Fast Secure and Robust Aggregation Learning Frameworks on Distributed and Federated Setups

Beom Jin Lee
Swanand Kadhe
Kannan Ramchandran

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2021-83
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2021/EECS-2021-83.html

May 14, 2021

Fast Secure and Robust Aggregation Learning Frameworks on Distributed and Federated Setups

by Beom Jin Lee

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences, University of California at Berkeley, in partial satisfaction of the requirements for the degree of **Master of Science, Plan II**.

Approval for the Report and Comprehensive Examination:

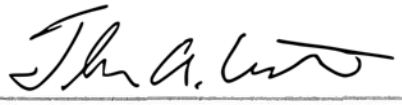
Committee:

Professor Kannan Ramchandran
Research Advisor

05/11/2021

(Date)

* * * * * * *



Professor Thomas Courtade
Second Reader

May 11, 2021

(Date)

Fast Secure and Robust Aggregation Learning Frameworks on Distributed and Federated Setups

by

Beom Jin Lee

A thesis submitted in partial satisfaction of the

requirements for the degree of

Master of Science

in

Electrical Engineering and Computer Sciences

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Kannan Ramchandran, Chair
Professor Thomas Courtade

Spring 2021

Abstract

Fast Secure and Robust Aggregation Learning Frameworks on Distributed and Federated Setups

by

Beom Jin Lee

Master of Science in Electrical Engineering and Computer Sciences

University of California, Berkeley

Professor Kannan Ramchandran, Chair

The plethora of data and increasing computational complexity of deep neural networks have led to deep learning schemes that require the use of multiple nodes in a cluster setup. Yet, distributing a computation over multiple machines has two main flaws: (1) it induces a higher risk of failures and (2) it induces heavy communication costs, which can sometimes outweigh the computational gains from using distributed learning. We study methods to tackle both problems by borrowing ideas from sketching and byzantine-worker literature. We show that our algorithm, SketchedRobustAgg, achieves similar runtime (measured by number of iterations) as without using sketching, even though the algorithm sends s-dimensional (where $s \ll d$) vectors between worker and parameter server.

At the same time, the plethora of data induces another challenge in privacy. Motivated by recent work around attacking federated learning schemes that demonstrate keeping training data on clients' devices do not provide sufficient privacy, we introduce FastSecAgg. We show that FastSecAgg, a secure aggregation protocol, is efficient in computation and communication, and also robust to client dropouts. FastSecAgg achieves significantly smaller computation cost, while achieving same communication cost asymptotically. We finally show that FastSecAgg performs well against benchmark federated learning datasets, even with aggressive quantization and sketching, and furthermore show empirically that it is possible to control tradeoff between computation/communication complexities and test accuracies.

Contents

List of Figures

List of Tables

Acknowledgments

I would like to first thank Swanand Kadhe and Kannan Ramchandran for giving me this opportunity to work on this research project. Thank you for giving me a proper introduction to the distributed and federated learning. I am very grateful to have worked with you during my time at Berkeley, and I have learned so much about doing research under your guidances.

I would also like to thank my friends, who colored my college career. Lastly, this thesis would not have been possible without the support of my parents, Su Jeong Han and Gwang Moo Lee. Thank you for supporting my career in every way, and always being there for me.

Chapter 1

Introduction

1.1 Background

The plethora of data and increasing computational complexity of deep neural networks have led to deep learning schemes that require the use of multiple nodes in a cluster setup, and much of industry-standard deep learning training setups use cluster setups, for example, [32, 47, 48]. A classic way of using distributed learning is to use a parameter-server framework [35], where k worker nodes run stochastic gradient descent on their mini-batch and send the gradients to the parameter server, where it then aggregates the gradients (oftentimes by taking the average). Because we distribute work across multiple workers, this method allows us to converge to an optimum in fewer iterations.

However, distributing computation over multiple machines has two main flaws: (1) it induces a higher risk of failures and (2) it incurs high communication costs, which can sometimes outweigh the computational gains from using distributed learning. In particular, failures can come from multiple sources including stalled processes, crashed and computational errors, but worse yet, byzantine attackers trying to compromise the entire system. Indeed, the most robust system is one that can tolerate even Byzantine failures, which are completely arbitrary behaviors of some processes. Likewise, heavy communication costs are often incurred from sending model parameters back and forth between the server and worker nodes.

