Straggler-Resilient Secure Aggregation for Federated Learning

Reent Schlegel*, Siddhartha Kumar*, Eirik Rosnes*, and Alexandre Graell i Amat†*
*Simula UiB, Bergen, Norway
†Department of Electrical Engineering, Chalmers University of Technology, Gothenburg, Sweden

***Abstract*—We present CodedSecAgg, a straggler-resilient secure aggregation scheme for federated learning. CodedSecAgg introduces redundancy on the devices' data across the network, which is leveraged during the iterative learning phase at the central server to update the global model based on the responses of a subset of the devices. Compared to other schemes in the literature, which deal with device dropouts by ignoring the contribution of dropped devices, the proposed scheme does not suffer from the client-drift problem. We apply CodedSecAgg to a classification problem on the MNIST dataset. For a scenario with 120 devices, we show that CodedSecAgg outperforms state-of-the-art LightSecAgg in terms of latency by a factor of 6.6 to 15.8, depending on the number of colluding agents, for an accuracy of 95%.**

I. Introduction

Federated learning [1], [2] is a distributed machine learning framework in which multiple devices and a central server jointly train a global model on the devices' private data. The key idea is that the devices do not exchange their data with the central server. More precisely, at each epoch, the devices train a local model on their local data and send the corresponding locally-trained models to the central server. The central server aggregates the local models to update the global model, which is sent to the devices for the next epoch of the training. Federated learning has been used in real-world applications, e.g., for medical data [3], text predictions on mobile devices [4], or by Apple to personalize Siri.

While in federated learning the devices do not share their local data with the central server, hence some level of privacy is preserved, model inversion attacks [5], [6] can allow the central server to infer information about devices' local data from their local models. To circumvent this problem, secure aggregation protocols [7]–[14] have been proposed to allow the aggregation of local models at the central server without revealing devices' individual models.

Training over multiple devices, with potentially very different computational capabilities and unstable connectivity, can suffer from so-called straggler devices, i.e., devices that take a long time to compute their local gradients. Dropouts, which can be seen as an extreme case of straggling, may also occur during the secure aggregation phase. The schemes in [7]–[14] provide security against inversion attacks and resiliency against dropouts by hiding devices' local models via masking and exploiting secret sharing ideas. The masks have an additive structure so that they can be removed when aggregated at the central server. To provide resiliency against dropouts, secret sharing of the random seeds that generate the masks between the devices is performed, so that the central server can cancel the masks belonging to dropped devices. Among these schemes, LightSecAgg is one of the most efficient ones.

The schemes [7]–[14] ignore the contribution of dropped devices. However, ignoring dropped (or straggling) devices makes these schemes sensitive to the client-drift problem, i.e., the global model tends toward local solutions of the participating devices.

In this paper, we propose CodedSecAgg, a secure aggregation scheme that provides resiliency against stragglers (and hence dropouts) and does not suffer from the client-drift problem. Borrowing ideas from coded distributed computing for straggler mitigation in data centers [15]–[18], CodedSecAgg provides straggler resiliency by introducing redundancy on the devices' local data via Shamir's secret sharing [19]. Particularly, the proposed scheme consists of two phases. In the first phase, each device encodes its data using Shamir's secret sharing scheme (SSS) and sends one piece of the encoded data to each of the other devices. In the second phase, the devices and the central server iteratively and collaboratively train the global model. CodedSecAgg provides information-theoretic security against model inversion attacks up to a given number of colluding malicious agents (including the central server). The proposed scheme is tailored to linear regression. However, it can be applied to nonlinear models via kernel embedding. We apply CodedSecAgg to a classification problem on the MNIST dataset. For a scenario with 120 devices, CodedSecAgg achieves a speed-up factor of 6.6 for 60 colluding agents and up to 15.8 for a single malicious agent compared to LightSecAgg for an accuracy of 95%.

In the context of traditional federated learning (i.e., without secure aggregation) [20], [21] proposed code-based schemes for straggler mitigation. However, the scheme in [20] leaks more information than traditional federated learning. Our scheme in [21] provides straggler resiliency and yields the same level of privacy as traditional federated learning. CodedSecAgg applies similar ideas as in [21] to secure aggregation. To the best of our knowledge, CodedSecAgg is the first scheme that provides resiliency against straggling devices for secure aggregation.

