Secure Aggregation for Buffered Asynchronous Federated Learning

Abstract

Federated learning (FL) typically relies on synchronous training, which is slow due to stragglers. While asynchronous training handles stragglers efficiently, it does not ensure privacy due to the incompatibility with the secure aggregation protocols. A buffered asynchronous training protocol known as FedBuff has been proposed recently which bridges the gap between synchronous and asynchronous training to mitigate stragglers and to also ensure privacy simultaneously. The key idea of FedBuff is that it allows the users to send their updates asynchronously, while ensuring privacy by storing the updates in a trusted execution environment (TEE) enabled private buffer. TEEs, however, have limited memory which limits the buffer size and the privacy guarantee of this approach. Motivated by these limitations, we develop a buffered asynchronous secure aggregation (`BASecAgg`) protocol that does not rely on TEEs. The conventional secure aggregation protocols cannot be applied in the buffered asynchronous setting since the buffer may have local models corresponding to different rounds in the buffer and hence the masks that the users use to protect their models may not cancel out. `BASecAgg` addresses this challenge by carefully designing the masks such that they cancel out even if they correspond to different rounds. Our convergence analysis and experiments show that `BASecAgg` almost has the same convergence guarantees as FedBuff without relying on TEEs.

1 Introduction

Federated learning (FL) allows a large number of users to collaboratively train a machine learning model without sharing their data and while protecting the privacy of each user [14]. The training in FL is typically coordinated by a central server. The main idea that enables decentralized training without sharing data is that each user trains a local model using its dataset and the global model maintained by the server. The users then only share their local models with the server which updates the global model and pushes it again to the users for the next training round until convergence. Recent studies, however, showed that sharing the local models still can breach the privacy of the users through inference or inversion attacks e.g., [8, 15, 25, 9]. To overcome this challenge, secure aggregation protocols have been developed to ensure that the server only learn the global model without revealing the local models of the users [4, 20, 11, 24, 7, 2].

FL protocols commonly rely on synchronous training [14]. Synchronous training, however, severely suffers from stragglers as a result of waiting for the updates of a sufficient number of users before completing each training round. Asynchronous FL addresses this challenge by incorporating the updates of the users as soon as they arrive at the server without waiting for the other users [22, 21, 5, 6]. While asynchronous FL handles stragglers efficiently, it does not protect the privacy of the individual users. Specifically, asynchronous FL is not compatible with the secure aggregation protocols designed particularly for synchronous FL. This is because in the conventional secure aggregation protocols, the users mask their models such that these masks cancel out when the server aggregates the local models. This ensures that the server only learns the aggregate model of the users while not revealing their local models. Such secure aggregation protocols securely aggregate many local models together each time the global model is updated and hence they are not suitable for asynchronous FL in which each single local model updates the global model. Another approach that can be applied in asynchronous FL to protect the privacy of the users is local differential privacy (LDP). In this approach, each user adds a noise to the local model before sharing it with the server. This approach, however, degrades the training accuracy.

In [16], an asynchronous aggregation protocol known as FedBuff has been proposed to mitigate stragglers and enable secure aggregation jointly. FedBuff enables secure aggregation in asynchronous FL through trusted execution environments (TEEs) as Intel software guard extensions (SGX) [12]. Specifically, the individual updates are not incorporated by the server as soon they arrive. Instead, the server keeps the local models that it receives in a TEE-enabled *secure buffer*. The server then updates the global model when K local models are collected in the secure buffer as shown in Fig. 1, where K is a tunable parameter. This idea has been shown to be 3.8 times faster than the conventional synchronous FL schemes.

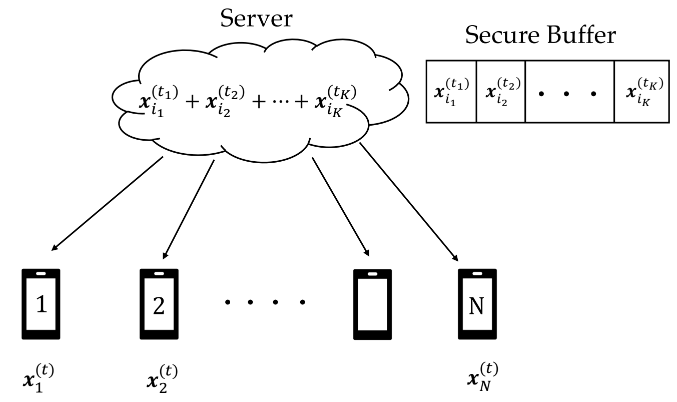
Server
$\boldsymbol{x}_{i_1}^{(t_1)} + \boldsymbol{x}_{i_2}^{(t_2)} + \cdots + \boldsymbol{x}_{i_K}^{(t_K)}$
Secure Buffer
$\boldsymbol{x}_{i_1}^{(t_1)}$
$\boldsymbol{x}_{i_2}^{(t_2)}$
$\boldsymbol{x}_{i_K}^{(t_K)}$
1
2
N
$\boldsymbol{x}_1^{(t)}$
$\boldsymbol{x}_2^{(t)}$
$\boldsymbol{x}_N^{(t)}$

Figure 1: In FedBuff, the server securely collects K local models and then updates the global model. This secure buffer is enabled through TEEs such as Intel SGX.

