LightSecAgg: A Lightweight and Versatile Design for Secure Aggregation in Federated Learning

Jinhyun So [*1] **Chaoyang He** [*1] **Chien-Sheng Yang** [*2] **Songze Li** [3 4] **Qian Yu** [5] **Ramy E. Ali** [1] **Basak Guler** [6] **Salman Avestimehr** [1]

Abstract

Secure model aggregation is a key component of federated learning (FL) that aims at protecting the privacy of each user's individual model while allowing for their global aggregation. It can be applied to any aggregation-based FL approach for training a global or personalized model. Model aggregation needs to also be resilient against likely user dropouts in FL systems, making its design substantially more complex. State-of-the-art secure aggregation protocols rely on secret sharing of the random-seeds used for mask generations at the users to enable the reconstruction and cancellation of those belonging to the dropped users. The complexity of such approaches, however, grows substantially with the number of dropped users. We propose a new approach, named `LightSecAgg`, to overcome this bottleneck by changing the design from "random-seed reconstruction of the dropped users" to "one-shot aggregate-mask reconstruction of the active users via mask encoding/decoding". We show that `LightSecAgg` achieves the same privacy and dropout-resiliency guarantees as the state-of-the-art protocols while significantly reducing the overhead for resiliency against dropped users. We also demonstrate that, unlike existing schemes, `LightSecAgg` can be applied to secure aggregation in the *asynchronous* FL setting. Furthermore, we provide a modular system design and optimized on-device parallelization for scalable implementation, by enabling computational overlapping between model training and on-device encoding, as well as improving the speed of concurrent receiving and sending of chunked masks. We evaluate `LightSecAgg` via extensive experiments for training diverse models (logistic regression, shallow CNNs, MobileNetV3, and EfficientNet-B0) on various datasets (MNIST, FEMNIST, CIFAR-10, GLD-23K) in a realistic FL system with large number of users and demonstrate that `LightSecAgg` significantly reduces the total training time.

1 Introduction

Federated learning (FL) has emerged as a promising approach to enable distributed training over a large number of users while protecting the privacy of each user (McMahan et al., 2017; 2021; Wang et al., 2021). The key idea of FL is to keep users' data on their devices and instead train local models at each user. The locally trained models are then aggregated via a server to update a global model, which is then pushed back to users. Due to model inversion attacks (e.g., (Geiping et al., 2020; Wang et al., 2019; Zhu & Han, 2020)), a critical consideration in FL design is to also ensure that the server does not learn the locally trained model of each user during model aggregation. Furthermore, model aggregation should be robust against likely user dropouts (due to poor connectivity, low battery, unavailability, etc) in FL systems. As such, there have been a series of works that aim at developing secure aggregation protocols for FL that protect the privacy of each user's individual model while allowing their global aggregation amidst possible user dropouts (Bonawitz et al., 2017; Kadhe et al., 2020; So et al., 2021d).

The state-of-the-art secure aggregation protocols essentially rely on two main principles: (1) pairwise random-seed agreement between users to generate masks that hide users' models while having an additive structure that allows their cancellation when added at the server and (2) secret sharing of the random-seeds to enable the reconstruction and cancellation of masks belonging to dropped users. The main drawback of such approaches is that the number of mask reconstructions at the server substantially grows as

[*]Equal contribution [1]University of Southern California, Los Angeles, California, USA [2]Mediatek Inc., Hsinchu, Taiwan [3]Internet of Things Thrust, The Hong Kong University of Science and Technology (Guangzhou), Guangzhou, China [4]Department of Computer Science and Engineering, The Hong Kong University of Science and Technology, Hong Kong SAR, China [5]Electrical and Computer Engineering, Princeton University, New Jersey, USA [6]Department of Electrical and Computer Engineering, University of California at Riverside, Riverside, California, USA. Correspondence to: Jinhyun So <jinhyuns@usc.edu>, Chaoyang He <chaoyang.he@usc.edu>.

This paper is accepted to *Proceedings of the 5^{th} MLSys Conference*, Santa Clara, CA, USA, 2022.

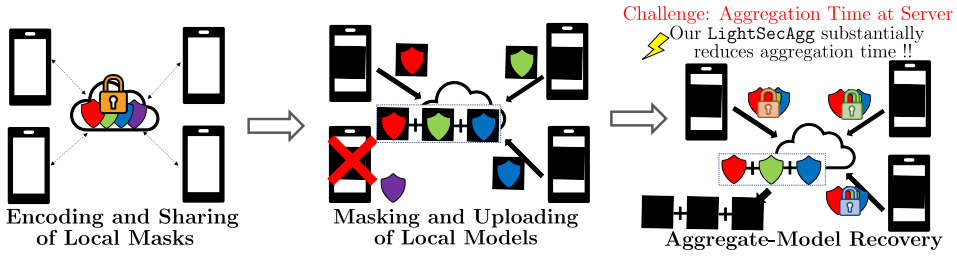


Figure 1. Illustration of our proposed LightSecAgg protocol. (1) **Sharing encoded mask**: users encode and share their generated local masks. (2) **Masking model**: each user masks its model by random masks, and uploads its masked model to the server. (3) **Reconstructing aggregate-mask**: The surviving users upload the aggregate of encoded masks to reconstruct the desired aggregate-mask. The server recovers the aggregate-model by canceling out the reconstructed aggregate-mask.

more users are dropped, causing a major computational bottleneck. For instance, the execution time of the SecAgg protocol proposed in (Bonawitz et al., 2017) is observed to be significantly limited by mask reconstructions at the server (Bonawitz et al., 2019b). SecAgg+ (Bell et al., 2020), an improved version of SecAgg, reduces the overhead at the server by replacing the complete communication graph of SecAgg with a sparse random graph, such that secret sharing is only needed within a subset of users rather than all users pairs. However, the number of mask reconstructions in SecAgg+ still increases as more users drop, eventually limits the scalability of FL systems. There have also been several other approaches, such as (So et al., 2021d; Kadhe et al., 2020), to alleviate this bottleneck, however they either increase round/communication complexity or compromise the dropout and privacy guarantees.

Contributions. We propose a new perspective for secure model aggregation in FL by turning the design focus from "pairwise random-seed reconstruction of the dropped users" to "one-shot aggregate-mask reconstruction of the surviving users". Using this viewpoint, we develop a new protocol named LightSecAgg that provides the same level of privacy and dropout-resiliency guarantees as the state-of-the-art while substantially reducing the aggregation (hence runtime) complexity. As illustrated in Figure 1, the main idea of LightSecAgg is that each user protects its local model using a locally generated random mask. This mask is then encoded and shared to other users in such a way that the aggregate-mask of any sufficiently large set of surviving users can be directly reconstructed at the server. In sharp contrast to prior schemes, in this approach the server only needs to reconstruct *one* mask in the recovery phase, independent of the number of dropped users.

Moreover, we provide a modular federated training system design and optimize on-device parallelization to improve the efficiency when secure aggregation and model training interact at the edge devices. This enables computational overlapping between model training and on-device encoding, as well as improving the speed of concurrent receiving and sending of chunked masks. To the best of our knowledge, this provides the first open-sourced and secure aggregation-enabled FL system that is built on the modern deep learning framework (PyTorch) and neural architecture (e.g., ResNet) with system and security co-design.

We further propose system-level optimization methods to improve the run-time. In particular, we design a federated training system and take advantage of the fact that the generation of random masks is independent of the computation of the local model, hence each user can parallelize these two operations via a multi-thread processing, which is beneficial to all evaluated secure aggregation protocols in reducing the total running time.

In addition to the synchronous FL setting, where all users train local models based on the same global model and the server performs a synchronized aggregation at each round, we also demonstrate that LightSecAgg enables secure aggregation when no synchrony between users' local updates are imposed. This is unlike prior secure aggregation protocols, such as SecAgg and SecAgg+, that are not compatible with asynchronous FL. To the best of our knowledge, in the asynchronous FL setting, this is the first work to protect the privacy of the individual updates without relying on differential privacy (Truex et al., 2020) or trusted execution environments (TEEs) (Nguyen et al., 2021).

We run extensive experiments to empirically demonstrate the performance of LightSecAgg in a real-world cross-device FL setting with up to 200 users and compare it with two state-of-the-art protocols SecAgg and SecAgg+. To provide a comprehensive coverage of realistic FL settings, we train various machine learning models including logistic regression, convolutional neural network (CNN) (McMahan et al., 2017), MobileNetV3 (Howard et al., 2019), and EfficientNet-B0 (Tan & Le, 2019), for image classification over datasets of different image sizes: low resolution images (FEMNIST (Caldas et al., 2018), CIFAR-10 (Krizhevsky et al., 2009)), and high resolution images (Google Landmarks Dataset 23k (Weyand et al., 2020)). The empirical results show that LightSecAgg provides significant speedup for all considered FL training tasks, achieving a

performance gain of 8.5×-12.7× over SecAgg and 2.9×-4.4× over SecAgg+, in realistic bandwidth settings at the users. Hence, compared to baselines, LightSecAgg can even survive and speedup the training of large deep neural network models on high resolution image datasets. Breakdowns of the total running time further confirm that the primary gain lies in the complexity reduction at the server provided by LightSecAgg, especially when the number of users are large.

Related works. Beyond the secure aggregation protocols proposed in (Bonawitz et al., 2017; Bell et al., 2020), there have been also other works that aim towards making secure aggregation more efficient. TurboAgg (So et al., 2021d) utilizes a circular communication topology to reduce the communication overhead, but it incurs an additional round complexity and provides a weaker privacy guarantee than SecAgg as it guarantees model privacy in the average sense rather than in the worst-case scenario. FastSecAgg (Kadhe et al., 2020) reduces per-user overhead by using the Fast Fourier Transform multi-secret sharing, but it provides lower privacy and dropout guarantees compared to the other state-of-the-art protocols. The idea of one-shot reconstruction of the aggregate-mask was also employed in (Zhao & Sun, 2021), where the aggregated masks corresponding to each user dropout pattern was prepared by a trusted third party, encoded and distributed to users prior to model aggregation. The major advantages of LightSecAgg over the scheme in (Zhao & Sun, 2021) are 1) not requiring a trusted third party; and 2) requiring significantly less randomness generation and a much smaller storage cost at each user. Furthermore, there is also a lack of system-level performance evaluations of (Zhao & Sun, 2021) in FL experiments. Finally, we emphasize that our LightSecAgg protocol can be applied to any aggregation-based FL approach (e.g., FedNova (Wang et al., 2020), FedProx (Li et al., 2018), FedOpt (Asad et al., 2020)), personalized FL frameworks (T. Dinh et al., 2020; Li et al., 2020; Fallah et al., 2020; Mushtaq et al., 2021; He et al., 2021e), communication-efficient FL (Shlezinger et al., 2020; Reisizadeh et al., 2020; Elkordy & Avestimehr, 2020), and asynchronous FL, and their applications in computer vision (He et al., 2021d; 2020a;b), natural language processing (Lin et al., 2021; He et al., 2021c), data mining (He et al., 2021a; Ezzeldin et al., 2021; Liang et al., 2021; He et al., 2021b; 2020c), and Internet of things (IoTs) (Zhang et al., 2021a;b).

2 Problem Setting

FL is a distributed training framework for machine learning, where the goal is to learn a global model $\mathbf{x}$ with dimension d using data held at edge devices. This can be represented by minimizing a global objective function F: $F(\mathbf{x}) = \sum_{i=1}^{N} p_i F_i(\mathbf{x})$, where N is the total number of users, F_i is the local objective function of user i, and $p_i \geq 0$ is a weight parameter assigned to user i to specify the relative impact of each user such that $\sum_{i=1}^{N} p_i = 1$.[1]

Training in FL is performed through an iterative process, where the users interact through a server to update the global model. At each iteration, the server shares the current global model, denoted by $\mathbf{x}(t)$, with the edge users. Each user i creates a local update, $\mathbf{x}_i(t)$. The local models are sent to the server and then aggregated by the server. Using the aggregated models, the server updates the global model $\mathbf{x}(t+1)$ for the next iteration. In FL, some users may potentially drop from the learning procedure for various reasons such as having unreliable communication connections. The goal of the server is to obtain the sum of the surviving users' local models. This update equation is given by $\mathbf{x}(t+1) = \frac{1}{|\mathcal{U}(t)|} \sum_{i \in \mathcal{U}(t)} \mathbf{x}_i(t)$, where $\mathcal{U}(t)$ denotes the set of surviving users at iteration t. Then, the server pushes the updated global model $\mathbf{x}(t+1)$ to the edge users.

Local models carry extensive information about the users' datasets, and in fact their private data can be reconstructed from the local models by using a model inversion attack (Geiping et al., 2020; Wang et al., 2019; Zhu & Han, 2020). To address this privacy leakage from local models, secure aggregation has been introduced in (Bonawitz et al., 2017). A secure aggregation protocol enables the computation of the aggregated global model while ensuring that the server (and other users) learn no information about the local models beyond their aggregated model. In particular, the goal is to securely recover the aggregate of the local models $\mathbf{y} = \sum_{i \in \mathcal{U}} \mathbf{x}_i$, where the iteration index t is omitted for simplicity. Since secure aggregation protocols build on cryptographic primitives that require all operations to be carried out over a finite field, we assume that the elements of $\mathbf{x}_i$ and $\mathbf{y}$ are from a finite field $\mathbb{F}_q$ for some field size q. We require a secure aggregation protocol for FL to have the following key features.

- **Threat model and privacy guarantee.** We consider a threat model where the users and the server are honest but curious. We assume that up to T (out of N) users can collude with each other as well as with the server to infer the local models of other users. The secure aggregation protocol has to guarantee that nothing can be learned beyond the aggregate-model, even if up to T users cooperate with each other. We consider privacy leakage in the strong information-theoretic sense. This requires that for every subset of users $\mathcal{T} \subseteq [N]$ of size at most T, we must have mutual information $I(\{\mathbf{x}_i\}_{i \in [N]}; \mathbf{Y} | \sum_{i \in \mathcal{U}} \mathbf{x}_i, \mathbf{Z}_{\mathcal{T}}) = 0$, where $\mathbf{Y}$ is the collection of information at the server, and $\mathbf{Z}_{\mathcal{T}}$ is the collection of information at the users in $\mathcal{T}$.

[1] For simplicity, we assume that all users have equal-sized datasets, i.e., $p_i = \frac{1}{N}$ for all $i \in [N]$.

- **Dropout-resiliency guarantee.** In the FL setting, it is common for users to be dropped or delayed at any time during protocol execution for various reasons, e.g., delayed/interrupted processing, poor wireless channel conditions, low battery, etc. We assume that there are at most D dropped users during the execution of protocol, i.e., there are at least $N - D$ surviving users after potential dropouts. The protocol must guarantee that the server can correctly recover the aggregated models of the surviving users, even if up to D users drop.

- **Applicability to asynchronous FL.** Synchronizing all users for training at each round of FL can be slow and costly, especially when the number of users are large. Asynchronous FL handles this challenge by incorporating the updates of the users in asynchronous fashion (Xie et al., 2019; van Dijk et al., 2020; Chai et al., 2020; Chen et al., 2020). This asynchrony, however, creates a mismatch of staleness among the users, which causes the incompatibility of the existing secure aggregation protocols (such as (Bonawitz et al., 2017; Bell et al., 2020)). More specifically, since it is not known a priori which local models will be aggregated together, the current secure aggregation protocols that are based on pairwise random masking among the users fail to work. We aim at designing a versatile secure aggregation protocol that is applicable to both synchronous and asynchronous FL.

Goal. We aim to design an efficient and scalable secure aggregation protocol that simultaneously achieves strong privacy and dropout-resiliency guarantees, scaling linearly with the number of users N, e.g., simultaneously achieves privacy guarantee $T = \frac{N}{2}$ and dropout-resiliency guarantee $D = \frac{N}{2} - 1$. Moreover, the protocol should be compatible with both synchronous and asynchronous FL.

