\eta} \rangle$ where $\boldsymbol{\lambda}^i$ is a public vector (associated with round i) and $\boldsymbol{\eta}$ is a vector of independent noise samples.

While we shall give a simple example illustrating how such correlated noise can help improve privacy/utility trade-offs later (see 1.1), for now, observe that the straightforward template for securely realizing DP-SGD via secure aggregation does not work here. Critical to that implementation was the fact that noise samples were independent in each learning step and, hence, amenable to local sampling by clients before aggregation.

To apply the DP-FTRL paradigm, the underlying noise vector $\boldsymbol{\eta}$ must persist throughout the entire learning process across the life of many cohorts, and it is not clear how to efficiently do this with secure aggregation alone. To date, no one has successfully realized private federating learning via such an approach without leveraging untenable trust assumptions on the central server.

We introduce a new primitive, *secure stateful aggregation*, that enables a seamless realization of the DP-FTRL approach to differentially private federated learning. We provide a simple, scalable secure stateful aggregation protocol in the federated multiparty computation setting.

1.1 Our Results

We begin by introducing our conceptual contributions: the secure stateful aggregation functionality and the federated multiparty computation model. Then, we will sketch our stateful aggregation protocol and illustrate its applicability with a simple example: computing private partial sums.

Secure Stateful Aggregation. Secure stateful aggregation is a reactive functionality that can be thought of as a simple append-only data structure with two operations:

1. Store: Appends the sum of current inputs to the data structure state.
2. Reveal: Outputs a linear function of the current data structure state to the server.

The actual functionality rolls these two operations together, but for the sake of clarity we provide this equivalent, albeit less efficient presentation. We refer the reader to Section 2 for further details on the functionality and Figure 1 in particular.

A secure realization of this functionality reveals nothing beyond its input/output behavior. The state of the data structure and any aggregated inputs will remain private, up to the linear functions that are revealed.

It is easy to see that this functionality is a mild generalization of the secure aggregation functionality. Moreover, this functionality allows for the secure aggregation of data across many cohorts. Moreover, this pared-down formulation allows for extremely efficient and nonetheless suffices for powerful applications in federated learning.

The (γ, β)-secure federated MPC paradigm. As mentioned above, we introduce a new paradigm for designing MPC protocols that aligns closely with many large scale distributed protocol deployments.

Concretely, a federated MPC (FMPC) protocol is broken into a sequence of rounds. A powerful stateful server persists throughout the computation and is capable communicating with all participants. In each round, lightweight ephemeral clients are scheduled to arrive in a cohort. These clients have limited communication and computational capability and can only participate for at most a few rounds, sometimes just one.

Communication takes place on a bulletin board with a PKI: clients can send secure, private messages to clients in the successive cohort, but metadata about messages is visible to all participants (even if their contents are not). However, given the massive number of clients, each client can only send and receive a

few short messages with other clients. While larger messages to (and from) the server are possible, this communication should also be as close to the information-theoretic minimum as possible.

We assume all communication is effectively synchronous.

The adversary can corrupt an γ-fraction of any given cohort in addition to the server. Additionally, a β-fraction of the clients in any given cohort may drop out, failing to complete their roles in the protocol. In this work, we consider a semi-honest adversary making static corruptions and guaranteed output delivery.

The FMPC paradigm can be seen as a hybrid of two emerging trends in MPC: protocols with ephemeral participants (such as Fluid MPC [Choudhuri et al.(2021)] and YOSO [Gentry et al.(2021)]) and protocols with a strong central party and very weak clients (such as GMPC [Alon et al.(2024)]). Federated MPC is comprised of a powerful persistent central party with a massive number of weak ephemeral clients. In contrast to Fluid MPC and YOSO, where minimal interaction is prized above all else, we assume a single persistent party. In contrast to the GMPC, we assume a much more reliable and transparent communication infrastructure for clients, and on the other hand that the clients are shortlived and unreliable.

It is our hope that this loosely-defined paradigm will help bring theory and practice closer together, at least in certain settings.

A lightweight protocol from RLWE. We provide a simple lightweight realization of stateful aggregation in the federated learning setting. Our protocol scales well with high-dimensional data and massive client cohorts.

The key ingredient is a (high-rate) linearly homomorphic secret key encryption scheme that also admits a kind of key homomorphism (enabling distributed encryption and decryption), which we instantiate via the Ring Learning with Errors (RLWE) assumption [Lyubashevsky et al.(2010)].

The high level idea is very straightforward. Throughout, a persistent global secret key is reshared from cohort to cohort using additive secret sharing (reminiscent of DC-nets [Chaum(1988)]).

1. To Store an aggregate: clients encrypt their private inputs using their share of the secret key and the server aggregates the ciphertexts to produce an encryption of the aggregate under the global key and appends the result to its state. This is possible due to the key homomorphism of the scheme.

2. To Reveal a linear function of the secret state: the server homomorphically evaluates the linear function over ciphertexts it is holding. Then the clients use their secret shares to run a distributed decryption of the resulting ciphertext.

Ensuring that this does not inadvertently compromise semantic security of the state (up to the linear function output) is slightly delicate. An elementary committee-based approach is proposed to handle client dropouts. We refer readers to Section 4 for more details.

In our protocol, communication to the server approaches the size of the aggregated elements. Clients send roughly κ^2 messages of length κ to other clients, where $2^{-\kappa}$ is the desired security level. The server is completely silent and hence the protocol is actually secure against a malicious server by default. (We only guarantee security against semi-honest clients.) Precise benchmarking can be found in Section 5.

Computational demands on clients are comparable to the communication costs.

Application 1: Releasing private partial sums (introduction to correlated-noise mechanisms). Let x_i denote the value of the aggregated inputs in cohort i. Consider the task of releasing a differentially-private running sum:

$$x_1, x_1 + x_2, x_1 + x_2 + x_3, \ldots, \sum_{j=1}^{i} x_j, \ldots, \sum_{j=1}^{N} x_j$$

A naive mechanism for doing this (assuming a trusted central curator) is to simply add independent noise (be it Gaussian or Laplacian or otherwise) to each output above, treating each output as a single count mechanism. Then by composition, we can argue that the whole release is differentially private. The downside of this approach is that the input x_1 appears in every single output, which means (according to the

DP composition theorem) that privacy will degrade by a factor of N. This means that to achieve ϵ-privacy we need to use ϵ/N-private noise on each component, which potentially drowns out the partial sums entirely. Note that it is unclear how to realize even this naive mechanism using secure aggregation, but it can be trivially realized using stateful secure aggregation: in each round, clients can store their aggregated inputs and noise samples separately and reveal their aggregatation of all inputs.

A clever mechanism introduced by Dwork et al. [Dwork et al.(2010)] (referred to as Tree Aggregation in the literature) proposes to instead think about a complete binary tree with $x_1, x_2, ..., x_N$ at its leaves. Label each internal node with the sum of the nodes beneath it (so the root is labeled with $\sum_{j=1}^{N} x_j$). Now, simply add independent noise to the label of each node in the tree and release all the noisy labels. To estimate the $i$the partial sum, one can compute a linear function of the noisy labels.[3] Because any input appears in at most $\log_2 N$ labels, to achieve ϵ-privacy one need only use $\epsilon/\log N$ noise for each internal node.

Now, it is easy to see how to create a mechanism that releases this entire noisy tree using secure stateful aggregation. However, recall that the ultimate i partial sum is a linear function, $\boldsymbol{\lambda}^i$, of the noisy internal labels. This estimator is constructed so that the output is equal to the partial sum plus a linear function of all the independent noise in the tree $\boldsymbol{\eta}$:

$$\sum_{j=1}^{i} x_j + \langle \boldsymbol{\lambda}^i, \boldsymbol{\eta} \rangle.$$

From this perspective, we can imagine a new mechanism that samples persistent independent noise $\boldsymbol{\eta}$ (possibly in an offline phase, although it is possible to construct this mechanism so the noise can be generated in an online manner, one new independent component per partial sum), then at step i directly releases the value above. To realize this new mechanism via secure stateful aggregation is again quite straightforward: clients aggregate their inputs and separately aggregate a fresh noise sample. Then, the expression above can be released by computing the appropriate linear function of the state: releasing the value described above.[4]

We can rephrase this last mechanism in linear algebraic terms. Let $\boldsymbol{C}$ be the matrix that maps to cohort input aggregates $\boldsymbol{x} = (x_1, \ldots, x_N)$ to tree labels as described above. Then let $\boldsymbol{B}$ denote the matrix such that the ith row is $\boldsymbol{\lambda}^i$, the linear function that produces the ith partial sum estimator. Then, in this notation our mechanism will output

$$\boldsymbol{B}(\boldsymbol{C}\boldsymbol{x} + \boldsymbol{\eta}) = \boldsymbol{y},$$

where $\boldsymbol{y} = (y_1, \ldots, y_n)$ and $y_i = \sum_{j=1}^{i} x_j + \langle \boldsymbol{\lambda}^i, \boldsymbol{\eta} \rangle$.

Application 2: Differentially-private federated learning via DP-FTRL. Kairouz et al. [Kairouz et al.(2021)] developed a new approach for training models with differential privacy using batch gradients. At a very high level their idea was that for convex optimization, one can bound regret by looking at a linear function of the loss of the model at time i. This means that the next model step can be computed using a linear function of the batch gradients seen thus far, essentially a partial sum of these gradients. Thus training a model privately effectively reduces to building a mechanism for releasing iterative partial sums as in Application 1 above!

Thus we can effectively use the same mechanism, albeit with a different choice of matrices $\boldsymbol{C}$ and $\boldsymbol{B}$, as the one described above to perform state-of-the-art differentially private learning at scale. In particular, we will choose $\boldsymbol{B}$ and $\boldsymbol{C}$ such that $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{C}$ where $\boldsymbol{A}$ is the linear function that maps the sequence of batch gradient described above. A "good" factorization of $\boldsymbol{A} = \boldsymbol{B}\boldsymbol{C}$ yielding nearly optimal privacy/utility tradeoff for this paradigm can be found using semi-definite programming. [Kairouz et al.(2021), Choquette-Choo et al.(2024)] The stateful secure aggregation then need only output

$$\boldsymbol{B}(\boldsymbol{C} + \boldsymbol{\eta}) = \boldsymbol{A}\boldsymbol{x} + \boldsymbol{B}\boldsymbol{\eta}.$$

[3] A straightforward approach simply sums the roots of the any left children on path to ith leaf (and the ith leaf), but better approaches are possible [Honaker(2015), Kairouz et al.(2021)].

[4] The advantage of this alternate approach (over releasing the entire tree), is that the partial sum estimator can be released in one shot as soon as the data is available. Releasing the whole noisy tree, while also possible and private, not only requires many releases at certain times but also requires more releases (leading to a higher cost overall).

At the ith training step, this is simply $\boldsymbol{A}^i\boldsymbol{x} + \boldsymbol{B}^i\boldsymbol{\eta}$ where $\boldsymbol{A}^i, \boldsymbol{B}^i$ are the ith rows of $\boldsymbol{A}$ and $\boldsymbol{B}$, respectively.

This new approach has been shown to yield dramatic improvements over other methods such as DP-SGD approaches, both theoretically [Kairouz et al.(2021), Choquette-Choo et al.(2023)] and empirically [Kairouz et al.(2021), Choquette-Choo et al.(2024)]. Moreover this is already being deployed at scale for private next word prediction in the Google Keyboard [Xu et al.(2023)], albeit with a trusted server.[5]

1.2 Limitations and Future Directions

We suspect that the protocols listed here are maliciously secure, assuming the presence of a PKI and a means of choosing which parties should take part in each round. However, in practice those would be very big assumptions in the presence of a malicious server so we have not prioritized showing this formally. Demonstrating malicious security and finding ways to integrate with realistic ways to resist Sybil attacks would be a useful contirbution.

Our implementation of MF-DP-FTRL requires the matrix $\boldsymbol{C}$ to be banded. In the central model concurrent work has shown that is is possible to work with Buffered Toeplitz matrices [McMahan et al.(2024)] giving slightly better results. That approach however doesn't interact well with discretization, so we cannot straight forwardly extend to it. Finding a way to make that work is a possible future direction.

Of course it isn't obvious that the communication and computation couldn't be reduced by practically meaningful constant factors by some other approach. This could also be a aim of future work.

2 Stateful Aggregation

The system consists of a server and a sequence of r cohorts of clients. The ith cohort C_i is available to make a submission at time step i. It must also know the public keys for C_{i+1} and C_{i+2} at that time, thus C_i must have been chosen and provided their public keys by time $i-2$.

In each step, multiple aggregations can be conducted. Each of these aggregations computes a linear combination of inputs from the clients that step and values stored in the protocol's memory in previous steps. The result of each aggregation can either be stored in memory or revealed to the server. For notational clarity we will assume that exactly one aggregation is conducted per step.

An aggregation instruction has three arguments: one indicating whether the result should be revealed, taking values in $\{\mathsf{Store}, \mathsf{Reveal}\}$; a rule explaining what value each client should provide for aggregation, taking values in a set $\mathcal{I}$; and the weights to be applied to stored values for inclusion in the aggregation which form a value in $\mathbb{F}_q$ for some prime q. An aggregation in the ith round can thus be written as either $\mathsf{Agg}\left(\mathsf{Store}, I_i, \{\lambda_{i,k}\}_{k<i}\right)$ or $\mathsf{Agg}\left(\mathsf{Reveal}, I_i, \{\lambda_{i,k}\}_{k\leq i}\right)$. The input rule I_i can be any object that the clients know how to use to derive their inputs to the specific aggregation, we will write $\boldsymbol{x}_{i,j}$ for the input that the jth client in C_i derives from I_i. A program P for our system consists of a sequence of aggregations, one per round. The ith entry in the sequence being the aggregation for the ith cohort to perform. We say a program of that form is valid if the dependency graph defined by input and outputs of its instructions is acyclic.

We make two simplifying assumptions, neither is a hard restriction but they will keep things simpler and hold for the applications we have in mind. Firstly, we assume that the program is known in advance, although it would be possible to decide the ith entry of the sequence after the $(i-1)$th cohort have spoken. In the malicious server case the ith cohort would have to check that the ith entry of the sequence was in some sense legitimate or the adversary could ask for the submission of secrets it shouldn't learn, but this would be doable for many applications. Secondly, as mentioned above, each cohort conducts at most one Store and one Reveal aggregation. Let v_i be the value stored or revealed in the ith round. We define the following ideal functionality.