Contributions. Since TEEs have limited memory and are inefficient compared to the untrusted hardware, we instead develop a buffered asynchronous secure aggregation protocol that does not rely on TEEs. The main challenge of leveraging the conventional secure aggregation protocols in the buffered asynchronous setting is that the pairwise masks may not cancel out. This is because of the asynchronous nature of this setting which may result in local models of different rounds in the buffer, while the pairwise masks cancel out if they belong to the same round. This requires a careful design of the masks such that they can be cancelled even if they do not correspond to the same round. Specifically, our contributions are as follows.

1. We propose a buffered asynchronous secure aggregation protocol that extends a recently proposed synchronous secure aggregation protocol known as `LightSecAgg` [23] to this buffered asynchronous setting. The key idea of our protocol, `BASecAgg`, is that we design the masks such that they cancel out even if they correspond to different training rounds.
2. We extend the convergence analysis of [16] to the case where the local updates are quantized, which is necessary for the secure aggregation protocols.
3. Our extensive experiments on MNIST and CIFAR datasets show that `BASecAgg` almost has the same convergence guarantees as FedBuff despite the quantization.

2 Related Works

Secure aggregation protocols are essential in FL to protect the privacy of the individual users. Secure aggregation protocols typically rely on exchanging pairwise random-seeds and then secret sharing the random-seeds to tolerate users' dropouts [4, 20, 11, 2]. The running time of such approaches, however, increases significantly with the number of dropped users. This is because the server needs to reconstruct the mask of each dropped user, which significantly limits the scalability of such approaches. Recently, a secure aggregation protocol known as `LightSecAgg` has been proposed to address this scalability challenge [23]. In `LightSecAgg` , unlike the prior works, the server does not reconstruct the pairwise random-seeds of each dropped user. Instead, the server directly reconstructs the aggregate masks of all surviving users. That is, the server only reconstructs the aggregate mask unlike the prior approaches in which the server reconstructs the mask of each dropped user. This one-shot reconstruction of the masks of all surviving users results in a much faster training compared to the baselines [4].

Prior secure aggregation protocols [4, 20, 11, 2] are designed for the synchronous FL algorithms such as FedAvg [14], which suffer from stragglers. Asynchronous FL handles this problem by updating the global model as soon as the server receives any local update [22, 21, 5, 6]. The larger staleness is of the local model in such an asynchronous setting, the greater is the error when updating the global model [22]. To address this staleness problem, an asynchronous protocol known as `FedAsync` has been developed in [22] that updates the global model through staleness-aware weighted averaging of the old global model and the received local model. In [5], an asynchronous protocol known as `FedAt`, which bridges the gap between synchronous FL and asynchronous FL by developing a semi-synchronous protocol that groups the users and synchronously updates the model of each group and then asynchronously updates the global model across groups. Similarly, a semi-synchronous FL protocol has been developed in [17] to handle the staleness problem and also mitigates Byzantine users simultaneously.

Asynchronous FL, however, is not compatible with secure aggregation. A potential approach to ensure privacy then is through DP approaches that add noise to the local models before sharing them with the sever as in [21]. A similar approach has been also leveraged in [10] to develop a privacy-preserving protocol for a limited class of learning problems as linear regression, logistic regression and least-squares support vector machine in the vertically partitioned (VP) asynchronous decentralized FL setting. Adding noise, however, degrades the training accuracy. In [16], an asynchronous aggregation protocol known as FedBuff ha been proposed to mitigate stragglers while ensuring privacy. The key idea of FedBuff is that the server does not incorporate the local updates as soon as they arrive. The server instead stores the local models in a TEE-enabled secure buffer of size K until the buffer is full and then securely aggregates them. Due to the memory limitations of TEEs, this approach is only feasible when K is small. This motivates us in this work to develop a buffered asynchronous secure aggregation protocol that is not based on TEEs.

3 Synchronous Secure Aggregation

In this section, we provide an overview of secure aggregation of synchronous FL.