3 Overview of Baseline Protocols: SecAgg and SecAgg+

We first review the state-of-the-art secure aggregation protocols SecAgg (Bonawitz et al., 2017) and SecAgg+ (Bell et al., 2020) as our baselines. Essentially, SecAgg and SecAgg+ require each user to mask its local model using random keys before aggregation. In SecAgg, the privacy of the individual models is protected by pairwise random masking. Through a key agreement (e.g., Diffie-Hellman (Diffie & Hellman, 1976)), each pair of users $i, j \in [N]$ agree on a pairwise random seed $a_{i,j} = \text{Key.Agree}(sk_i, pk_j) = \text{Key.Agree}(sk_j, pk_i)$ where sk_i and pk_i are the private and public keys of user i, respectively. In addition, user i creates a private random seed b_i to prevent the privacy breaches that may occur if user i is only delayed rather than dropped, in which case the pairwise masks alone are not sufficient for privacy protection. User $i \in [N]$ then masks its model $\mathbf{x}_i$ as $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \text{PRG}(b_i) + \sum_{j:i<j} \text{PRG}(a_{i,j}) - \sum_{j:i>j} \text{PRG}(a_{j,i})$, where PRG is a pseudo random generator, and sends it to the server. Finally, user i secret shares its private seed b_i as well as private key sk_i with the other users via Shamir's secret sharing (Yao, 1982). From the subset of users who survived the previous stage, the server collects either the shares of the private key belonging to a dropped user, or the shares of the private seed belonging to a surviving user (but not both). Using the collected shares, the server reconstructs the private seed of each surviving user, and the pairwise seeds of each dropped user. The server then computes the aggregated model as follows

$$\sum_{i\in\mathcal{U}} \mathbf{x}_i = \sum_{i\in\mathcal{U}} (\tilde{\mathbf{x}}_i - \text{PRG}(b_i)) + \sum_{i\in\mathcal{D}} \left(\sum_{j:i<j} \text{PRG}(a_{i,j}) - \sum_{j:i>j} \text{PRG}(a_{j,i}) \right), \quad (1)$$

where $\mathcal{U}$ and $\mathcal{D}$ represent the set of surviving and dropped users, respectively. SecAgg protects model privacy against T colluding users and is robust to D user dropouts as long as $N - D > T$.

We now illustrate SecAgg through a simple example. Consider a secure aggregation problem in FL, where there are $N = 3$ users with $T = 1$ privacy guarantee and dropout-resiliency guarantee $D = 1$. Each user $i \in \{1, 2, 3\}$ holds a local model $\mathbf{x}_i \in \mathbb{F}_q^d$ where d is the model size and q is the size of the finite field. As shown in Figure 2, SecAgg is composed of the following three phases.

Offline pairwise agreement. User 1 and user 2 agree on pairwise random seed $a_{1,2}$. User 1 and user 3 agree on pairwise random seed $a_{1,3}$. User 2 and user 3 agree on pairwise random seed $a_{2,3}$. In addition, user $i \in \{1, 2, 3\}$ creates a private random seed b_i. Then, user i secret shares b_i and its private key sk_i with the other users via Shamir's secret sharing. In this example, a 2 out of 3 secret sharing is used to tolerate 1 curious user.

Masking and uploading of local models. To provide the privacy of each individual model, user $i \in \{1, 2, 3\}$ masks its model $\mathbf{x}_i$ as follows:

$$\tilde{\mathbf{x}}_1 = \mathbf{x}_1 + \mathbf{n}_1 + \mathbf{z}_{1,2} + \mathbf{z}_{1,3},$$
$$\tilde{\mathbf{x}}_2 = \mathbf{x}_2 + \mathbf{n}_2 + \mathbf{z}_{2,3} - \mathbf{z}_{1,2},$$
$$\tilde{\mathbf{x}}_3 = \mathbf{x}_3 + \mathbf{n}_3 - \mathbf{z}_{1,3} - \mathbf{z}_{2,3},$$

where $\mathbf{n}_i = \text{PRG}(b_i)$ and $\mathbf{z}_{i,j} = \text{PRG}(a_{i,j})$ are the random masks generated by a pseudo random generator. Then user $i \in \{1, 2, 3\}$ sends its masked local model $\tilde{\mathbf{x}}_i$ to the server.

Aggregate-model recovery. Suppose that user 1 drops in the previous phase. The goal of the server is to compute the aggregate of models $\mathbf{x}_2 + \mathbf{x}_3$. Note that

$$\mathbf{x}_2 + \mathbf{x}_3 = \tilde{\mathbf{x}}_2 + \tilde{\mathbf{x}}_3 + (\mathbf{z}_{1,2} + \mathbf{z}_{1,3} - \mathbf{n}_2 - \mathbf{n}_3). \quad (2)$$

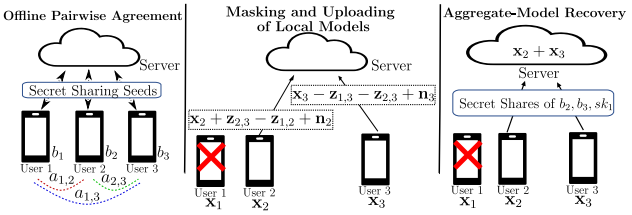


Figure 2. An illustration of SecAgg in the example of 3 users is depicted. The users first agree on pairwise random seeds, and secret share their private random seeds and private keys. The local models are protected by the pairwise random masking. Suppose that user 1 drops. To recover the aggregate-mask, the server first reconstructs the private random seeds of the surviving users and the private key of user 1 by collecting the secret shares for each of them. Then, the server recovers $\mathbf{z}_{1,2}$, $\mathbf{z}_{1,3}$, $\mathbf{n}_2$ and $\mathbf{n}_3$, which incurs the computational cost of $4d$ at the server.

Hence, the server needs to reconstruct masks $\mathbf{n}_2$, $\mathbf{n}_3$, $\mathbf{z}_{1,2}$, $\mathbf{z}_{1,3}$ to recover $\mathbf{x}_2 + \mathbf{x}_3$. To do so, the server has to collect two shares for each of b_2, b_3, sk_1, and then compute the aggregate model by (2). Since the complexity of evaluating a PRG scales linearly with its size, the computational cost of the server for mask reconstruction is $4d$.

We note that SecAgg requires the server to compute a PRG function on *each* of the reconstructed seeds to recover the aggregated masks, which incurs the overhead of $O(N^2)$ (see more details in Section 5) and dominates the overall execution time of the protocol (Bonawitz et al., 2017; 2019b). SecAgg+ reduces the overhead of mask reconstructions from $O(N^2)$ to $O(N \log N)$ by replacing the complete communication graph of SecAgg with a sparse random graph of degree $O(\log N)$ to reduce both communication and computational loads. Reconstructing pairwise random masks in SecAgg and SecAgg+ poses a major bottleneck in scaling to a large number of users.

Remark 1. *(Incompatibility of SecAgg and SecAgg+ with Asynchronous FL).* It is important to note that SecAgg and SecAgg+ cannot be applied to asynchronous FL as the cancellation of the pairwise random masks based on the key agreement protocol is not guaranteed. This is because the users do not know a priori which local models will be aggregated together, hence the masks cannot be designed to cancel out in these protocols. We explain this in more detail in Appendix F.2. It is also worth noting that a recently proposed protocol known as FedBuff (Nguyen et al., 2021) enables secure aggregation in asynchronous FL through a trusted execution environment (TEE)-enabled buffer, where the server stores the local models that it receives in this *private* buffer. The reliance of FedBuff on TEEs, however, limits the buffer size in this approach as TEEs have limited memory. It would also limit its application to FL settings where TEEs are available.

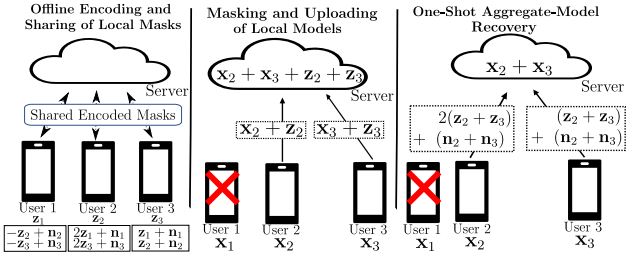


Figure 3. An illustration of LightSecAgg in the example of 3 users is depicted. Each user first generates a single mask. Each mask of a user is encoded and shared to other users. Each user's local model is protected by its generated mask. Suppose that user 1 drops during the execution of protocol. The server directly recovers the aggregate-mask in one shot. In this example, LightSecAgg reduces the computational cost at the server from $4d$ to d.

4 LightSecAgg Protocol

Before providing a general description of LightSecAgg, we first illustrate its key ideas through the previous 3-user example in the synchronous setting. As shown in Figure 3, LightSecAgg has the following three phases.

Offline encoding and sharing of local masks. User $i \in \{1,2,3\}$ randomly picks $\mathbf{z}_i$ and $\mathbf{n}_i$ from $\mathbb{F}_q^d$. User $i \in \{1,2,3\}$ creates the masked version of $\mathbf{z}_i$ as

$$\tilde{\mathbf{z}}_{1,1} = -\mathbf{z}_1 + \mathbf{n}_1,\ \tilde{\mathbf{z}}_{1,2} = 2\mathbf{z}_1 + \mathbf{n}_1,\ \tilde{\mathbf{z}}_{1,3} = \mathbf{z}_1 + \mathbf{n}_1;$$
$$\tilde{\mathbf{z}}_{2,1} = -\mathbf{z}_2 + \mathbf{n}_2,\ \tilde{\mathbf{z}}_{2,2} = 2\mathbf{z}_2 + \mathbf{n}_2,\ \tilde{\mathbf{z}}_{2,3} = \mathbf{z}_2 + \mathbf{n}_2;$$
$$\tilde{\mathbf{z}}_{3,1} = -\mathbf{z}_3 + \mathbf{n}_3,\ \tilde{\mathbf{z}}_{3,2} = 2\mathbf{z}_3 + \mathbf{n}_3,\ \tilde{\mathbf{z}}_{3,3} = \mathbf{z}_3 + \mathbf{n}_3;$$

and user $i \in \{1,2,3\}$ sends $\tilde{\mathbf{z}}_{i,j}$ to each user $j \in \{1,2,3\}$. Thus, user $i \in \{1,2,3\}$ receives $\tilde{\mathbf{z}}_{j,i}$ for $j \in \{1,2,3\}$. In this case, this procedure provides robustness against 1 dropped user and privacy against 1 curious user.

Masking and uploading of local models. To make each individual model private, each user $i \in \{1,2,3\}$ masks its local model as follows:

$$\tilde{\mathbf{x}}_1 = \mathbf{x}_1 + \mathbf{z}_1, \quad \tilde{\mathbf{x}}_2 = \mathbf{x}_2 + \mathbf{z}_2, \quad \tilde{\mathbf{x}}_3 = \mathbf{x}_3 + \mathbf{z}_3, \quad (3)$$

and sends its masked model to the server.

One-shot aggregate-model recovery. Suppose that user 1 drops in the previous phase. To recover the aggregate of models $\mathbf{x}_2 + \mathbf{x}_3$, the server only needs to know the aggregated masks $\mathbf{z}_2 + \mathbf{z}_3$. To recover $\mathbf{z}_2 + \mathbf{z}_3$, the surviving user 2 and user 3 send $\tilde{\mathbf{z}}_{2,2} + \tilde{\mathbf{z}}_{3,2}$ and $\tilde{\mathbf{z}}_{2,3} + \tilde{\mathbf{z}}_{3,3}$,

$$\tilde{\mathbf{z}}_{2,2} + \tilde{\mathbf{z}}_{3,2} = 2(\mathbf{z}_2 + \mathbf{z}_3) + \mathbf{n}_2 + \mathbf{n}_3,$$
$$\tilde{\mathbf{z}}_{2,3} + \tilde{\mathbf{z}}_{3,3} = (\mathbf{z}_2 + \mathbf{z}_3) + \mathbf{n}_2 + \mathbf{n}_3,$$

to the server, respectively. After receiving the messages from user 2 and user 3, the server can directly recover the aggregated masks via an one-shot computation as follows:

$$\mathbf{z}_2 + \mathbf{z}_3 = \tilde{\mathbf{z}}_{2,2} + \tilde{\mathbf{z}}_{3,2} - (\tilde{\mathbf{z}}_{2,3} + \tilde{\mathbf{z}}_{3,3}). \quad (4)$$

Then, the server recovers the aggregate-model $\mathbf{x}_2+\mathbf{x}_3$ by subtracting $\mathbf{z}_2+\mathbf{z}_3$ from $\tilde{\mathbf{x}}_2+\tilde{\mathbf{x}}_3$. As opposed to SecAgg which has to reconstruct the random seeds of the dropped users, LightSecAgg enables the server to reconstruct the desired aggregate of masks via a one-shot recovery. Compared with SecAgg, LightSecAgg reduces the server's computational cost from $4d$ to d in this simple example.

4.1 General Description of LightSecAgg for Synchronous FL

We formally present LightSecAgg, whose idea is to encode the local generated random masks in a way that the server can recover the aggregate of masks from the encoded masks via an one-shot computation with a cost that does not scale with N. LightSecAgg has three design parameters: (1) $0 \leq T \leq N-1$ representing the privacy guarantee; (2) $0 \leq D \leq N-1$ representing the dropout-resiliency guarantee; (3) $1 \leq U \leq N$ representing the targeted number of surviving users. In particular, parameters T, D, and U are selected such that $N-D \geq U > T \geq 0$.

LightSecAgg is composed of three main phases. First, each user first partitions its local random mask to $U-T$ pieces and creates encoded masks via a Maximum Distance Separable (MDS) code (Roth & Lempel, 1989; Yu et al., 2019; Tang et al., 2021; So et al., 2021c) to provide robustness against D dropped users and privacy against T colluding users. Each user sends one of the encoded masks to one of the other users for the purpose of one-shot recovery. Second, each user uploads its masked local model to the server. Third, the server reconstructs the aggregated masks of the surviving users to recover their aggregate of models. Each surviving user sends the aggregated encoded masks to the server. After receiving U aggregated encoded masks from the surviving users, the server recovers the aggregate-mask and the desired aggregate-model. The pseudo code of LightSecAgg is provided in Appendix A. We now describe each of these phases in detail.

Offline encoding and sharing of local masks. User $i \in [N]$ picks $\mathbf{z}_i$ uniformly at random from $\mathbb{F}_q^d$ and partitions it to $U-T$ sub-masks $[\mathbf{z}_i]_k \in \mathbb{F}_q^{\frac{d}{U-T}}$, $k \in [U-T]$. With the randomly picked $[\mathbf{n}_i]_k \in \mathbb{F}_q^{\frac{d}{U-T}}$ for $k \in \{U-T+1, \ldots, U\}$, user $i \in [N]$ encodes sub-masks $[\mathbf{z}_i]_k$'s as

$$[\tilde{\mathbf{z}}_i]_j = ([\mathbf{z}_i]_1, \ldots, [\mathbf{z}_i]_{U-T}, [\mathbf{n}_i]_{U-T+1}, \ldots, [\mathbf{n}_i]_U) \cdot W_j, \tag{5}$$

where W_j is j'th column of a T-*private* MDS matrix $W \in \mathbb{F}_q^{U \times N}$. In particular, we say an MDS matrix[2] is T-private iff the submatrix consisting of its $\{U-T+1, ..., U\}$-th rows is also MDS. A T-private MDS matrix guarantees that $I(\mathbf{z}_i; \{[\tilde{\mathbf{z}}_i]_j\}_{j \in \mathcal{T}}) = 0$, for any $i \in [N]$ and any $\mathcal{T} \subseteq [\mathcal{N}]$ of size T, if $[\mathbf{n}_i]_k$'s are jointly uniformly random. We can always find T-private MDS matrices for any U, N, and T (e.g., (Shamir, 1979; Yu et al., 2019; Roth & Lempel, 1989)). Each user $i \in [N]$ sends $[\tilde{\mathbf{z}}_i]_j$ to user $j \in [N] \backslash \{i\}$. In the end of offline encoding and sharing of local masks, each user $i \in [N]$ has $[\tilde{\mathbf{z}}_j]_i$ from $j \in [N]$.[3]

Masking and uploading of local models. To protect the local models, each user i masks its local model as $\tilde{\mathbf{x}}_i = \mathbf{x}_i + \mathbf{z}_i$, and sends it to the server. Since some users may drop in this phase, the server identifies the set of surviving users, denoted by $\mathcal{U}_1 \subseteq [N]$. The server intends to recover $\sum_{i \in \mathcal{U}_1} \mathbf{x}_i$. We note that before masking the model, each user quantizes the local model to convert from the domain of real numbers to the finite field (Appendix F.3.2).

One-shot aggregate-model recovery. After identifying the surviving users in the previous phase, user $j \in \mathcal{U}_1$ is notified to send its aggregated encoded sub-masks $\sum_{i \in \mathcal{U}_1} [\tilde{\mathbf{z}}_i]_j$ to the server for the purpose of one-shot recovery. We note that each $\sum_{i \in \mathcal{U}_1} [\tilde{\mathbf{z}}_i]_j$ is an encoded version of $\sum_{i \in \mathcal{U}_1} [\mathbf{z}_i]_k$ for $k \in [U-T]$ using the MDS matrix W (see more details in Appendix B). Thus, the server is able to recover $\sum_{i \in \mathcal{U}_1} [\mathbf{z}_i]_k$ for $k \in [U-T]$ via MDS decoding after receiving a set of any U messages from the participating users. The server obtains the aggregated masks $\sum_{i \in \mathcal{U}_1} \mathbf{z}_i$ by concatenating $\sum_{i \in \mathcal{U}_1} [\mathbf{z}_i]_k$'s. Lastly, the server recovers the desired aggregate of models for the set of participating users $\mathcal{U}_1$ by subtracting $\sum_{i \in \mathcal{U}_1} \mathbf{z}_i$ from $\sum_{i \in \mathcal{U}_1} \tilde{\mathbf{x}}_i$.