[5] User gradients in this process are aggregated securely, yielding some privacy guarantees. However, while the model outputs preserved differentially-privacy to external parties and users, Google's internal view was not differentially-private. By using our secure stateful aggregation, the view of *all* parties remains differentially-private.

Stateful Aggregation Functionality $\mathcal{I}$

Public Parameters:

- A program to be executed $P := \langle t_1, \ldots, t_r \rangle$. Each instruction t_i is of the form $(\mathsf{Mode}_i, I_i, \{\lambda_{i,k}\}_{k<i})$, with $\mathsf{Mode} \in \{\mathsf{Store}, \mathsf{Reveal}\}$.
- Vector length ℓ, input domain $\mathbb{F}^\ell$.

Parties:

- A Server $\mathcal{S}$.
- A sequence of (not necessarily disjoint) cohorts $C_1, \ldots, C_r$ of clients. Each cohort contains n clients. We denote by $C_{i,j}$ the jth client in the ith cohort. Each client holds a private database $D_{i,j}$.
- A trusted party $\mathcal{I}$ holding state $\mathsf{State} := \langle (v_i) \in \mathbb{F}^\ell \rangle_{i \in [\leq r]}$, initially $\langle\rangle$ (empty).

Private Parameters: Client $C_{i,j}$ may hold private input $\boldsymbol{x}_{i,j} := I_i(D_{i,j})$ for round i.

Functionality:

For each round $i \in [r]$:

- Let $(\mathsf{Mode}_i, I_i, \{\lambda_{i,k}\}_{k<i}) = t_i$.
- Each client $C_{i,j}$ sends $\boldsymbol{x}_{i,j}$ to $\mathcal{I}$.
- $\mathcal{I}$ computes $v_i := \sum_{j \in [n]} \boldsymbol{x}_{i,j} + \sum_{k<i} \lambda_{i,k} v_k$.
- $\mathcal{I}$ updates $\mathsf{State} := \mathsf{State} \mid\mid v_i$.
- If $\mathsf{Mode}_{i-1} = \mathsf{Reveal}$ then $\mathcal{I}$ sends v_{i-1} it to $\mathcal{S}$.

Figure 1: The Stateful Aggregation Functionality

Long running aggregation. As a basic example, consider the case where the server just wants to compute the sum of the private input from clients across different round. Let `input` be a rule that just returns the client's input to the server. In this case each cohort except the last one will run $\mathsf{Agg}(\mathsf{Store}, \texttt{input}, \{\mu_{i,k} := 0\}_{k<i})$. The final cohort will run only $\mathsf{Agg}(\mathsf{Reveal}, \texttt{input}, \{\nu_{i,k} := 1\}_{k<i})$.

3 Applications of Stateful Aggregation in Distributed Differential Privacy

In this section, we describe a couple of programs for releasing prefix sums with distributed differential privacy using our system. We have a sequence of cohorts, each client in these cohorts has a vector as input, let the sum of inputs in cohort i be x_i. The server should after each cohort receive an estimate of the sum of all inputs from all clients who have submitted so far, which we call S_i, i.e. after cohort i it should receive an estimate of $S_i := \sum_{j \leq i} x_j$. This is useful in federated learning, where the inputs are user contributed updates to a model and the current parameters are the sum of the initial parameters and all inputs so far.

In both cases the noise will be added by the clients making the submissions and for privacy we will assume that at most a fraction γ of them are corrupt (which is also an assumption for the security of the protocol so this is no extra assumption).

Following a standard trick, when we want a Gaussian random variable with variance α^2 for differential privacy, each client can provide Gaussian noise with variance $\alpha^2/n(1-\gamma)$. The sum of honest contributions will then have the correct variance. For privacy purposes any extra noise added can be considered post-processing. The effect of this approach is to inflate the variance of the added noise by a factor of $1/(1-\gamma)$, this will be small compared to our other gains.

Throughout we assume that each client's input has a bounded sensitivity and that in order to achieve the

required DP epsilon locally the required variance of Gaussian noise (for each coordinate) would be $(1-\gamma)\sigma^2$ to each entry. The choice of σ will in practice depend on sensitivity, γ and ϵ, but the total noise in all of the following methods will depend on those parameters only through σ, thus this will simplify presentation.

3.1 Baselines

If we don't require differential privacy we can have a secure aggregation for each cohort providing the sum x_i of that cohort's inputs. This provides the server with the difference between their output for this round and their output for the previous round, which they would learn anyway and from which they can compute their current output $\sum_{j\leq i} x_i$ as x_i plus their previous output $\sum_{j<i} x_i$.

This extends naturally to the baseline differentially private idea of using a separate secure aggregation with each cohort to produce $x_i + z_i$ where the z_i is a Gaussian noise with variance σ^2. However, the prefix sum that the server outputs in the jth round is $\sum_{i\leq j} x_i + z_i$ which has noise $\sum_{i\leq j} z_i$ which has variance proportional to $j\sigma^2$, we can do better than this.

3.2 Prefix Tree Aggregation

Dwork et. al. [Dwork et al.(2010)] suggested the following DP mechanism that has since come to be known as Tree Aggregation. Suppose we have 2^h cohorts for some integer h. Assign the cohort's inputs x_i to the leaves l_i of the tree from left to right. Assign Gaussian noise samples z_i to each of the left child nodes in the tree and to the root, such that z_i's node n_i has l_i as its rightmost descendant (about half the time $n_i = l_i$). Let $\upsilon(i)$ be the index of the leftmost descendant of n_i. At time step i the curator calculates $r_i = z_i + \sum_{j=\upsilon(i)}^{i} x_j$, that is the true sum of the leaves descended from n_i masked by the noise at n_i. It then adds the output from step $\upsilon(i) - 1$ (or nothing if $\upsilon(i) = 1$) and outputs the result.

As the full sequence of outputs is a post processing of the r_i it suffices to show that the r_i are cumulatively DP. As each input x_i has at most h ancestors amongst the n_i, it is included in at most h of the r_i. By advanced composition it is thus sufficient for each of the noises to have variance $O(k\sigma^2)$. Each output is the sum of at most h of the r_i and thus the variance of the noise on each output is $O(h^2\sigma^2)$ i.e. $O(\log_2(r)^2\sigma^2)$. The constants resulting from this protocol can be optimized at the cost of extra computation was described by Honaker [Honaker(2015)], we will not bother implementing a version of that as it is more complicated (so not interesting for exposition) and superfluous given MF-DP-FTRL.

We now provide a program $P = \langle t_1, ..., t_{2^{h+1}} \rangle$ for our functionality that will implement the above. In this program the odd numbered cohorts will not provide data, only Gaussian noise to be used for DP and the even numbered cohorts will be the cohorts providing the noise, thus x_i will be uploaded in round $2i$.

We remark that there is no reason why the same physical devices couldn't play the role of cohorts $2i-1$ and $2i$, further this would avoid the cost of transferring the key for that change, that would further allow the work for the two rounds to be done in parallel. This is probably how the protocol would be run in practice but we haven't explained the details here to keep the exposition and interface simple.

Let G_{σ^2} be a function that generates and returns a vector in $\mathbb{F}^l$ of discrete Gaussians with variance σ^2/n. In round $2i-1$ we will have the cohort generate and store the noise z_i that is we take $t_{2i-1} = (\mathsf{Store}, G_{\sigma^2}, 0_k)$. In round $2i$ we will have the server learn the x_i plus the noise that we want applied to S_i minus the noise that was applied to S_{i-1}. The server can then add this to its previous output to get the output for round i.

Let 2^{h_i} be the largest power of two dividing i. The difference of the noise for S_i and S_{i-1} is given by $z_i - \sum_{d=0}^{h_i-1} z_{i-2^d}$. We let I map a client's data to the input we want them to provide to the aggregation and define $\lambda_{2i,k} = -1$ if $k \in \{2i - 2^d - 1 | d \in \{1, ..., h_i\}\}$ and $\lambda_{2i,k} = 0$ otherwise. Then the even indexed instructions in our program are given by $t_{2i} = (\mathsf{Reveal}, I_{2i}, \{\lambda_{2i,k}\}_{k<2i})$.

3.3 MF-DP-FTRL

The state of the art in central model DP federated learning is given by the matrix factorization approach [Choquette-Choo et al.(2024)]. We now describe an outline of this procedure and the optimizations provided in that paper.

Define $\boldsymbol{A}$ to be the lower triangular matrix with all entries (on and below the diagonal) equal to one. Note that if $\boldsymbol{x}$ is a vector (of vectors) with the ith cohort's contribution in the ith place then the task we are aiming for is to estimate $\boldsymbol{Ax}$ in a streaming fashion.

The matrix factorization in the name is of $\boldsymbol{A}$ into two components $\boldsymbol{A} = \boldsymbol{BC}$. The calculated result will be given by $\boldsymbol{B}(\boldsymbol{CX} + \boldsymbol{\eta})$ for some Gaussian noise $\boldsymbol{\eta}$. To prove that this is DP it is enough to show that $\boldsymbol{CX} + \boldsymbol{\eta}$ is DP, which is done by requiring $\boldsymbol{C}$ to have Frobenius norm at most one and setting the variance of $\boldsymbol{\eta}$ to be the same as would make $\boldsymbol{X} + \boldsymbol{\eta}$ DP. Thus the factorization is usually chosen to minimise $\boldsymbol{B\eta}$ subject to the bound on the norm of $\boldsymbol{C}$. This optimization is then done numerically and results in substantial practical improvements over the Tree Aggregation idea above.

In practice it is also important that the necessary matrix multiplications can be calculated efficiently in an online fashion. Efficiently here largely means that the server doesn't want to have to store $\Omega(r)$ vectors in memory at any point. To achieve this it is recommended to choose $\boldsymbol{C}$ to be banded with band width b. The result can then be calculated as $\boldsymbol{AC}^{-1}(\boldsymbol{CX} + \boldsymbol{\eta})$ using online algorithms for multiplying by a known banded matrix or its inverse that each require storing only b vectors at any point (these are given by Algorithms 8 and 9 in [Choquette-Choo et al.(2024)]). Adding the restriction that $\boldsymbol{C}$ is bounded is shown numerically to lead to little loss in utility and so this restriction is recommended.

In our protocol we will also use the fact that $\boldsymbol{C}$ is banded to get an efficient protocol in runtime and storage in much the same way. We will add one more restriction on $\boldsymbol{C}$ which is that we require it to be discrete. This is a minimal change because we can discretize at any fixed level of precision. Thus we propose optimizing $\boldsymbol{C}$ over the reals as in the central model and then rounding each entry in $\boldsymbol{C}$ to discrete values. This may increase the Frobenius norm of $\boldsymbol{C}$ to slightly more than 1, if this happens then rounding down some of the entries that were barely rounded up should bring it back down without significantly damaging the fidelity of the approximation. If all rounding is toward 0 then the Frobenius norm will not increase and the fidelity will still be good for sufficiently fine discretizations.

We note that the multiplication by $\boldsymbol{AC}^{-1}$ can be considered post processing and so can be done in the clear. It is enough to implement the online processing of $\boldsymbol{CX} + \boldsymbol{\eta}$ using our system.

Again to have i cohorts provide inputs we will run a program P with $2i$ instructions. The same possibilities for combining these rounds apply as in the Tree Aggregation case. In the Tree Aggregation case where the noise was stored and then applied to each input as it was revealed. In this case we will store the inputs in the odd numbered rounds and then in each even numbered round reveal a new instance of noise with the appropriate linear combination of the inputs on top. The odd indexed instructions are thus $t_{2i-1} = (\mathsf{Store}, I, \{0\})$ and the even ones are $t_{2i} = (\mathsf{Reveal}, G_{\sigma^2}, \{\lambda_{2i,2k-1} = C_{i,k}\}_{k \leq i})$.

The outputs from this Program can then be scaled back from fixed point to floating point encodings and then online multiplied by $\boldsymbol{B} = \boldsymbol{AC}^{-1}$ as in the central model.

4 Realizing Secure Stateful Aggregation

For clarity, we begin by describing a protocol for securely realizing this functionality in the fully-synchronous (or no client dropouts) semi-honest setting, and provide some intuition for its security. (A formal security proof for this protocol can be found in Appendix B) This setting captures the key ideas in realizing stateful secure aggregation. In Section 4.3, we describe how to augment this basic protocol to achieve resilience to client dropouts.

Before continuing, let us recall the stateful secure aggregation functionality. A stateful secure aggregation program consists of a sequence of instructions $t_i = (\mathsf{Mode}_i, I_i, \boldsymbol{\lambda}^i)$ and maintains an append-only data structure whose state at time i we denote $\boldsymbol{v}$.

At time i when instruction t_i is executed, all clients currently present in cohort C_i ($|C_i| = n$) submit their inputs $\boldsymbol{x}_{i,1}, \ldots, \boldsymbol{x}_{i,n}$ to the server. Then the sum of these inputs with a linear function, $\boldsymbol{\lambda}^i \in \mathbb{Z}_q^*$ for some prime q, of the prior state is appended to the state: $\boldsymbol{v} \leftarrow \boldsymbol{v} || v_i$ where $v_i = \sum_{j \in [n]} x_{i,j} + \langle \boldsymbol{\lambda}^i, \boldsymbol{v} \rangle$.

If $\mathsf{Mode}_i = \mathsf{Store}$, then this is all that happens. No output is produced. Otherwise, if $\mathsf{Mode}_i = \mathsf{Reveal}$, then value just appended to the state, v_i, will be released in the next time step.

We will show how to securely implement this functionality in the presence of a semi-honest adversary who can corrupt at most an γ-fraction of any cohort and the central server. We then show how to augment this protocol to achieve robust correctness (and security) guarantees in the presence of a fail-stop adversary who can force corrupt clients to drop out.

4.1 Preliminaries: Linearly Homomorphic Encryption with Distributed Encryption/Decryption via (R)LWE

The key ingredient in our scheme is a simple symmetric key encryption scheme based on learning with errors (LWE) assumptions that admits both key and message homomorphism. In particular, given a key $\boldsymbol{A}, \boldsymbol{s}$ where $\boldsymbol{A}$ can be public, a message $\boldsymbol{x}$ is encrypted as

$$\mathsf{Enc}_{\boldsymbol{A},\boldsymbol{s}}(\boldsymbol{x}) \to \boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e} + \boldsymbol{x}$$

where e is sample from some appropriate small noise distribution and T is an appropriately chosen scalar.