A recent line of work [57, 19] studied the first problem of risk of failures, and tackles the problem from two main points of views. The first point of view coming from using l_2 distance between generated gradients [9], and the second using coding theory to ensure robustness during distributed learning [19, 43]. A number of works also study the second problem, approaching the problem from quantization [3], sparsity and compression methods [8].

Along with the increasing computational complexity, another issue that arises from the plethora of data is privacy considerations. In this work, in addition to distributed learning

setups to improve runtime complexities, we also study the problem of Federated Learning, a distributed learning paradigm that enables a large number of clients to coordinate with a central server to learn a shared model while keeping all the training data on clients' devices. Federated Learning setups ensure that every client keeps their local data on their own device, and sends only model updates that are functions of their dataset.

Federated Learning suffers from two similar problems: (1) model parameters can leak information about clients' sensitive data and (2) existing aggregation protocols incur high computation/communication costs. Indeed, recent attacks on Federated Learning demonstrate that keeping the training data on clients' devices does not provide sufficient privacy, as the model parameters shared by clients can leak information about their training data. In fact, certain neural networks (e.g., generative text models) trained on sensitive data (e.g., private text messages) can memorize the training data [18]. This makes it possible for an adversary can launch various types of inversion and inference attacks on the model parameters [23, 45, 24].

A recent line of work around secure aggregation protocols can be used to provide strong privacy guarantees in FL. At a high level, secure aggregation is a secure multi-party computation (MPC) protocol that enables the server to compute the sum of clients' model updates without learning any information about any client's individual update [11]. Secure aggregation can be efficiently composed with differential privacy to further enhance privacy guarantees [11, 26, 55]. Indeed, secure aggregation problem for securely computing the summation has received significant research attention in the past few years [26].

Sketching, or data dimensionality reduction, is a common technique to quickly reduce the size of a large-scale optimization problem while preserving the solution space. Sketching has been widely used in numerical linear algebra and machine learning, and it has led to near-optimal algorithms for a number of fundamental problems in this area. Typically, one is given a large data matrix X, and we choose Π, such that ΠX can compress the original matrix X, and one can perform the same optimization on the smaller matrix ΠX

1.2 Related Works

Byzantine Robustness in Distributed Learning

Byzantine robustness is often guaranteed by using gradient aggregation rules. A classical method for improving fault tolerance are based on robust statistics, for example, [53] uses geometric median as the aggregation rule, and [59] establishes statistical error rates for marginal trimmed mean as the aggregation rule. Another line of work does not use robust statistics. For example, Krum [9] was proposed as a method of selecting candidates with minimal local sum of l_p distances, and DRACO [19], DETOX uses coding theory to ensure Byzantine robustness. However, all these works suffer from communication complexity issues

arising from sending the full model vector between the parameter and worker servers. In this paper, our proposed algorithm works under the same Byzantine settings, and we use Krum aggregation rule [9] to ensure Byzantine robustness.

A number of works also improve communication efficiency. A dominant line of work in reducing communication cost in synchronous distributed stochastic gradient descent either quantize or sparsify gradients. For example, SignSGD [8] achieve a reduction in communication cost per iteration by quantizing the gradients to their mantissa, and QSGD [3] achieves a reduction in communication cost at the expense of a increase in the number of iterations (with no asymptotic improvement). Another line of work is in sketching the gradients to lower dimensions. In particular, Ivkin, et al. [28] introduces Sketched-SGD that sends $\mathcal{O}(\log d)$-length gradients where d is the number of model parameters. While [8, 3] are great improvements, they only reduce communication by a constant factor, where as Sketched-SGD reduces communication complexity to a log factor.

However, to our knowledge, no work has focused on building algorithms to tackle both the issue of Byzantine robustness and communication complexity. In our work, we take inspiration from both literature and use them as black-boxes to build SketchedRobustAgg.