Remark 2. Note that it is not necessary to have a stable communication link between every pair of users in LightSecAgg. Specifically, given the design parameter U, LightSecAgg only requires at least U surviving users at any time during the execution. That is, even if up to $N-U$ users drop or get delayed due to unstable communication links, the server can still reconstruct the aggregate-mask.

Remark 3. We note that LightSecAgg directly applies for secure aggregation of weighted local models. The sharing of the masking keys among the clients does not require the knowledge of the weight coefficients. For example, LightSecAgg can work for the case in which all users do not have equal-sized datasets. Suppose that user i holds a dataset with a number of samples s_i. Rather than directly masking the local model $\mathbf{x}_i$, user i first computes $\mathbf{x}_i^{'} = s_i \mathbf{x}_i$. Then, user i uploads $\mathbf{x}_i^{'} + \mathbf{z}_i$ to the server. Through the LightSecAgg protocol, the server can recover $\sum_{i \in \mathcal{U}} \mathbf{x}_i^{'} = \sum_{i \in \mathcal{U}} s_i \mathbf{x}_i$ securely. By dividing by $\sum_{i \in \mathcal{U}} s_i$, the server can obtain the desired aggregate of weighted model $\sum_{i \in \mathcal{U}} p_i \mathbf{x}_i$ where $p_i = \frac{s_i}{\sum_{i \in \mathcal{U}} s_i}$.

[2] A matrix $W \in \mathbb{F}_q^{U \times N}$ $(U < N)$ is an MDS matrix if any $U \times U$ sub-matrix of W is non-singular.

[3] All users communicate through secure (private and authenticated) channels, i.e., the server would only receive the encrypted version of $[\tilde{\mathbf{z}}_i]_j$'s. Such secure communication is also used in prior works on secure aggregation, e.g., SecAgg, SecAgg+.

4.2 Extension to Asynchronous FL

We now describe how LightSecAgg can be applied to asynchronous FL. We consider the asynchronous FL setting with bounded staleness as considered in (Nguyen et al., 2021), where the updates of the users are not synchronized and the staleness of each user is bounded by $\tau_{\max}$. In this setting, the server stores the models that it receives in a buffer of size K and updates the global model once the buffer is full. More generally, LightSecAgg may apply to any asynchronous FL setting where a group of local models are aggregated at each round. That is, the group size does not need to be fixed in all rounds. While the baselines are not compatible with this setting, LightSecAgg can be applied by encoding the local masks in a way that the server can recover the aggregate of masks from the encoded masks via a one-shot computation, even though the masks are generated in different training rounds. Specifically, the users share the encoded masks with the timestamp to figure out which encoded masks should be aggregated for the reconstruction of the aggregate of masks. As the users aggregate the encoded masks after the server stores the local updates in the buffer, the users can aggregate the encoded masks according to the timestamp of the stored updates. Due to the commutative property of MDS coding and addition, the server can reconstruct the aggregate of masks even though the masks are generated in different training rounds. We postpone the detailed description of the LightSecAgg protocol for the asynchronous setting to Appendix F.

5 Theoretical Analysis

5.1 Theoretical Guarantees

We now state our main result for the theoretical guarantees of the LightSecAgg protocol.

Theorem 1. Consider a secure aggregation problem in federated learning with N users. Then, the proposed LightSecAgg protocol can *simultaneously* achieve (1) privacy guarantee against up to any T colluding users, and (2) dropout-resiliency guarantee against up to any D dropped users, for any pair of privacy guarantee T and dropout-resiliency guarantee D such that $T + D < N$.

The proof of Theorem 1, which is applicable to both synchronous and asynchronous FL settings, is presented in Appendix B.

Remark 4. Theorem 1 provides a trade-off between privacy and dropout-resiliency guarantees, i.e., LightSecAgg can increase the privacy guarantee by reducing the dropout-resiliency guarantee and vice versa. As SecAgg (Bonawitz et al., 2017), LightSecAgg achieves the worst-case dropout-resiliency guarantee. That is, for any privacy guarantee T and the number of dropped users $D < N - T$, LightSecAgg ensures that any set of dropped users of size D in secure aggregation can be tolerated. Differently, SecAgg+ (Bell et al., 2020), FastSecAgg (Kadhe et al., 2020), and TurboAgg (So et al., 2021d) relax the worst-case constraint to random dropouts and provide a probabilistic dropout-resiliency guarantee, i.e., the desired aggregate-model can be correctly recovered with high probability.

Remark 5. From the training convergence perspective, LightSecAgg only adds a quantization step to the local model updates of the users. The impact of this model quantization on FL convergence is well studied in the synchronous FL (Reisizadeh et al., 2020; Elkordy & Avestimehr, 2020). In the asyncrhonous FL, however, we need to analyze the convergence of LightSecAgg. We provide this analysis in the smooth and non-convex setting in Appendix F.4.

5.2 Complexity Analysis of LightSecAgg

We measure the storage cost, communication load, and computational load of LightSecAgg in units of elements or operations in $\mathbb{F}_q$ for a single training round. Recall that U is a design parameter chosen such that $N - D \geq U > T$.

Offline storage cost. Each user i independently generates a random mask $\mathbf{z}_i$ of length d. Additionally, each user i stores a coded mask $[\tilde{\mathbf{z}}_j]_i$ of size $\frac{d}{U-T}$, for $j \in [N]$. Hence, the total offline storage cost at each user is $(1 + \frac{N}{U-T})d$.

Offline communication and computation loads. For each iteration of secure aggregation, before the local model is computed, each user prepares offline coded random masks and distributes them to the other users. Specifically, each user encodes U local data segments with each of size $\frac{d}{U-T}$ into N coded segments and distributes each of them to one of N users. Hence, the offline computational and communication load of LightSecAgg at each user is $O(\frac{dN \log N}{U-T})$ and $O(\frac{dN}{U-T})$, respectively.

Communication load during aggregation. While each user uploads a masked model of length d, in the phase of aggregate-model recovery, no matter how many users drop, each surviving user in $\mathcal{U}_1$ sends a coded mask of size $\frac{d}{U-T}$. The server is guaranteed to recover the aggregate-model of the surviving users in $\mathcal{U}_1$ after receiving messages from any U users. The total required communication load at the server in the phase of mask recovery is therefore $\frac{U}{U-T}d$.

Computation load during aggregation. The major computational bottleneck of LightSecAgg is the decoding process to recover $\sum_{j\in\mathcal{U}_1} \mathbf{z}_j$ at the server. This involves decoding a U-dimensional MDS code from U coded symbols, which can be performed with $O(U \log U)$ operations on elements in $\mathbb{F}_q$, hence a total computational load of $\frac{U \log U}{U-T}d$.

We compare the communication and computational complexities of LightSecAgg with baseline protocols. In particular, we consider the case where secure aggregation

Table 1. Complexity comparison between SecAgg, SecAgg+, and LightSecAgg. Here N is the total number of users, d is the model size, s is the length of the secret keys as the seeds for PRG ($s \ll d$). In the table, U stands for User and S stands for Server.

	SecAgg	SecAgg+	LightSecAgg
offline comm. (U)	$O(sN)$	$O(s\log N)$	$O(d)$
offline comp. (U)	$O(dN + sN^2)$	$O(d\log N + s\log^2 N)$	$O(d\log N)$
online comm. (U)	$O(d + sN)$	$O(d + s\log N)$	$O(d)$
online comm. (S)	$O(dN + sN^2)$	$O(dN + sN\log N)$	$O(dN)$
online comp. (U)	$O(d)$	$O(d)$	$O(d)$
reconstruction (S)	$O(dN^2)$	$O(dN\log N)$	$O(d\log N)$

protocols aim at providing privacy guarantee $T = \frac{N}{2}$ and dropout-resiliency guarantee $D = pN$ simultaneously for some $0 \le p < \frac{1}{2}$. As shown in Table 1, by choosing $U = (1-p)N$, LightSecAgg significantly improves the computational efficiency at the server during aggregation. SecAgg and SecAgg+ incurs a total computational load of $O(dN^2)$ and $O(dN\log N)$, respectively at the server, while the server complexity of LightSecAgg remains nearly constant with respect to N. It is expected to substantially reduce the overall aggregation time for a large number of users, which is bottlenecked by the server's computation in SecAgg (Bonawitz et al., 2017; 2019b). More detailed discussions, as well as a comparison with another recently proposed secure aggregation protocol (Zhao & Sun, 2021), which achieves similar server complexity as LightSecAgg, are carried out in Appendix C.

6 System Design and Optimization

Apart from theoretical design and analysis, we have further designed a FL training system to reduce the overhead of secure model aggregation and enable realistic evaluation of LightSecAgg in cross-device FL.

The software architecture is shown in Figure 4. In order to keep the software architecture lightweight and maintainable, we do not over-design and only modularize the system as the foundation layer and the algorithm layer.

The foundation layer (blocks below the dashed line) contains the communicator and training engine. The communicator can support multiple communication protocols (PyTorch RPC (trp, 2021), and gRPC (grp, 2021)), but it provides a unified communication interface for the algorithmic layer. In the training engine, in addition to standard PyTorch for GPU, we also compile the ARM-based PyTorch for embedded edge devices (e.g., Raspberry Pi).

In the algorithm layer, Client Manager calls Trainer in the foundation layer to perform on-device training. Client Manager also integrates Client Encoder to complete the secure aggregation protocol, which is supported by security primitive APIs. In Server Manager, Secure Aggregator maintains the cache

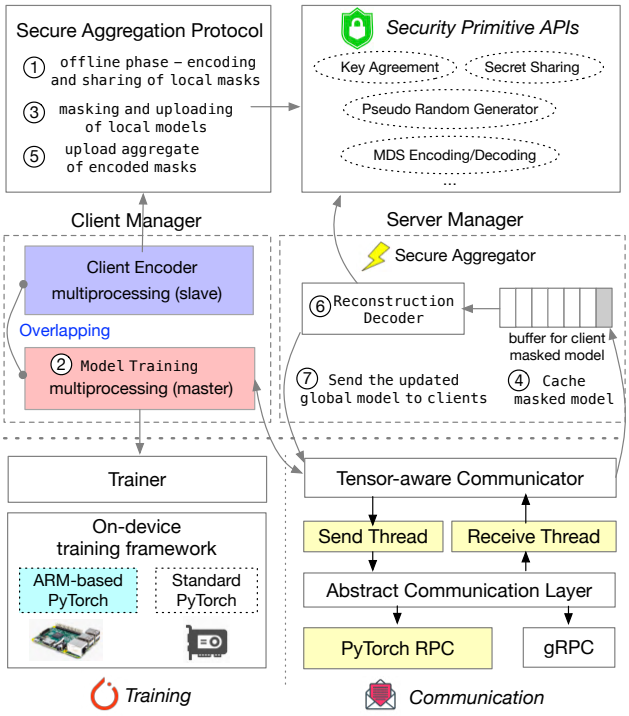


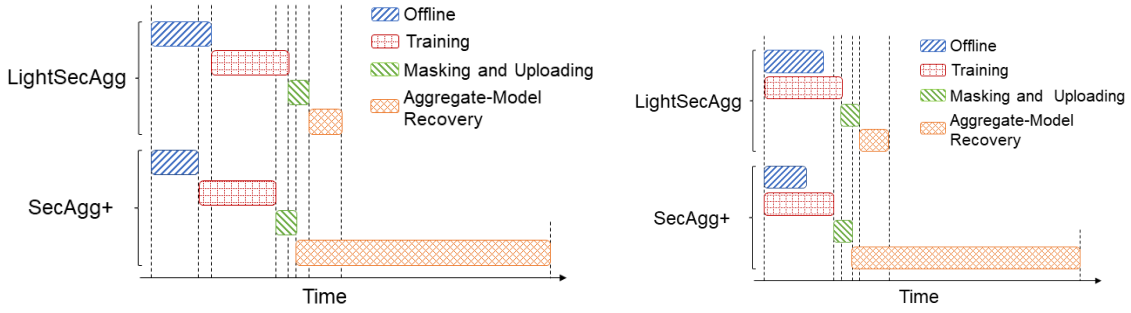
Figure 4. Overview of the System Design

for masked models, and once the cache is full, it starts reconstruction based on aggregated masks uploaded by clients. The server then synchronizes the updated global model to clients for the next round of training. In Figure 4, we mark the 7 sequential steps in a single FL round as circled numbers to clearly show the interplay between federated training and secure aggregation protocol.

This software architecture has two special designs that can further reduce the computational and communication overhead of the secure aggregation protocol.

Parallelization of offline phase and model training. We note that for all considered protocols, LightSecAgg, SecAgg, and SecAgg+, the communication and computation time to generate and exchange the random masks in the offline phase can be *overlapped* with model training. Hence, in our design, we reduce the offline computation and communication overhead by allowing each user to train the model and carry out the offline phase simultaneously by running two parallel processes (multi-threading performs relatively worse due to Python GIL, Global Interpreter Lock), as shown as purple and red colors in Figure 4. We also demonstrate the timing diagram of the overlapped implementation in a single FL training round in Figure 5. We will analyze its impact on overall acceleration in section 7.2.

Optimized federated training system and communication APIs via tensor-aware RPC (Remote Procedure Call). As the yellow blocks in Figure 4 show, we specially design the sending and receiving queues to accelerate the



(a) Non-overlapped (b) Overlapped

Figure 5. The timing diagram of the overlapped implementation in LightSecAgg and SecAgg+ (Bell et al., 2020) for a single FL training round to train MobileNetV3 (Howard et al., 2019) with CIFAR-100 dataset (Krizhevsky et al., 2009). SecAgg (Bonawitz et al., 2017) is not included as it takes much longer than other two protocols.

Table 2. Summary of four implemented machine learning tasks and performance gain of LightSecAgg with respect to SecAgg and SecAgg+. All learning tasks are for image classification. MNIST, FEMNIST and CIFAR-10 are low-resolution datasets, while images in GLD-23K are high resolution, which cost much longer training time; LR and CNN are shallow models, but MobileNetV3 and EfficientNet-B0 are much larger models, but they are tailored for efficient edge training and inference.

No.	Dataset	Model	Model Size (d)	Gain		
				Non-overlapped	Overlapped	Aggregation-only
1	MNIST (LeCun et al., 1998)	Logistic Regression	7,850	6.7×, 2.5×	8.0×, 2.9×	13.0×, 4.1×
2	FEMNIST (Caldas et al., 2018)	CNN (McMahan et al., 2017)	1,206,590	11.3×, 3.7×	12.7×, 4.1×	13.2×, 4.2×
3	CIFAR-10 (Krizhevsky et al., 2009)	MobileNetV3 (Howard et al., 2019)	3,111,462	7.6×, 2.8×	9.5×, 3.3×	13.1×, 3.9×
4	GLD-23K (Weyand et al., 2020)	EfficientNet-B0 (Tan & Le, 2019)	5,288,548	3.3×, 1.6×	3.4×, 1.7×	13.0×, 4.1×

scenario that the device has to be sender and receiver simultaneously. As such, the offline phase of LightSecAgg can further be accelerated by parallelizing the transmission and reception of $[\tilde{\mathbf{z}}_i]_j$. This design can also speed up the offline pairwise agreement in SecAgg and SecAgg+. Moreover, we choose PyTorch RPC (trp, 2021) as the communication backend rather than gRPC (grp, 2021) and MPI (mpi) because its tensor-aware communication API can reduce the latency in scenarios where the communicator is launched frequently, i.e., each client in the offline mask exchanging phase needs to distribute N coded segments to N users.

With the above design, we can deploy LightSecAgg in both embedded IoT devices and AWS EC2 instances. AWS EC2 instances can also represent a realistic cross-device setting because, in our experiments, we use AWS EC2 m3.medium instances, which are CPU-based and have the same hardware configuration as modern smartphones such as iOS and Android devices. Furthermore, we package our system as a Docker image to simplify the system deployment to hundreds of edge devices.