II. Preliminaries

A. Notation

We use uppercase and lowercase bold letters for matrices and vectors, respectively, italics for sets, and sans-serif letters for random variables, e.g., $\boldsymbol{X}$, $\boldsymbol{x}$, $\mathcal{X}$, and X represent a matrix, a

This work was financially supported by the Swedish Research Council under grant 2020-03687.

vector, a set, and a random variable, respectively. An exception to this rule is $\boldsymbol{\epsilon}^{(e)}$, which will denote a matrix. Vectors are represented as row vectors throughout the paper. The transpose of a matrix $\boldsymbol{X}$ is denoted as $\boldsymbol{X}^\top$. The gradient of a function $f(\boldsymbol{X})$ with respect to $\boldsymbol{X}$ is denoted by $\nabla_{\boldsymbol{X}} f(\boldsymbol{X})$. Furthermore, we represent the Euclidean norm of a vector $\boldsymbol{x}$ by $\|\boldsymbol{x}\|$, while the Frobenius norm of a matrix $\boldsymbol{X}$ is denoted by $\|\boldsymbol{X}\|_{\mathrm{F}}$. Given integers $a, b \in \mathbb{Z}$, $a < b$, we define $[a, b] \triangleq \{a, \ldots, b\}$, where $\mathbb{Z}$ is the set of integers, and $[a] \triangleq \{1, \ldots, a\}$ for a positive integer a. For a real number e, $\lfloor e \rfloor$ is the largest integer less than or equal to e. The expectation of a random variable $\mathsf{\Lambda}$ is denoted by $\mathbb{E}[\mathsf{\Lambda}]$, while $\mathsf{I}(\cdot;\cdot)$ denotes mutual information and $\mathsf{H}(\cdot|\cdot)$ conditional entropy. The finite field of order q is denoted by $\mathrm{GF}(q)$.

B. Shamir's Secret Sharing Scheme

Shamir's SSS [19] with parameters (n, k) encodes a secret $x \in \mathrm{GF}(q)$ into n shares $s_1, \ldots, s_n$ such that any subset of less than k shares do not reveal any information about x whereas any subset of k or more shares contain sufficient information to recover x. More precisely, for any $\mathcal{I} \subset \{s_1, \ldots, s_n\}$ with $|\mathcal{I}| < k$ and any $\mathcal{J} \subseteq \{s_1, \ldots, s_n\}$ with $|\mathcal{J}| \geq k$, we have $\mathsf{I}(x; \mathcal{I}) = 0$ and $\mathsf{H}(x|\mathcal{J}) = 0$.

Shamir's SSS encodes x together with $k-1$ independent and uniformly-distributed random numbers $\mathsf{r}_1, \ldots, \mathsf{r}_{k-1} \in \mathrm{GF}(q)$ using a nonsystematic (n, k) Reed-Solomon code. The random numbers $\{\mathsf{r}_i\}$ ensure that any $k-1$ or less shares are independent and uniformly random, i.e., they are statistically independent of x, resulting in $\mathsf{I}(x; \mathcal{I}) = 0$. Additionally, the maximum distance separable property of Reed-Solomon codes ensures that x can be recovered from any k shares, i.e., $\mathsf{H}(x|\mathcal{J}) = 0$.

C. Fixed-Point Numbers

As secret sharing cannot be directly applied to real-valued data, our scheme considers a fixed-point arithmetic representation of the real data and subsequently fixed-point arithmetic operations.

Fixed-point numbers are a datatype to represent rational numbers with a finite amount of bits. Fixed-point numbers of length ℓ and resolution f can be seen as integers from $\mathbb{Z}_{\langle \ell \rangle} = [-2^{\ell-1}, 2^{\ell-1} - 1]$ scaled by 2^{-f}. More precisely, a fixed-point number $\tilde{x}$ is given as $\tilde{x} = \bar{x} \cdot 2^{-f}$, for some $\bar{x} \in \mathbb{Z}_{\langle \ell \rangle}$. We denote the set of all fixed-point numbers of length ℓ and resolution f as $\mathbb{Q}_{\ell,f}$.

Addition of two fixed-point numbers can be performed via standard integer addition with a subsequent modulo operation. To this end, we define $(\cdot)_{\mathbb{Z}_{\langle \ell \rangle}}$ as the map from the integers onto the set $\mathbb{Z}_{\langle \ell \rangle}$ given by the modulo operation. Now, let $\tilde{a}, \tilde{b} \in \mathbb{Q}_{\ell,f}$, with $\tilde{a} = \bar{a} 2^{-f}$ and $\tilde{b} = \bar{b} 2^{-f}$. For $\tilde{c} = \tilde{a} + \tilde{b}$, with $\tilde{c} = \bar{c} 2^{-f}$, we have $\bar{c} = (\bar{a} + \bar{b})_{\mathbb{Z}_{\langle \ell \rangle}}$.