To decrypt a ciphertext $\boldsymbol{c} = \boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e} + \boldsymbol{x}$, one subtracts $\boldsymbol{A}\boldsymbol{s}$ and removes the noise:

$$\mathsf{Dec}_{\boldsymbol{A},\boldsymbol{s}}(\boldsymbol{c}) = \boldsymbol{c} - \boldsymbol{A}\boldsymbol{s} \mod T = (\boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e} + \boldsymbol{x}) - \boldsymbol{A}\boldsymbol{s} \mod T = T\boldsymbol{e} + \boldsymbol{x} \mod T = \boldsymbol{x}$$

The first key property that we will rely on is linear message homomorphism: given encryptions of $\boldsymbol{x}$ and $\boldsymbol{y}$ under the same secret key (but possibly different public keys), one can produce an encryption of $a \cdot \boldsymbol{x} + b \cdot \boldsymbol{y}$ (albeit with respect to a different public key).

$$a \cdot (\boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e} + \boldsymbol{x}) + b \cdot (\boldsymbol{B}\boldsymbol{s} + T\boldsymbol{f} + \boldsymbol{y}) = (a\boldsymbol{A} + b\boldsymbol{B})\boldsymbol{s} + T(a\boldsymbol{e} + b\boldsymbol{f}) + a\boldsymbol{x} + b\boldsymbol{f}$$

So long as the coefficients a and b are appropriately bounded (and hence the noise $a\boldsymbol{e} + b\boldsymbol{f}$ and message $a\boldsymbol{x} + b\boldsymbol{y}$ are not too large), this can be correctly decrypted.

The second key property we rely on is key homomorphism, which enables a form of distributed encryption and decryption. In what follows, imagine Alice and Bob are holding $\boldsymbol{s}_1$ and $\boldsymbol{s}_2$ additive shares of the secret key $\boldsymbol{s}$ such that $\boldsymbol{s}_1 + \boldsymbol{s}_2 = \boldsymbol{s}$.

To compute a distributed encryption of the sum of their inputs ($\boldsymbol{x}_1, \boldsymbol{x}_2$ respectively), Alice can send $\boldsymbol{A}\boldsymbol{s}_1 + \boldsymbol{x}_1 + T\boldsymbol{e}_1$ and Bob can send $\boldsymbol{A}\boldsymbol{s}_2 + \boldsymbol{x}_2 + T\boldsymbol{e}_2$. The server can sum the result to get an encryption of $\boldsymbol{x}_1 + \boldsymbol{x}_2 : \boldsymbol{c} = \boldsymbol{A}\boldsymbol{s} + (\boldsymbol{x}_1 + \boldsymbol{x}_2) + T(\boldsymbol{e}_1 + \boldsymbol{e}_2)$. So long as $\boldsymbol{e}_1 + \boldsymbol{e}_2$ is small (which is the case if $\boldsymbol{e}_1$ and $\boldsymbol{e}_2$ are small, $\boldsymbol{c}$ can later be correctly decrypted.

Now to see how key homomorphism enables distributed decryption, imagine the server is holding a cipher text $\boldsymbol{c} = \boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e} + \boldsymbol{x}$. Now, Alice and Bob can simply compute and send $\boldsymbol{A}\boldsymbol{s}_1$ and $\boldsymbol{A}\boldsymbol{s}_2$ respectively. This enables the server to recover $\boldsymbol{x}$

$$(\boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e} + \boldsymbol{x}) - \boldsymbol{A}\boldsymbol{s}_1 - \boldsymbol{A}\boldsymbol{s}_2 \equiv_T \boldsymbol{x}$$

Unfortunately, this also allows the server to recover $\boldsymbol{e}$, and in turn $\boldsymbol{s}$. While this may be ok in a one-time scenario, we will require a distributed decryption that only reveals "safe" leakage on $\boldsymbol{e}$ (or following the terminology of Lee et al. [Lee et al.(2018), Cheon et al.(2021), Bell et al.(2023b)]: hints about $\boldsymbol{e}$) that won't compromise $\boldsymbol{s}$. Following Bell et al. [Bell et al.(2023b)], we note that semantic security on correlated ciphertexts can be preserved without impinging upon correctness if Alice and Bob add some noise to their messages (effectively sending encryption of 0 using their private keys):

$$(\boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e} + \boldsymbol{x}) - (\boldsymbol{A}\boldsymbol{s}_1 - T\boldsymbol{e}_1) - (\boldsymbol{A}\boldsymbol{s}_2 - T\boldsymbol{e}_2) = \boldsymbol{x} + T(\boldsymbol{e} + \boldsymbol{e}_1 + \boldsymbol{e}_2) \equiv_T \boldsymbol{x}$$

The server now can learn $\boldsymbol{e} + \boldsymbol{e}_1 + \boldsymbol{e}_2$, but as shown in [Lee et al.(2018), Bell et al.(2023b)] this preserves semantic security ($\boldsymbol{u}$ is uniformly random below):

$$(\boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e}, \boldsymbol{e} + \boldsymbol{e}_1) \approx (\boldsymbol{u}, \boldsymbol{e} + \boldsymbol{e}_1)$$

Before continuing, we note that these properties are satisfied by other encryption schemes.[6] However, in our study, the scheme below instantiated with Ring Learning with Errors yielded the best practical parameters.

4.2 Fully-Synchronous Semi-Honest Protocol (No dropouts)

We begin by showing how to securely realize the stateful secure aggregation functionality in the absence of client dropouts. We will informally describe this protocol and give intuition for its security. The formal description of the protocol can be found in Figure 2 and Figure 3. A formal security proof can be found in Appendix B.

The high-level idea of our secure stateful aggregation protocol is relatively straightforward.

A persistent secret key. At the outset, clients in the first cohort, C_1, locally and independently sample uniformly random secret keys, $\boldsymbol{s}_{1,1}, \ldots, \boldsymbol{s}_{1,n}$. This implicitly defines a global secret key $\boldsymbol{s} = \sum_{j=1}^{n} \boldsymbol{s}_{1,j}$. Throughout the protocol, we will maintain the invariant that the clients of any particular cohort are holding an additive secret sharing of $\boldsymbol{s}$.

To do this, we use a simple trick reminiscent of Chaum's dining cryptographers [Chaum(1988)]. If the jth client in cohort C_i is holding a share $\boldsymbol{s}_{i,j}$, that client simply additively shares $\boldsymbol{s}_{i,j}$ into $\boldsymbol{s}^1_{i,j}, \ldots, \boldsymbol{s}^d_{i,j}$ such that $\boldsymbol{s}^1_{i,j}, \ldots, \boldsymbol{s}^d_{i,j}$ are uniform conditioned on $\sum_{k=1}^{d} \boldsymbol{s}^k_{i,j} = \boldsymbol{s}_{i,j}$. Then $C_{i,j}$ sends those shares to d randomly chosen clients in the next cohort. Provided that every honest client sends a message to some other honest client and receives at least one message from some other honest client, $C_{i,j}$'s share $\boldsymbol{s}_{i,j}$ remains perfectly hidden. Clients of the next cohort simply sum up the shares they receive to produce their own share. In particular, if the kth client in cohort $i+1$, receives $\bar{\boldsymbol{s}}^1, \ldots, \bar{\boldsymbol{s}}^{d'}$, then its share of the secret key will be $\boldsymbol{s}_{i+1,k} = \sum_{j=1}^{d'} \bar{\boldsymbol{s}}^j$.

It is easy to verify that the invariant is maintained and so long as d is sufficiently large, no honest party's secret key share will be compromised.

We will use this persistent secret key to encrypt the state of the protocol, $\boldsymbol{v} = (\boldsymbol{v}_i)_{i \le r}$. In particular, the server will hold an ever growing sequence of ciphertexts $\boldsymbol{v} = (\hat{\boldsymbol{v}}_i)_{i \le r}$ where the ith ciphertext is an encryption of $\boldsymbol{v}_i$ relative to a public matrix $\boldsymbol{A}_i$ and the global secret key $\boldsymbol{s}$.

Writing to the secret state. Having established how to maintain private random keys that sum to the same key $\boldsymbol{s}$ at any given time, we next describe a simple mechanism for updating an encrypted state held by the server. Each client j in cohort C_i simply uses their secret key share $\boldsymbol{s}_{i,j}$ to encrypt their input $\boldsymbol{x}_{i,j}$. The clients then send these ciphertexts to the server. The server, holding an encryption of the old state $\boldsymbol{v}$, uses the linearly holomorphic property of the encryption scheme to compute an encryption of $\langle \boldsymbol{\lambda}^i, \boldsymbol{v} \rangle$. The server then simply sums the resulting correlated ciphertext with all ciphertext received from cohort C_i. The result, an encryption of $\langle \boldsymbol{\lambda}^i, \boldsymbol{v} \rangle + \sum_{j=1}^{n} \boldsymbol{x}_{i,j}$, is then appended to its encrypted state.

Revealing parts of the secret state. If $\mathsf{Mode}_{i-1} = \mathsf{Reveal}$, then we will use the distributed decryption property to open the last part of the state. Namely, clients in cohort C_i send messages for distributed decryption of the last part of the state. The server then uses these messages to reveal that part of the state.

Security intuition. Arguing security amounts to proving that these distributed decryptions of homomorphically evaluated ciphertexts preserve semantic security of the underlying ciphertexts sent by clients when writing to the state, up to some linear constraints, even when this is done repeatedly.

We do this via a hybrid argument over the individual messages, but the step is in arguing that opening linear functions of a sequence of ciphertexts is indistinguishable from uniform (up to the outputs of the

[6] For example, our framework can be instantiated with ElGamal (provided one sufficiently constrains the message space—which suffices for our applications). However, this significantly degrades the complexity of communication relative to the RLWE-based approach.

linear functions). We argue this by reducing to a generalization of HintMLWE introduced by Kim et al. [Kim et al.(2023)] (Def. 6[7]) that allows for multiple leaks or hints on the noise:

$$(\boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e}, \boldsymbol{e} + \boldsymbol{f}^1, \ldots, \boldsymbol{f}^\ell) \approx (\boldsymbol{u}, \boldsymbol{e} + \boldsymbol{f}^1, \ldots, \boldsymbol{e} + \boldsymbol{f}^\ell).$$

By a direct reduction (see Corollary 1 for precise statement and details), this implies that for $\boldsymbol{\lambda}^1 \ldots \boldsymbol{\lambda}^\ell$ there exist some noise distributions $\boldsymbol{f}^1, \ldots, \boldsymbol{f}^\ell$ such that

$$\left(\underbrace{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_r}_{\text{public matrices}}, \underbrace{\boldsymbol{A}_1\boldsymbol{s} + T\boldsymbol{e}_1 + \boldsymbol{x}_1, \ldots, \boldsymbol{A}_r\boldsymbol{s} + T\boldsymbol{e}_r + \boldsymbol{x}_r}_{\text{server's encrypted state (aggregated Store messages)}}, \underbrace{\sum_{i=1}^r \boldsymbol{\lambda}_i^1 (T\boldsymbol{f}^1 - \boldsymbol{A}_i\boldsymbol{s}), \ldots, \sum_{i=1}^r \boldsymbol{\lambda}_i^\ell (T\boldsymbol{f}^\ell - \boldsymbol{A}_i\boldsymbol{s})}_{\text{client's distributed decryptions (aggregated Reveal messages)}} \right)$$

$$\approx \left(\underbrace{\boldsymbol{A}_1, \ldots, \boldsymbol{A}_r}_{\text{public matrices}}, \underbrace{\boldsymbol{u}_1, \ldots, \boldsymbol{u}_r}_{\text{uniform}}, \underbrace{\sum_{i=1}^r \boldsymbol{\lambda}_i^1 (T\boldsymbol{f}_i^1 + T\boldsymbol{e}_1 - \boldsymbol{u}_i) + \langle \boldsymbol{\lambda}^1, \boldsymbol{x} \rangle, \ldots, \sum_{i=1}^r \boldsymbol{\lambda}_i^\ell (T\boldsymbol{f}_i^\ell + T\boldsymbol{e}_1 - \boldsymbol{u}_i) + \langle \boldsymbol{\lambda}^\ell, \boldsymbol{x} \rangle}_{\text{simulated decryption aggregation}} \right)$$

The first r components are the public matrices in both distributions. The second r component can be thought of as the server's encrypted state (the sum of the Store messages). The third ℓ components are distributed decryptions (the sum of the Reveal messages).

Critically note that if one considers the function

$$\phi^i : (\boldsymbol{A}_1, \ldots, \boldsymbol{A}_r, \boldsymbol{b}_1, \ldots, \boldsymbol{b}_r, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_\ell) \mapsto \boldsymbol{c}_i + \sum_{j=1}^r \boldsymbol{\lambda}_j^i \boldsymbol{b}_j,$$

then ϕ^i applied to the either distribution yields

$$\sum_{j=1}^r \boldsymbol{\lambda}_j^i \boldsymbol{x}_j + T \left(\sum_{j=1}^r \boldsymbol{\lambda}_j^i (\boldsymbol{f}_j^i + \boldsymbol{e}_j) \right).$$

Thus, provided $\boldsymbol{\lambda}^i$ is appropriately bounded, the server can correctly recover $\sum_{j=1}^r \boldsymbol{\lambda}_j^i \boldsymbol{x}_j$.

On the other hand, the indistinguishability of these two distributions means that (provided the client's messages are securely aggregated) nothing is leaked to the server about the state beyond precisely $\langle \boldsymbol{\lambda}^1, \boldsymbol{x} \rangle, \ldots, \langle \boldsymbol{\lambda}^\ell, \boldsymbol{x} \rangle$. From there is simply a matter of arguing that the aggregation is secure using a hybrid argument (similar to Bell at al. [Bell et al.(2023b)]).

This is summarized in the following theorem:

Theorem 1. *Assuming that a semi-honest PPT adversary corrupts at most an γ-fraction of any user cohort, in addition to the server, The protocol given in Figures 2, 3 securely implements the functionality in Figure 1.*

Full proof of this theorem can be found in Appendix B.