7 Experimental Results

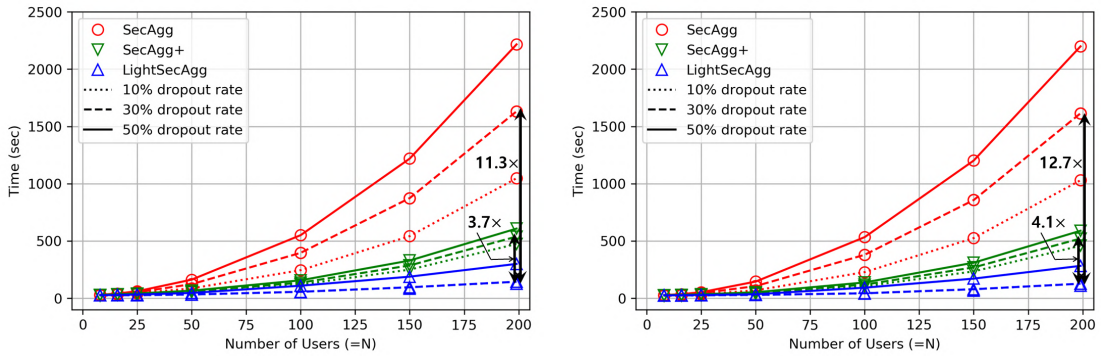
7.1 Setup

Dataset and models. To provide a comprehensive coverage of realistic FL settings, we train four models over computer vision datasets of different sizes, summarized in Table 2. The hyper-parameter settings are provided in Appendix D.

Dropout rate. To model the dropped users, we randomly select pN users where p is the dropout rate. We consider the worst-case scenario (Bonawitz et al., 2017), where the selected pN users artificially drop after uploading the masked model. All three protocols provide privacy guarantee $T = \frac{N}{2}$ and resiliency for three different dropout rates, $p = 0.1$, $p = 0.3$, and $p = 0.5$, which are realistic values according to the industrial observation in real FL system (Bonawitz et al., 2019a). As we can see that when carefully selecting devices which may be stable online during the time period of training, the dropout rate is as high as 10%; when considering intermittently connected devices, only up to 10K devices can participate simultaneously when there are 10M daily active devices ($1 : 1000$).

Number of users and Communication Bandwidth. In our experiments, we train up to $N = 200$ users. The measured real bandwidth is 320Mb/s. We also consider two other bandwidth settings of 4G (LTE-A) and 5G cellular networks as we discuss later.

Baselines. We analyze and compare the performance of LightSecAgg with two baseline schemes: SecAgg and SecAgg+ described in Section 3. While there are also other secure aggregation protocols (e.g., TurboAgg (So et al., 2021d) and FastSecAgg (Kadhe et al., 2020)), we use SecAgg and SecAgg+ for our baselines since other



(a) Non-overlapped (b) Overlapped

Figure 6. Total running time of LightSecAgg versus the state-of-the-art protocols (SecAgg and SecAgg+) to train CNN (McMahan et al., 2017) on the FEMNIST dataset (Caldas et al., 2018), as the number of users increases, for various dropout rates.

schemes weaken the privacy guarantees as we discussed in Related Works part of Section 1.

7.2 Overall Evaluation and Performance Analysis

For the performance analysis, we measure the total running time for a single round of global iteration which includes model training and secure aggregation with each protocol while increasing the number of users N gradually for different user dropouts. Our results from training CNN (McMahan et al., 2017) on the FEMNIST dataset (Caldas et al., 2018) are demonstrated in Figure 6. The performance gain of LightSecAgg with respect to SecAgg and SecAgg+ to train the other models is also provided in Table 2. More detailed experimental results are provided in Appendix D. We make the following key observations.

Impact of dropout rate: the total running time of SecAgg and SecAgg+ increases monotonically with the dropout rate. This is because their total running time is dominated by the mask recovery at the server, which increases quadratically with the number of users.

Non-overlapping v.s. Overlapping: In the non-overlapped implementation, LightSecAgg provides a speedup of up to $11.3\times$ and $3.7\times$ over SecAgg and SecAgg+, respectively, by significantly reducing the server's execution time; in the overlapped implementation, LightSecAgg provides a further speedup of up to $12.7\times$ and $4.1\times$ over SecAgg and SecAgg+, respectively. This is due to the fact that LightSecAgg requires more communication and a higher computational cost in the offline phase than the baseline protocols, and the overlapped implementation helps to mitigate this extra cost.

Impact of model size: LightSecAgg provides a significant speedup of the aggregate-model recovery phase at the server over the baseline protocols in all considered model sizes. When training EfficientNet-B0 on GLD-23K dataset, LightSecAgg provides the smallest speedup in the most training-intensive task. This is because training time is dominant in this task, and training takes almost the same time in LightSecAgg and baseline protocols.

Aggregation-only: When comparing the aggregation time only, the speedup remains the same for various model sizes as shown in Table 2. We note that speeding up the aggregation phase by itself is still very important because local training and aggregation phases are not necessarily happening one immediately after the other. For example, local training may be done sporadically and opportunistically throughout the day (whenever resources are available), while global aggregation may be postponed to a later time when a large fraction of the users are done with local training, and they are available for aggregation (e.g., 2 am).

Impact of U: LightSecAgg incurs the smallest running time for the case when $p = 0.3$, which is almost identical to the case when $p = 0.1$. Recall that LightSecAgg can select the design parameter U between $T = 0.5N$ and $N - D = (1 - p)N$. Within this range, while increasing U reduces the size of the symbol to be decoded, it also increases the complexity of decoding each symbol. The experimental results suggest that the optimal choices for the cases of $p = 0.1$ and $p = 0.3$ are both $U = \lfloor 0.7N \rfloor$, which leads to a faster execution than when $p = 0.5$, where U can only be chosen as $U = 0.5N + 1$.

Table 3. Performance gain in different bandwidth settings.

Protocols	4G (98 Mbps)	320 Mbps	5G (802 Mbps)
SecAgg	8.5×	12.7×	13.5×
SecAgg+	2.9×	4.1×	4.4×

Impact of Bandwidth: We have also analyzed the impact of communication bandwidth at the users. In addition to the default bandwidth setting used in this section, we have considered two other edge scenarios: 4G (LTE-A) and 5G cellular networks using realistic bandwidth settings of 98 and 802 Mbps respectively (Minovski et al., 2021; Scheuner & Leitner, 2018)). The results are reported in Table 3 for a single FL round to train CNN over FEMNIST.

Table 4. Breakdown of the running time (sec) of LightSecAgg and the state-of-the-art protocols (SecAgg (Bonawitz et al., 2017) and SecAgg+ (Bell et al., 2020)) to train CNN (McMahan et al., 2017) on the FEMNIST dataset (Caldas et al., 2018) with $N = 200$ users, for dropout rate $p = 10\%, 30\%, 50\%$.

Protocols	Phase	Non-overlapped			Overlapped		
		$p = 10\%$	$p = 30\%$	$p = 50\%$	$p = 10\%$	$p = 30\%$	$p = 50\%$
LightSecAgg	Offline	69.3	69.0	191.2	75.1	74.9	196.9
	Training	22.8	22.8	22.8			
	Uploading	12.4	12.2	21.6	12.6	12.0	21.4
	Recovery	40.9	40.7	64.5	40.7	41.0	64.9
	Total	145.4	144.7	300.1	123.4	127.3	283.2
SecAgg	Offline	95.6	98.6	102.6	101.2	102.3	101.3
	Training	22.8	22.8	22.8			
	Uploading	10.7	10.9	11.0	10.9	10.8	11.2
	Recovery	911.4	1499.2	2087.0	911.2	1501.3	2086.8
	Total	1047.5	1631.5	2216.4	1030.3	1614.4	2198.9
SecAgg+	Offline	67.9	68.1	69.2	73.9	73.8	74.2
	Training	22.8	22.8	22.8			
	Uploading	10.7	10.8	10.7	10.7	10.8	10.9
	Recovery	379.1	436.7	495.5	378.9	436.7	497.3
	Total	470.5	538.4	608.2	463.6	521.3	582.4

7.3 Performance Breakdown

To further investigate the primary gain of LightSecAgg, we provide the breakdown of total running time for training CNN (McMahan et al., 2017) on the FEMNIST dataset (Caldas et al., 2018) in Table 4. The breakdown of the running time confirms that the primary gain lies in the complexity reduction at the server provided by LightSecAgg, especially for a large number of users.

7.4 Convergence Performance in Asynchronous FL

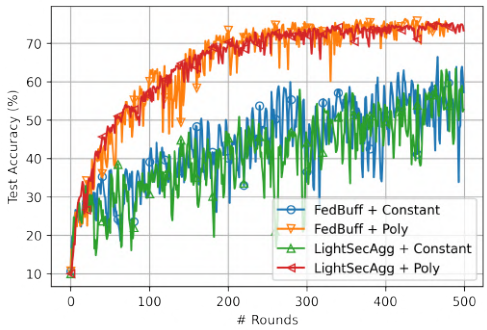


Figure 7. Accuracy of asynchronous LightSecAgg and FedBuff on CIFAR-10 dataset (Krizhevsky et al., 2009) with two strategies for mitigating the staleness: a constant function $s(\tau) = 1$ named *Constant*; and a polynomial function $s_\alpha(\tau) = (1+\tau)^{-\alpha}$ named *Poly* where $\alpha = 1$. The accuracy is reasonable since we use a variant of LeNet-5 (Xie et al., 2019).

As described in Remark 1, SecAgg and SecAgg+ are not applicable to asynchronous FL, and hence we cannot compare the total running time of LightSecAgg with these baseline secure aggregation protocols. As such, in our experiments here we instead focus on convergence performance of LightSecAgg compared to FedBuff (Nguyen et al., 2021) to investigate the impact of asynchrony and quantization in performance. In Figure 7, we demonstrate that LightSecAgg has almost the same performance as FedBuff on CIFAR-10 dataset while LightSecAgg includes quantization noise to protect the privacy of individual local updates of users. The details of the experiment setting and additional experiments for asynchronous FL are provided in Appendix F.5.

8 Conclusion and Future Works

This paper proposed LightSecAgg, a new approach for secure aggregation in synchronous and asynchronous FL. Compared with the state-of-the-art protocols, LightSecAgg reduces the overhead of model aggregation in FL by leveraging one-shot aggregate-mask reconstruction of the surviving users, while providing the same privacy and dropout-resiliency guarantees. In a realistic FL framework, via extensive empirical results it is also shown that LightSecAgg can provide substantial speedup over baseline protocols for training diverse machine learning models. While we focused on privacy in this work (under the honest but curious threat model), an interesting future research is to combine LightSecAgg with state-of-the-art Byzantine robust aggregation protocols (e.g., (He et al., 2020d; So et al., 2021b; Elkordy et al., 2021; Karimireddy et al., 2021)) to also mitigate Byzantine users while ensuring privacy.

REFERENCES

Open mpi: Open source high performance computing. https://grpc.io/.

gRPC: A high performance, open source universal RPC framework. https://grpc.io/, 2021.

Pytorch rpc: Distributed deep learning built on tensor-optimized remote procedure calls. https://pytorch.org/docs/stable/rpc.html, 2021.

Alistarh, D., Grubic, D., Li, J., Tomioka, R., and Vojnovic, M. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pp. 1709–1720, 2017.

Asad, M., Moustafa, A., and Ito, T. Fedopt: Towards communication efficiency and privacy preservation in federated learning. *Applied Sciences*, 10(8):2864, 2020.

Bell, J. H., Bonawitz, K. A., Gascón, A., Lepoint, T., and Raykova, M. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1253–1269, 2020.

Bonawitz, K., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H. B., Patel, S., Ramage, D., Segal, A., and Seth, K. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečnỳ, J., Mazzocchi, S., McMahan, H. B., et al. Towards federated learning at scale: System design. *arXiv preprint arXiv:1902.01046*, 2019a.

Bonawitz, K., Eichner, H., Grieskamp, W., Huba, D., Ingerman, A., Ivanov, V., Kiddon, C., Konečný, J., Mazzocchi, S., McMahan, B., Van Overveldt, T., Petrou, D., Ramage, D., and Roselander, J. Towards federated learning at scale: System design. In *Proceedings of Machine Learning and Systems*, volume 1, pp. 374–388, 2019b. URL https://proceedings.mlsys.org/paper/2019/file/bd686fd640be98efaae0091fa301e613-Paper.pdf.

Bonawitz, K., Salehi, F., Konečnỳ, J., McMahan, B., and Gruteser, M. Federated learning with autotuned communication-efficient secure aggregation. In *2019 53rd Asilomar Conference on Signals, Systems, and Computers*, pp. 1222–1226. IEEE, 2019c.

Caldas, S., Duddu, S. M. K., Wu, P., Li, T., Konečnỳ, J., McMahan, H. B., Smith, V., and Talwalkar, A. Leaf: A benchmark for federated settings. *arXiv preprint arXiv:1812.01097*, 2018.

Chai, Z., Chen, Y., Zhao, L., Cheng, Y., and Rangwala, H. FedAt: A communication-efficient federated learning method with asynchronous tiers under non-iid data. *arXiv preprint arXiv:2010.05958*, 2020.

Chen, Y., Ning, Y., Slawski, M., and Rangwala, H. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pp. 15–24. IEEE, 2020.

Diffie, W. and Hellman, M. New directions in cryptography. *IEEE transactions on Information Theory*, 22(6):644–654, 1976.

Elkordy, A. R. and Avestimehr, A. S. Secure aggregation with heterogeneous quantization in federated learning. *arXiv preprint arXiv:2009.14388*, 2020.

Elkordy, A. R., Prakash, S., and Avestimehr, A. S. Basil: A fast and byzantine-resilient approach for decentralized training. *arXiv preprint arXiv:2109.07706*, 2021.

Ezzeldin, Y. H., Yan, S., He, C., Ferrara, E., and Avestimehr, S. Fairfed: Enabling group fairness in federated learning. *ICML 2021 - International Workshop on Federated Learning for User Privacy and Data Confidentiality*, 2021.

Fallah, A., Mokhtari, A., and Ozdaglar, A. Personalized federated learning with theoretical guarantees: A model-agnostic meta-learning approach. *Advances in Neural Information Processing Systems*, 33, 2020.

Geiping, J., Bauermeister, H., Dröge, H., and Moeller, M. Inverting gradients - how easy is it to break privacy in federated learning? In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 16937–16947. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/c4ede56bbd98819ae6112b20ac6bf145-Paper.pdf.

He, C., Annavaram, M., and Avestimehr, S. Group knowledge transfer: Federated learning of large cnns at the edge. *NeurIPS 2020 (Advances in Neural Information-Processing Systems 2020)*, 2020a.

He, C., Annavaram, M., and Avestimehr, S. Fednas: Federated deep learning via neural architecture search. *CVPR 2020 Workshop on Neural Architecture Search and Beyond for Representation Learning*, pp. arXiv–2004, 2020b.

He, C., Tan, C., Tang, H., Qiu, S., and Liu, J. Central server free federated learning over single-sided trust social networks. *NeurIPS 2020 (Advances in Neural Information Processing Systems 2020) - Federated Learning Workshop*, 2020c.

He, C., Balasubramanian, K., Ceyani, E., Yang, C., Xie, H., Sun, L., He, L., Yang, L., Yu, P. S., Rong, Y., et al. Fedgraphnn: A federated learning system and benchmark for graph neural networks. *DPML@ICLR 2021 and GNNSys@MLSys 2021*, 2021a.

He, C., Ceyani, E., Balasubramanian, K., Annavaram, M., and Avestimehr, S. Spreadgnn: Serverless multi-task federated learning for graph neural networks. *International Workshop on Federated Learning for User Privacy and Data Confidentiality in Conjunction with ICML 2021 (FL-ICML'21) and Deep Learning on Graphs: Method and Applications with KDD 2021 (DLG-KDD'21)*, 2021b.

He, C., Li, S., Soltanolkotabi, M., and Avestimehr, S. Pipetransformer: Automated elastic pipelining for distributed training of large-scale models. In *International Conference on Machine Learning*, pp. 4150–4159. PMLR, 2021c.

He, C., Shah, A. D., Tang, Z., Sivashunmugam, D. F. N., Bhogaraju, K., Shimpi, M., Shen, L., Chu, X., Soltanolkotabi, M., and Avestimehr, S. Fedcv: A federated learning framework for diverse computer vision tasks. *arXiv preprint arXiv:2111.11066*, 2021d.

He, C., Yang, Z., Mushtaq, E., Lee, S., Soltanolkotabi, M., and Avestimehr, S. Ssfl: Tackling label deficiency in federated learning via personalized self-supervision. *arXiv preprint arXiv:2110.02470*, 2021e.

He, L., Karimireddy, S. P., and Jaggi, M. Secure byzantine-robust machine learning. *arXiv preprint arXiv:2006.04747*, 2020d.

Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., et al. Searching for mobilenetv3. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 1314–1324, 2019.

Kadhe, S., Rajaraman, N., Koyluoglu, O. O., and Ramchandran, K. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2009.11248*, 2020.