Multiplication over $\mathbb{Q}_{\ell,f}$ is performed via standard integer multiplication, followed by a scaling over the reals to retain the precision, and lastly, a modulo operation. In particular, for $\tilde{d} = \tilde{a} \cdot \tilde{b}$, with $\tilde{d} = \bar{d} 2^{-f}$, we have $\bar{d} = (\lfloor \bar{a} \cdot \bar{b} \cdot 2^{-f} \rfloor)_{\mathbb{Z}_{\langle \ell \rangle}}$.

III. System Model

We consider a scenario where n devices want to collaboratively train a global model on local data with the help of a central server. Each device i has its local data $\{(\boldsymbol{x}_j^{(i)}, \boldsymbol{y}_j^{(i)}) | j \in [m_i]\}$ consisting of m_i training examples, where $\{\boldsymbol{x}_j^{(i)}\}$ are the feature vectors, of dimension d, and $\{\boldsymbol{y}_j^{(i)}\}$ the corresponding labels, of dimension c. More precisely, we consider the case where the devices wish to learn the linear model

$$\boldsymbol{y} = \boldsymbol{x}\boldsymbol{\Theta}\,, \tag{1}$$

where $\boldsymbol{\Theta}$ contains the parameters of the model. To this end, the devices utilize federated learning to train the global model $\boldsymbol{\Theta}$ on the $m = \sum_i m_i$ training examples in the network.

To represent the data, the devices utilize the fixed-point datatype $\mathbb{Q}_{\ell,f}$. In particular, the local data at device i is given as $\boldsymbol{x}_j^{(i)} \in \mathbb{Q}_{\ell,f}^d$ and $\boldsymbol{y}_j^{(i)} \in \mathbb{Q}_{\ell,f}^c$, and we represent the data in matrix form as

$$\boldsymbol{X}^{(i)} = \begin{pmatrix} \boldsymbol{x}_1^{(i)} \\ \vdots \\ \boldsymbol{x}_{m_i}^{(i)} \end{pmatrix} \text{ and } \boldsymbol{Y}^{(i)} = \begin{pmatrix} \boldsymbol{y}_1^{(i)} \\ \vdots \\ \boldsymbol{y}_{m_i}^{(i)} \end{pmatrix}.$$

Furthermore, we represent the global dataset consisting of all m training examples as two matrices $\boldsymbol{X}$ and $\boldsymbol{Y}$ given by

$$\boldsymbol{X} = \begin{pmatrix} \boldsymbol{x}_1 \\ \vdots \\ \boldsymbol{x}_m \end{pmatrix} = \begin{pmatrix} \boldsymbol{X}^{(1)} \\ \vdots \\ \boldsymbol{X}^{(n)} \end{pmatrix} \text{ and } \boldsymbol{Y} = \begin{pmatrix} \boldsymbol{y}_1 \\ \vdots \\ \boldsymbol{y}_m \end{pmatrix} = \begin{pmatrix} \boldsymbol{Y}^{(1)} \\ \vdots \\ \boldsymbol{Y}^{(n)} \end{pmatrix},$$

where $\boldsymbol{X}$ is of size $m \times d$ and $\boldsymbol{Y}$ of size $m \times c$.

A. Federated Gradient Descent

In federated gradient descent, the devices in the network collaboratively and iteratively train a global model $\boldsymbol{\Theta}$ on local data available at the devices. The model $\boldsymbol{\Theta}$ is the solution of the minimization problem

$$\boldsymbol{\Theta} = \arg\min_{\boldsymbol{\Theta}'} \; f(\boldsymbol{\Theta}')\,,$$

where $f(\boldsymbol{\Theta})$ is the global loss function. For a linear model,

$$f(\boldsymbol{\Theta}) \triangleq \frac{1}{2m} \sum_{l=1}^{m} \|\boldsymbol{x}_l \boldsymbol{\Theta} - \boldsymbol{y}_l\|^2 + \frac{\lambda}{2} \|\boldsymbol{\Theta}\|_{\mathrm{F}}^2\,,$$

with regularization parameter λ. Let the local loss function at device i be

$$f_i(\boldsymbol{\Theta}) = \frac{1}{2m_i} \sum_{j=1}^{m_i} \|\boldsymbol{x}_j^{(i)} \boldsymbol{\Theta} - \boldsymbol{y}_j^{(i)}\|^2\,,$$

such that we can express the global loss function as

$$f(\boldsymbol{\Theta}) = \sum_{i=1}^{n} \frac{m_i}{m} f_i(\boldsymbol{\Theta}) + \frac{\lambda}{2} \|\boldsymbol{\Theta}\|_{\mathrm{F}}^2\,.$$