Remark 1. *We note that the communication complexity between clients, the biggest bottleneck, can be improved beyond the naive implementation specified above. Recall that client-to-client communication is comprised exclusively of additive secret sharing of the client's secret. It can be advantageous to choose parameters in the encryption scheme so that the secret key is quite large. However, the communication complexity between parties will then grow accordingly as the size of each share is exactly the size of the secret key, in the naive implementation described above.*

To reduce communication costs, client i holding a secret key share $\boldsymbol{s}_i$ can, instead of additively sharing $\boldsymbol{s}_i$ directly, sample a sequence of independent random seeds $r_1, \ldots, r_d$ that expand (using a PRG) to pseudo-random strings $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_d$. By setting $\boldsymbol{y}^* = \boldsymbol{s}_i - \sum_{j=1}^d \boldsymbol{y}_j$, the client can then send the seeds $\boldsymbol{r}_1, \ldots, \boldsymbol{r}_d$ to

[7] Our definition is slightly different than that of Kim et al. [Kim et al.(2023)]: we consider a variant with uniformly random secrets (as opposed to Gaussian) but do not reveal leakage on the secret.

Server $\mathcal{S}$

Public parameters:

- Clients' public keys (via PKI)
- Vector length ℓ, input domain $\mathbb{F}^\ell$.
- Key sharing parameter d
- Additive secret sharing scheme AShare, threshold secret sharing scheme TShare.
- The ith instruction, $t_i = (\mathsf{Mode}_i, I_i, \{\boldsymbol{\lambda}_k^i\}_{k<i})$, from the program P to be executed
- A means of generating public random matrices A_k indexed by corresponding round k
- Discrete Gaussian distribution D_σ for generating noise

For $i > 1$:

1. Send to C_i:
   - Encrypted shares of the key from C_{i-1}
2. Receive from each client of C_i:
   - Encrypted key resharings for C_{i+1}
   - Receive w_j^i from client j
3. Computation and Output:
   - Aggregate $w^i = \sum_{j\in[n]} w_j^i$
   - If $\mathsf{Mode}_{i-1} = \mathsf{Reveal}$, compute and output the remainder $(w^{i-1} + \sum_{k<i-1} \boldsymbol{\lambda}_k^{i-1} w^k)$ modulo T

Figure 2: Server: No dropout resilience

clients in the next cohort and $\boldsymbol{y}^*$ to the server. The future client j receiving a batch of $\boldsymbol{r}_k$'s will then expand them to $\boldsymbol{y}_k$'s and set their secret key $\boldsymbol{s}_j = \sum \boldsymbol{y}_k$. The server can use $\boldsymbol{y}^*$ to "correct" the output of reveal by subtracting off $\boldsymbol{A}\boldsymbol{y}^*$ from the result, for the appropriately computed $\boldsymbol{A}$.

This results in client-to-client communication that is just $d\kappa$ bits, where κ is the security parameter at the cost of sending an additional $|\boldsymbol{s}|$ bits to the powerful and persistent server.

4.3 Adding Dropout Resilience

We now describe how to augment the simple protocol present above to be resilient to client dropouts.

Recall that security and correctness of the protocol above effectively reduces to maintaining the invariant that at any point in time, the persistent secret key $\boldsymbol{s}$ is safely additively secret shared across the current cohort of clients: clients are holding $\bar{\boldsymbol{s}}_1, \ldots, \bar{\boldsymbol{s}}_n$ that are uniformly random conditioned on $\boldsymbol{s} = \sum_{i=1}^n \bar{\boldsymbol{s}}_i$.

This is maintained by having each client re-share their share to some random subset of the next cohort: client i holding $\bar{\boldsymbol{s}}_i$ samples uniformly random $\bar{\boldsymbol{s}}_i^1, \ldots, \bar{\boldsymbol{s}}_i^d$ subject to $\sum_{j=1}^d \bar{\boldsymbol{s}}_i^j = \bar{\boldsymbol{s}}_i$ and sends each share $\bar{\boldsymbol{s}}_i^j$ to a random client in the next cohort. Client i's secret share $\bar{\boldsymbol{s}}_i$ was in turn the result of summing the shares sent to it by clients in the previous cohort subject to $\bar{\boldsymbol{s}}_i = \sum_{j=1}^{d'} \bar{\boldsymbol{s}}'_j$.

We will augment this maintenance procedure with a simple mechanism to ensure that if any client drops out, their share can be recovered to maintain the persistent global secret key. Some delicacy is required to avoid compromising clients that send *any* sensitive information (even if they are not able to complete a full protocol round).

Client $C_{i,j}$

Public parameters:

- Clients' public keys (via PKI)
- Vector length ℓ, input domain $\mathbb{F}^\ell$.
- Key sharing parameter d
- Additive secret sharing scheme AShare, threshold secret sharing scheme TShare.
- The ith instruction, $t_i = (\mathsf{Mode}_i, I_i, \{\boldsymbol{\lambda}_k^i\}_{k<i})$, from the program P to be executed
- A means of generating public random matrices A_k indexed by corresponding round k
- Discrete Gaussian distribution D_σ for generating noise.

Private input: Possibly an input vector $x_{i,j} := I_i(D_{i,j})$ based on private data $D_{i,j}$.

1. Receive messages from $\mathcal{S}$:
   - Unless $i = 1$, receive encrypted shares of key from C_{i-1}
2. Local computation:
   - If $i = 1$, sample uniformly random $s_{1,j}$ from $\mathbb{F}^\ell$.
   - If $i > 1$, decrypt shares and compute $s_{i,j} \leftarrow \sum_r s_{i-1,j}^r$
   - Compute $(s_{i,j}^1, \ldots, s_{i,j}^d) \leftarrow \mathsf{AShare}(s_{i,j})$
   - If $\mathsf{Mode}_i = \mathsf{Store}$, generate and set $M_i = A_i$. Sample $g^i \leftarrow D_\sigma$.
   - If $\mathsf{Mode}_i = \mathsf{Reveal}$, set $M_i = -\sum_{k<i} \lambda_k^i A_k$. Sample $g^i \leftarrow \sum_{k<i} \lambda_k^i D_\sigma$.
   - Compute $w_j^i = M_i s_{i,j} + T g^i$.
3. Send to the server:
   - Send encrypted $(s_{i,j}^1, \ldots, s_{i,j}^d)$ to d random clients in C_{i+1} via server
   - Upload w_j^i to server

Figure 3: Client j in cohort i: No dropout resilience

Server $\mathcal{S}$

Public parameters:

- Clients' public keys (via PKI).
- Input domain $\mathbb{F}^l$.
- Key sharing parameter d.
- Additive secret sharing scheme AShare, threshold secret sharing scheme TShare.
- The ith instruction, $t_i = (\mathsf{Mode}_i, I_i, \{\lambda_{i,r}\}_{r<i})$, from the program P to be executed
- A list of dropped clients $\mathcal{D}_i$ indexed by the ith cohort
- A means of generating public random matrices A_i indexed by corresponding round i
- A pseudorandom generator $\mathsf{PRG} : \mathbb{Z}_q \to R_q^\ell$.
- A sate Z (initialized to 0) that aggregates missing secret shares.
- Discrete Gaussian distribution D_σ for generating noise.

For $i = 2$ to N:

1. Send to C_i:
   - Encrypted additive shares of key from C_{i-1}
   - **[Dropout Recovery]** Encrypted threshold shares from C_{i-2} that recover the dropped clients in C_{i-1}
   - **[Remove Mask]** Encrypted threshold shares from non-dropped clients of C_{i-1}
2. Receive from each client in C_i:
   - Register dropped clients from C_i in the global list $\mathcal{D}_i$.
   - Encrypted key resharings for C_{i+1}
   - Receive w_j^i from client j for every $j \in [n] \backslash \mathcal{D}_i$
   - Encrypted threshold shares of each additive key share to C_{i+2}
   - Encrypted threshold shares of self-mask secret to C_{i+1}
   - The decryption of all received threshold that server requested
3. Computation and Output:
   - Aggregate $\bar{w}^i = \sum_{j \in [n] \backslash \mathcal{D}_i} w_j^i$
   - Reconstruct missing key shares in C_{i-1}
     - Recover shares $s'_{k,j}$ that client k sent to (dropped) client j for all $k \in [n] \setminus \mathcal{D}^{i-2}, j \in \mathcal{D}^{i-1}$
     - Update $Z \leftarrow Z + \sum_{k,j} s'_{k,j}$
   - Reconstruct self-mask of C_{i-1}
     - Recover $b_{i-1,j}$ for all non-dropped client j in the $(i-1)$th cohort
     - Compute $\mathsf{MASK}_j^{i-1} \leftarrow \mathsf{PRG}(b_{i-1,j})$
   - If $\mathsf{Mode}_{i-1} = \mathsf{Store}$, set $w^{i-1} \leftarrow \bar{w}^{i-1} + \boldsymbol{A}_{i-1} Z - \sum_{j \in [n] \backslash \mathcal{D}_{i-1}} \mathsf{MASK}_j^{i-1}$
   - If $\mathsf{Mode}_{i-1} = \mathsf{Reveal}$, compute and output the remainder $(\bar{w}^{i-1} + \sum_{k<i-1} \lambda_{i-1,k}(w^k - \boldsymbol{A}_k Z) - \sum_{j \in [n] \backslash \mathcal{D}_{i-1}} \mathsf{MASK}_j^{i-1})$ modulo T

Figure 4: Server: With dropout resilience

Our approach is to associate with each client a random committee of *chaperones* in some future cohort. The chaperones for client i will effectively hold threshold secret shares of client i's secret key so that if client i drops out, a quorum of the chaperones can help the server to reconstruct $\bar{\boldsymbol{s}}_i$. The server can then use $\bar{\boldsymbol{s}}_i$ to ensure Reveal outputs are computed correctly (by subtracting off from the result $\boldsymbol{A}\bar{\boldsymbol{s}}_i$ for the appropriate matrix $\boldsymbol{A}$).

To enable the chaperones to do this, every client j who sends a share $\boldsymbol{s}'_j$ to client i (so that client i's share $\bar{\boldsymbol{s}}_i = \sum_{j=1}^{d'} \boldsymbol{s}'_j$) will additionally send *threshold secret shares* of $\boldsymbol{s}'_j$ to a randomly chosen set of chaperones in the cohort after i's.

If client i fails to send *any* message specified by the protocol (recall that we assume all traffic is visible, but not its contents), then these chaperones will release their shares of $\boldsymbol{s}'_j$ enabling the server to recover $\boldsymbol{s}'_j$. Because this happens for all shares $\boldsymbol{s}'_j$ sent to client i, the server can reconstruct $\bar{\boldsymbol{s}}_i$. Provided that the chaperone committees do not have too many corrupted members, client i's share $\bar{\boldsymbol{s}}_i$ will remain perfectly secret.

The problem with what we have sketched so far, is that client i may have sent an encryption of the input $\boldsymbol{x}_i$ under $\bar{\boldsymbol{s}}_i$ before dropping out. Therefore, if the procedure outlined above is followed, $\bar{\boldsymbol{s}}_i$ can be used to decrypt client i's private input $\boldsymbol{x}_i$!

To avoid this, we introduce one last simple mechanism: client i masks their encryption with a pseudorandom mask $\boldsymbol{y}_i$. The short seed for this pseudorandom mask is then threshold secret shared with a random committee of chaperones in the next cohort. If the client successfully sends all messages, i.e. does *not* drop out, then the chaperones release their shares so the server can reconstruct the mask. If the client fails to send all messages as protocol defined, i.e. the client is registered as having dropped out, the chaperones will not release the shares of the mask. Thus, provided there are not too many corrupt chaperones, the mask will remain pseudorandom and ensure the privacy of the client's input $\boldsymbol{x}_i$.

Remark 2. *An alternative approach is to associate a single publicly known committee of chaperones with each client that will be used for all tasks above (instead of choosing a random committee for each underlying message in the dropout-free protocol). This committee could even be the same for all clients if one has good reason to believe that not too many will be corrupted. This may enable simpler implementation, albeit at the cost of making any one such committee easier to corrupt (as the membership is known in advance).*

Additionally, instead of automatically relying on the chaperones to release the mask for a client's encryption of their input, $\boldsymbol{x}_i$. An alternate approach is to ask the client to send that directly in the next round, after checking all their messages were delivered successfully. Only if the client does not stay online would the chaperones need to reconstruct the mask. This way, clients who do not drop out will reduce the overall communication cost.

5 Benchmarking

In this section we discuss concrete parameter selection and performance costs for our protocol. Given that RLWE encryption and decryption very fast in practice using FFT friendly parameters we focus on ciphertext expansion, and overall communication costs for clients.

5.1 Parameter Selection

We use the lattice estimator [Albrecht et al.(2015)] to estimate the hardness of RLWE problem used and set parameters to have at least 128 bits of computational security. The secret distribution is a Gaussian with standard deviation $\sigma_s = \sqrt{2}\sigma$, and the noise distribution used in fresh encryptions is Gaussian with standard deviation $\sigma_n = 2\sigma\sqrt{r+1}$, where $\sigma = 3.2$ is the stdev used to estimate RLWE security, and r is the number of rounds/releases in the protocol. Since we only apply additive homomorphic operations on ciphertexts, we can track the l_∞ norm on the coefficient embedding of the error polynomial, as well as the plaintext.

Security of the our scheme relies on a variant of Kim et al. [Kim et al.(2023)] assumption which proves that by using a uniformly random secret and Gaussian error with standard deviation σ_n as set above, the

n	ℓ	N	q	packing factor	client communication	ciphertext expansion
10^3	10^3	2048	44	1	16.76 KB	8.38x
10^5	10^3	2048	54	1	20.57 KB	10.29x
10^7	10^3	4096	64	1	40.77 KB	20.38x
10^3	10^5	4096	96	3	449.16 KB	2.25x
10^5	10^5	4096	87	2	588.29 KB	2.94x
10^7	10^5	4096	103	2	696.49 KB	3.48x
10^3	10^7	16384	434	16	34.88 MB	1.74x
10^5	10^7	16384	413	12	43.87 MB	2.19x
10^7	10^7	16384	417	10	52.98 MB	2.65x

Table 1: Parameters for some common settings (number of rounds is 1000 in all cases, and input domain is $[2^{16}]^{\ell}$).

resulting aggregation protocol is as secure as standalone HE scheme with $\sigma = 3.2$. Note that the specific choice of $\sigma = 3.2$ as the error distribution for standalone homomorphic encryption schemes is suggested by the Homomorphic Encryption Standard [Albrecht et al.(2018)], and is widely accepted and used in practice.