Karimireddy, S. P., He, L., and Jaggi, M. Learning from history for byzantine robust optimization. In *International Conference on Machine Learning*, pp. 5311–5319. PMLR, 2021.

Krizhevsky, A., Hinton, G., et al. Learning multiple layers of features from tiny images. 2009.

LeCun, Y., Bottou, L., Bengio, Y., and Haffner, P. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

Li, T., Sahu, A. K., Zaheer, M., Sanjabi, M., Talwalkar, A., and Smith, V. Federated optimization in heterogeneous networks. *arXiv preprint arXiv:1812.06127*, 2018.

Li, T., Hu, S., Beirami, A., and Smith, V. Ditto: Fair and robust federated learning through personalization. *arXiv: 2012.04221*, 2020.

Li, X., Huang, K., Yang, W., Wang, S., and Zhang, Z. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2019.

Liang, J., Li, S., Jiang, W., Cao, B., and He, C. Omnilytics: A blockchain-based secure data market for decentralized machine learning. *ICML 2021 - International Workshop on Federated Learning for User Privacy and Data Confidentiality*, 2021.

Lin, B. Y., He, C., Zeng, Z., Wang, H., Huang, Y., Soltanolkotabi, M., Ren, X., and Avestimehr, S. Fednlp: A research platform for federated learning in natural language processing. *arXiv preprint arXiv:2104.08815*, 2021.

McMahan, B., Moore, E., Ramage, D., Hampson, S., and y Arcas, B. A. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*, pp. 1273–1282. PMLR, 2017.

McMahan, H. B. et al. Advances and open problems in federated learning. *Foundations and Trends® in Machine Learning*, 14(1), 2021.

Minovski, D., Ogren, N., Ahlund, C., and Mitra, K. Throughput prediction using machine learning in lte and 5g networks. *IEEE Transactions on Mobile Computing*, 2021.

Mushtaq, E., He, C., Ding, J., and Avestimehr, S. Spider: Searching personalized neural architecture for federated learning. *arXiv preprint arXiv:2112.13939*, 2021.

Nguyen, J., Malik, K., Zhan, H., Yousefpour, A., Rabbat, M., Esmaeili, M. M., and Huba, D. Federated learning with buffered asynchronous aggregation. *arXiv preprint arXiv:2106.06639*, 2021.

Reddi, S., Charles, Z., Zaheer, M., Garrett, Z., Rush, K., Konečnỳ, J., Kumar, S., and McMahan, H. B. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

Reisizadeh, A., Mokhtari, A., Hassani, H., Jadbabaie, A., and Pedarsani, R. Fedpaq: A communication-efficient federated learning method with periodic averaging and quantization. In *International Conference on Artificial Intelligence and Statistics*, pp. 2021–2031. PMLR, 2020.

Roth, R. M. and Lempel, A. On mds codes via cauchy matrices. *IEEE transactions on information theory*, 35 (6):1314–1319, 1989.

Scheuner, J. and Leitner, P. A cloud benchmark suite combining micro and applications benchmarks. In *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, pp. 161–166, 2018.

Shamir, A. How to share a secret. *Communications of the ACM*, 22(11):612–613, 1979.

Shlezinger, N., Chen, M., Eldar, Y. C., Poor, H. V., and Cui, S. Uveqfed: Universal vector quantization for federated learning. *IEEE Transactions on Signal Processing*, 69: 500–514, 2020.

So, J., Ali, R. E., Guler, B., Jiao, J., and Avestimehr, S. Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning. *arXiv preprint arXiv:2106.03328*, 2021a.

So, J., Güler, B., and Avestimehr, A. S. Byzantine-resilient secure federated learning. *IEEE Journal on Selected Areas in Communications*, 39(7):2168–2181, 2021b.

So, J., Güler, B., and Avestimehr, A. S. Codedprivateml: A fast and privacy-preserving framework for distributed machine learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):441–451, 2021c.

So, J., Güler, B., and Avestimehr, A. S. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):479–489, 2021d.

T. Dinh, C., Tran, N., and Nguyen, J. Personalized federated learning with moreau envelopes. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M. F., and Lin, H. (eds.), *Advances in Neural Information Processing Systems*, volume 33, pp. 21394–21405. Curran Associates, Inc., 2020. URL https://proceedings.neurips.cc/paper/2020/file/f4f1f13c8289ac1b1ee0ff176b56fc60-Paper.pdf.

Tan, M. and Le, Q. Efficientnet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning*, pp. 6105–6114. PMLR, 2019.

Tang, T., Ali, R. E., Hashemi, H., Gangwani, T., Avestimehr, S., and Annavaram, M. Verifiable coded computing: Towards fast, secure and private distributed machine learning. *arXiv preprint arXiv:2107.12958*, 2021.

Truex, S., Liu, L., Chow, K.-H., Gursoy, M. E., and Wei, W. Ldp-fed: Federated learning with local differential privacy. In *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, pp. 61–66, 2020.

van Dijk, M., Nguyen, N. V., Nguyen, T. N., Nguyen, L. M., Tran-Dinh, Q., and Nguyen, P. H. Asynchronous federated learning with reduced number of rounds and with differential privacy from less aggregated gaussian noise. *arXiv preprint arXiv:2007.09208*, 2020.

Wang, J., Liu, Q., Liang, H., Joshi, G., and Poor, H. V. Tackling the objective inconsistency problem in heterogeneous federated optimization. *arXiv preprint arXiv:2007.07481*, 2020.

Wang, J., Charles, Z., Xu, Z., Joshi, G., McMahan, H. B., Al-Shedivat, M., Andrew, G., Avestimehr, S., Daly, K., Data, D., et al. A field guide to federated optimization. *arXiv preprint arXiv:2107.06917*, 2021.

Wang, Z., Song, M., Zhang, Z., Song, Y., Wang, Q., and Qi, H. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE Conference on Computer Communications*, pp. 2512–2520. IEEE, 2019.

Weyand, T., Araujo, A., Cao, B., and Sim, J. Google landmarks dataset v2-a large-scale benchmark for instance-level recognition and retrieval. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 2575–2584, 2020.

Xie, C., Koyejo, S., and Gupta, I. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

Yao, A. C. Protocols for secure computations. In *23rd annual symposium on foundations of computer science (sfcs 1982)*, pp. 160–164. IEEE, 1982.

Yu, Q., Li, S., Raviv, N., Kalan, S. M. M., Soltanolkotabi, M., and Avestimehr, S. A. Lagrange coded computing: Optimal design for resiliency, security, and privacy. In *The 22nd International Conference on Artificial Intelligence and Statistics*, pp. 1215–1225. PMLR, 2019.

Zhang, T., He, C., Ma, T., Gao, L., Ma, M., and Avestimehr, A. S. Federated learning for internet of things. *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021a.

Zhang, T., He, C., Ma, T., Gao, L., Ma, M., and Avestimehr, A. S. Federated learning for internet of things. *Proceedings of the 19th ACM Conference on Embedded Networked Sensor Systems*, 2021b.

Zhao, Y. and Sun, H. Information theoretic secure aggregation with user dropouts. *arXiv preprint arXiv:2101.07750*, 2021.

Zhu, L. and Han, S. Deep leakage from gradients. In *Federated Learning*, pp. 17–31. Springer, 2020.

APPENDIX

A PSEUDO CODE OF LIGHTSECAGG

Algorithm 1 The LightSecAgg protocol

Input: T (privacy guarantee), D (dropout-resiliency guarantee), U (target number of surviving users)

Server Executes:
// phase: offline encoding and sharing of local masks
for each user $i = 1, 2, \ldots, N$ **in parallel do**
  $\mathbf{z}_i \leftarrow$ randomly picks from $\mathbb{F}_q^d$
  $[\mathbf{z}_i]_1, \ldots, [\mathbf{z}_i]_{U-T} \leftarrow$ obtained by partitioning $\mathbf{z}_i$ to $U-T$ pieces
  $[\mathbf{n}_i]_{U-T+1}, \ldots, [\mathbf{n}_i]_U \leftarrow$ randomly picks from $\mathbb{F}_q^{\frac{d}{U-T}}$
  $\{[\tilde{\mathbf{z}}_i]_j\}_{j\in[N]} \leftarrow$ obtained by encoding $[\mathbf{z}_i]_k$'s and $[\mathbf{n}_i]_k$'s using (5)
  sends encoded mask $[\tilde{\mathbf{z}}_i]_j$ to user $j \in [N]\backslash\{i\}$
  receives encoded mask $[\tilde{\mathbf{z}}_j]_i$ from user $j \in [N]\backslash\{i\}$
end for
// phase: masking and uploading of local models
for each user $i = 1, 2, \ldots, N$ **in parallel do**
  // user i obtains $\mathbf{x}_i$ after the local update
  $\tilde{\mathbf{x}}_i \leftarrow \mathbf{x}_i + \mathbf{z}_i$ // masks the local model
  uploads masked model $\tilde{\mathbf{x}}_i$ to the server
end for
identifies set of surviving users $\mathcal{U}_1 \subseteq [N]$
gathers masked models $\tilde{\mathbf{x}}_i$ from user $i \in \mathcal{U}_1$
// phase: one-shot aggregate-model recovery
for each user $i \in \mathcal{U}_1$ **in parallel do**
  computes aggregated encoded masks $\sum_{j\in\mathcal{U}_1}[\tilde{\mathbf{z}}_j]_i$
  uploads aggregated encoded masks $\sum_{j\in\mathcal{U}_1}[\tilde{\mathbf{z}}_j]_i$ to the server
end for
collects U messages of aggregated encoded masks $\sum_{j\in\mathcal{U}_1}[\tilde{\mathbf{z}}_j]_i$ from user $i \in \mathcal{U}_1$
// recovers the aggregated-mask
$\sum_{i\in\mathcal{U}_1}\mathbf{z}_i \leftarrow$ obtained by decoding the received U messages
// recovers the aggregate-model for the surviving users
$\sum_{i\in\mathcal{U}_1}\mathbf{x}_i \leftarrow \sum_{i\in\mathcal{U}_1}\tilde{\mathbf{x}}_i - \sum_{i\in\mathcal{U}_1}\mathbf{z}_i$

B PROOF OF THEOREM 1

We prove the dropout-resiliency guarantee and the privacy guarantee for a single FL training round. As all randomness is independently generated across each round, one can extend the dropout-resiliency guarantee and the privacy guarantee for all training rounds for both synchronous and asynchronous FL setting. For simplicity, round index t is omitted in this proof.

For any pair of privacy guarantee T and dropout-resiliency guarantee D such that $T + D < N$, we select an arbitrary U such that $N - D \geq U > T$. In the following, we show that LightSecAgg with chosen design parameters T, D and U can simultaneously achieve (1) privacy guarantee against up to any T colluding users, and (2) dropout-resiliency guarantee against up to any D dropped users. We denote the concatenation of $\{[\mathbf{n}_i]_k\}_{k\in U-T+1,\ldots,U}$ by $\mathbf{n}_i$ for $i \in [N]$.

(Dropout-resiliency guarantee) We now focus on the phase of one-shot aggregate-model recovery. Since each user encodes its sub-masks by the same MDS matrix W, each $\sum_{i\in\mathcal{U}_1}[\tilde{\mathbf{z}}_i]_j$ is an encoded version of $\sum_{i\in\mathcal{U}_1}[\mathbf{z}_i]_k$ for $k \in [U-T]$ and $\sum_{i\in\mathcal{U}_1}[\mathbf{n}_i]_k$ for $k \in \{U-T+1, \ldots, U\}$ as follows:

$$\sum_{i\in\mathcal{U}_1}[\tilde{\mathbf{z}}_i]_j = (\sum_{i\in\mathcal{U}_1}[\mathbf{z}_i]_1, \ldots, \sum_{i\in\mathcal{U}_1}[\mathbf{z}_i]_{U-T}, \sum_{i\in\mathcal{U}_1}[\mathbf{n}_i]_{U-T+1}, \ldots, \sum_{i\in\mathcal{U}_1}[\mathbf{n}_i]_U) \cdot W_j, \tag{6}$$

where W_j is the j'th column of W.

Since $N-D \geq U$, there are at least U surviving users after user dropouts. Thus, the server is able to recover $\sum_{i\in\mathcal{U}_1}[\mathbf{z}_i]_k$ for $k \in [U-T]$ via MDS decoding after receiving a set of any U messages from the surviving users. Recall that $[\mathbf{z}_i]_k$'s are sub-masks of $\mathbf{z}_i$, so the server can successfully recover $\sum_{i\in\mathcal{U}_1}\mathbf{z}_i$. Lastly, the server recovers the aggregate-model for the set of surviving users $\mathcal{U}_1$ by $\sum_{i\in\mathcal{U}_1}\mathbf{x}_i = \sum_{i\in\mathcal{U}_1}\tilde{\mathbf{x}}_i - \sum_{i\in\mathcal{U}_1}\mathbf{z}_i = \sum_{i\in\mathcal{U}_1}(\mathbf{x}_i+\mathbf{z}_i) - \sum_{i\in\mathcal{U}_1}\mathbf{z}_i$.

(Privacy guarantee) We first present Lemma 1, whose proof is provided in Appendix E.

Lemma 1. For any $\mathcal{T} \subseteq [N]$ of size T and any $\mathcal{U}_1 \subseteq [N]$, $|\mathcal{U}_1| \geq U$ such that $U > T$, if the random masks $[\mathbf{n}_i]_k$'s are jointly uniformly random, we have

$$I(\{\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}; \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}) = 0. \tag{7}$$

We consider the worst-case scenario in which all the messages sent from the users are received by the server during the execution of LightSecAgg, i.e., the users identified as dropped are delayed. Thus, the server receives $\mathbf{x}_i+\mathbf{z}_i$ from user $i\in[N]$ and $\sum_{j\in\mathcal{U}_1}[\tilde{\mathbf{z}}_j]_i$ from user $i\in\mathcal{U}_1$. We now show that LightSecAgg provides privacy guarantee T, i.e., for an arbitrary set of colluding users $\mathcal{T}$ of size T, the following holds,

$$I\left(\{\mathbf{x}_i\}_{i\in[N]}; \{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]}, \{\sum_{j\in\mathcal{U}_1}[\tilde{\mathbf{z}}_j]_i\}_{i\in\mathcal{U}_1} \middle| \sum_{i\in\mathcal{U}_1}\mathbf{x}_i, \{\mathbf{x}_i\}_{i\in\mathcal{T}}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right) = 0. \tag{8}$$