The goal in FL is to collaboratively learn a global model $\boldsymbol{x}$ with dimension d, using the local datasets of N users without sharing the datasets. This learning problem can be formulated as minimizing a global loss function as follows

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

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

This problem is solved iteratively. At round t, the server sends the global model $\boldsymbol{x}^{(t)}$ to the users. Some of the users may dropout due to various reasons such as wireless connectivity or battery issues. We assume that at most D users may dropout in any round. We denote the set of the surviving users at round t by $\mathcal{U}^{(t)}$ and the set of dropped users by $\mathcal{D}^{(t)}$. User $i \in [N]$ that receives the global model updates the global model by carrying out E (≥ 1) local stochastic gradient descent (SGD) steps. The goal of the server is to get the sum of the local models of the surviving users to update its global model as follows

$$\boldsymbol{x}^{(t)} = \frac{1}{|\mathcal{U}^{(t)}|} \sum_{i\in\mathcal{U}^{(t)}} \boldsymbol{x}_i^{(t)}. \tag{2}$$

The server then sends $\boldsymbol{x}^{(t)}$ to the users for the next round. While the users do not share their data with the server and just share their local models, the local models of the users still can reveal significant information about their datasets [8, 15, 25, 9]. To address this challenge, a secure aggregation protocol known as `SecAgg` was developed in [3] to ensure that the server does not learn anything about the local models except their sum $\sum_{i\in\mathcal{U}^{(t)}} \boldsymbol{x}_i^{(t)}$ at round t. Specifically, we assume that up to T users can collude with each other as well as with the server to reveal the local models of other users. The secure aggregation protocol then must ensure that nothing is revealed beyond the aggregate model despite such collusions.

3.1 Overview of `SecAgg`

We now provide an overview of SecAgg. In this discussion, we omit the round index t for simplicity since the procedure is the same at each round. SecAgg ensures privacy against any subset of up to T colluding users and resiliency against D colluding workers provided that $N > D + T$.

In SecAgg , the users mask their models before sharing them with the server using random keys. Specifically, each pair of users $i, j \in [N]$ agree on a pairwise random seed $a_{i,j}$. Moreover, user i also uses a private random seed b_i that is used when the update of this user is delayed but eventually reaches the server. The model of user i is masked as follows

$$\boldsymbol{y}_i = \boldsymbol{x}_i + \mathrm{PRG}(b_i) + \sum_{j:i<j} \mathrm{PRG}(a_{i,j}) - \sum_{j:i>j} \mathrm{PRG}(a_{j,i}), \tag{3}$$

where PRG is a pseudo random generator. The server then recover the aggregate model of the surviving users as follows

$$\sum_{i\in\mathcal{U}} \boldsymbol{x}_i = \sum_{i\in\mathcal{U}} (\boldsymbol{y}_i - \mathrm{PRG}(b_i)) + \sum_{i\in\mathcal{D}} \left(\sum_{j:i<j} \mathrm{PRG}(a_{i,j}) - \sum_{j:i>j} \mathrm{PRG}(a_{j,i}) \right). \tag{4}$$

3.2 Overview of `LightSecAgg`

Next, we provide an overview of LightSecAgg. LightSecAgg has three parameters T that represents the privacy guarantee, D that represents that dropout guarantee and U which represents the targeted number of surviving users. These parameters must be selected such that $N - D \geq U \geq T$. In LightSecAgg, user i selects a random mask $\boldsymbol{z}_i$ and partitions it to $U - T$ sub-masks denoted by $[\boldsymbol{z}_i]_1, \cdots, [\boldsymbol{z}_i]_{U-T}$. User i also selects another T random masks denoted by $[\boldsymbol{n}_i]_{U-T+1}, \cdots, [\boldsymbol{n}_i]_U$. These U partitions $[\boldsymbol{z}_i]_1, \cdots, [\boldsymbol{z}_i]_{U-T}, [\boldsymbol{n}_i]_{U-T+1}, \cdots, [\boldsymbol{n}_i]_U$ are then encoded through an (N, U) Maximum Distance Separable (MDS) code as follows

$$[\tilde{\boldsymbol{z}}_i]_j = ([\boldsymbol{z}_i]_1, \cdots, [\boldsymbol{z}_i]_{U-T}, [\boldsymbol{n}_i]_{U-T+1}, \cdots, [\boldsymbol{n}_i]_U)\, \boldsymbol{v}_j, \tag{5}$$

where $\boldsymbol{v}_j$ is the j-th column of a Vandermonde matrix $\mathbf{V} \in \mathbb{F}_q^{U\times N}$. After that, user i sends $[\tilde{\boldsymbol{z}}_i]_j$ to user $j \in [N] \setminus \{i\}$. User i then masks its model as follows