Secure Aggregation in Federated Learning

Bonawitz et al. [11] presented the first secure aggregation protocol SecAgg for FL, wherein clients use a key exchange protocol to agree on pairwise additive masks to achieve privacy. SecAgg, in the honest-but-curious setting, can achieve $T = \alpha N$ for any $\alpha \in (0, 1)$ and $D = N - T - 1$, and provides worst-case dropout resilience against any D users dropping out. However, SecAgg incurs significant computation cost of $\mathcal{O}\left(LN^2\right)$ at the server. This limits its scalability to several hundred clients as observed in [12].

Truex et al. [55] uses threshold homomorphic encryption and Xu et al. [58] uses functional encryption to perform secure aggregation. However, these schemes assume a trusted third party for key distribution, which typically does not hold in the FL setup.

The scheme by So et al. [50], which we call TurboAgg, uses additive secret sharing for security combined with erasure codes for dropout tolerance. TurboAgg allows $T = D = \alpha N$ for any $\alpha \in (0, 1/2)$. However, it has two main drawbacks. First, it divides N clients into $N/\log N$ groups, and each client in a group needs to communicate to every client in the next group. This results in per client communication cost of $\mathcal{O}\left(L \log N\right)$. Moreover, processing in groups requires at least $\log N$ rounds. Second, it can tolerate only non-adaptive adversaries, i.e., client corruptions happen before the clients are partitioned into groups. On the other hand, FastSecAgg results in $\mathcal{O}\left(L\right)$ communication per client, runs in 3 rounds, and is robust against an adaptive adversary which can corrupt clients during the protocol execution. On the other hand, [11] provide strong privacy guarantees, but incur significant computation

costs. In particular, [11] provides the first secure aggregation protocol for federated learning. It is a four round interactive protocol, which ensures the privacy by using *pairwise random masks* for every pair of clients. However, in the final round while recovering the sum, the server needs to recover and remove the random mask corresponding to each dropped out which blows up the computation cost at server.

Recently, Bell et al. [5] proposed an improved version of SecAgg, which we call SecAgg+. Their key idea is to replace the complete communication graph of SecAgg by a sparse random graph to reduce the computation and communication costs. FastSecAgg achieves smaller computation cost than SecAgg+ at the server and the same (orderwise) communication cost per client as SecAgg+ when $L = \Omega(N)$. Moreover, FastSecAgg is robust against adaptive adversaries, whereas SecAgg+ can mitigate only non-adaptive adversaries where client corruptions happen before the protocol execution starts. On the other hand, SecAgg+ achieves smaller communication cost in absolute numbers than FastSecAgg.

There are several recent works [40, 27, 49] which consider secure aggregation when a fraction of clients are Byzantine. Our focus, on the other hand, is on honest-but-curious setting, and we leave the case of Byzantine clients as a future work.

Sketching

Sketches $\Pi_f(X)$ of some dataset X with respect to some function f is a compression of X that allow us to compute (or approximately compute) $f(X)$ given only $\Pi(X)$. In our work, we compress our gradient vector denoted g into a sketch $\Pi(g) \in \mathbb{R}^s$ using Count-Sketch [15], Subsampled Randomized Hadamard Transform and random uniform projections.

We focus our attention specifically on using popular sketching methods to project our gradients to a lower dimension. This is a novel approach in that sketching is typically used to project the data matrix to a lower dimension, but we instead use it on the gradients themselves, and then recover them at the server/parameter node. We use the well-known Count-Sketch, Subsampling and Subsampled Randomized Hadamard Transform [56].

Count-Sketch In sketching, our primary interest is in finding the large coordinates (or "heavy hitter") of a gradient vector $g \in \mathbb{R}^d$. Let us suppose that we want to sketch a d-dimensional vector $\mathbf{x}$ to a s-dimensional vector $\mathbf{x}'$. Then, Count-Sketch is specified a 2-wise independent hash function $h : [d] \to [s]$ and 2-wise independent sign function $g : [d] \to \{-1, +1\}$. When we apply it to $\mathbf{x}$, the value at coordinate i of the output, such that $i = 1, \ldots, d'$ is $\sum_{j,h(j)=i} g(j)\mathbf{x}_j$. Then, Count-Sketch matrix can be represented as a $s \times d$ matrix where j-th column contains a single non-zero entry $g(j)$ in the $h(j)$-th row. Importantly, Count-Sketch provides a guarantee that the point query returns an estimate equal to $x_i \pm \|x\|_2 / \sqrt{k}$, and it has been widely adopted in distributed systems due to linearity

of COUNT-SKETCH (i.e., let $\Pi(g_1)$ and $\Pi(g_2)$ be two sketched gradients, then we can merge them as $\Pi(g) = \Pi(g_1 + g_2) = \Pi(g_1) + \Pi(g_2)$).