We prove it as follows:

$$I\left(\{\mathbf{x}_i\}_{i\in[N]}; \{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]}, \{\sum_{j\in\mathcal{U}_1}[\tilde{\mathbf{z}}_j]_i\}_{i\in\mathcal{U}_1} \middle| \sum_{i\in\mathcal{U}_1}\mathbf{x}_i, \{\mathbf{x}_i\}_{i\in\mathcal{T}}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right) \tag{9}$$

$$=H\left(\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]}, \{\sum_{j\in\mathcal{U}_1}[\tilde{\mathbf{z}}_j]_i\}_{i\in\mathcal{U}_1} \middle| \sum_{i\in\mathcal{U}_1}\mathbf{x}_i, \{\mathbf{x}_i\}_{i\in\mathcal{T}}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)$$

$$-H\left(\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]}, \{\sum_{j\in\mathcal{U}_1}[\tilde{\mathbf{z}}_j]_i\}_{i\in\mathcal{U}_1} \middle| \{\mathbf{x}_i\}_{i\in[N]}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right) \tag{10}$$

$$=H\left(\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]}, \sum_{i\in\mathcal{U}_1}\mathbf{z}_i, \sum_{i\in\mathcal{U}_1}\mathbf{n}_i \middle| \sum_{i\in\mathcal{U}_1}\mathbf{x}_i, \{\mathbf{x}_i\}_{i\in\mathcal{T}}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)$$

$$-H\left(\{\mathbf{z}_i\}_{i\in[N]}, \sum_{i\in\mathcal{U}_1}\mathbf{z}_i, \sum_{i\in\mathcal{U}_1}\mathbf{n}_i \middle| \{\mathbf{x}_i\}_{i\in[N]}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right) \tag{11}$$

$$=H\left(\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}, \sum_{i\in\mathcal{U}_1}\mathbf{z}_i, \sum_{i\in\mathcal{U}_1}\mathbf{n}_i \middle| \sum_{i\in\mathcal{U}_1}\mathbf{x}_i, \{\mathbf{x}_i\}_{i\in\mathcal{T}}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)$$

$$-H\left(\{\mathbf{z}_i\}_{i\in[N]}, \sum_{i\in\mathcal{U}_1}\mathbf{z}_i, \sum_{i\in\mathcal{U}_1}\mathbf{n}_i \middle| \{\mathbf{x}_i\}_{i\in[N]}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right) \tag{12}$$

$$=H\left(\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}} \middle| \sum_{i\in\mathcal{U}_1}\mathbf{x}_i, \{\mathbf{x}_i\}_{i\in\mathcal{T}}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)$$

$$+H\left(\sum_{i\in\mathcal{U}_1}\mathbf{z}_i, \sum_{i\in\mathcal{U}_1}\mathbf{n}_i \middle| \{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}, \sum_{i\in\mathcal{U}_1}\mathbf{x}_i, \{\mathbf{x}_i\}_{i\in\mathcal{T}}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)$$

$$-H\left(\{\mathbf{z}_i\}_{i\in[N]} \middle| \{\mathbf{x}_i\}_{i\in[N]}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)$$

$$-H\left(\sum_{i\in\mathcal{U}_1}\mathbf{z}_i, \sum_{i\in\mathcal{U}_1}\mathbf{n}_i \middle| \{\mathbf{z}_i\}_{i\in[N]}, \{\mathbf{x}_i\}_{i\in[N]}, \{\mathbf{z}_i\}_{i\in\mathcal{T}}, \{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right) \tag{13}$$

$$
\begin{aligned}
=&H\left(\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}\middle|\sum_{i\in\mathcal{U}_1}\mathbf{x}_i,\{\mathbf{x}_i\}_{i\in\mathcal{T}},\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)\\
&+H\left(\sum_{i\in\mathcal{U}_1}\mathbf{n}_i\middle|\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}},\sum_{i\in\mathcal{U}_1}\mathbf{x}_i,\{\mathbf{x}_i\}_{i\in\mathcal{T}},\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)\\
&-H\left(\{\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}\middle|\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)-H\left(\sum_{i\in\mathcal{U}_1}\mathbf{n}_i\middle|\{\mathbf{z}_i\}_{i\in[N]},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right) \qquad (14)\\
=&H\left(\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}\middle|\sum_{i\in\mathcal{U}_1}\mathbf{x}_i,\{\mathbf{x}_i\}_{i\in\mathcal{T}},\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)\\
&+H\left(\sum_{i\in\mathcal{U}_1}\mathbf{n}_i\middle|\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}},\sum_{i\in\mathcal{U}_1}\mathbf{x}_i,\{\mathbf{x}_i\}_{i\in\mathcal{T}},\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right)\\
&-H\left(\{\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}\right)-H\left(\sum_{i\in\mathcal{U}_1}\mathbf{n}_i\middle|\{\mathbf{z}_i\}_{i\in[N]},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}\right) \qquad (15)\\
=&0, \qquad (16)
\end{aligned}
$$

where (11) follows from the fact that $\{\sum_{j\in\mathcal{U}_1}[\tilde{\mathbf{z}}_j]_i\}_{i\in\mathcal{U}_1}$ is invertible to $\sum_{i\in\mathcal{U}_1}\mathbf{z}_i$ and $\sum_{i\in\mathcal{U}_1}\mathbf{n}_i$. Equation (12) holds since $\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in\mathcal{T}}$ is a deterministic function of $\{\mathbf{z}_i\}_{i\in\mathcal{T}}$ and $\{\mathbf{x}_i\}_{i\in\mathcal{T}}$. Equation (13) follows from the chain rule. In equation (14), the second term follows from the fact that $\sum_{i\in\mathcal{U}_1}\mathbf{z}_i$ is a deterministic function of $\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}$, $\sum_{i\in\mathcal{U}_1}\mathbf{x}_i$, $\{\mathbf{x}_i\}_{i\in\mathcal{T}}$,$\{\mathbf{z}_i\}_{i\in\mathcal{T}}$; the third term follows from the independence of $\mathbf{x}_i$'s and $\mathbf{z}_i$'s; the last term follows from the fact that $\sum_{i\in\mathcal{U}_1}\mathbf{z}_i$ is a deterministic function of $\{\mathbf{z}_i\}_{i\in[N]}$ and the independence of $\mathbf{n}_i$'s and $\mathbf{x}_i$'s. In equation (15), the third term follows from Lemma 1. Equation (16) follows from 1) $\sum_{i\in\mathcal{U}_1}\mathbf{n}_i$ is a function of $\{\mathbf{x}_i+\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}$, $\sum_{i\in\mathcal{U}_1}\mathbf{x}_i$,$\{\mathbf{x}_i\}_{i\in\mathcal{T}}$, $\{\mathbf{z}_i\}_{i\in\mathcal{T}}$ and $\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}$; 2) $\sum_{i\in\mathcal{U}_1}\mathbf{n}_i$ is a function of $\{\mathbf{z}_i\}_{i\in\mathcal{U}_1}$ $\{[\tilde{\mathbf{z}}_j]_i\}_{j\in\mathcal{U}_1,i\in\mathcal{T}}$; 3) $\mathbf{z}_i$ is uniformly distributed and hence it has the maximum entropy in $\mathbb{F}_q^d$, combined with the non-negativity of mutual information.

C Discussion

As shown in Table 5, compared with the SecAgg protocol (Bonawitz et al., 2017), LightSecAgg significantly improves the computational efficiency at the server during aggregation. While SecAgg requires the server to retrieve $T+1$ secret shares of a secret key for *each* of the N users, and to compute a single PRG function if the user survives, or $N-1$ PRG functions to recover $N-1$ pairwise masks if the user drops off, yielding a total computational load of $O(N^2d)$ at the server. In contrast, as we have analyzed in Section 5.2, for $U=O(N)$, LightSecAgg incurs an almost constant ($O(d\log N)$) computational load at the server. This admits a scalable design and is expected to achieve a much faster end-to-end execution for a large number of users, given the fact that the overall execution time is dominated by the server's computation in SecAgg (Bonawitz et al., 2017; 2019b). SecAgg has a smaller storage overhead than LightSecAgg as secret shares of keys with small sizes (e.g., as small as an integer) are stored, and the model size d is much larger than the number of users N in typical FL scenarios. This effect will also allow SecAgg to have a smaller communication load in the phase of aggregate-model recovery. Finally, we would like to note that another advantage of LightSecAgg over SecAgg is the reduced dependence on cryptographic primitives such as public key infrastructure and key agreement mechanism, which further simplifies the implementation of the protocol. SecAgg+ (Bell et al., 2020) improves both communication and computational load of SecAgg by considering a sparse random graph of degree $O(\log N)$, and the complexity is reduced by factor of $O(\frac{N}{\log N})$. However, SecAgg+ still incurs $O(dN\log N)$ computational load at the server, which is much larger than $O(d\log N)$ computational load at the server in LightSecAgg when $U=O(N)$.

Compared with a recently proposed secure aggregation protocol in (Zhao & Sun, 2021), LightSecAgg achieves similar complexities in communication and computation during the aggregation process. The main advantage of LightSecAgg over the scheme in (Zhao & Sun, 2021) lies in how the randomness is generated and stored offline at the users and the resulting reduced storage cost. For the scheme in (Zhao & Sun, 2021), all randomness are generated at some external trusted party, and for each subset of $\mathcal{U}_1$ of size $|\mathcal{U}_1|\geq U$ the trusted party needs to generate T random symbols in $\mathbb{F}_q^{\frac{d}{U-T}}$, which account to a total amount of randomness that increases exponentially with N. In sharp contrast, LightSecAgg does not require a trusted third party, and each user generates *locally* a set of T random symbols. It significantly improves the

Table 5. Complexity comparison between SecAgg (Bonawitz et al., 2017), SecAgg+ (Bell et al., 2020), and LightSecAgg. Here N is the total number of users. The parameters d and s respectively represent the model size and the length of the secret keys as the seeds for PRG, where $s \ll d$. LightSecAgg and SecAgg provide *worst-case* privacy guarantee T and dropout-resiliency guarantee D for any T and D as long as $T + D < N$. SecAgg+ provides *probabilistic* privacy guarantee T and dropout-resiliency guarantee D. LightSecAgg selects three design parameters T, D and U such that $T < U \leq N - D$.

	SecAgg	SecAgg+	LightSecAgg
Offline storage per user	$O(d + Ns)$	$O(d + s\log N)$	$O(d + \frac{N}{U-T}d)$
Offline communication per user	$O(sN)$	$O(s\log N)$	$O(d\frac{N}{U-T})$
Offline computation per user	$O(dN + sN^2)$	$O(d\log N + s\log^2 N)$	$O(d\frac{N\log N}{U-T})$
Online communication per user	$O(d + sN)$	$O(d + s\log N)$	$O(d + \frac{d}{U-T})$
Online communication at server	$O(dN + sN^2)$	$O(dN + sN\log N)$	$O(dN + d\frac{U}{U-T})$
Online computation per user	$O(d)$	$O(d)$	$O(d + d\frac{U}{U-T})$
Decoding complexity at server	$O(sN^2)$	$O(sN\log^2 N)$	$O(d\frac{U\log U}{U-T})$
PRG complexity at server	$O(dN^2)$	$O(dN\log N)$	–

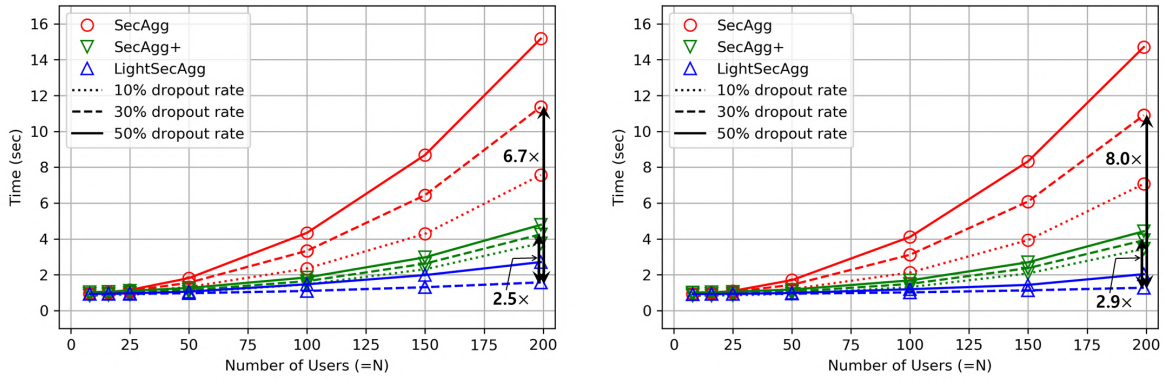
Table 6. Comparison of storage cost (in the number of symbols in $\mathbb{F}_q^{\frac{d}{U-T}}$) between protocol in (Zhao & Sun, 2021) and LightSecAgg.

	Protocol in (Zhao & Sun, 2021)	LightSecAgg
Total amount of randomness needed	$N(U-T) + T\sum_{u=U}^{N}\binom{N}{u}$	NU
Offline storage per user	$U - T + \sum_{u=U}^{N}\binom{N}{u}\frac{u}{N}$	$U - T + N$

practicality of LightSecAgg to maintain model security, and further reduces the total amount of needed randomness to scale linearly with N. Consequently, the local offline storage of each user in LightSecAgg scales linearly with N, as opposed to scaling exponentially in (Zhao & Sun, 2021). We compare the amount of generated randomness and the offline storage cost between the scheme in (Zhao & Sun, 2021) and LightSecAgg in Table 6.

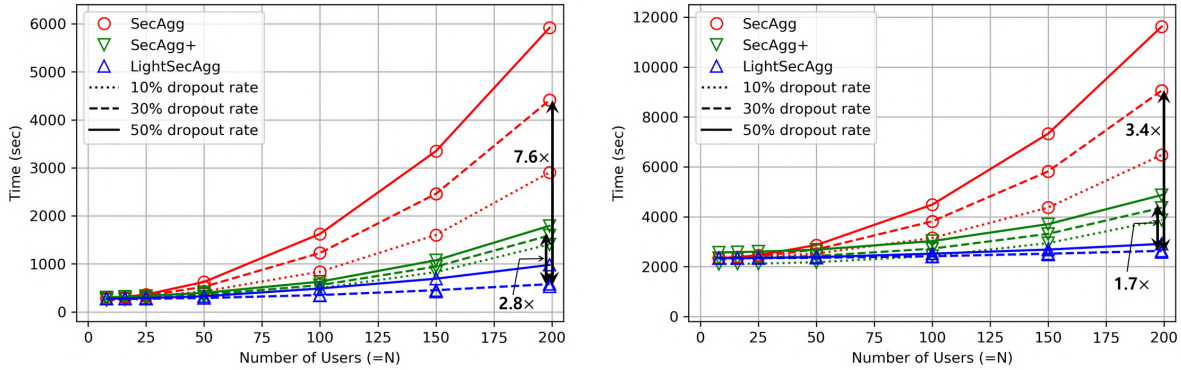
D Experimental Details

In this section, we provide experimental details of Section 7. Aside from the results of training CNN (McMahan et al., 2017) on the FEMNIST dataset (Caldas et al., 2018) as shown in Figure 6, we also demonstrate the total running time of LightSecAgg versus two baseline protocols SecAgg (Bonawitz et al., 2017) and SecAgg+ (Bell et al., 2020) to train logistic regression on the MNIST dataset (LeCun et al., 1998), MobileNetV3 (Howard et al., 2019) on the CIFAR-10 dataset (Krizhevsky et al., 2009), and EfficientNet-B0 (Tan & Le, 2019) on the GLD23k dataset (Weyand et al., 2020) in Figure 8, Figure 9, and Figure 10, respectively. For all considered FL training tasks, each user locally trains its model with $E = 5$ local epochs, before masking and uploading its model. We can observe that LightSecAgg provides significant speedup for all considered FL training tasks in the running time over SecAgg and SecAgg+.



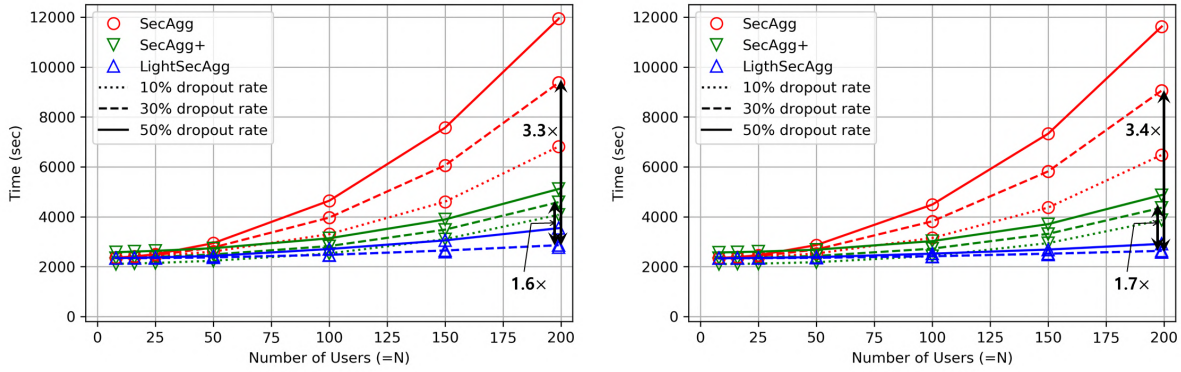
(a) Non-overlapped (b) Overlapped

Figure 8. Total running time of LightSecAgg versus the state-of-the-art protocols (SecAgg (Bonawitz et al., 2017) and SecAgg+ (Bell et al., 2020)) to train logistic regression on the MNIST (LeCun et al., 1998) with an increasing number of users, for various dropout rate.



(a) Non-overlapped (b) Overlapped

Figure 9. Total running time of LightSecAgg versus the state-of-the-art protocols (SecAgg (Bonawitz et al., 2017) and SecAgg+ (Bell et al., 2020)) to train MobileNetV3 (Howard et al., 2019) on the CIFAR-10 (Krizhevsky et al., 2009) with an increasing number of users, for various dropout rate.



(a) Non-overlapped (b) Overlapped

Figure 10. Total running time of LightSecAgg versus the state-of-the-art protocols (SecAgg (Bonawitz et al., 2017) and SecAgg+ (Bell et al., 2020)) to train EfficientNet-B0 (Tan & Le, 2019) on the GLD23k (Weyand et al., 2020) with an increasing number of users, for various dropout rate.