$$\boldsymbol{y}_i = \boldsymbol{x}_i + \boldsymbol{z}_i. \tag{6}$$

The goal of the server now is to recover the aggregate model $\sum_{i\in\mathcal{U}_1} \boldsymbol{x}_i$, where $\mathcal{U}_1$ is the set of surviving users in this phase. To do so, each surviving users $j \in \mathcal{U}_1$ sends $\sum_{i\in\mathcal{U}_1} [\tilde{\boldsymbol{z}}_i]_j$ to the server. The server can then recover $\sum_{i\in\mathcal{U}_1} [\boldsymbol{z}_i]_k$ for $k \in [U - T]$ through MDS decoding when it receives at least U messages from the surviving users. We denote this subset of the surviving users by $\mathcal{U}_2$, where $|\mathcal{U}_2| = U$. Finally, the server recovers the aggregate model as $\sum_{i\in\mathcal{U}_1} \boldsymbol{x}_i = \sum_{i\in\mathcal{U}_1} \boldsymbol{y}_i - \sum_{i\in\mathcal{U}_1} \boldsymbol{z}_i$.

4 Buffered Asynchronous Secure Aggregation

In this section, we provide a brief overview of FedBuff [16]. Then we illustrate the incompatibility of SecAgg with asynchronous FL in Section 4.1. Later on, in Section 4.2, we introduce BASecAgg.

In asynchronous FL, the updates of the users are not synchronized while the goal is the same as the synchronous FL, to minimize the global loss function in (1). The server stores each local model that it receives in a buffer of size K and updates the global model when the buffer is full. In our setting, this buffer is *not a secure buffer*, hence our goal is to design the secure aggregation protocol where users send the masked updates to protect the privacy in a way that the server can aggregate the local updates while the server (and potential colluding users) learns no information about the local updates $\boldsymbol{x}_i(t)$ beyond the aggregate of updates stored in the buffer.

Threat Model. We consider a threat model where the users and the server are honest but curious adversaries who follow the protocol but try to infer the local updates of the other users. The secure aggregation protocol guarantees that nothing beyond the aggregate of the local updates is revealed, even if up to T users collude with the server and each other. We consider information-theoretic privacy where from every subset of users $\mathcal{T} \subseteq [N]$ of size at most T, we must have mutual information $I(\{\boldsymbol{x}_i\}_{i\in[N]}; \mathbf{Y} | \sum_{i\in\mathcal{U}} \boldsymbol{x}_i, \mathbf{Z}_{\mathcal{T}}) = 0$, 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 $\mathcal{T}$.

FedBuff. Before presenting our secure aggregate protocol, `BASecAgg`, we first provide an overview about the buffered asynchronous aggregation framework, named FedBuff [16], and describe the challenges that render `SecAgg` incompatible with this framework. The key intuition of FedBuff is to introduce a new design parameter K, buffer size at the server, so that FedBuff has two degrees of freedom, K and the concurrency C while the synchronous FL frameworks have only one degree of freedom, concurrency. The concurrency is the number of users training concurrently and is an important parameter to provide a trade-off between the training time and the data inefficiency. Synchronous FL speeds up training time by increasing the concurrency, but higher concurrency results in data inefficiency [16]. In Fedbuff, however, a high concurrency coupled with a proper value of K results in fast training. In other words, the additional degree of freedom K allows the server to update more frequently than concurrency, which enables FedBuff to achieve data efficiency at high concurrency.

At round t, C users are locally training 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

$$\Delta_i^{(t;t_i)} = \boldsymbol{x}^{(t_i)} - \boldsymbol{x}_i^{(E)}, \tag{7}$$

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

$$\boldsymbol{x}_i^{(e)} = \boldsymbol{x}_i^{(e-1)} - \eta_l \nabla F_i(\boldsymbol{x}_i^{(e-1)}), \tag{8}$$

for $e = 1, \ldots, E$ where $\boldsymbol{x}_i^{(0)} = \boldsymbol{x}^{(t_i)}$, η_l denotes learning rate of the local updates, and F_i is the local loss function of user i defined in (1).

The server stores the received local update in a buffer of size K. When the number of stored local updates in the buffer is K, the server updates the global model by subtracting the aggregate of all local updates from the current global model. Specifically, the global model at the server is updated as follows

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \frac{\eta_g}{\sum_{i\in\mathcal{S}^{(t)}} s(t-t_i)} \sum_{i\in\mathcal{S}^{(t)}} s(t-t_i)\Delta_i^{(t;t_i)}, \tag{9}$$

where $\mathcal{S}^{(t)}$ is an index set of users sending the local updates to the server at round t and η_g is the learning rate of the global updates. $s(\tau)$ is a function to compensate for the staleness satisfying $s(0) = 1$ and monotonically decreasing 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 [22].