5.2 Communication costs

Table 5.2 shows some parameters for common settings of the protocol. We find parameters by minimizing the dominant communication costs (bits sent to server) while doing a grid search over secure parameters. It can be observed that as soon as vector length is > 1000 ciphertext packing pays off, and ciphextext expansion stays within small single digits (2-5x). By ciphertext packing we mean encoding several entries of the vectors to be encrypted in the same coefficient of a ciphertext. This gives flexibility when finding parameters as it allows to use the plaintext domain optimally. This explain the large values of N and q in the table. An important optimization well-known in practice is to drop unused coefficients of a ciphertext.

We now turn our attention to the prefix sum application. In Figure 6, we show how per-client communication scales with input vector length, compared to a baseline insecure protocol where plaintexts are submitted in the clear. As shown in the plot, the communication overhead is small even for small ϵ (privacy parameter). The plaintext domain has to account for differentially private noise. As the corresponding distribution is a centered Gaussian, this is not a huge increase over the requirement that the plaintext domain has to fit the sum of n clients' contributions, i.e. n, as every input is a binary vector in this case.

We do not report the client-to-client communication cost as this is (a) comparatively much smaller and (b) invariant regardless of how the encryption scheme is instantiated (see Remark 1).

References

[Abadi et al.(2016)] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.

[Agarwal et al.(2021)] Naman Agarwal, Peter Kairouz, and Ziyu Liu. 2021. The Skellam Mechanism for Differentially Private Federated Learning. arXiv:2110.04995 [cs.LG] `https://arxiv.org/abs/2110.04995`

[Albrecht et al.(2018)] Martin Albrecht, Melissa Chase, Hao Chen, Jintai Ding, Shafi Goldwasser, Sergey Gorbunov, Shai Halevi, Jeffrey Hoffstein, Kim Laine, Kristin Lauter, Satya Lokam, Daniele Micciancio, Dustin Moody, Travis Morrison, Amit Sahai, and Vinod Vaikuntanathan. 2018. *Homomorphic Encryption Security Standard*. Technical Report. HomomorphicEncryption.org, Toronto, Canada.

[Albrecht et al.(2015)] Martin R. Albrecht, Rachel Player, and Sam Scott. 2015. On the concrete hardness of Learning with Errors. *J. Math. Cryptol.* 9, 3 (2015), 169–203.

[Alon et al.(2024)] Bar Alon, Moni Naor, Eran Omri, and Uri Stemmer. 2024. MPC for Tech Giants (GMPC): Enabling Gulliver and the Lilliputians to Cooperate Amicably. In *Advances in Cryptology - CRYPTO 2024 - 44th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2024, Proceedings, Part VIII (Lecture Notes in Computer Science, Vol. 14927)*, Leonid Reyzin and Douglas Stebila (Eds.). Springer, 74–108. https://doi.org/10.1007/978-3-031-68397-8_3

[Bassily et al.(2014)] Raef Bassily, Adam Smith, and Abhradeep Thakurta. 2014. Private empirical risk minimization: Efficient algorithms and tight error bounds. In *2014 IEEE 55th annual symposium on foundations of computer science*. IEEE, 464–473.

[Bell et al.(2023a)] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. 2023a. ACORN: Input Validation for Secure Aggregation. In *USENIX Security Symposium*. USENIX Association, 4805–4822.

[Bell et al.(2023b)] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. 2023b. ACORN: Input Validation for Secure Aggregation. In *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, Joseph A. Calandrino and Carmela Troncoso (Eds.). USENIX Association, 4805–4822. https://www.usenix.org/conference/usenixsecurity23/presentation/bell

[Bell et al.(2020)] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 1253–1269.

[Bonawitz et al.(2019)] Kallista A. Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chloé Kiddon, Jakub Konečný, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *Proceedings of the Second Conference on Machine Learning and Systems, SysML 2019, Stanford, CA, USA, March 31 - April 2, 2019*, Ameet Talwalkar, Virginia Smith, and Matei Zaharia (Eds.). mlsys.org. https://proceedings.mlsys.org/paper_files/paper/2019/hash/7b770da633baf74895be22a8807f1a8f-Abstract.html

[Bonawitz et al.(2017)] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, Bhavani Thuraisingham, David Evans, Tal Malkin, and Dongyan Xu (Eds.). ACM, 1175–1191. https://doi.org/10.1145/3133956.3133982

[Chaum(1988)] David Chaum. 1988. The Dining Cryptographers Problem: Unconditional Sender and Recipient Untraceability. *J. Cryptol.* 1, 1 (1988), 65–75. https://doi.org/10.1007/BF00206326

[Chen et al.(2022a)] Wei-Ning Chen, Christopher A. Choquette-Choo, Peter Kairouz, and Ananda Theertha Suresh. 2022a. The Fundamental Price of Secure Aggregation in Differentially Private Federated Learning. arXiv:2203.03761 [cs.LG] https://arxiv.org/abs/2203.03761

[Chen et al.(2022b)] Wei-Ning Chen, Ayfer Özgür, and Peter Kairouz. 2022b. The Poisson binomial mechanism for secure and private federated learning. arXiv:2207.09916 [cs.CR] https://arxiv.org/abs/2207.09916

[Cheon et al.(2021)] Jung Hee Cheon, Dongwoo Kim, Duhyeong Kim, Joohee Lee, Junbum Shin, and Yongsoo Song. 2021. Lattice-Based Secure Biometric Authentication for Hamming Distance. In *Information*

Security and Privacy - 26th Australasian Conference, ACISP 2021, Virtual Event, December 1-3, 2021, Proceedings (Lecture Notes in Computer Science, Vol. 13083), Joonsang Baek and Sushmita Ruj (Eds.). Springer, 653–672. https://doi.org/10.1007/978-3-030-90567-5_33

[Choquette-Choo et al.(2023)] Christopher A Choquette-Choo, Krishnamurthy Dvijotham, Krishna Pillutla, Arun Ganesh, Thomas Steinke, and Abhradeep Thakurta. 2023. Correlated noise provably beats independent noise for differentially private learning. *arXiv preprint arXiv:2310.06771* (2023).

[Choquette-Choo et al.(2024)] Christopher A Choquette-Choo, Arun Ganesh, Ryan McKenna, H Brendan McMahan, John Rush, Abhradeep Guha Thakurta, and Zheng Xu. 2024. (Amplified) Banded Matrix Factorization: A unified approach to private training. *Advances in Neural Information Processing Systems* 36 (2024).

[Choudhuri et al.(2021)] Arka Rai Choudhuri, Aarushi Goel, Matthew Green, Abhishek Jain, and Gabriel Kaptchuk. 2021. Fluid MPC: Secure Multiparty Computation with Dynamic Participants. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, 94–123. https://doi.org/10.1007/978-3-030-84245-1_4

[Dwork et al.(2010)] Cynthia Dwork, Moni Naor, Toniann Pitassi, and Guy N. Rothblum. 2010. Differential privacy under continual observation. In *Proceedings of the 42nd ACM Symposium on Theory of Computing, STOC 2010, Cambridge, Massachusetts, USA, 5-8 June 2010*, Leonard J. Schulman (Ed.). ACM, 715–724. https://doi.org/10.1145/1806689.1806787

[Gentry et al.(2021)] Craig Gentry, Shai Halevi, Hugo Krawczyk, Bernardo Magri, Jesper Buus Nielsen, Tal Rabin, and Sophia Yakoubov. 2021. YOSO: You Only Speak Once - Secure MPC with Stateless Ephemeral Roles. In *Advances in Cryptology - CRYPTO 2021 - 41st Annual International Cryptology Conference, CRYPTO 2021, Virtual Event, August 16-20, 2021, Proceedings, Part II (Lecture Notes in Computer Science, Vol. 12826)*, Tal Malkin and Chris Peikert (Eds.). Springer, 64–93. https://doi.org/10.1007/978-3-030-84245-1_3

[Hartmann and Kairouz(2023)] Florian Hartmann and Peter Kairouz. 2023. Distributed differential privacy for federated learning. https://research.google/blog/distributed-differential-privacy-for-federated-learning/.

[Honaker(2015)] James Honaker. 2015. Efficient use of differentially private binary trees. *Theory and Practice of Differential Privacy (TPDP 2015), London, UK* 2 (2015), 26–27.

[Kairouz et al.(2022)] Peter Kairouz, Ziyu Liu, and Thomas Steinke. 2022. The Distributed Discrete Gaussian Mechanism for Federated Learning with Secure Aggregation. arXiv:2102.06387 [cs.LG] https://arxiv.org/abs/2102.06387

[Kairouz et al.(2021)] Peter Kairouz, Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu. 2021. Practical and private (deep) learning without sampling or shuffling. In *International Conference on Machine Learning*. PMLR, 5213–5225.

[Karthikeyan and Polychroniadou(2024)] Harish Karthikeyan and Antigoni Polychroniadou. 2024. OPA: One-shot Private Aggregation with Single Client Interaction and its Applications to Federated Learning. Cryptology ePrint Archive, Paper 2024/723. https://eprint.iacr.org/2024/723 https://eprint.iacr.org/2024/723.

[Kim et al.(2023)] Duhyeong Kim, Dongwon Lee, Jinyeong Seo, and Yongsoo Song. 2023. Toward Practical Lattice-Based Proof of Knowledge from Hint-MLWE. In *Advances in Cryptology - CRYPTO 2023 - 43rd Annual International Cryptology Conference, CRYPTO 2023, Santa Barbara, CA, USA, August 20-24, 2023, Proceedings, Part V (Lecture Notes in Computer Science, Vol. 14085)*, Helena Handschuh and Anna Lysyanskaya (Eds.). Springer, 549–580. https://doi.org/10.1007/978-3-031-38554-4_18

[Lee et al.(2018)] Joohee Lee, Dongwoo Kim, Duhyeong Kim, Yongsoo Song, Junbum Shin, and Jung Hee Cheon. 2018. Instant Privacy-Preserving Biometric Authentication for Hamming Distance. *IACR Cryptol. ePrint Arch.* (2018), 1214. https://eprint.iacr.org/2018/1214

[Li et al.(2023)] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. 2023. LERNA: secure single-server aggregation via key-homomorphic masking. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 302–334.

[Lyubashevsky et al.(2010)] Vadim Lyubashevsky, Chris Peikert, and Oded Regev. 2010. On ideal lattices and learning with errors over rings. In *Advances in Cryptology–EUROCRYPT 2010: 29th Annual International Conference on the Theory and Applications of Cryptographic Techniques, French Riviera, May 30–June 3, 2010. Proceedings 29*. Springer, 1–23.

[Ma et al.(2023)] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. 2023. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 477–496.

[McMahan et al.(2018)] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. arXiv:1710.06963 [cs.LG] https://arxiv.org/abs/1710.06963

[McMahan et al.(2024)] H. Brendan McMahan, Zheng Xu, and Yanxiang Zhang. 2024. A Hassle-free Algorithm for Private Learning in Practice: Don't Use Tree Aggregation, Use BLTs. arXiv:2408.08868 [cs.LG] https://arxiv.org/abs/2408.08868

[Micciancio and Regev(2004)] D. Micciancio and O. Regev. 2004. Worst-case to average-case reductions based on Gaussian measures. In *45th Annual IEEE Symposium on Foundations of Computer Science*. 372–381. https://doi.org/10.1109/FOCS.2004.72

[Song et al.(2013)] Shuang Song, Kamalika Chaudhuri, and Anand D Sarwate. 2013. Stochastic gradient descent with differentially private updates. In *2013 IEEE global conference on signal and information processing*. IEEE, 245–248.

[Tramer and Boneh(2020)] Florian Tramer and Dan Boneh. 2020. Differentially private learning needs better features (or much more data). *arXiv preprint arXiv:2011.11660* (2020).

[Xu et al.(2023)] Zheng Xu, Yanxiang Zhang, Galen Andrew, Christopher A Choquette-Choo, Peter Kairouz, H Brendan McMahan, Jesse Rosenstock, and Yuanbo Zhang. 2023. Federated learning of gboard language models with differential privacy. *arXiv preprint arXiv:2305.18465* (2023).

A Preliminaries

A.1 Cryptographic Building Blocks

A.1.1 Lattices, Rings, and RLWE Encryption

Definition 1 (Decisional LWE assumption)**.** *Given a prime q, a matrix A sampled uniformly from $\mathbb{Z}_q^{M\times N}$, a vector s uniformly sampled from $\mathbb{Z}_q^N$, an error vector $e \in \mathbb{Z}_q^N$ sampled from ϕ. We say that Decisional LWE is hard if $(A, As+e)$ is computationally indistinguishable from the uniform.*

Definition 2 (LWE encryption)**.** *Let A be sampled uniformly from $\mathbb{Z}_q^{M\times N}$ and ϕ be an error distribution in which LWE is hard. LWE encryption scheme consists of the following three algorithms:*

- $\mathsf{LWEGen}(1^\lambda) \to (s)$*: sample a random vector s from $\mathbb{Z}_q^N$, sample $e \leftarrow \phi$.*
- $\mathsf{LWEEnc}(A, s, x) \to c$*: compute $c = As + Te + x \mod q$, where $x \in \mathbb{Z}_T^M$, T is coprime to q.*
- $\mathsf{LWEDec}(A, s, c) \to x$*: compute $x = (c - As) \mod T$.*

We say that LWE encryption is secure is secure if it has CPA security. Note that if Decisional LWE assumption holds, then LWE encryption is secure.

A.1.2 Secret Sharing

Definition 3 (Additive secret sharing)**.** *An additive secret-sharing scheme consists of the following algorithm:*

- $\mathsf{AShare}(sk) \to (x_1, \ldots, x_n)$*: take in a secret sk, generate randomness ρ, output n random messages such that $\sum_{i=1}^n x_i = sk$.*

Definition 4 (Threshold secret sharing)**.** *A threshold secret-sharing scheme consists of the following algorithm:*

- $\mathsf{TShare}(sk) \to (x_1, \ldots, x_m)$*: take in a secret sk, generate randomness ρ, output m random messages to help with the reconstruction of sk.*
- $\mathsf{TRec}(\{x_j\}_{j\in S, |S|\geq t}) \to sk$*: take in at least t threshold secret shares, output sk.*

We say that the threshold secret sharing scheme is secure if the probability that, given less than t threshold shares, an adversary can recover sk is negligible.