The training in federated gradient descent is performed iteratively over multiple epochs. In each epoch, devices train locally on their data using the local loss function and upload their

local model updates to the central server. The central server aggregates all local model updates to obtain the new global model for the next epoch. In particular, let $\boldsymbol{\Theta}^{(e)}$ be the global model at epoch e. Device i computes

$$\boldsymbol{G}_i^{(e)} = m_i \nabla_{\boldsymbol{\Theta}} f_i(\boldsymbol{\Theta}^{(e)}) = \boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}\boldsymbol{\Theta}^{(e)} - \boldsymbol{X}^{(i)\top}\boldsymbol{Y}^{(i)}\,, \tag{2}$$

and sends $\boldsymbol{G}_i^{(e)}$ to the central server. The central server then computes $\boldsymbol{G}^{(e)} = \sum_i \boldsymbol{G}_i^{(e)}$ and updates the global model as

$$\nabla_{\boldsymbol{\Theta}} f(\boldsymbol{\Theta}^{(e)}) = \frac{1}{m}\boldsymbol{G}^{(e)} + \lambda\boldsymbol{\Theta}^{(e)}\,, \tag{3}$$

$$\boldsymbol{\Theta}^{(e+1)} = \boldsymbol{\Theta}^{(e)} - \mu\nabla_{\boldsymbol{\Theta}} f(\boldsymbol{\Theta}^{(e)})\,, \tag{4}$$

with learning rate μ. The central server sends the updated model $\boldsymbol{\Theta}^{(e+1)}$ to the devices for the next epoch. The computations in (2)–(4) are repeated until convergence, i.e., until $\boldsymbol{\Theta}^{(e+1)} \approx \boldsymbol{\Theta}^{(e)}$.

B. Computation and Communication Model

We assume the time it takes a device to finish a computation is random. We model the computation time as a shifted exponentially-distributed random variable as it is common in the literature [22]. In this model, the computation time is comprised of a deterministic part—the shift—that corresponds to the time it takes a device to perform the calculations in its processing unit and a random delay—exponentially distributed—that corresponds to random events such as memory access and tasks running in the background.

Let $\mathsf{T}_i^{\mathsf{comp}}$ be the time it takes device i to perform ρ_i multiply and accumulate (MAC) operations. We then have

$$\mathsf{T}_i^{\mathsf{comp}} = \frac{\rho_i}{\tau_i} + \Lambda_i\,,$$

with τ_i being the deterministic number of MAC operations device i performs per second and Λ_i the random exponentially-distributed setup time with $\mathbb{E}[\Lambda_i] = 1/\eta_i$.

In practice, the communication links are unreliable and may fail. In case the link fails, the devices will simply retransmit until a successful transmission occurs. Let $\mathsf{N}_i^{\mathsf{u}}$ and $\mathsf{N}_i^{\mathsf{d}}$ be geometrically-distributed random variables with success probability $1-p_i$ that model respectively the number of tries it takes device i to successfully upload and download a message to and from the central server through a channel with failure probability p_i. Furthermore, let γ^{u} and γ^{d} be the data rate in the upload and download, respectively. Then, the time it takes device i to upload and download b bits is given as

$$\mathsf{T}_i^{\mathsf{u}} = \frac{\mathsf{N}_i^{\mathsf{u}}}{\gamma^{\mathsf{u}}}b \quad \text{and} \quad \mathsf{T}_i^{\mathsf{d}} = \frac{\mathsf{N}_i^{\mathsf{d}}}{\gamma^{\mathsf{d}}}b\,,$$

respectively.

In the sharing phase, communication between any two devices is performed via a secure, i.e., encrypted and authenticated, channel and routed through the central server. This enables efficient device-to-device communication between any two devices in the network regardless of spatial separation of the devices.

C. Threat Model and Goal

We assume that all agents, i.e., the n devices and the central server, are honest-but-curious. We further assume that up to z agents in the network collude to infer information about the local datasets of other devices. The goal is to ensure device data privacy against the z colluding agents while providing straggler mitigation in the federated learning scenario. Privacy in this setting means that malicious devices do not gain any information about the local datasets of other devices and that the central server only learns the aggregate of all local gradients to prevent a model inversion attack.

IV. Coded Secure Aggregation

In this section, we present our proposed scheme, CodedSecAgg, which provides straggler mitigation for secure aggregation in federated learning. CodedSecAgg consists of two phases. In the first phase, the devices share their data in a secure manner with other devices. Subsequently, the devices train the global model on their own and shared data in such a way that the computation results of a subset of the devices is sufficient to recover the aggregated model.