4.1 Incompatibility of `SecAgg` with Asynchronous FL

As described in Section 3.1, `SecAgg` is 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 generates random vector by carrying out PRG based on the random seed of $a_{i,j}^{(t)}$ to mask the local update. The additive structure of `SecAgg` has the unique property that these pairwise random vectors cancel out each other when the server aggregates the masked models $\boldsymbol{y}_i^{(t)}$ in (3) because user $i(< j)$ add PRG $\left(a_{i,j}^{(t)}\right)$ to $\boldsymbol{x}_i^{(t)}$ and user $j(> i)$ subtract PRG $\left(a_{i,j}^{(t)}\right)$ from $\boldsymbol{x}_j^{(t)}$. We note that this key agreement protocol must be carried out at each round to prevent the potential privacy leakage.

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 users. Specifically, at round t, user $i \in \mathcal{S}^{(t)}$ sends the masked model $\boldsymbol{y}_i^{(t)}$ to the server and $\boldsymbol{y}_i^{(t)}$ is given by

$$\boldsymbol{y}_i^{(t)} = \Delta_i^{(t;t_i)} + \mathrm{PRG}\left(b_i^{(t_i)}\right) + \sum_{j:i<j} \mathrm{PRG}\left(a_{i,j}^{(t_i)}\right) - \sum_{j:i>j} \mathrm{PRG}\left(a_{j,i}^{(t_i)}\right), \tag{10}$$

where $\Delta_i^{(t;t_i)}$ is local update defined in (7). When $t_i \neq t_j$ for $i, j \in \mathcal{S}^{(t)}$, pairwise random vectors in $\boldsymbol{y}_i^{(t)}$ and $\boldsymbol{y}_j^{(t)}$ are not canceled out as $a_{i,j}^{(t_i)}$ and $a_{i,j}^{(t_j)}$ are not the same. 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.

4.2 The Proposed `BASecAgg` Protocol

Add paragraph to describe a key intuition behind `BASecAgg`: the server can reconstruct the aggregate of random masks in one-shot even though users in $\mathcal{S}^{(t)}$ have different staleness. Three design parameters T, U, D are defined in the same way as in Section 3.2.

`BASecAgg` is composed of three main phases. First, each user generates a random mask to protect the privacy of the local update, and further creates encoded masks via T*-private* Maximum Distance Separable (MDS) code to provide privacy against T colluding users. Each user sends one of the encoded masks to one of other users for the purpose of one-shot recovery. Second, each user trains a local update and converts it from the domain of real numbers to the finite field as generating random masks and MDS encoding are required to be carried out in the finite field to provide information-theoretic privacy. Then, the quantized model is masked by the random mask generated in the first phase, and sent to the server. The server stores the masked update in the buffer. Third, when the buffer is full, the server aggregates K masked updates in the buffer. To remove the randomness in the aggregate of the masked updates, the server reconstructs the aggregated masks in the buffer. To do so, each surviving user sends the aggregate of encoded masks to the server. After receiving a sufficient number of aggregated encoded masks, the server reconstructs the aggregate of masks and the desired result, aggregate of K local updates. We now described the details of these three phases.

4.2.1 Offline Encoding and Sharing of Local Masks

User i generates $\boldsymbol{z}_i^{(t_i)}$ 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. Mask $\boldsymbol{z}_i^{(t_i)}$ is partitioned into $U - T$ sub-masks denoted by $[\boldsymbol{z}_i^{(t_i)}]_1, \cdots, [\boldsymbol{z}_i^{(t_i)}]_{U-T}$. User i also selects another T random masks denoted by $[\boldsymbol{n}_i^{(t_i)}]_{U-T+1}, \cdots, [\boldsymbol{n}_i^{(t_i)}]_U$. These U partitions $[\boldsymbol{z}_i^{(t_i)}]_1, \cdots, [\boldsymbol{z}_i^{(t_i)}]_{U-T}, [\boldsymbol{n}_i^{(t_i)}]_{U-T+1}, \cdots, [\boldsymbol{n}_i^{(t_i)}]_U$ are then encoded through an (N, U) Maximum Distance Separable (MDS) code as follows

$$[\tilde{\boldsymbol{z}}_i^{(t_i)}]_j = \left([\boldsymbol{z}_i^{(t_i)}]_1, \cdots, [\boldsymbol{z}_i^{(t_i)}]_{U-T}, [\boldsymbol{n}_i^{(t_i)}]_{U-T+1}, \cdots, [\boldsymbol{n}_i^{(t_i)}]_U\right) \boldsymbol{v}_j, \tag{11}$$

where $\boldsymbol{v}_j$ is the j-th column of a Vandermonde matrix $\mathbf{V} \in \mathbb{F}_q^{U \times N}$. After that, user i sends $[\tilde{\boldsymbol{z}}_i^{(t_i)}]_j$ to user $j \in [N] \setminus \{i\}$. At the end of this phase, each user $i \in [N]$ has $[\tilde{\boldsymbol{z}}_j^{(t_j)}]_i$ from $j \in [N]$.