B Fully-Synchronous (no dropouts) semi-honest security

We use a variant case of the Hint-MLWE problem of Kim et al. [Kim et al.(2023)]. Kim et al. additionally allow leakage on the MLWE secret, s, but we do allow the adversary this. On the other hand, we do assume the secret key is uniformly random (in contrast to Gaussian). Additionally, our variant considers multiplying the noise by a factor of T (where T is coprime to the modulus q).

Definition 5 (MLWE [Kim et al.(2023)])**.** *Let d, m, q be positive integers, and χ be a distribution over R^{d+m}. Then, the goal of the Module-LWE (MLWE) problem is to distinguish $(\boldsymbol{A}, \boldsymbol{u})$ from $(\boldsymbol{A}, \boldsymbol{As}+\boldsymbol{e})$ for $\boldsymbol{A} \xleftarrow{u} R_q^{m\times d}, \boldsymbol{u} \leftarrow \mathcal{U}(R_q^m)$, $\boldsymbol{e} \leftarrow \chi$, and $\boldsymbol{s} \xleftarrow{u} R_q^d$. We say that a PPT adversary $\mathcal{A}$ has advantage ε in solving $\mathsf{MLWE}_{R,d,m,q,\chi}$ if*

$$\Pr[\mathcal{A}(\boldsymbol{A}, \boldsymbol{As}+\boldsymbol{e}) = 1] - \Pr[\mathcal{A}(\boldsymbol{A}, \boldsymbol{u}) = 1] \geq \varepsilon.$$

Definition 6 (Hint-MLWE [Kim et al.(2023)])**.** *Let d, m, ℓ be positive integers, χ, ξ be distributions over R^m, χ a distribution over R^d. The Hint-MLWE problem, denoted by $\mathsf{HintMLWE}^{\ell,\xi}_{R,d,m,q,T,\chi,\chi'}$ asks an adversary $\mathcal{A}$ to distinguish the following two cases:*

1. $(\boldsymbol{A}, \boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e}, \boldsymbol{e} + \boldsymbol{f}_1, \ldots, \boldsymbol{e} + \boldsymbol{f}_\ell)$,
2. $(\boldsymbol{A}, \boldsymbol{u}, \boldsymbol{e} + \boldsymbol{f}_1, \ldots, \boldsymbol{e} + \boldsymbol{f}_\ell)$;

where in both distributions above $\boldsymbol{A} \xleftarrow{u} R_q^{m \times d}$, $\boldsymbol{s} \xleftarrow{u} \chi'$, $\boldsymbol{e} \leftarrow \chi$, $\boldsymbol{f}_i \leftarrow \xi$ *for all* $i \in [\ell]$, *and* $\boldsymbol{u} \xleftarrow{u} R_q^m$.

We take $\mathsf{HintMLWE}^{\ell,\sigma_2}_{R,d,m,q,T,\sigma_1}$ *to be the case where* ξ *is a spherical Gaussian distribution with width* σ_2 *and* χ *is a spherical Gaussian distributions with width* σ_1, *and* χ' *is uniformly random.*

We verify that Kim et al.'s proof of the hardness of Hint-MLWE can similarly be adapted to our variant following the observations of Bell et al. for HintLWE [Bell et al.(2023b)] and noticing that if the secret isn't leaked then conditional sampling need not be invoked on the secret key.

Definition 7 (Smoothing parameter [Micciancio and Regev(2004)])**.** *For an* n*-dimensional lattice* Λ *and positive real* $\varepsilon > 0$, *the smoothing parameter* $\eta_\varepsilon(\Lambda)$ *is the smallest* s *such that* $\rho_{1/s}(\Lambda^* \backslash \{\boldsymbol{0}\}) \leq \varepsilon$.

Theorem 2 ([Kim et al.(2023)])**.** *Let* d, k, m, q, ℓ *be positive integers. For any* T *coprime to* q, $\ell > 0$, *and* $\sigma_1, \sigma_2, \sigma > 0$ *such that* $\frac{1}{\sigma^2} \geq 2\left(\frac{1}{\sigma_1^2} + \frac{\ell}{\sigma_2^2}\right)$. *If* $\sigma \geq \sqrt{2}\eta_\varepsilon(\mathbb{Z}^n)$ *for* $0 < \varepsilon \leq 1/2$, *then there exists an efficient reduction from* $\mathsf{MLWE}_{R,d,m,q,\sigma}$ *to* $\mathsf{HintMLWE}^{\ell,\sigma_2}_{R,d,m,q,T,\sigma_1}$.

From this we derive a sanity check for our security proof that says that the leakage of the reveal operations does not reveal anything about the encrypted state held by the server. The full required for the proof amounts to incorporating a hybrid argument with this observation. Note that here we give a general proof for the high dimensional case. Our protocol actually works under Hint-RLWE assumption, where d is set to 1.

Corollary 1. *Assuming the hardness of* $\mathsf{HintMLWE}^{\ell,\sigma_2}_{R,d,m,q,T,\sigma_1}$ *, for any* $\boldsymbol{\lambda}^{(1)}, \ldots, \boldsymbol{\lambda}^{(\ell)} \in \mathbb{Z}_q^m$, *the following distributions are indistinguishable for any PPT adversary*

1. $(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_r, \boldsymbol{A}_1\boldsymbol{s} + T\boldsymbol{e}_1, \ldots, \boldsymbol{A}_r\boldsymbol{s} + T\boldsymbol{e}_r, \sum_{i=1}^r \boldsymbol{\lambda}_i^{(1)}(T\boldsymbol{f}_i^{(1)} - \boldsymbol{A}_i\boldsymbol{s}), \sum_{i=1}^r \boldsymbol{\lambda}_i^{(\ell)}(T\boldsymbol{f}_i^{(\ell)} - \boldsymbol{A}_i\boldsymbol{s}))$,
2. $(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_r, \boldsymbol{u}_1, \ldots, \boldsymbol{u}_r, \sum_{i=1}^r \boldsymbol{\lambda}_i^{(1)}(T\boldsymbol{f}_i^{(1)} + T\boldsymbol{e}_i - \boldsymbol{u}_i), \sum_{i=1}^r \boldsymbol{\lambda}_i^{(\ell)}(T\boldsymbol{f}_i^{(\ell)} - T\boldsymbol{e}_i - \boldsymbol{u}_i))$;

where in both distributions above $\boldsymbol{A}_i \xleftarrow{u} R_q^{m \times d}$, $\boldsymbol{s} \leftarrow D_{d,\sigma_1}$, $\boldsymbol{e} \leftarrow D_{m,\sigma_1}$, $\boldsymbol{f}_i^{(j)} \leftarrow D_{\sigma_2}$, *and* $\boldsymbol{u}_i \xleftarrow{u} R_q^m$ *for all* $i \in [r], j \in [\ell]$.

Proof. This follows by observing that each distribution is a linear function from the corresponding HintMLWE distribution. Namely, if we take

$$\boldsymbol{A} := \begin{bmatrix} \boldsymbol{A}_1 \\ \cdots \\ \boldsymbol{A}_n \end{bmatrix} \quad \& \quad \boldsymbol{u} := \begin{bmatrix} \boldsymbol{u}_1 \\ \cdots \\ \boldsymbol{u}_n \end{bmatrix} \quad \& \quad \boldsymbol{e} := \begin{bmatrix} \boldsymbol{e}_1 \\ \cdots \\ \boldsymbol{e}_n \end{bmatrix} \quad \& \quad \boldsymbol{f}^{(i)} := \begin{bmatrix} \boldsymbol{f}_1^{(i)} \\ \cdots \\ \boldsymbol{f}_n^{(i)} \end{bmatrix}$$

for all $i \in [\ell]$ and define the operator

$$M(\boldsymbol{\lambda}) := [\boldsymbol{\lambda}_1 I | \cdots \boldsymbol{\lambda}_n I] = \boldsymbol{\lambda}^\top \otimes I$$

then we can equivalently formulate the above distributions as

1. $\big(\boldsymbol{A}, \boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e}, M(\lambda^{(1)})(T\boldsymbol{f}^{(1)} - \boldsymbol{A}\boldsymbol{s}), \ldots, M(\lambda^{(\ell)})(T\boldsymbol{f}^{(\ell)} - \boldsymbol{A}\boldsymbol{s})\big)$,
2. $\big(\boldsymbol{A}, \boldsymbol{u}, M(\lambda^{(1)})(T\boldsymbol{f}^{(1)} + T\boldsymbol{e} - \boldsymbol{u}), \ldots, M(\lambda^{(\ell)})(T\boldsymbol{f}^{(\ell)} + T\boldsymbol{e} - \boldsymbol{u})\big)$.

The corollary then follows from the simple reduction F that given $(\boldsymbol{A}, \boldsymbol{b}, \boldsymbol{y}^{(1)}, \ldots, \boldsymbol{y}^{(\ell)})$ and $\boldsymbol{\lambda}^{(1)}, \ldots, \boldsymbol{\lambda}^{(\ell)}$ simply outputs

$$(\boldsymbol{A}, \boldsymbol{b}, M(\boldsymbol{\lambda}^{(1)})(T\boldsymbol{y}^{(1)} - \boldsymbol{b}), \ldots, M(\boldsymbol{\lambda}^{(\ell)})(T\boldsymbol{y}^{(\ell)} - \boldsymbol{b})).$$

To see this suffices note that for $\boldsymbol{y}^{(i)} = \boldsymbol{e} + \boldsymbol{f}^{(i)}$,

$$M(\boldsymbol{\lambda}^{(i)})(\boldsymbol{y}^{(i)} - \boldsymbol{b}) = \begin{cases} M(\boldsymbol{\lambda}^{(i)})(T\boldsymbol{f}^{(i)} - \boldsymbol{A}\boldsymbol{s}) & \text{if } \boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e}, \\ M(\boldsymbol{\lambda}^{(i)})(T\boldsymbol{f}^{(i)} + T\boldsymbol{e} - \boldsymbol{u}) & \text{if } \boldsymbol{b} = \boldsymbol{u}. \end{cases}$$

□

However, as mentioned, this corollary alone is not quite enough for our proof. We need to show that not only does the encrypted state remain secure under the Reveal operations but moreover that the Store operations themselves remain secure given these releases: encrypting under correlated keys remain secure under leakage.

Lemma 1. *Assuming the hardness of* $\mathsf{HintMLWE}^{\ell,\sigma_2}_{R,d,m,q,T,\sigma_1}$*, for any* $\boldsymbol{\lambda}^1,\ldots,\boldsymbol{\lambda}^\ell \in \mathbb{Z}_q^m, \boldsymbol{x}_1,\ldots,\boldsymbol{x}_t \in R_q^m$ *the following distributions are indistinguishable for any PPT adversary*

1. $\big(\boldsymbol{A}_1,\ldots,\boldsymbol{A}_r, \boldsymbol{A}_1\boldsymbol{s}+T\boldsymbol{e}_1+\boldsymbol{x}_1,\ldots,\boldsymbol{A}_r\boldsymbol{s}+T\boldsymbol{e}_r+\boldsymbol{x}_r, \sum_{i=1}^r \boldsymbol{\lambda}_i^1(T\boldsymbol{f}_i^1-\boldsymbol{A}_i\boldsymbol{s}-\boldsymbol{x}_i)+y_1,\ldots,\sum_{i=1}^r \boldsymbol{\lambda}_i^\ell(T\boldsymbol{f}_i^\ell-\boldsymbol{A}_i\boldsymbol{s}-\boldsymbol{x}_i)+y_\ell\big)$,
2. $\big(\boldsymbol{A}_1,\ldots,\boldsymbol{A}_r, \boldsymbol{u}_1,\ldots,\boldsymbol{u}_r, \sum_{i=1}^r \boldsymbol{\lambda}_i^1(T\boldsymbol{f}_i^1+T\boldsymbol{e}_i-\boldsymbol{u}_i)+y_1,\ldots,\sum_{i=1}^r \boldsymbol{\lambda}_i^\ell(T\boldsymbol{f}_i^\ell+T\boldsymbol{e}_i-\boldsymbol{u}_i)+y_\ell\big)$;

where in both distributions above $\boldsymbol{A}_i \xleftarrow{u} R_q^{m\times d}$, $\boldsymbol{s}\leftarrow D_{d,\sigma_1}$, $\boldsymbol{e}_i\leftarrow D_{m,\sigma_1}$, $\boldsymbol{f}_i^{(j)}\leftarrow D_{\sigma_2}$ *for all* $i\in[r], j\in[\ell]$, *and* $\boldsymbol{u}_1,\ldots,\boldsymbol{u}_r \xleftarrow{u} R_q^m$ *and we define* $y_k := \sum_{i=1}^r \boldsymbol{\lambda}_i^k \boldsymbol{x}_i$ *for* $k\in[\ell]$.

The proof of this lemma combines the simple transformation above with a hybrid argument.

Proof. We begin by defining the hybrids $H_0,\ldots,H_r$. In particular, H_i is the distribution

$$(\boldsymbol{A}_1,\ldots,\boldsymbol{A}_r,\boldsymbol{u}_1,\ldots,\boldsymbol{u}_i,\boldsymbol{A}_{i+1}\boldsymbol{s}+\boldsymbol{x}_{i+1}+T\boldsymbol{e}_{i+1},\ldots,\boldsymbol{A}_r\boldsymbol{s}+\boldsymbol{x}_r+T\boldsymbol{e}_r,z_1,\ldots z_\ell)$$

where for $k\in[\ell]$,

$$z_k=\sum_{i=1}^r \boldsymbol{\lambda}_i^{(k)}\left(-\sum_{j=1}^i \boldsymbol{u}_j-\sum_{j=i+1}^r \boldsymbol{A}_j\boldsymbol{s}+T\boldsymbol{f}_i^k+T\sum_{j=1}^i \boldsymbol{e}_j-\sum_{j=i+1}^r \boldsymbol{x}_j\right)+y_k$$

and $\boldsymbol{u}_1,\ldots,\boldsymbol{u}_r \xleftarrow{u} R_q^m, \boldsymbol{A}_i \xleftarrow{u} R_q^{m\times d}, \boldsymbol{s}\leftarrow D_{d,\sigma_1}, \boldsymbol{e}_i\leftarrow D_{m,\sigma_1}, \boldsymbol{f}_i^{(j)}\leftarrow D_{\sigma_2}$ for $i\in[r], j\in[\ell]$.