Random Uniform Projections Subsampling is a subspace embedding where one uniformly randomly samples from a data matrix X.

Subsampled Randomized Hadamard Transform (SRHT) SRHT [56] is a fast Johnson-Lindenstrauss transform, and is a subspace embedding $\Pi \in \mathbb{R}^{m \times n}$ such that,

$$\forall x \in E, (1-\epsilon)\|x\|_2^2 \leq \|\Pi x\|_2^2 \leq (1+\epsilon)\|x\|_2^2$$

and

$$\Pi = \frac{1}{\sqrt{m}} SHD$$

where S uniformly samples rows of HD, H is a Hadamard matrix defined recursively,

$$H_n = \begin{bmatrix} H_{n/2} & H_{n/2} \\ H_{n/2} & -H_{n/2} \end{bmatrix}$$

and D is a diagonal with uniformly i.i.d. ± 1 on the diagonal. We note that HD is an orthogonal matrix that essentially rotates $\mathbf{v}$, and denote it as $R = HD$. SRHT is often viewed as a standard reference point for comparing sketching algorithms. Moreover, for many applications, random projections with i.i.d. entries perform worse compared to orthogonal projection. More recently, this observation has also found some theoretical support in limited contexts. Other works also showed the guaranteed improved performance in accuracy and/or speed. Consequently, along with computational considerations, these results favor the SRHT over Gaussian projections.

1.3 Contributions

In this work, we largely consider Sketching to enable Fast and Robust Aggregation schemes in distributed and federated learning setups. Our contributions are twofold.

1. **Distributed Learning**: We propose SKETCHEDROBUSTAGG in this section of the work. We use ideas from sketching and byzantine-worker literature to address both the issue of byzantine workers and communication efficiency, by using sketching methods to decrease the amount of communication required between the server and worker nodes. We show that our algorithm achieves similar runtime (measured by number of iterations) as without using sketching, even though the algorithm sends s-dimensional (where $s \ll d$) vectors between worker and parameter server.

2. **Federated Learning**: We propose a secure aggregation protocol, FastSecAgg, that is efficient in terms of computation and communication, and robust to client dropouts. The main building block of FastSecAgg is a novel multi-secret sharing scheme, FastShare, based on the Fast Fourier Transform (FFT), which may be of independent interest. Like SketchedRobustAgg, we add ideas from Sketching to reduce communication and computation costs at the server. This part of the thesis pertaining to FastShare and cryptographic analysis of FastSecAgg is largely a reorganization of [31], and additional discussion around Sketching and additional results and Federated Learning optimizations are new contributions.

Chapter 2

Fast Robust Aggregation for Distributed Learning

2.1 Preliminaries

Problem Setup

In this work, we let $\|\cdot\|$ represent the l_2 norm of a vector and the operator norm of a matrix, unless otherwise specified. We denote input data vectors using bolded $\mathbf{x}$, label vectors as $\mathbf{y}$ and vector of all weights and model parameters as $\mathbf{w}$. We also include a table of all the common notations in Table 2.1.

Common Notation	Meaning
$\mathbf{x}$	Training data
$\mathbf{y}$	Training labels
$\mathbf{w}$	Model parameters
n	Number of training data samples
d	Dimension of the original model parameters
s	Sketched dimension of the model parameters
m	Dimension of one training point
P	Number of servers
k	Number of Byzantine servers

Table 2.1: Common Notations Used

Distributed Learning

Distributed Training The process of training a neural network from data X is often cast as empirical risk minimization (ERM) problem, where our objective is:

$$\min_{\mathbf{w}} \frac{1}{n} \sum_{i=1}^{n} \ell(\mathbf{w}; \mathbf{x}_i)$$

where $\mathbf{x}_i \in \mathbb{R}^m$ represents the ith data point, n is the number of data points, $w \in \mathbb{R}^d$ represents the model parameters and $\ell(\cdot;\cdot)$ is the loss function.