4.2.2 Training, Quantizing, Masking, and Uploading of Local Updates

Each user i trains the local model from (7) and (8). 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 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 function 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 [1], which represents the sign of a negative number separately from its magnitude. `BASecAgg` addresses this challenge with a simple stochastic quantization strategy combined with the two's complement representation as described next. For any positive integer $c \geq 1$, we first define a stochastic rounding function as

$$Q_c(x) = \begin{cases} \frac{\lfloor cx \rfloor}{c} & \text{with prob. } 1-(cx-\lfloor cx \rfloor) \\ \frac{\lfloor cx \rfloor + 1}{c} & \text{with prob. } cx - \lfloor cx \rfloor, \end{cases} \tag{12}$$

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, which will be described in Lemma xxx in Section 5 in detail. We then define the quantized update

$$\overline{\Delta}_i^{(t;t_i)} := \phi\left(c_l \cdot Q_{c_l}\left(\Delta_i^{(t;t_i)}\right)\right), \tag{13}$$

where the function Q_c from (12) is carried out element-wise, and c_l is a positive integer parameter to determine the quantization level of the local updates. A mapping function $\phi : \mathbb{R} \to \mathbb{F}_p$ is defined to represent a negative integer in the finite field by using the two's complement representation,

$$\phi(x) = \begin{cases} x & \text{if } x \geq 0 \\ p + x & \text{if } x < 0. \end{cases} \tag{14}$$

To protect the privacy of the local updates, user i masks the quantized update $\overline{\Delta}_i^{(t;t_i)}$ in (13) as

$$\tilde{\Delta}_i^{(t;t_i)} = \overline{\Delta}_i^{(t;t_i)} + \boldsymbol{z}_i^{(t_i)}, \tag{15}$$

and sends the pair of $\left\{\tilde{\Delta}_i^{(t;t_i)}, t_i\right\}$ to the server. The local round index t_i will be 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 4.2.3.

4.2.3 One-shot Aggregate-update Recovery and Global Model Update

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

$$\sum_{i \in \mathcal{S}^{(t)}} s(t-t_i)\Delta_i^{(t;t_i)}, \tag{16}$$

where $\Delta_i^{(t;t_i)}$ is the local update in the real domain defined in (7), $\mathcal{S}^{(t)}$ ($\left|\mathcal{S}^{(t)}\right| = K$) is the index set of users sending the local updates to the server at round t, and $s(\tau)$ is the staleness function defined in (9). To do so, the first step is to reconstruct $\sum_{i \in \mathcal{S}^{(t)}} s(t-t_i)\boldsymbol{z}_i^{(t_i)}$. This requires a challenging task as decoding for this 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 $\overline{s} : \{0, 1, \dots, \} \to \mathbb{F}_q$,

$$\overline{s}_{c_s}(\tau) = c_s Q_{c_s}\left(s(\tau)\right), \tag{17}$$

where $Q_s(\cdot)$ is a stochastic rounding function defined in (12), and c_s is a positive integer to determine the quantization level of staleness function. Then, the server broadcasts information of $\left\{\mathcal{S}^{(t)}, \{t_i\}_{i \in \mathcal{S}^{(t)}}, c_s\right\}$ to all surviving users. After identifying the selected users in $\mathcal{S}^{(t)}$, local round indices $\{t_i\}_{i \in \mathcal{S}^{(t)}}$ and corresponding staleness, user $j \in [N]$ aggregates its encoded sub-masks

$$\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_s}(t-t_i)\left[\tilde{\boldsymbol{z}}_i^{(t_i)}\right]_j, \tag{18}$$

and sends it to the server for the purpose of one-shot recovery. We note that each $\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_s}(t-t_i)\left[\tilde{\boldsymbol{z}}_i^{(t_i)}\right]_j$ in (18) is an encoded version of $\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_s}(t-t_i)\left[\boldsymbol{z}_i^{(t_i)}\right]_k$ for $k \in [U-T]$ using the MDS matrix (or Vandermonde matrix) $\mathbf{V}$ defined in (11). Thus, after receiving a set of any U results from surviving users in $\mathcal{U}_2$ where $|\mathcal{U}_2| = U$, the server can reconstruct $\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_s}(t-t_i)\left[\boldsymbol{z}_i^{(t_i)}\right]_k$ for $k \in [U-T]$ via MDS decoding. By concatenating $U-T$ aggregated sub-masks $\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_s}(t-t_i)\left[\boldsymbol{z}_i^{(t_i)}\right]_k$, the server can recover $\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_s}(t-t_i)\boldsymbol{z}_i^{(t_i)}$. Finally, the server obtains the desired global update as follows