Notice that H_0 coincides with distribution 1 and H_r coincides with distribution 2. Now, we argue that for any $i\in[r]$, $H_{i-1}\approx H_i$ by the following reduction from $\mathsf{HintMLWE}$ (Definition 6).

We can build a reduction $\mathcal{B}$ that takes input $(\boldsymbol{A},\boldsymbol{b},\boldsymbol{w}^{(1)},\ldots,\boldsymbol{w}^{(\ell)})$ from $\mathsf{HintMLWE}$ such that it outputs

$$(\boldsymbol{A}_1,\ldots,\boldsymbol{A}_r,\boldsymbol{u}_1,\ldots,\boldsymbol{u}_{i-1},\boldsymbol{v}_i,\boldsymbol{A}_{i+1}\boldsymbol{s}+\boldsymbol{x}_{i+1}+T\boldsymbol{e}_{i+1},\ldots,\boldsymbol{A}_r\boldsymbol{s}+\boldsymbol{x}_r+T\boldsymbol{e}_r,\hat{z}_1,\ldots,\hat{z}_\ell),$$

where $\boldsymbol{v}_i=\boldsymbol{b}+\boldsymbol{x}_i$ and

$$\hat{z}_k=\sum_{i=1}^r \boldsymbol{\lambda}_i^k\left(-\boldsymbol{b}+\boldsymbol{w}^{(k)}-\sum_{j=1}^{i-1}\boldsymbol{u}_j-\sum_{j=i+1}^r \boldsymbol{A}_j\boldsymbol{s}+T\sum_{j=1}^{i-1}\boldsymbol{e}_j-\sum_{j=i}^r \boldsymbol{x}_j\right)+y_k$$

Consider that $\boldsymbol{b}=\boldsymbol{A}\boldsymbol{s}+T\boldsymbol{e}$, $\boldsymbol{w}^{(i)}=T\boldsymbol{e}+T\boldsymbol{f}^{(i)}$ for $i\in[\ell]$, $\boldsymbol{e}\leftarrow D_{m,\sigma_1}, \boldsymbol{f}^{(i)}\leftarrow D_{\sigma_2}$. We rewrite using fresh variable names $\boldsymbol{A}_i=\boldsymbol{A}, \boldsymbol{e}_i=\boldsymbol{e}, \boldsymbol{f}_i^k=\boldsymbol{f}^{(i)}$. As a result, we have $\boldsymbol{v}_i=\boldsymbol{A}_i\boldsymbol{s}+\boldsymbol{x}_i+T\boldsymbol{e}_i$ and

$$\hat{z}_k=\sum_{i=1}^r \boldsymbol{\lambda}_i^k\left(-\sum_{j=1}^{i-1}\boldsymbol{u}_j-\sum_{j=i}^r \boldsymbol{A}_j\boldsymbol{s}+T\boldsymbol{f}_i^k+T\sum_{j=1}^{i-1}\boldsymbol{e}_j-\sum_{j=i}^r \boldsymbol{x}_j\right)+y_k$$

So the output of the reduction is the same as H_{i-1}.

On the other hand, consider that $\boldsymbol{b}=\boldsymbol{u}, \boldsymbol{w}^{(i)}=T\boldsymbol{e}+T\boldsymbol{f}^{(i)}$ for $i\in[\ell]$, $\boldsymbol{e}\leftarrow D_{m,\sigma_1}, \boldsymbol{f}^{(i)}\leftarrow D_{\sigma_2}$. We rewrite using fresh variable names $\boldsymbol{u}_i=\boldsymbol{u}, \boldsymbol{e}_i=\boldsymbol{e}, \boldsymbol{f}_i^k=\boldsymbol{f}^{(i)}$. As a result, we have $\boldsymbol{v}_i=\boldsymbol{u}_i+\boldsymbol{x}_i$ and

$$\hat{z}_k=\sum_{i=1}^r \boldsymbol{\lambda}_i^k\left(-\sum_{j=1}^{i}\boldsymbol{u}_j-\sum_{j=i+1}^r \boldsymbol{A}_j\boldsymbol{s}+T\boldsymbol{f}_i^k+T\sum_{j=1}^{i}\boldsymbol{e}_j-\sum_{j=i}^r \boldsymbol{x}_j\right)+y_k$$

Since $\boldsymbol{b} = \boldsymbol{u}$ is uniformly random, $\hat{z}_k$ follows exactly the distribution H_i. By the hardness assumption of $\mathsf{HintMLWE}$, for any $i \in [r]$, H_{i-1} and H_i are indistinguishable for any PPT adversary. In particular, H_0 and H_r are indistinguishable for any PPT adversary, which proves the lemma.

□

Theorem 3. *Assuming that a semi-honest adversary corrupts at most an γ-fraction of any user cohort, in addition to the server, The protocol given in Figures 2,3 securely implements the functionality in figure 1.*

The proof follows more or less immediately from Lemma 1.

Sketch. For simplicity, we assume secure point-to-point channels between clients in successive cohorts. This assumption is easily relaxed in usual manner by introducing key infrastructure. Also for simplicity we assume the inputs are statically chosen, although the proof can easily be modified to securely simulate a reactive version of the functionality.

The lazy protocol. In this proof we consider, without loss of generality, a *lazy protocol* where all linear function evaluation is pushed to the time of reveal. The lazy server simply appends the sum of the clients' Store messages to it's state, instead of also adding a linear combination of the prior states. Then, when it is time to reveal the server and clients recursively compute the appropriate linear function, $\bar{\boldsymbol{\lambda}}^i$, consistent with all real linear operations that were to have taken place up to this point, $\boldsymbol{\lambda}^1, \ldots, \boldsymbol{\lambda}^{i-1}$. By linearity, this too is just another linear function that can be computed by copying the state and performing the sequence of necessary linear evaluations or, alternatively, simply evaluating the with $\bar{\boldsymbol{\lambda}}^i$ where $\bar{\boldsymbol{\lambda}}^i_j := \sum_{j=k_1<k_2<\cdots<k_\ell=i} \boldsymbol{\lambda}^{k_\ell}_{k_{\ell-1}} \boldsymbol{\lambda}^{k_{\ell-1}}_{k_{\ell-2}} \cdots \boldsymbol{\lambda}^{k_3}_{k_2} \boldsymbol{\lambda}^{k_2}_{k_1}$ for $j < i$ and $\bar{\boldsymbol{\lambda}}^i_i = 1$.

For example if the sum of inputs at round i is x_i, then the real server has state

$$\boldsymbol{v} = (\underbrace{x_1}_{v_1}, \underbrace{x_2 + \lambda_1^2 v_1}_{v_2}, \underbrace{x_3 + \lambda_2^3 v_2 + \lambda_1^3 v_1}_{v_3}, \ldots)$$

and when asked to reveal simply outputs v_i.

In the lazy protocol, the state is simply

$$\bar{\boldsymbol{v}} = (x_1, x_2, x_3, \ldots)$$

and when asked to reveal the output is computed as $\langle \bar{\boldsymbol{\lambda}}^i, \bar{\boldsymbol{v}} \rangle = \sum_{j=1}^i \bar{\boldsymbol{\lambda}}^i_j x_j$.

Simulating client-to-client communication. We begin by noting that if an adversary controls at most an γ fraction of the next cohort, C^{i+1}, the probability that any specific client in cohort C^i who sends messages to d i.i.d. randomly chosen clients in C^{i+1} fails to send a message to an honest client is γ^d. If there are a total of at most r cohorts, each containing n clients, then for $d \geq \frac{\log(1/\delta)+\log(2nr)}{1-\gamma}$ every honest client sends a message to some other honest client and receives at least one message from some other honest client with probability at least $1 - 2nre^{d(\gamma-1)} \geq 1 - \delta$.

In this case, because messages from honest clients to other clients are perfect additive secret shares as we assume point-to-point channels between clients, the adversary's view of such messages is uniformly random and independent of all other communication in the protocol seen by the adversary. Thus, we can focus exclusively on simulating messages to the corrupt server (which comprises all other messages from honest clients seen by the server). Next, we describe how to simulate these messages.

Simulating messages to the adversarial server. Let $\hat{\boldsymbol{v}}_i$ denote the state of the idealized functionality at round i if only adversarial inputs are incorporated. Let $\hat{\boldsymbol{v}}_i{}'$ denote the state of the idealized functionality using only honest outputs at round i, then the state at round i (incorporating both honest and adversarial inputs) is simply $\hat{\boldsymbol{v}}_i + \hat{\boldsymbol{v}}_i{}' = \boldsymbol{v}_i$. Recall, additionally, that if there is a Reveal operation at round i, then the idealized functionality will output $\boldsymbol{v}_i$ at the next round.

The simulator is as follows:

1. The messages sent by honest parties when $\mathsf{Mode}_i = \mathsf{Store}$ are simply uniformly random.

2. When $\mathsf{Mode}_i = \mathsf{Reveal}$, the simulator additionally sends random values (in the next round) such that when summed with the contribution from server, $y_i = \langle \bar{\boldsymbol{\lambda}}^i, \bar{\boldsymbol{v}} \rangle$, where $\bar{\boldsymbol{v}} = (x_1, x_2, x_3, \ldots)$ is the server's state at time i, and the messages from the corrupt parties, the result is $y_i + T \sum_{j=1}^{t'_i} \langle \bar{\boldsymbol{\lambda}}^i, \boldsymbol{f}^{i,j} \rangle$ where $\boldsymbol{f}^{i,j}$ are sampled according to $D_{\sigma_1} + D_{\sigma_2}$.

We now argue that this simulation is indistinguishable from the real protocol using a hybrid argument. However before continuing, we set up some notation that will be consistent across the hybrid argument. Suppose there are r instructions with $\mathsf{Mode} = \mathsf{Store}$ and ℓ instructions with $\mathsf{Mode} = \mathsf{Reveal}$. We denote the view of the adversarial server (rearranged) as

$$(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_r, \alpha_1^1, \ldots, \alpha_{t_1}^1, \ldots, \alpha_1^r, \ldots, \alpha_{t_r}^r, \beta_1^1, \ldots, \beta_{t'_1}^1, \ldots, \beta_1^\ell \ldots, \beta_{t'_\ell}^\ell)$$

where α_j^i denotes the Store message sent by the jth honest client in round i (here the honest clients are indexed from 1 to t_i) and β_j^i denotes the message sent by the jth honest client in round i if $\mathsf{Mode}_{i-1} = \mathsf{Reveal}$ (here the honest clients are indexed from 1 to t'_i). Caution: the distribution of these variables will depend on the specific hybrid.

So the simulator sets all α_j^i to be uniformly random and independent. Similarly, β_j^i are uniformly random conditioned on $\sum_{j=1}^{t'_i} \beta_j^i = y_i + T \sum_{j=1}^{t'_i} \langle \bar{\boldsymbol{\lambda}}^i, \boldsymbol{f}^{i,j} \rangle - \boldsymbol{c}_i$, where $\boldsymbol{c}_i$ is the message from the dishonest parties.

We now define the sequence of hybrids:

Hybrid 0. H_0 is the adversary's real view. We can rewrite how this is sampled as follows:

1. Sample $\boldsymbol{A}_i \xleftarrow{u} R_q^{m \times d}$ for all $i \in [r]$.
2. Sample $\boldsymbol{s} \xleftarrow{u} R_q^m$.
3. Sample $\boldsymbol{e}_j^i \leftarrow D_{m,\sigma_1}$ for all $i \in [r], j \in [t_i]$ and $\boldsymbol{f}_{k,j}^i \leftarrow D_{\sigma_2}$ for all $i \in [\ell], j \in [t'_i], k \in [r]$.
4. For all $i \in [r], \boldsymbol{s}_j^i$ are uniformly random variables conditioned on $\sum_{j=1}^{t_i} \boldsymbol{s}_j^i + \sum_{k \text{ corrupt}} \boldsymbol{s}_k^i = \boldsymbol{s}$, where $\boldsymbol{s}_k^i$ for corrupt k are computed according to the protocol.
5. For all $j \in [t_i]$, set $\alpha_j^i = \boldsymbol{A}_i \boldsymbol{s}_j^i + T\boldsymbol{e}_j^i + x_j^i$, where x_j^i is the secret input of the jth honest party in round i.
6. For all $j \in [t'_i]$, set $\beta_j^i = \sum_{k=1}^r \bar{\boldsymbol{\lambda}}_k^i (T\boldsymbol{f}_{k,j}^i - \boldsymbol{A}_k \boldsymbol{s}_j^i)$, where $\bar{\boldsymbol{\lambda}}^i \in \mathbb{Z}_q^m$ is the weights applied in round i.
7. $\hat{\alpha}_k^i, \hat{\beta}_k^i$ for corrupt k are computed according to protocol and prior messages.

Hybrid 1. In this hybrid, we can equivalently sample this by first sampling uniformly random $\alpha_j^i \xleftarrow{u} R_q^m$ for all $i \in [r]$ and $j \geq 2$. Then we set $\alpha_1^i = \sum_{j=1}^{t_i} \boldsymbol{A}_i s_j^i + T\boldsymbol{e}_j^i + x_j^i - \sum_{j=2}^{t_i} \alpha_j^i$ for all $i \in [r]$.

Notice that all clients need to follow the protocol regardless of whether honest or not. So the invariant $\sum_{j=1}^{t_i} \boldsymbol{A}_i s_j^i + \sum_{k \text{ corrupt}} \boldsymbol{A}_i s_k^i = \boldsymbol{A}_i \boldsymbol{s}$ always holds. So we can rewrite α_1^i as

$$\boldsymbol{A}_i \boldsymbol{s} + T\boldsymbol{e}_i + \boldsymbol{x}_i - \boldsymbol{c}_i - \sum_{j=2}^{t_i} \alpha_j^i$$

where $\boldsymbol{e}_i = \sum_{j=1}^{t_i} \boldsymbol{e}_j^i$, $\boldsymbol{x}_i = \sum_{j=1}^{t_i} \boldsymbol{x}_j^i$, and $\boldsymbol{c}_i = \sum_{k \text{ corrupt}} \boldsymbol{A}_i s_k^i$.