One way to approximately solve ERM optimization problem is by using stochastic gradient descent, where we initialize the model at $\mathbf{w}_0$ and update according to:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \alpha \nabla \ell(\mathbf{w}_t; \mathbf{x}_{i_k})$$

where $\alpha > 0$ is the learning rate, and $i_k \in [n]$ is a random data-point index. To take advantage of the additional computational power, we often use mini-batch gradient descent, where at each iteration, we select some subset $S_k \in \mathbb{R}^{s \times m}$ of size s of the data, and update our model according to:

$$\mathbf{w}_{t+1} \leftarrow \mathbf{w}_t - \frac{\alpha}{m} \sum_{i \in S_k} \nabla \ell(\mathbf{w}_t; \mathbf{x}_i)$$

In the distributed setting, every worker gets access to a unique mini-batch, updates the model accordingly and send the model parameters to the parameter servers. At the parameter server, the server often does model averaging, where it takes a global step with:

$$\mathbf{w}_{\text{Global}} \leftarrow \frac{1}{P} \sum_{i=1}^{P} \mathbf{w}_i$$

where $\mathbf{w}_i$ represents the model parameters coming from the ith worker and P represents the total number of workers.

Byzantine Worker Node We consider the setting where some subset of size k of P worker nodes are byzantine and act adversarially against the training process. We consider a worker node to be Byzantine if it does not return the correct gradient updates given its allocated samples. In fact, it has been shown that mini-batch SGD can fail to converge even with the presence of a single adversarial node.

More specifically, suppose that in iteration i, we sample correct vectors $\{v_j^i : j \in [P]\}$ are i.i.d. samples drawn from random variable $G = \nabla f(x, \varepsilon)$ where $\mathbb{E}[G] = g$ is an unbiased estimator of the gradient. Then, with Byzantine workers, the vectors received by the parameter server are as follows,

$$\tilde{v}_j^i = \begin{cases} v_j^i \text{ if the } i\text{th worker is not Byzantine} \\ \text{arbitrary if the } i\text{th worker is Byzantine} \end{cases} \tag{2.1}$$

We note further that the indices of the Byzantine workers can change (though for simplicity of proofs, we often omit this assumption), and further, that serve nodes do not know which workers are Byzantine. Moreover, we assume that these Byzantine workers have full knowledge of the system, including the aggregation rule and gradients proposed by the workers.

Krum KRUM is a popular robust aggregation method using the l_p norm between generated gradients. At any time step t, updates $(g_1^t, \ldots, g_n^t)$ are received at the server. For each g_i^t, the $n-k-2$ closest updates are chosen to form a set C_i, and their distances are added up to give $S(g_i^t) = \sum_{g \in C_i} \|g_i^t - g\|$, and Krum chooses the g_{Krum} with lowest score to add to $g_{\text{Global}}^{t+1} = g_{\text{Global}}^t + g_{\text{Krum}}$.

2.2 Fast Robust Aggregation

In traditional distributed learning paradigms, the main parameter server (MAIN) collects gradients g_i from every worker i then aggregates them using their average. In other words,

$$g_{\text{MAIN}} = \frac{1}{n} \sum_{i=1}^{n} g_i$$

However, this aggregation rule is not Byzantine-tolerant, and one can show that a single Byzantine worker can make this rule always select an arbitrary vector U by proposing $g_n = \frac{1}{n} \cdot U - \sum_{i=1}^{n-1} g_i$.

Below, we propose a method using robust aggregation scheme KRUM and sketching methods SKETCH.

Algorithm 1 One Update Step

$\{b_i\}_{i=1}^n \leftarrow$ Random Batches of Data
Initialize BUFFER

for $i \leftarrow 0$ n **do** $g_i \leftarrow$ ONE-MACHINE-TRAIN(b_i)
BUFFER $\leftarrow$ BUFFER + SKETCH(g_i)
$k \leftarrow$ KRUM(BUFFER)
$g_{\text{MAIN}} \leftarrow$ UNSKETCH(k)

For simplicity of proof, we consider using SRHT as our primary sketching and unsketching algorithms in analysis of time complexity below. We assume, also, that $s = O(\log d)$, such that $s \ll d$. We note that similar analyses can be carried out for other sketches.