E PROOF OF LEMMA 1

We show that for an arbitrary set of colluding users $\mathcal{T}$ of size T, we have

$$I(\{\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}};\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}})=0. \tag{17}$$

The T-private MDS matrix used in LightSecAgg guarantees $I(\mathbf{z}_i;\{[\tilde{\mathbf{z}}_i]_j\}_{j\in\mathcal{T}})=0$. Thus,

$$\begin{aligned}
&I(\{\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}};\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}) && (18)\\
=&H(\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}})-H(\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}|\{\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}}) && (19)\\
=&H(\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}})-H(\{\mathbf{z}_i\}_{i\in\mathcal{T}}|\{\mathbf{z}_i\}_{i\in[N]\setminus\mathcal{T}})-H(\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}|\{\mathbf{z}_i\}_{i\in[N]}) && (20)\\
=&H(\{\mathbf{z}_i\}_{i\in\mathcal{T}},\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}})-H(\{\mathbf{z}_i\}_{i\in\mathcal{T}})-H(\{[\tilde{\mathbf{z}}_j]_i\}_{j\in[N],i\in\mathcal{T}}) && (21)\\
=&0, && (22)
\end{aligned}$$

where equation (20) follows from the chain rule. Equation (21) follows from the independence of $\mathbf{z}_i$'s and $I(\mathbf{z}_i;\{[\tilde{\mathbf{z}}_i]_j\}_{j\in\mathcal{T}})=0$. Equation (22) follows from the fact that joint entropy is less than or equal to the sum of the individual entropies, combined with the non-negativity of mutual information.

F APPLICATION OF LIGHTSECAGG TO ASYNCHRONOUS FL

In this Appendix, we provide a brief overview of asynchronous FL in Appendix F.1. Then, we illustrate the incompatibility of the conventional secure aggregation protocols, SecAgg and SecAgg+, with the asynchronous FL in Appendix F.2. Later on, in Appendix F.3, we demonstrate how LightSecAgg can be applied to the asynchronous FL setting to protect the privacy of individual updates.

F.1 General Description of Asynchronous FL

We consider the general asynchronous FL setting where the updates of the users are not synchronized while the goal is the same as synchronous FL, to collaboratively learn a global model $\mathbf{x}\in\mathbb{R}^d$, using the local datasets of N users without sharing them. This problem is formulated as minimizing a global loss function as follows

$$\min_{\mathbf{x}\in\mathbb{R}^d} F(\mathbf{x})=\sum_{i=1}^{N} p_i F_i(\mathbf{x}), \tag{23}$$

where F_i is the local loss function of user $i\in[N]$ and $p_i\geq 0$ are the weight parameters that indicate the relative impact of the users and are selected such that $\sum_{i=1}^{N} p_i=1$. This problem is solved iteratively in asynchronous FL.

At round t, each user locally trains the model by carrying out $E\geq 1$ local SGD steps. When the local update is done, user i sends the difference between the downloaded global model and updated local model to the server. The local update of user i sent to the server at round t is given by

$$\Delta_i^{(t;t_i)}=\mathbf{x}^{(t_i)}-\mathbf{x}_i^{(E;t_i)}, \tag{24}$$

where t_i is the latest round index when the global model is downloaded by user i and t is the round index when the local update is sent to the server, hence the staleness of user i is given by $\tau_i=t-t_i$. $\mathbf{x}_i^{(E;t_i)}$ denotes the local model after E local SGD steps and the local model at user i is updated as

$$\mathbf{x}_i^{(e;t_i)}=\mathbf{x}_i^{(e-1;t_i)}-\eta_l g_i(\mathbf{x}_i^{(e-1;t_i)};\xi_i) \tag{25}$$

for $e=1,\ldots,E$, where $\mathbf{x}_i^{(0;t_i)}=\mathbf{x}^{(t_i)}$, η_l denotes learning rate of the local updates. $g_i(\mathbf{x};\xi_i)$ denotes the stochastic gradient with respect to the random sampling ξ_i on user i, and we assume $\mathbb{E}_{\xi_i}[g_i(\mathbf{x};\xi_i)]=\nabla F_i(\mathbf{x})$ for all $\mathbf{x}\in\mathbb{R}^d$ where F_i is the local loss function of user i defined in (23). When the server receives $\Delta_i^{(t;t_i)}$, the global model at the server is updated as

$$\mathbf{x}^{(t+1)}=\mathbf{x}^{(t)}-\frac{\eta_g}{\sum_{i\in\mathcal{S}^{(t)}} s(t-t_i)}\sum_{i\in\mathcal{S}^{(t)}} s(t-t_i)\Delta_i^{(t;t_i)}, \tag{26}$$

where $\mathcal{S}^{(t)}$ is an index set of the users whose local models are sent to the server at round t and η_g is the learning rate of the global updates. $s(\tau)$ is a function that compensates for the staleness satisfying $s(0)=1$ and decreases

monotonically as τ increases. There are many functions that satisfy these two properties and we consider a polynomial function $s_\alpha(\tau) = (\tau+1)^{-\alpha}$ as it shows similar or better performance than the other functions e.g., Hinge or Constant stale function (Xie et al., 2019).

As discussed in Section 4.2, we focus on extending LightSecAgg to the *buffered* asynchronous FL setting of FedBuff (Nguyen et al., 2021), where the server stores the local updates in buffer of size K and updates the global model once the buffer is full. This is a special case of the general asynchronous FL setting described above, where $|\mathcal{S}^{(t)}| = K$ for all t. In principle the same approach for generalizing LightSecAgg can be used in other asynchronous FL settings where $|\mathcal{S}^{(t)}|$ changes over time, however since the convergence of FL in those settings are yet not understood, we do not consider them in the paper.

F.2 Incompatibility of SecAgg and SecAgg+ with Asynchronous FL

As described in Section 3, SecAgg (Bonawitz et al., 2017) and SecAgg+ (Bell et al., 2020) are designed for synchronous FL. At round t, each pair of users $i, j \in [N]$ agree on a pairwise random-seed $a_{i,j}^{(t)}$, and generate a random vector by running a PRG based on the random seed of $a_{i,j}^{(t)}$ to mask the local update. This additive structure has the unique property that these pairwise random vectors cancel out when the server aggregates the masked models because user $i(<j)$ adds $\mathrm{PRG}(a_{i,j}^{(t)})$ to $\mathbf{x}_i^{(t)}$ and user $j(>i)$ subtracts $\mathrm{PRG}(a_{i,j}^{(t)})$ from $\mathbf{x}_j^{(t)}$.

In asynchronous FL, however, the cancellation of the pairwise random masks based on the key agreement protocol is not guaranteed due to the mismatch in staleness between the users. Specifically, at round t, user $i \in \mathcal{S}^{(t)}$ sends the masked model $\mathbf{y}_i^{(t;t_i)}$ to the server that is given by

$$\mathbf{y}_i^{(t;t_i)} = \Delta_i^{(t;t_i)} + \mathrm{PRG}\left(b_i^{(t_i)}\right) + \sum_{j:i<j} \mathrm{PRG}\left(a_{i,j}^{(t_i)}\right) - \sum_{j:i>j} \mathrm{PRG}\left(a_{j,i}^{(t_i)}\right), \tag{27}$$

where $\Delta_i^{(t;t_i)}$ is the local update defined in (24). When $t_i \neq t_j$, the pairwise random vectors in $\mathbf{y}_i^{(t;t_i)}$ and $\mathbf{y}_j^{(t;t_j)}$ are not canceled out as $a_{i,j}^{(t_i)} \neq a_{i,j}^{(t_j)}$. We note that the identity of the staleness of each user is not known a priori, hence each pair of users cannot use the same pairwise random-seed.

F.3 Asynchronous LightSecAgg

We now demonstrate how LightSecAgg can be applied to the asynchronous FL setting where the server stores each local update in a buffer of size K and updates the global model by aggregating the stored updates when the buffer is full. Our key intuition is to encode the local masks in a way that the server can recover the aggregate of masks from the encoded masks via a one-shot computation even though the masks are generated in different training rounds. The asynchronous LightSecAgg protocol also consists of three phases with three design parameters D, T, U which are defined in the same way as the synchronous LightSecAgg.

Synchronous and asynchronous LightSecAgg have two key differences: (1) In asynchronous FL, the users share the encoded masks with the time stamp in the first phase to figure out which encoded masks should be aggregated for the reconstruction of aggregate of masks in the third phase. Due to the commutative property of coding and addition, the server can reconstruct the aggregate of masks even though the masks are generated in different training rounds; (2) In asynchronous FL, the server compensates the staleness of the local updates. This is challenging as this compensation should be carried out over the masked model in the finite field to provide the privacy guarantee while the conventional compensation functions have real numbers as outputs (Xie et al., 2019; Nguyen et al., 2021).

We now describe the three phases in detail.

F.3.1 Offline Encoding and Sharing of Local Masks

User i generates $\mathbf{z}_i^{(t_i)}$ uniformly at random from the finite field $\mathbb{F}_q^d$, where t_i is the global round index when user i downloads the global model from the server. The mask $\mathbf{z}_i^{(t_i)}$ is partitioned into $U-T$ sub-masks denoted by $[\mathbf{z}_i^{(t_i)}]_1, \cdots, [\mathbf{z}_i^{(t_i)}]_{U-T}$, where U denotes the targeted number of surviving users and $N - D \geq U \geq T$. User i also selects another T random masks denoted by $[\mathbf{n}_i^{(t_i)}]_{U-T+1}, \cdots, [\mathbf{n}_i^{(t_i)}]_U$. These U partitions $[\mathbf{z}_i^{(t_i)}]_1, \cdots, [\mathbf{z}_i^{(t_i)}]_{U-T}, [\mathbf{n}_i^{(t_i)}]_{U-T+1}, \cdots, [\mathbf{n}_i^{(t_i)}]_U$ are then

encoded through an (N, U) Maximum Distance Separable (MDS) code as follows

$$[\widetilde{\mathbf{z}}_i^{(t_i)}]_j = \left([\mathbf{z}_i^{(t_i)}]_1, \cdots, [\mathbf{z}_i^{(t_i)}]_{U-T}, [\mathbf{n}_i^{(t_i)}]_{U-T+1}, \cdots, [\mathbf{n}_i^{(t_i)}]_U\right) \mathbf{W}_j, \tag{28}$$

where $\mathbf{W}_j$ is the Vandermonde matrix defined in (5). User i sends $[\widetilde{\mathbf{z}}_i^{(t_i)}]_j$ to user $j \in [N] \setminus \{i\}$. At the end of this phase, each user $i \in [N]$ has $[\widetilde{\mathbf{z}}_j^{(t_j)}]_i$ from $j \in [N]$.

F.3.2 Training, Quantizing, Masking, and Uploading of Local Updates

Each user i trains the local model as in (24) and (25). User i quantizes its local update $\Delta_i^{(t;t_i)}$ from the domain of real numbers to the finite field $\mathbb{F}_q$ as masking and MDS encoding are carried out in the finite field to provide information-theoretic privacy. The field size q is assumed to be large enough to avoid any wrap-around during secure aggregation.

The quantization is a challenging task as it should be performed in a way to ensure the convergence of the global model. Moreover, the quantization should allow the representation of negative integers in the finite field, and enable computations to be carried out in the quantized domain. Therefore, we cannot utilize well-known gradient quantization techniques such as in (Alistarh et al., 2017), which represents the sign of a negative number separately from its magnitude. LightSecAgg addresses this challenge with a simple stochastic quantization strategy combined with the two's complement representation as described subsequently. For any positive integer $c \geq 1$, we first define a stochastic rounding function as

$$Q_c(x) = \begin{cases} \frac{\lfloor cx \rfloor}{c} & \text{with prob. } 1 - (cx - \lfloor cx \rfloor) \\ \frac{\lfloor cx \rfloor + 1}{c} & \text{with prob. } cx - \lfloor cx \rfloor, \end{cases} \tag{29}$$

where $\lfloor x \rfloor$ is the largest integer less than or equal to x, and this rounding function is unbiased, i.e., $\mathbb{E}_Q[Q_c(x)] = x$. The parameter c is a design parameter to determine the number of quantization levels. The variance of $Q_c(x)$ decreases as the value of c increases. We then define the quantized update

$$\overline{\Delta}_i^{(t;t_i)} := \phi\left(c_l \cdot Q_{c_l}\left(\Delta_i^{(t;t_i)}\right)\right), \tag{30}$$

where the function Q_c from (29) is carried out element-wise, and c_l is a positive integer parameter to determine the quantization level of the local updates. The mapping function $\phi : \mathbb{R} \to \mathbb{F}_q$ is defined to represent a negative integer in the finite field by using the two's complement representation,

$$\phi(x) = \begin{cases} x & \text{if } x \geq 0 \\ q + x & \text{if } x < 0. \end{cases} \tag{31}$$

To protect the privacy of the local updates, user i masks the quantized update $\overline{\Delta}_i^{(t;t_i)}$ in (30) as

$$\widetilde{\Delta}_i^{(t;t_i)} = \overline{\Delta}_i^{(t;t_i)} + \mathbf{z}_i^{(t_i)}, \tag{32}$$

and sends the pair of $\left\{\widetilde{\Delta}_i^{(t;t_i)}, t_i\right\}$ to the server. The local round index t_i is used in two cases: (1) when the server identifies the staleness of each local update and compensates it, and (2) when the users aggregate the encoded masks for one-shot recovery, which will be explained in Section F.3.3.

F.3.3 One-shot Aggregate-update Recovery and Global Model Update

The server stores $\widetilde{\Delta}_i^{(t;t_i)}$ in the buffer, and when the buffer of size K is full, the server aggregates the K masked local updates. In this phase, the server intends to recover

$$\sum_{i \in \mathcal{S}^{(t)}} s(t - t_i) \Delta_i^{(t;t_i)}, \tag{33}$$

where $\Delta_i^{(t;t_i)}$ is the local update in the real domain defined in (24), $\mathcal{S}^{(t)}$ ($\left|\mathcal{S}^{(t)}\right| = K$) is the index set of users whose local updates are stored in the buffer and aggregated by the server at round t, and $s(\tau)$ is the staleness function defined in (26). To

do so, the first step is to reconstruct $\sum_{i\in\mathcal{S}^{(t)}} s(t-t_i)\mathbf{z}_i^{(t_i)}$. This is challenging as the decoding should be performed in the finite field, but the value of $s(\tau)$ is a real number. To address this problem, we introduce a quantized staleness function $\overline{s}:\{0,1,\dots,\}\to\mathbb{F}_q$,

$$\overline{s}_{c_g}(\tau)=c_g Q_{c_g}\left(s(\tau)\right), \tag{34}$$

where $Q_c(\cdot)$ is a stochastic rounding function defined in (29), and c_g is a positive integer to determine the quantization level of staleness function. Then, the server broadcasts information of $\left\{\mathcal{S}^{(t)},\{t_i\}_{i\in\mathcal{S}^{(t)}},c_g\right\}$ to all surviving users. After identifying the selected users in $\mathcal{S}^{(t)}$, the local round indices $\{t_i\}_{i\in\mathcal{S}^{(t)}}$ and the corresponding staleness, user $j\in[N]$ aggregates its encoded sub-masks $\sum_{i\in\mathcal{S}^{(t)}}\overline{s}_{c_g}(t-t_i)\left[\widetilde{\mathbf{z}}_i^{(t_i)}\right]_j$ and sends it to the server for the purpose of one-shot recovery. The key difference between the asynchronous LightSecAgg and the synchronous LightSecAgg is that in the asynchronous LightSecAgg, the time stamp t_i for encoded masks $\left[\widetilde{\mathbf{z}}_i^{(t_i)}\right]_j$ for each $i\in\mathcal{S}^{(t)}$ can be different, hence user $j\in[N]$ must aggregate the encoded mask with the proper round index. Due to the commutative property of coding and linear operations, each $\sum_{i\in\mathcal{S}^{(t)}}\overline{s}_{c_g}(t-t_i)\left[\widetilde{\mathbf{z}}_i^{(t_i)}\right]_j$ is an encoded version of $\sum_{i\in\mathcal{S}^{(t)}}\overline{s}_{c_g}(t-t_i)\left[\mathbf{z}_i^{(t_i)}\right]_k$ for $k\in[U-T]$ using the MDS matrix (or Vandermonde matrix) $\mathbf{V}$ defined in (28). Thus, after receiving a set of any U results from surviving users in $\mathcal{U}_2$, where $|\mathcal{U}_2|=U$, the server reconstructs $\sum_{i\in\mathcal{S}^{(t)}}\overline{s}_{c_g}(t-t_i)\left[\mathbf{z}_i^{(t_i)}\right]_k$ for $k\in[U-T]$ via MDS decoding. By concatenating the $U-T$ aggregated sub-masks $\sum_{i\in\mathcal{S}^{(t)}}\overline{s}_{c_g}(t-t_i)\left[\mathbf{z}_i^{(t_i)}\right]_k$, the server can recover $\sum_{i\in\mathcal{S}^{(t)}}\overline{s}_{c_g}(t-t_i)\mathbf{z}_i^{(t_i)}$. Finally, the server obtains the desired global update as follows

$$\mathbf{g}^{(t)}=\frac{1}{c_g c_l\sum_{i\in\mathcal{S}^{(t)}}s_{c_g}(t-t_i)}\phi^{-1}\left(\sum_{i\in\mathcal{S}^{(t)}}\overline{s}_{c_g}(t-t_i)\widetilde{\Delta}_i^{(t;t_i)}-\sum_{i\in\mathcal{S}^{(t)}}\overline{s}_{c_g}(t-t_i)\mathbf{z}_i^{(t_i)}\right), \tag{35}$$

where c_l is defined in (30) and $\phi^{-1}:\mathbb{F}_q\to\mathbb{R}$ is the demapping function defined as follows

$$\phi^{-1}(\overline{x})=\begin{cases}\overline{x} & \text{if}\quad 0\le\overline{x}<\frac{q-1}{2}\\ \overline{x}-q & \text{if}\quad \frac{q-1}{2}\le\overline{x}<q.\end{cases} \tag{36}$$

Finally, the server updates the global model as $\mathbf{x}^{(t+1)}=\mathbf{x}^{(t)}-\eta_g\mathbf{g}^{(t)}$, which is equivalent to

$$\mathbf{x}^{(t+1)}=\mathbf{x}^{(t)}-\frac{\eta_g}{\sum_{i\in\mathcal{S}^{(t)}}Q_{c_g}\left(s(t-t_i)\right)}\sum_{i\in\mathcal{S}^{(t)}}Q_{c_g}\left(s(t-t_i)\right)Q_{c_l}\left(\Delta_i^{(t;t_i)}\right), \tag{37}$$

where Q_{c_l} and Q_{c_g} are the stochastic rounding function defined in (29) with respect to quantization parameters c_l and c_g, respectively.