To show this hybrid is indistinguishable from **Hybrid 0**, we define the following partial hybrids $H_i^1, \ldots, H_i^{t_i}$ for each $i \in [r]$. In particular, H_i^j is the distribution:

$$\left(\boldsymbol{A}_i, \sum_{k=1}^{j} \boldsymbol{A}_i \boldsymbol{s}_k^i + \boldsymbol{x}_k^i + T\boldsymbol{e}_k^i - \sum_{k=2}^{j} \boldsymbol{u}_k, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_j, \boldsymbol{A}_i \boldsymbol{s}_{j+1}^i + \boldsymbol{x}_{j+1}^i + T\boldsymbol{e}_{j+1}^i, \ldots, \boldsymbol{A}_i \boldsymbol{s}_{t_i}^i + \boldsymbol{x}_{t_i}^i + T\boldsymbol{e}_{t_i}^i \right)$$

where $\boldsymbol{u}_2, \ldots, \boldsymbol{u}_{t_i} \xleftarrow{u} R_q^m, \boldsymbol{A}_i \xleftarrow{u} R_q^{m \times d}, \boldsymbol{s} \xleftarrow{u} R_q^d, \boldsymbol{e}_1^i, \ldots, \boldsymbol{e}_{t_i}^i \leftarrow D_{m,\sigma_1}$, and $\boldsymbol{s}_j^i$ are uniformly random variables conditioned on $\sum_{j=1}^{t_i} \boldsymbol{s}_j^i + \sum_{k \text{ corrupt}} \boldsymbol{s}_k^i = \boldsymbol{s}$, where $\boldsymbol{s}_k^i$ for corrupt k are computed according to the protocol.

Notice that H_i^1 coincides with the partial view in **Hybrid 0** and $H_i^{t_i}$ coincides with the partial view in **Hybrid 1**. Now, we argue that for any $j \in \{2, \ldots, t_i\}$, $H_i^{j-1} \approx H_i^j$ by the following reduction from MLWE (Definition 5).

We can build a reduction $\mathcal{B}$ that takes input $(\boldsymbol{A}, \boldsymbol{b})$ from MLWE such that it outputs

$$\left(\boldsymbol{A}_i, \boldsymbol{z}, \boldsymbol{u}_2, \ldots, \boldsymbol{u}_{j-1}, \boldsymbol{v}_j, \boldsymbol{A}_i \boldsymbol{s}_{j+1}^i + \boldsymbol{x}_{j+1}^i + T\boldsymbol{e}_{j+1}^i, \ldots, \boldsymbol{A}_i \boldsymbol{s}_{t_i}^i + \boldsymbol{x}_{t_i}^i + T\boldsymbol{e}_{t_i}^i\right),$$

where $\boldsymbol{v}_j = \boldsymbol{b} + \boldsymbol{x}_j^i$ and

$$\boldsymbol{z} = \sum_{k=1}^{j} \boldsymbol{A}_i \boldsymbol{s}_k^i + \boldsymbol{x}_k^i + T\boldsymbol{e}_k^i - \sum_{k=2}^{j-1} \boldsymbol{u}_k - \boldsymbol{b} - \boldsymbol{x}_j^i$$

Consider that $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{s} + T\boldsymbol{e}$, $\boldsymbol{e} \leftarrow D_{m,\sigma_1}$. We rewrite using fresh variable names $\boldsymbol{A}_i = \boldsymbol{A}, \boldsymbol{s}_j^i = \boldsymbol{s}, \boldsymbol{e}_j^i = \boldsymbol{e}$. As a result, we have $\boldsymbol{v}_j = \boldsymbol{A}_i \boldsymbol{s}_j^i + \boldsymbol{x}_j^i + T\boldsymbol{e}_j^i$ and

$$\boldsymbol{z} = \sum_{k=1}^{j-1} \boldsymbol{A}_i \boldsymbol{s}_k^i + \boldsymbol{x}_k^i + T\boldsymbol{e}_k^i - \sum_{k=2}^{j-1} \boldsymbol{u}_k$$

So the output of the reduction is the same as H_i^{j-1}.

On the other hand, consider that $\boldsymbol{b} = \boldsymbol{u}$. We rewrite using fresh variable names $\boldsymbol{u}_j = \boldsymbol{u}$. As a result, we have $\boldsymbol{v}_j = \boldsymbol{u}_j + \boldsymbol{x}_j^i$ and

$$\boldsymbol{z} = \sum_{k=1}^{j} \boldsymbol{A}_i \boldsymbol{s}_k^i + \boldsymbol{x}_k^i + T\boldsymbol{e}_k^i - \sum_{k=2}^{j} \boldsymbol{u}_k - \boldsymbol{x}_j^i$$

Since $\boldsymbol{b} = \boldsymbol{u}$ is uniformly random, $\boldsymbol{z}$ follows exactly the distribution H_i^j. By the hardness assumption of MLWE, for any $j \in \{2, \ldots, t_i\}$, H_i^{j-1} and H_i^j are indistinguishable for any PPT adversary. In particular, H_i^1 and $H_i^{t_i}$ are indistinguishable for any PPT adversary.

Combining all the partial hybrids, we can show that any PPT adversary cannot distinguish between **Hybrid 0** and **Hybrid 1**.

Hybrid 2. In this hybrid, we set $\alpha_i^1 = \boldsymbol{A}_i \boldsymbol{s} + T\boldsymbol{e}_i + \boldsymbol{x}_i - \boldsymbol{c}_i$ for all $i \in [r]$. This hybrid is indistinguishable from **Hybrid 1** by the pseudorandomness of MLWE samples.

Hybrid 3. In this hybrid, we can equivalently sample the distribution in the previous hybrid by first sampling uniformly random $\beta_{j,k}^i \xleftarrow{u} R_q^m$ for all $i \in [\ell], k \in [r]$ and $j > 2$. Then we set $\beta_j^i = \sum_{k=1}^r \bar{\boldsymbol{\lambda}}_k^i \beta_{j,k}^i$ for $j > 2$, and $\beta_i^1 = \sum_{j=1}^{t_i'} \sum_{k=1}^r \bar{\boldsymbol{\lambda}}_k^i (T\boldsymbol{f}_{k,j}^i - \boldsymbol{A}_k \boldsymbol{s}_j^i) - \sum_{j=2}^{t_j'} \beta_j^i$ for all $i \in [\ell]$. By the same invariant, we can rewrite β_i^1 as

$$\beta_i^1 = \sum_{k=1}^{r} \bar{\boldsymbol{\lambda}}_k^i (T\boldsymbol{f}_k^i - \boldsymbol{A}_k \boldsymbol{s}) - \boldsymbol{c}_i' - \sum_{j=2}^{t_j'} \beta_j^i$$

where $\boldsymbol{f}_k^i = \sum_{j=1}^{t_i'} \boldsymbol{f}_{k,j}^i$, and $c_i' = \sum_{j=1}^r \sum_{k \text{ corrupt}} -\bar{\boldsymbol{\lambda}}_j^i \boldsymbol{A}_j s_k^i = -\sum_{i=1}^r \bar{\boldsymbol{\lambda}}_j^i \boldsymbol{c}_i$.

We can apply the same partial hybrid technique to show that **Hybrid 2** and **Hybrid 3** are indistinguishable.

Hybrid 4. In this hybrid, we set $\beta_i^1 = \sum_{k=1}^r \bar{\boldsymbol{\lambda}}_k^i (T\boldsymbol{f}_k^i - \boldsymbol{A}_k \boldsymbol{s}) - \boldsymbol{c}_i'$, and sample uniformly random $\beta_j^i \xleftarrow{u} R_q^m$. This hybrid is indistinguishable from **Hybrid 3** by the pseudorandomness of MLWE samples and by the fact that $\bar{\boldsymbol{\lambda}}^i \in \mathbb{Z}_q^m \setminus \{\boldsymbol{0}\}$ and q is a prime.

After **Hybrid 4**, the view becomes

$$(\boldsymbol{A}_1, \ldots, \boldsymbol{A}_r, \alpha_1^1 := \boldsymbol{A}_1\boldsymbol{s} + T\boldsymbol{e}_1 + \boldsymbol{x}_1 - \boldsymbol{c}_1, \alpha_2^1, \ldots, \alpha_{t_1}^1, \ldots, \alpha_1^r := \boldsymbol{A}_r\boldsymbol{s} + T\boldsymbol{e}_r + \boldsymbol{x}_r - \boldsymbol{c}_r, \alpha_2^r, \ldots, \alpha_{t_r}^r,$$
$$\beta_1^1 := \sum_{i=1}^{r} \bar{\boldsymbol{\lambda}}_i^1 (T\boldsymbol{f}_i^1 - \boldsymbol{A}_i\boldsymbol{s} + \boldsymbol{c}_i), \beta_2^1, \ldots, \beta_{t_r'}^1, \ldots, \beta_1^\ell := \sum_{i=1}^{r} \bar{\boldsymbol{\lambda}}_i^\ell (T\boldsymbol{f}_i^\ell - \boldsymbol{A}_i\boldsymbol{s} + \boldsymbol{c}_\ell), \beta_2^\ell, \ldots, \beta_{t_r'}^\ell)$$

where α_j^i and β_k^i are independent uniformly random variables over R_q^m for all $j > 2$ and $k > 2$.

Hybrid 5. In this hybrid, we sample uniformly random α_1^i for all $i \in [r]$, and set β_1^k to $\sum_{i=1}^r \bar{\boldsymbol{\lambda}}_i^k (T\boldsymbol{f}_i^k + T\boldsymbol{e}_i - \alpha_1^i) + \langle \bar{\boldsymbol{\lambda}}^i, \boldsymbol{x} \rangle$ for all $k \in [\ell]$. Because all α_1^i, β_1^k are independent of other random variables. So by Lemma 1, this hybrid is indistinguishable from **Hybrid 4**.

The last hybrid **Hybrid 5** can be computed from the simulator's input. Therefore, the security claim follows.

□

Client $C_{i,j}$

Public parameters:

- Clients' public keys (via PKI)
- Input domain $\mathbb{F}^l$
- Key sharing parameter d
- Additive secret sharing scheme AShare, threshold secret sharing scheme TShare.
- Chaperone parameter h
- The ith instruction, $t_i = (\mathsf{Mode}_i, I_i, \{\lambda_{i,r}\}_{r<i})$, from the program P to be executed
- A list of dropped clients $\mathcal{D}_i$ indexed by the ith cohort
- A means of generating public random matrices A_i indexed by corresponding round i
- A pseudorandom generator $\mathsf{PRG} : \mathbb{Z}_q \to R_q^m$.
- A sate Z (initialized to 0) that aggregates missing secret shares.
- Discrete Gaussian distribution D_σ for generating noise.

Private input: Possibly an input vector $\boldsymbol{x}_{i,j} := I_i(D_{i,j})$ based on private data $D_{i,j}$.

1. Receive messages from $\mathcal{S}$:
   - Unless $i = 1$, receive additive encrypted shares of key from C_{i-1}
   - **[Dropout Recovery]** Encrypted threshold shares from C_{i-2} that recover the dropped clients in C_{i-1}
   - **[Remove Mask]** Encrypted threshold shares from non-dropped clients of C_{i-1}
2. Local computation:
   - If $i = 1$, sample uniformly random $s_{1,j}$ from $\mathbb{Z}_q^*$.
   - If $i > 1$, decrypt shares and compute $s_{i,j} \leftarrow \sum_r s_{i-1,j}^r$
   - Compute $(s_{i,j}^1, \ldots, s_{i,j}^d) \leftarrow \mathsf{AShare}(s_{i,j})$
   - For each $s_{i,j}^k$, $k \in [d]$, compute $(t_{i,j}^{k,1}, \ldots, t_{i,j}^{k,h}) \leftarrow \mathsf{TShare}(s_{i,j}^k)$
   - Sample uniformly random secret $b_{i,j}$ from $\mathbb{Z}_q$ and compute $\mathsf{MASK}_j^i \leftarrow \mathsf{PRG}(b_{i,j})$
   - Compute $(u_{i,j}^1, \ldots, u_{i,j}^h) \leftarrow \mathsf{TShare}(b_{i,j})$
   - If $\mathsf{Mode}_i = \mathsf{Store}$, generate and set $M_i = A_i$. Sample $g^i \leftarrow D_\sigma$.
   - If $\mathsf{Mode}_i = \mathsf{Reveal}$, set $M_i = -\sum_{k<i} \lambda_k^i A_k$. Sample $g^i \leftarrow \sum_{k<i} \lambda_k^i D_\sigma$.
   - Compute $w_j^i = M_i s_{i,j} + T g^i + \mathsf{MASK}_j^i$.
3. Send to the server:
   - Send encrypted $(s_{i,j}^1, \ldots, s_{i,j}^d)$ to d random clients in C_{i+1} via server
   - For each $k \in [d]$, send encrypted $(t_{i,j}^{k,1}, \ldots, t_{i,j}^{k,h})$ to h random chaperones of client k in C^{i+2} via server.
   - Send encrypted $(u_{i,j}^1, \ldots, u_{i,j}^h)$ to h random chaperones of client j in C^{i+1} via server.
   - The decryption of all received threshold that server requested.
   - Upload w_j^i to server

Figure 5: Client j in cohort i: With dropout resilience

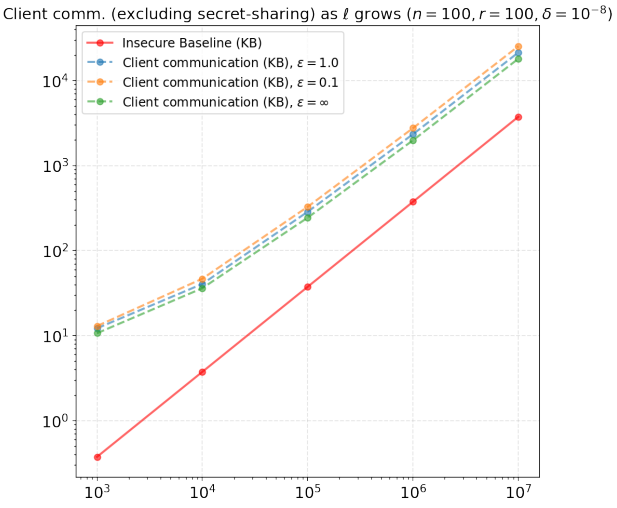

Figure 6: Client communication to server for several vector lengths for our protocol (1000 rounds), compared to the baseline where clients submit a vector in $\{0,1\}^\ell$ in the clear.