Lemma 2.2.1. *The time complexity of* KRUM$(g_1, \ldots, g_n)$*, where* $g_1, \ldots, g_n \in \mathbb{R}^d$ *is* $\mathcal{O}(n^2(d + \log n))$

Proof. We only provide a sketch of the proof, and defer a more detailed discussion to [9]. For each g_i, the parameter server computes the n squared distances $\|g_i - g_j\|^2$ in $\mathcal{O}(nd)$ time. Then the parameter server sorts these distances in $\mathcal{O}(n \log n)$ time and sums the first $n-k-1$ values in $\mathcal{O}(nd)$ time. Last, we take $\mathcal{O}(n)$ time to find the minimum score. Thus, computing the score of all the g_i's takes $\mathcal{O}(n^2(d + \log n))$ time. □

Lemma 2.2.2. *Time complexity for sketching and unsketching operations with SRHT takes* $\mathcal{O}(nd \log n)$ *time*

Corollary 2.2.2.1. *Time complexity of the algorithm at the server is* $\mathcal{O}(n \log n(n+d))$ *using SRHT sketch.*

Proof. We assume for simplicity that $s = \mathcal{O}(\log n)$. We use Lemmas 2.2.1 and 2.2.2 and concatenate them, since they occur in separate steps. □

Notice that our runtime complexity at the server using Sketched-Robust-Aggregation also outperforms runtime complexity using Krum as runtime complexity of Krum dominates the construction and sketching operations with SRHT. We remark further that time complexity can be further improved using Sketched-SGD, as opposed to unsketching and running normal SGD.

That said, our biggest gain is in communication complexity, where we send only $\mathcal{O}(s)$-length gradient vectors as opposed to $\mathcal{O}(d)$-length gradient vectors. In fact, we show, experimentally in section **??**, that we can handle even a 50-times reduction without compromising heavily on the number of iterations to convergence using SRHT and Count-Sketch.

2.3 Byzantine Robustness

In this section, we show that our algorithm above is indeed Byzantine-Robust, by using proof techniques analogous to [9]. Intuitively, our aggregation rule should output a vector g_{SRA} that is not far from the real gradient g.

Definition 2.3.1 ((α, f)-Byzantine Resilience)**.** *Let α be any angular value in $[0, \pi/2)$ and f be an integer in $[0, n]$. Likewise, let $V_1, \ldots, V_n$ be some independent and identically distributed random vectors in $\mathbb{R}^d$ such that $V_i \sim G$ and $\mathbb{E}[G] = g$. Furthermore, let $B_1, \ldots, B_f$ be any random vectors in $\mathbb{R}^d$, possibly dependent on the V_i's. A function F is said to be (α, f)-Byzantine resilient if, for any $1 \leq j_1 < \cdots < j_f \leq n$, the vector*

$$F = F(V_1, \ldots, \underbrace{B_1}_{j_1}, \ldots, \underbrace{B_f}_{j_f}, \ldots, V_n)$$

satisfies (i) $\langle \mathbb{E}F, g \rangle \geq (1 - \sin\alpha) \cdot \|g\|^2 > 0$ *and* (ii) *for $r = 2, 3, 4$, $\mathbb{E}\|F\|^r$ is bounded above by a linear combination of terms $\mathbb{E}\|G\|^{r_1} \ldots \mathbb{E}\|G\|^{r_{n-1}}$ with $r_1 + \cdots + r_{n-1} = r$.*

Definition 2.3.2 (Embedding)**.** *An embedding for a set $S \in \mathbb{R}^n$ with distortion ϵ is an $m \times n$ matrix Π such that*

$$\forall x \in S, (1-\epsilon)\|x\|^2 \leq \|\Pi x\|^2 \leq (1+\epsilon)\|x\|^2$$

Definition 2.3.3 (Subspace Embedding)**.** *A subspace embedding is an embedding for a set S where S is a d-dimensional linear subspace.*