$$\boldsymbol{g}^{(t)} = \frac{1}{c_s c_l \sum_{i \in \mathcal{S}^{(t)}} s_{c_s}(t-t_i)} \phi^{-1}\left(\sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_s}(t-t_i)\tilde{\Delta}_i^{(t;t_i)} - \sum_{i \in \mathcal{S}^{(t)}} \overline{s}_{c_s}(t-t_i)\boldsymbol{z}_i^{(t_i)}\right), \tag{19}$$

where c_l is the positive constant defined in (13) and $\phi^{-1} : \mathbb{F}_q \to \mathbb{R}$ is the demapping function defined as follows

$$\phi^{-1}(\overline{x}) = \begin{cases} \overline{x} & \text{if} \quad 0 \leq \overline{x} < \frac{q-1}{2} \\ \overline{x} - q & \text{if} \quad \frac{q-1}{2} \leq \overline{x} < q, \end{cases} \tag{20}$$

where q is the field size. Finally, the server updates the global model as

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta_g \boldsymbol{g}^{(t)}, \tag{21}$$

which is equivalent to

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \frac{\eta_g}{\sum_{i \in \mathcal{S}^{(t)}} Q_{c_g}\left(s(t-t_i)\right)} \sum_{i \in \mathcal{S}^{(t)}} Q_{c_g}\left(s(t-t_i)\right) Q_{c_l}\left(\Delta_i^{(t;t_i)}\right), \tag{22}$$

where Q_{c_l} and Q_{c_g} are the stochastic round function defined in (12) with respect to quantization parameter c_l and c_g, respectively.

5 Convergence Analysis

In this section, we provide the convergence guarantee of `BASecAgg` in the L-smooth and non-convex setting. For simplicity, we consider the constant staleness function $s(\tau) = 1$ for all τ in (22). Then, the global update equation of `BASecAgg` is given by

$$\boldsymbol{x}^{(t+1)} = \boldsymbol{x}^{(t)} - \eta_g \sum_{i \in \mathcal{S}^{(t)}} Q_{c_l}\left(\Delta_i^{(t;t_i)}\right), \tag{23}$$

where Q_{c_l} is the stochastic round function defined in (12) and c_l is the positive constant to determine the qunatization level. We first introduce our assumptions, which are commonly made in analyzing FL algorithms [13, 16, 18, 19].

Assumption 1 *$F_1, \ldots, F_N$ in (1) are all ρ-smooth: for all $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$ and $i \in [N]$, $F_i(\boldsymbol{a}) \leq F_i(\boldsymbol{b}) + (\boldsymbol{a} - \boldsymbol{b})^\top \nabla F_i(\boldsymbol{b}) + \frac{\rho}{2}\|\boldsymbol{a} - \boldsymbol{b}\|^2$.*

Assumption 2 *$F_1, \ldots, F_N$ in (1) are all μ-strongly convex: for all $\boldsymbol{a}, \boldsymbol{b} \in \mathbb{R}^d$ and $i \in [N]$, $F_i(\boldsymbol{a}) \geq F_i(\boldsymbol{b}) + (\boldsymbol{a} - \boldsymbol{b})^\top \nabla F_i(\boldsymbol{b}) + \frac{\mu}{2}\|\boldsymbol{a} - \boldsymbol{b}\|^2$.*

Assumption 3 *Let $\xi_i^{(t)}$ be a sample uniformly selected from the local dataset $\mathcal{D}_i$. The variance of the stochastic gradients at each user is bounded, i.e., $\mathbb{E}\|\nabla F_i(\boldsymbol{x}_i^{(t)}, \xi_i^{(t)}) - \nabla F_i(\boldsymbol{x}_i^{(t)})\|^2 \leq \sigma_i^2$ for $i \in [N]$.*

Assumption 4 *The expected squared norm of the stochastic gradients is uniformly bounded, i.e., $\mathbb{E}\|\nabla F_i(\boldsymbol{x}_i^{(t)}, \xi_i^{(t)})\|^2 \leq G^2$ for all $i \in [N]$.*

Our first lemma states the unbiasedness and bounded variance of the quantized gradient estimator $Q_c(g(\mathbf{w}, \xi))$ for any vector $\mathbf{w} \in \mathbb{R}^d$.

Lemma 1 *For the quantized gradient estimator $Q_c(g(\mathbf{w}, \xi))$ with a given vector $\mathbf{w} \in \mathbb{R}^d$ where ξ is a uniform random variable representing the sample drawn, g is a gradient estimator such that $\mathbb{E}_\xi[g(\mathbf{w}, \xi)] = \nabla F(\mathbf{w})$ and $\mathbb{E}_\xi\|g(\mathbf{w}, \xi) - \nabla F(\mathbf{w})\|^2 = d\sigma^2(\mathbf{w})$, and the stochastic rounding function Q_c is given in (12), the following holds,*

$$\mathbb{E}_{Q,\xi}[Q_c(g(\mathbf{w}, \xi))] = \nabla F(\mathbf{w}) \tag{24}$$

$$\mathbb{E}_{Q,\xi}\|Q_c(g(\mathbf{w}, \xi)) - \nabla F(\mathbf{w})\|^2 \leq d\sigma'^2(\mathbf{w}), \tag{25}$$

where $\sigma'(\mathbf{w}) = \sqrt{\frac{1}{4q^2} + \sigma^2(\mathbf{w})}$.