F.4 Convergence Analysis of Asynchronous LightSecAgg

We now provide the convergence guarantee of asynchronous LightSecAgg in the L-smooth and non-convex setting. The prior works mostly focus on the synchronous FL setting, but here we focus on the buffered asynchronous setting. While the convergence analysis in the buffered asynchronous setting has been considered recently in (Nguyen et al., 2021) and the effects of the buffer size and the staleness have been analyzed, LightSecAgg requires quantization to enable secure aggregation without TEEs. Hence, we extend this analysis here by taking the quantization's effect into account.

For simplicity, we consider the constant staleness function $s(\tau)=1$ for all τ in (37). Then, the global update equation of asynchronous LightSecAgg is given by

$$\mathbf{x}^{(t+1)}=\mathbf{x}^{(t)}-\frac{\eta_g}{K}\sum_{i\in\mathcal{S}^{(t)}}Q_{c_l}\left(\Delta_i^{(t;t_i)}\right), \tag{38}$$

where Q_{c_l} is the stochastic round function defined in (29), c_l is the positive constant to determine the quantization level, and $\Delta_i^{(t;t_i)}$ is the local update of user i defined in (24). We first introduce our assumptions, which are commonly made in analyzing FL algorithms (Li et al., 2019; Nguyen et al., 2021; Reddi et al., 2020; So et al., 2021a).

Assumption 1. *(Unbiasedness of local SGD). For all $i \in [N]$ and $\mathbf{x} \in \mathbb{R}^d$, $\mathbb{E}_{\xi_i}[g_i(\mathbf{x};\xi_i)] = \nabla F_i(\mathbf{x})$ where $g_i(\mathbf{x};\xi_i)$ is the stochastic gradient estimator of user i defined in (25).*

Assumption 2. *(Lipschitz gradient). $F_1, \ldots, F_N$ in (23) are all L-smooth: for all $\mathbf{a}, \mathbf{b} \in \mathbb{R}^d$ and $i \in [N]$, $\|\nabla F_i(\mathbf{a}) - \nabla F_i(\mathbf{b})\|^2 \leq L\|\mathbf{a} - \mathbf{b}\|^2$.*

Assumption 3. *(Bounded variance of local and global gradients). The variance of the stochastic gradients at each user is bounded, i.e., $\mathbb{E}_{\xi_i} \|\nabla g_i(\mathbf{x};\xi_i) - \nabla F_i(\mathbf{x})\|^2 \leq \sigma_l^2$ for $i \in [N]$ and $\mathbf{x} \in \mathbb{R}^d$. For the global loss function $F(\mathbf{x})$ defined in (23), $\frac{1}{N}\sum_{i=1}^N \|\nabla F_i(\mathbf{x}) - \nabla F(\mathbf{x})\|^2 \leq \sigma_g^2$ holds.*

Assumption 4. *(Bounded gradient). For all $i \in [N]$, $\|\nabla F_i(\mathbf{x})\|^2 \leq G$.*

In addition, we make an assumption on the staleness of the local updates under asynchrony (Nguyen et al., 2021).

Assumption 5. *(Bounded staleness). For each global round index t and all users $i \in [N]$, the delay $\tau_i^{(t)} = t - t_i$ is not larger than a certain threshold $\tau_{\max}$ where t_i is the latest round index when the global model is downloaded to user i.*

Now, we state our main result for the convergence guarantee of asynchronous LightSecAgg.

Theorem 2. *Selecting the constant learning rates η_l and η_g such that $\eta_l \eta_g KE \leq \frac{1}{L}$, the global model iterations in (38) achieve the following ergodic convergence rate*

$$\frac{1}{J}\sum_{t=0}^{J-1} \mathbb{E}\left[|\nabla F(\mathbf{x}^{(t)})|^2\right] \leq \frac{2F^*}{\eta_g \eta_l EKT} + \frac{L\eta_g \eta_l \sigma_{c_l}^2}{2} + 3L^2E^2\eta_l^2\left(\eta_g^2 K^2 \tau_{\max}^2\right)\sigma^2, \tag{39}$$

where $F^* = F(\mathbf{x}^{(0)}) - F(\mathbf{x}^*)$, $\sigma^2 = G + \sigma_g^2 + \sigma_{c_l}^2$, and $\sigma_{c_l}^2 = \frac{d}{4c_l^2} + \sigma_l^2$.

The proof of Theorem 2 directly follows from the following useful lemma that shows the unbiasedness and bounded variance still hold for the quantized gradient estimator $Q_c(g(\mathbf{x},\xi))$ for any $\mathbf{x} \in \mathbb{R}^d$.

Lemma 2. *For the quantized gradient estimator $Q_c(g(\mathbf{x},\xi))$ with a given vector $\mathbf{x} \in \mathbb{R}^d$ where ξ is a uniform random variable representing the sample drawn, g is a gradient estimator such that $\mathbb{E}_\xi[g(\mathbf{x},\xi)] = \nabla F(\mathbf{x})$ and $\mathbb{E}_\xi\|g(\mathbf{x},\xi) - \nabla F(\mathbf{x})\|^2 \leq \sigma_l^2$, and the stochastic rounding function Q_c is given in (29), the following holds,*

$$\mathbb{E}_{Q,\xi}[Q_c(g(\mathbf{x},\xi))] = \nabla F(\mathbf{x}) \tag{40}$$

$$\mathbb{E}_{Q,\xi}\|Q_c(g(\mathbf{x},\xi)) - \nabla F(\mathbf{x})\|^2 \leq \sigma_c^2, \tag{41}$$

where $\sigma_c^2 = \frac{d}{4c^2} + \sigma_l^2$.

Proof. (Unbiasedness). Given Q_c in (29) and any random variable x, it follows that,

$$\begin{aligned}\mathbb{E}_Q\left[Q_c(x) \mid x\right] &= \frac{\lfloor cx \rfloor}{c}\left(1 - (cx - \lfloor cx \rfloor)\right) + \frac{(\lfloor cx \rfloor + 1)}{c}(cx - \lfloor cx \rfloor) \\ &= x\end{aligned} \tag{42}$$

from which we obtain the unbiasedness condition in (40),

$$\begin{aligned}\mathbb{E}_{Q,\xi}[Q_c(g(\mathbf{x},\xi))] &= \mathbb{E}_\xi\left[\mathbb{E}_Q[Q_c(g(\mathbf{x},\xi)) \mid g(\mathbf{x},\xi)]\right] \\ &= \mathbb{E}_\xi\left[g(\mathbf{x},\xi)\right] \\ &= \nabla F(\mathbf{x}).\end{aligned} \tag{43}$$

(Bounded variance). Next, we observe that,

$$\begin{aligned}&\mathbb{E}_Q\left[\left(Q_c(x) - \mathbb{E}_Q[Q_c(x) \mid x]\right)^2 \mid x\right] \\ &= \left(\frac{\lfloor cx \rfloor}{c} - x\right)^2 (1 - (cx - \lfloor cx \rfloor)) + \left(\frac{\lfloor cx \rfloor + 1}{c} - x\right)^2 (cx - \lfloor cx \rfloor)\end{aligned}$$

$$= \frac{1}{c^2}\left(\frac{1}{4} - \left(cx - \lfloor cx \rfloor - \frac{1}{2}\right)^2\right)$$
$$\leq \frac{1}{4c^2} \tag{44}$$

from which we obtain the bounded variance condition in (41) as follows,

$$\begin{aligned}
&\mathbb{E}_{Q,\xi}\|Q_c(g(\mathbf{x},\xi)) - \nabla F(\mathbf{x})\|^2 \\
&\quad = \mathbb{E}_{\xi}\left[\mathbb{E}_Q[\|Q_c(g(\mathbf{x},\xi)) - \nabla F(\mathbf{x})\|^2 \mid g(\mathbf{x},\xi)]\right] \\
&\quad \leq \mathbb{E}_{\xi}\left[\mathbb{E}_Q[\|Q_c(g(\mathbf{x},\xi)) - g(\mathbf{x},\xi)\|^2 \mid g(\mathbf{x},\xi)]\right] + \mathbb{E}_{\xi}\left[\mathbb{E}_Q[\|g(\mathbf{x},\xi) - \nabla F(\mathbf{x})\|^2 \mid g(\mathbf{x},\xi)]\right] \quad (45) \\
&\quad \leq \frac{d}{4c^2} + \sigma_l^2 \quad (46) \\
&\quad = \sigma_c^2,
\end{aligned}$$

where (45) follows from the triangle inequality and (46) follows form (44). □

Now, the update equation of asynchronous LightSecAgg is equivalent to the update equation of FedBuff except that LightSecAgg has an additional random source, stochastic quantization Q_{c_l}, which also satisfies the unbiasedness and bounded variance. One can show the convergence rate of asynchronous LightSecAgg presented in Theorem 2 by exchanging $\mathbf{E}_\xi$ and variance-bound σ_l^2 in (Nguyen et al., 2021) with $\mathbf{E}_{Q_{c_l},\xi}$ and variance-bound $\sigma_{c_l}^2 = \frac{d}{4{c_l}^2} + \sigma_l^2$, respectively.

Remark 6. Theorem 2 shows that convergence rates of asynchronous LightSecAgg and FedBuff (see Corollary 1 in (Nguyen et al., 2021)) are the same except for the increased variance of the local updates due to the quantization noise in LightSecAgg. The amount of the increased variance $\frac{d}{4{c_l}^2}$ in $\sigma_{c_l}^2 = \frac{d}{4{c_l}^2} + \sigma_l^2$ is negligible for large c_l, which will be demonstrated in our experiments in Appendix F.5.

F.5 Experiments for Asynchronous LightSecAgg

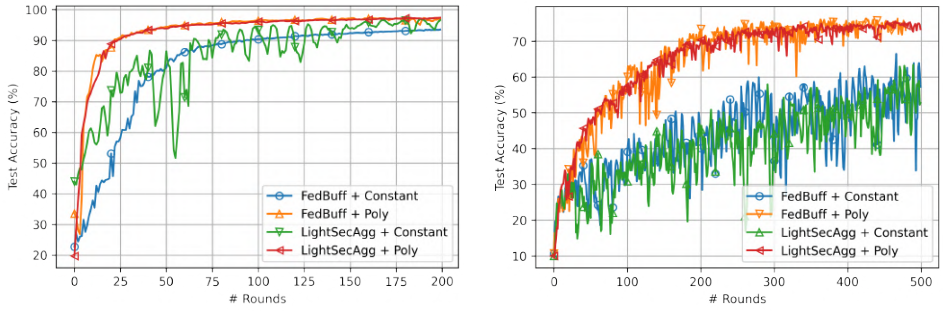
As described in the previous sections, there is no prior secure aggregation protocol applicable to asynchronous FL, and hence we cannot compare the the total running time of LightSecAgg with other baseline protocols, such as SecAgg and SecAgg+ that were considered in synchronous FL. As such, in our experiments here we instead focus on convergence performance of LightSecAgg compared to the buffered asynchronous FL scheme to highlight the impact of asynchrony and quantization in performance. We measure the performance in terms of the model accuracy evaluated over the test samples with respect to the global round index t.

Datasets and network architectures. We consider an image classification task on the MNIST (LeCun et al., 1998) and CIFAR-10 datasets (Krizhevsky et al., 2009). For the MNIST dataset, we train LeNet (LeCun et al., 1998). For the CIFAR-10 dataset, we train the convolutional neural network (CNN) used in (Xie et al., 2019). These network architectures are sufficient for our needs as our goal is to evaluate various schemes, and not to achieve the best accuracy.

Setup. We consider a buffered asynchronous FL setting with $N = 100$ users and a single server having the buffer of size $K = 10$. For the IID data distribution, the training samples are shuffled and partitioned into $N = 100$ users. For asynchronous training, we assume the staleness of each user is uniformly distributed over $[0, 10]$, i.e., $\tau_{\max} = 10$, as used in (Xie et al., 2019). We set the field size $q = 2^{32} - 5$, which is the largest prime within 32 bits.

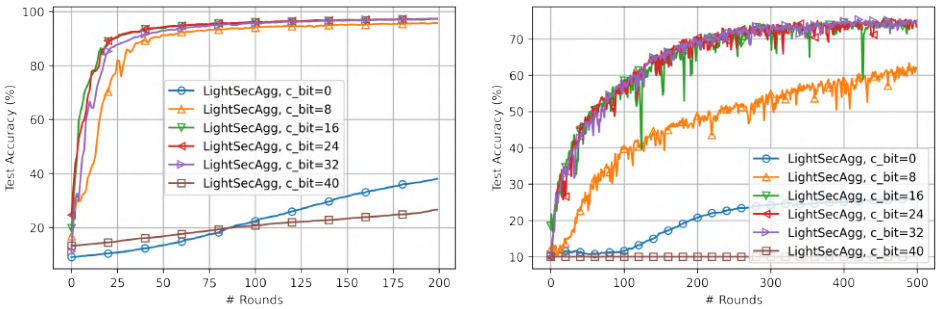
Implementations. We implement two schemes, FedBuff and LightSecAgg. The key difference between the two schemes is that in LightSecAgg, the local updates are quantized and converted into the finite field to provide privacy of the individual local updates while all operations are carried out over the domain of real numbers in FedBuff. For both schemes, to compensate the staleness of the local updates, we employ the two strategies for the weighting function: a constant function $s(\tau) = 1$ and a polynomial function $s_\alpha(\tau) = (1 + \tau)^{-\alpha}$.

Empirical results. In Figure 11(a) and 11(b), we demonstrate that LightSecAgg has almost the same performance as FedBuff on both MNIST and CIFAR-10 datasets, while LightSecAgg includes quantization noise to protect the privacy of individual local updates of users. This is because the quantization noise in LightSecAgg is negligible. To compensate the staleness of the local updates over the finite field in LightSecAgg, we implement the quantized staleness function



(a) MNIST dataset. (b) CIFAR-10 dataset.

Figure 11. Accuracy of asynchronous `LightSecAgg` and `FedBuff` with two strategies for the weighting function to mitigate the staleness: a constant function $s(\tau) = 1$ (no compensation) named *Constant*; and a polynomial function $s_\alpha(\tau) = (1+\tau)^{-\alpha}$ named *Poly* where $\alpha = 1$.



(a) MNIST dataset. (b) CIFAR-10 dataset.

Figure 12. Accuracy of asynchronous `LightSecAgg` and `FedBuff` with various values of the quantization parameter $c_l = 2^{c_{bit}}$.

defined in (34) with $c_g = 2^6$, which has the same performance in mitigating the staleness as the original staleness function carried out over the domain of real numbers.

Performance with various quantization levels. To investigate the impact of the quantization, we measure the performance with various values of the quantization parameter c_l on MNIST and CIFAR-10 datasets in Fig. 12. We observe that $c_l = 2^{16}$ has the best performance, while a small or a large value of c_l has poor performance. This is because the value of c_l provides a trade-off between two sources of quantization noise: 1) the rounding error from the stochastic rounding function defined in (29) and 2) the wrap-around error when modulo operations are carried out in the finite field. When c_l has small value the rounding error is dominant, while the wrap-around error is dominant when c_l has large value. To find a proper value of c_l, we can utilize the auto-tuning algorithm proposed in (Bonawitz et al., 2019c).