The model updates in CodedSecAgg slightly deviate from the standard federated gradient descent described in Section III-A. The global model at epoch e can be expressed as

$$\boldsymbol{\Theta}^{(e)} = \boldsymbol{\Theta}^{(1)} + \boldsymbol{\epsilon}^{(e)}\,,$$

where $\boldsymbol{\epsilon}^{(e)}$ is an update matrix and $\boldsymbol{\Theta}^{(1)}$ is the model at the first epoch. Hence, we can rewrite the local model updates in (2) as

$$\boldsymbol{G}_i^{(e)} = \boldsymbol{G}_i^{(1)} + \boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}\boldsymbol{\epsilon}^{(e)}\,.$$

As a result, the computations at the devices in each epoch are affine in $\boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}$ instead of quadratic in $\boldsymbol{X}^{(i)}$. At the end of epoch e, the central server simply has to send $\boldsymbol{\epsilon}^{(e+1)}$ instead of $\boldsymbol{\Theta}^{(e+1)}$ to the devices.

Remark 1. *Due to the use of Shamir's SSS to encode the devices' data, the proposed scheme applies to models in the form of* (1)*. However, it can also be applied to nonlinear problems via kernel embedding. In Section V, we will apply CodedSecAgg to a classification task.*

A. Data Sharing

The devices encode their transformed dataset $\boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}$ and the gradient at epoch 1 $\boldsymbol{G}_i^{(1)}$ using Shamir's (n,k) SSS to obtain $2n$ shares $\boldsymbol{S}_{\mathsf{x},1}^{(i)},\dots,\boldsymbol{S}_{\mathsf{x},n}^{(i)}$ and $\boldsymbol{S}_{\mathsf{g},1}^{(i)},\dots,\boldsymbol{S}_{\mathsf{g},n}^{(i)}$ (see Section II-B). To this end, a fixed-point number $\tilde{x} = \bar{x}2^{-f}$ is represented by its integer part $\bar{x}$. The integer $\bar{x} \in \mathbb{Z}_{\langle\ell\rangle}$ is then mapped into the field $\mathrm{GF}(q)$ for a prime $q > 2^{\ell+f}$. We need a field of size at least $2^{\ell+f}$ because we need to modify the multiplication in $\mathbb{Q}_{\ell,f}$ in our scheme. We postpone the scaling by 2^{-f} until after the decoding of the SSS at the central server. That means, to compute $\tilde{d} = \tilde{a}\cdot\tilde{b}$, the devices compute $\bar{d}' = \bar{a}'\cdot\bar{b}'$, where $\bar{a}', \bar{b}', \bar{d}' \in \mathrm{GF}(q)$. In turn, the central server will map $\bar{d}'$ to the integers and apply the scaling by 2^{-f} to get back the initial accuracy. In order to get the correct $\bar{d}$

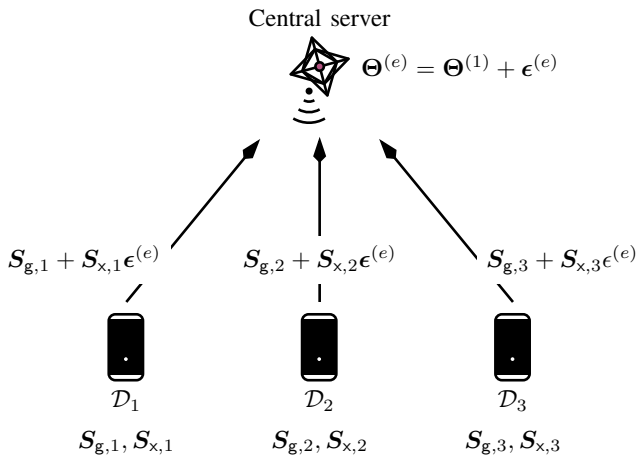

Fig. 1. An example showcasing an epoch of CodedSecAgg. The system consists of $n = 3$ devices, labeled D_1 to D_3, and a central server. Each device has access to one share of the global dataset.

corresponding to $\tilde{d}$ we have to ensure that no overflow in the finite field due to the multiplication without scaling occurs. Therefore, we have to facilitate integers in $\mathbb{Z}_{\langle \ell+f \rangle}$ and not just in $\mathbb{Z}_{\langle \ell \rangle}$ for which we need a field size $q > 2^{\ell+f}$ instead of $q > 2^{\ell}$. After mapping the entries of $\boldsymbol{X}^{(i)\top}\boldsymbol{X}^{(i)}$ and $\boldsymbol{G}_i^{(1)}$ to GF(q), the devices apply Shamir's SSS (see Section II-B).