6 Experiments

We need to include followings.

- Dataset
  - MNIST
  - CIFAR-10
- Performance comparison - Fedbuff (real domain) vs LightSecAgg (Finite Field)
- Impact of scaling constants on the performance

Remark 1 (How to select c_l and c_g?).

7 Conclusions

References

[1] Dan Alistarh, Demjan Grubic, Jerry Li, Ryota Tomioka, and Milan Vojnovic. Qsgd: Communication-efficient sgd via gradient quantization and encoding. In *Advances in Neural Information Processing Systems*, pages 1709–1720, 2017.

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

[3] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

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

[5] Zheng Chai, Yujing Chen, Liang Zhao, Yue Cheng, and Huzefa Rangwala. FedAt: A communication-efficient federated learning method with asynchronous tiers under non-iid data. *arXiv preprint arXiv:2010.05958*, 2020.

[6] Yujing Chen, Yue Ning, Martin Slawski, and Huzefa Rangwala. Asynchronous online federated learning for edge devices with non-iid data. In *2020 IEEE International Conference on Big Data (Big Data)*, pages 15–24. IEEE, 2020.

[7] Ahmed Roushdy Elkordy and A Salman Avestimehr. Secure aggregation with heterogeneous quantization in federated learning. *arXiv preprint arXiv:2009.14388*, 2020.

[8] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, pages 1322–1333, 2015.

[9] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients–how easy is it to break privacy in federated learning? *arXiv preprint arXiv:2003.14053*, 2020.

[10] Bin Gu, An Xu, Zhouyuan Huo, Cheng Deng, and Heng Huang. Privacy-preserving asynchronous vertical federated learning algorithms for multiparty collaborative learning. *IEEE Transactions on Neural Networks and Learning Systems*, 2021.

[11] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2009.11248*, 2020.

[12] Ryan Karl, Jonathan Takeshita, and Taeho Jung. Cryptonite: A framework for flexible time-series secure aggregation with non-interactive fault recovery.

[13] Xiang Li, Kaixuan Huang, Wenhao Yang, Shusen Wang, and Zhihua Zhang. On the convergence of fedavg on non-iid data. In *International Conference on Learning Representations*, 2019.

[14] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Int. Conf. on Artificial Int. and Stat. (AISTATS)*, pages 1273–1282, 2017.

[15] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*, pages 739–753. IEEE, 2019.

[16] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Michael Rabbat, Mani Malek Esmaeili, and Dzmitry Huba. Federated learning with buffered asynchronous aggregation. *arXiv preprint arXiv:2106.06639*, 2021.

[17] Jungwuk Park, Dong-Jun Han, Minseok Choi, and Jaekyun Moon. Sself: Robust federated learning against stragglers and adversaries. 2020.

[18] Sashank Reddi, Zachary Charles, Manzil Zaheer, Zachary Garrett, Keith Rush, Jakub Konečnỳ, Sanjiv Kumar, and H Brendan McMahan. Adaptive federated optimization. *arXiv preprint arXiv:2003.00295*, 2020.

[19] Jinhyun So, Ramy E Ali, Basak Guler, Jiantao Jiao, and Salman Avestimehr. Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning. *arXiv preprint arXiv:2106.03328*, 2021.

[20] Jinhyun So, Başak Güler, and A Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):479–489, 2021.

[21] Marten van Dijk, Nhuong V Nguyen, Toan N Nguyen, Lam M Nguyen, Quoc Tran-Dinh, and Phuong Ha Nguyen. Asynchronous federated learning with reduced number of rounds and with differential privacy from less aggregated gaussian noise. *arXiv preprint arXiv:2007.09208*, 2020.

[22] Cong Xie, Sanmi Koyejo, and Indranil Gupta. Asynchronous federated optimization. *arXiv preprint arXiv:1903.03934*, 2019.

[23] Chien-Sheng Yang, Jinhyun So, Chaoyang He, Songze Li, Qian Yu, and Salman Avestimehr. LightSecAgg: Rethinking secure aggregation in federated learning. *arXiv preprint*, 2021.

[24] Yizhou Zhao and Hua Sun. Information theoretic secure aggregation with user dropouts. *arXiv preprint arXiv:2101.07750*, 2021.

[25] Ligeng Zhu and Song Han. Deep leakage from gradients. In *Federated Learning*, pages 17–31. Springer, 2020.