Subsequently, device i sends $\boldsymbol{S}_{\mathsf{x},j}^{(i)}$ and $\boldsymbol{S}_{\mathsf{g},j}^{(i)}$ to device j for $j \in [n] \backslash i$. For each device, this encompasses transmitting $(n-1)(d^2+dc)$ elements from GF(q). After receiving all $n-1$ incoming transmissions, device i computes $\boldsymbol{S}_{\mathsf{x},i} = \sum_j \boldsymbol{S}_{\mathsf{x},i}^{(j)}$ and $\boldsymbol{S}_{\mathsf{g},i} = \sum_j \boldsymbol{S}_{\mathsf{g},i}^{(j)}$. Due to the linearity of Shamir's SSS, $\{\boldsymbol{S}_{\mathsf{x},i}\}$ is a secret sharing of $\boldsymbol{X}^\top \boldsymbol{X}$ and $\{\boldsymbol{S}_{\mathsf{g},i}\}$ is a secret sharing of $\boldsymbol{G}^{(1)}$.

B. Iterative Learning

The devices perform the training of the global model directly on their shares of the global data $\{\boldsymbol{S}_{\mathsf{x},i}\}$ and $\{\boldsymbol{S}_{\mathsf{g},i}\}$. In epoch e, device i computes

$$\tilde{\boldsymbol{G}}_i^{(e)} = \boldsymbol{S}_{\mathsf{g},i} + \boldsymbol{S}_{\mathsf{x},i}\boldsymbol{\epsilon}^{(e)} \,. \tag{5}$$

The result $\tilde{\boldsymbol{G}}_i^{(e)}$ is a share of the global gradient $\boldsymbol{G}^{(e)}$ at epoch e. Upon completion of (5), device i sends $\tilde{\boldsymbol{G}}_i^{(e)}$ to the central server. After receiving the result of (5) from k different devices, the central server can decode the SSS, map the result to $\mathbb{Z}_{\langle \ell+f \rangle}$, and apply the scaling by 2^{-f} to obtain $\boldsymbol{G}^{(e)}$. In turn, the central server updates the global model in the usual manner described in (3) and (4). The scheme is illustrated in Fig. 1.

In CodedSecAgg the aggregated gradient at each epoch of the gradient descent algorithm is equivalent to the gradient in traditional federated learning and hence, CodedSecAgg converges to the global optimum.

C. Analysis

In CodedSecAgg, any device has access to at most one share of the data of any other device. Shamir's SSS guarantees that any less than k shares do not leak any information about the secret. As a result, CodedSecAgg provides privacy against any $z < k$ colluding agents including the central server. At each epoch, the central server collects shares of the global gradient. The only information it can infer from these shares is the global gradient itself. The central server does not have access to any (shares of) local gradients or datasets and hence, a model inversion attack is prevented.

CodedSecAgg trades off an initial communication delay—due to the sharing phase—with much shorter epoch times—thanks to the introduced redundancy. The introduced redundancy can be leveraged at the central server in each epoch to recover the aggregated gradient from the responses of the k fastest devices in that epoch. This significantly speeds up the federated gradient descent compared to existing secure aggregation protocols, where the central server has to wait for all devices in the network to finish their computations in order to incorporate their model updates into the global model.

V. Numerical Results

We apply CodedSecAgg to a classification task on the MNIST dataset. We simulate a scenario with $n = 120$ devices and a central server. We utilize kernel embedding to linearize the classification problem via Python's radial basis function sampler of the sklearn package (with 5 as kernel parameter and 2000 features). We sort the training data by the labels prior to distributing it across the devices in order to simulate non-identically distributed data. As a result, in our simulation we have $d = 2000$, $c = 10$ (we use one-hot encoding for the labels), and $m_i = 500$. At the central server we use regularization parameter $\lambda = 9 \times 10^{-6}$ and an initial learning rate of $\mu = 6.0$, which is updated as $\mu \leftarrow 0.8\mu$ at epochs 200 and 350.

We represent the fixed-point numbers with $\ell = 48$ bits out of which $f = 24$ bits are for the fractional part. For CodedSecAgg, we assume that the encoding of the data is performed offline as no interaction is required for this step. For the communication we assume the devices utilize the LTE Cat 1 standard for Internet of Things devices. This results in data rates of $\gamma^{\mathsf{d}} = 10$ Mbit/s and $\gamma^{\mathsf{u}} = 5$ Mbit/s. We assume a channel failure probability of $p_i = 0.1$ and a 10% header overhead for each transmission. Furthermore, each device is capable of full-duplex transmission (as per the LTE Cat 1 standard) and we assume that there are sufficient orthogonal channels for all devices, i.e., we assume that all devices can simultaneously upload and download data to and from the central server.

To simulate a heterogeneous network, we sample the MAC rate τ_i of device i uniformly at random from $25 \cdot 10^6$, $5 \cdot 10^6$, $2.5 \cdot 10^6$, and $1.25 \cdot 10^6$ MAC/s. The setup times Λ_i at the devices are sampled at each epoch. We assume an expected value of half of the deterministic computation time, i.e., when computing ρ_i MAC operations at device i we have $\eta_i = \frac{2\tau_i}{\rho_i}$.

In Fig. 2, we compare CodedSecAgg with the state-of-the-art secure aggregation scheme LightSecAgg [12] for different numbers of colluding agents z. In LightSecAgg, devices compute the local model updates only on their data in each epoch. Subsequently, the devices secretly share the local gradients in order to securely aggregate them at the central

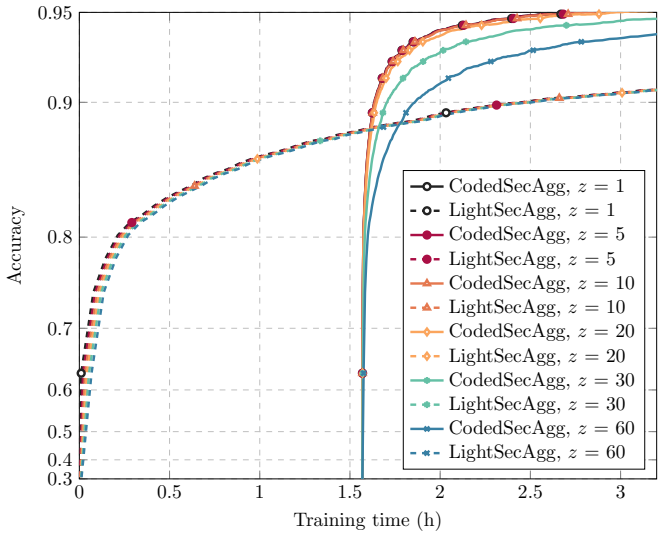

Fig. 2. Accuracy versus training time for a classification problem on the MNIST dataset for CodedSecAgg (solid curves) and LightSecAgg (dashed curves) with $n = 120$ devices and security against z colluding agents.

server. As a result, there is no big initial sharing phase as in CodedSecAgg. However, as there is no redundancy of data in the network, at each epoch of LightSecAgg the central server has to wait for all devices to finish their gradient computations before the secure aggregation can start. This results in longer epoch times than for CodedSecAgg. As we can see in Fig. 2, CodedSecAgg trades off the initial communication latency for drastically reduced epoch times. As a result, for meaningful levels of accuracy above 90%, CodedSecAgg yields significantly lower latency regardless of the privacy level z.

The impact of the privacy level z differs between the two schemes. Since LightSecAgg succumbs to the straggler problem, the computation times of the model upgrades are a major contributor to the overall latency of the scheme. Furthermore, during each epoch, the gradients have to be shared in a secure manner. Both the computation times and the sharing are independent of z. Hence, there is only a small increase in latency for LightSecAgg with increasing z. For CodedSecAgg, z has a direct impact on the straggler mitigation capabilities. Since the central server has to wait for the responses of the k fastest devices and we have $k > z$, the epoch times increase significantly with increasing z. However, even when half the devices in the network collude, i.e., $z = 60$, CodedSecAgg still gives a speed-up of 6.6 compared to LightSecAgg for an accuracy of 95%. For $z = 1$, this speed-up increases to 15.8.

VI. Conclusion

We presented CodedSecAgg, a straggler-resilient secure aggregation scheme for federated learning. CodedSecAgg relies on the linearity of the learning problem, but can be applied to nonlinear tasks via kernel embedding. For a classification task on the MNIST dataset and a scenario with 120 devices, CodedSecAgg outperforms state-of-the-art LightSecAgg in terms of latency by a factor of 6.6 for 60 colluding agents for a target accuracy of 95%. The speed-up factor increases to 15.8 for a single malicious agent.

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. Agüera y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Proc. Int. Conf. Artificial Intell. Stats. (AISTATS)*, Fort Lauderdale, FL, Apr. 2017, pp. 1273–1282.

[2] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, "Federated learning: Strategies for improving communication efficiency," in *NIPS Workshop Private Multi-Party Mach. Learn. (PMPML)*, Barcelona, Spain, Dec. 2016.

[3] A. Jochems *et al.*, "Developing and validating a survival prediction model for NSCLC patients through distributed learning across 3 countries," *Int. J. Radiat. Oncol. Biol. Phys.*, vol. 99, no. 2, pp. 344–352, Oct. 2017.

[4] K. Bonawitz *et al.*, "Towards federated learning at scale: System design," in *Proc. Mach. Learn. Syst. (MLSys)*, Stanford, CA, Mar./Apr. 2019, pp. 374–388.

[5] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Denver, CO, Oct. 2015, pp. 1322–1333.

[6] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, "Beyond inferring class representatives: User-level privacy leakage from federated learning," in *Proc. IEEE Int. Conf. Comp. Commun. (INFOCOM)*, Paris, France, Apr./May 2019, pp. 2512–2520.

[7] K. Bonawitz *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, Dallas, TX, Oct./Nov. 2017, pp. 1175–1191.

[8] S. Kadhe, N. Rajaraman, O. O. Koyluoglu, and K. Ramachandran, "FastSecAgg: Scalable secure aggregation for privacy-preserving federated learning," in *Int. Workshop Fed. Learn. User Privacy Data Confidentiality*, Vienna, Austria, Jul. 2020.

[9] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE J. Sel. Areas Inf. Theory*, vol. 2, no. 1, pp. 479–489, Mar. 2021.

[10] A. R. Elkordy and A. S. Avestimehr, "HeteroSAg: Secure aggregation with heterogeneous quantization in federated learning," *IEEE Trans. Commun.*, vol. 70, no. 4, pp. 2372–2386, Apr. 2022.

[11] J. So, R. E. Ali, B. Güler, and A. S. Avestimehr, "Secure aggregation for buffered asynchronous federated learning," in *1st NeurIPS Workshop New Frontiers Fed. Learn. (NFFL)*, online, Dec. 2021.

[12] J. So, C. He, C.-S. Yang, S. Li, Q. Yu, R. E. Ali, B. Güler, and S. Avestimehr, "LightSecAgg: a lightweight and versatile design for secure aggregation in federated learning," in *Proc. Mach. Learn. Syst. (MLSys)*, Santa Clara, CA, Aug./Sep. 2022.

[13] Y. Zhao and H. Sun, "Information theoretic secure aggregation with user dropouts," in *Proc. IEEE Int. Symp. Inf. Theory (ISIT)*, Melbourne, Australia, Jul. 2021, pp. 1124–1129.

[14] T. Jahani-Nezhad, M. A. Maddah-Ali, S. Li, and G. Caire, "SwiftAgg+: Achieving asymptotically optimal communication load in secure aggregation for federated learning," Mar. 2022, arXiv:2203.13060.

[15] S. Li, M. A. Maddah-Ali, and A. S. Avestimehr, "A unified coding framework for distributed computing with straggling servers," in *Proc. IEEE Globecom Workshops (GC Wkshps)*, Washington, DC, Dec. 2016.

[16] K. Lee, M. Lam, R. Pedersani, D. Papailiopoulos, and K. Ramachandran, "Speeding up distributed machine learning using codes," *IEEE Trans. Inf. Theory*, vol. 64, no. 3, pp. 1514–1529, Mar. 2018.

[17] A. Severinson, A. Graell i Amat, and E. Rosnes, "Block-diagonal and LT codes for distributed computing with straggling servers," *IEEE Trans. Commun.*, vol. 67, no. 3, pp. 1739–1753, Mar. 2019.

[18] S. Dutta, V. Cadambe, and P. Grover, ""Short-Dot": Computing large linear transforms distributedly using coded short dot products," *IEEE Trans. Inf. Theory*, vol. 65, no. 10, pp. 6171–6193, Oct. 2019.

[19] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, Nov. 1979.

[20] S. Prakash, S. Dhakal, M. R. Akdeniz, Y. Yona, S. Talwar, S. Avestimehr, and N. Himayat, "Coded computing for low-latency federated learning over wireless edge networks," *IEEE J. Sel. Areas Commun.*, vol. 39, no. 1, pp. 233–250, Jan. 2021.

[21] S. Kumar, R. Schlegel, E. Rosnes, and A. Graell i Amat, "Coding for straggler mitigation in federated learning," in *Proc. IEEE Int. Conf. Commun. (ICC)*, Seoul, Korea, May 2022.

[22] J. Zhang and O. Simeone, "On model coding for distributed inference and transmission in mobile edge computing systems," *IEEE Commun. Lett.*, vol. 23, no. 6, pp. 1065–1068, Jun. 2019.