Theorem 2.3.1. *Let $g_1, \ldots, g_n$ be any independent and identically distributed random d-dimensional vectors s.t $g_i \sim G$, with $\mathbb{E}G = g$ and $\mathbb{E}\|G-g\|^2 = d\sigma^2$. If $2k+2 < n$, and*

$$\eta(n,k) \underset{def}{=} \sqrt{2\left(n-k+\frac{k\cdot(n-k-2)+k^2\cdot(n-k-1)}{n-2k-2}\right)} = \begin{cases} O(n) & \text{if } k = O(n) \\ O(\sqrt{n}) & \text{if } k = O(1) \end{cases}$$

then Sketched-Robust-Aggregation *is (α, f)-Byzantine resilient where $0 \leq \alpha < \pi/2$ is defined by*

$$\sin\alpha = \frac{\eta(n,f)\cdot\sqrt{d}\cdot\sigma}{\|g\|}.$$

The condition on the norm of the gradient, $\eta(n,f)\cdot\sqrt{d}\cdot\sigma < \|g\|$, can be satisfied, to a certain extent, by having the (correct) workers computing their gradient estimates on mini-batches [13]. Indeed, averaging the gradient estimates over a mini-batch divides the deviation σ by the squared root of the size of the mini-batch.

Proof. We largely notice that this theorem is the same as that in [9], where Byzantine robustness of Krum function is proven. Because of our algorithm's dependence on Krum, we only have to prove that executing Krum on sketched gradients will result in Byzantine robustness. Importantly, we notice that proof of Krum, depends on the condition that $\mathbb{E}G = g$ and $\mathbb{E}\|G-g\|^2 = d\sigma^2$. For all the sketches we have chosen, we know that $\mathbb{E}G = g$ and Count-Sketch and SRHT are subspace embeddings of G and satisfy the second constraint. While random uniform projections do not satisfy the second constraint, in practice, we notice a good rate of convergence despite adversarial workers. □

2.4 Experiments for SketchedRobustAgg

Experiment Setup

Bit-flipping Attacks We show Byzantine-robustness by following recent literature like [57] and consider bit-flipping and label-flipping attacks. The bits that control the sign of the floating numbers are flipped, e.g., due to some hardware failure. A faulty worker pushes the negative gradient instead of the true gradient to the servers. To make the failure even worse, one of the faulty gradients is copied to and overwrites the other faulty gradients, which means that all the faulty gradients have the same value.

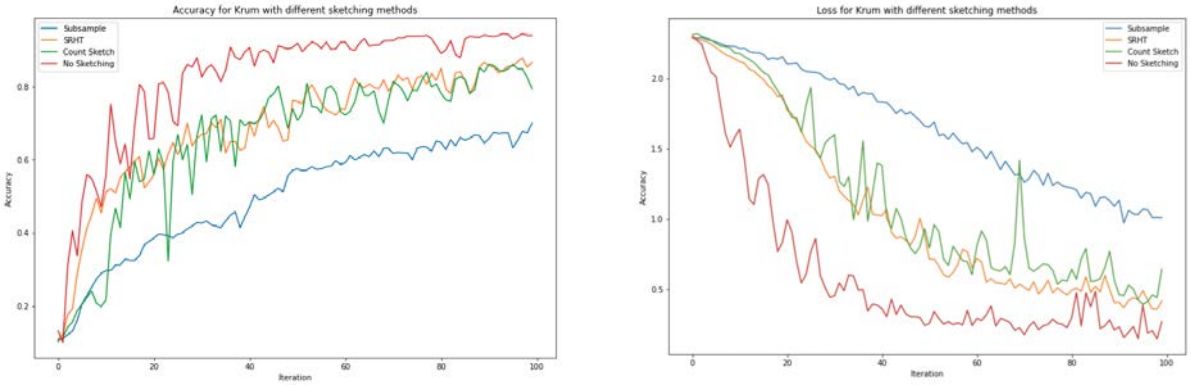
Label-flipping Attacks When such failures happen, the workers compute the gradients based on the training data with "flipped" labels, i.e., any label $\in \{0, \ldots, 9\}$ is replaced by 9 label. Such failures/attacks can be caused by data poisoning or software failures.

Datasets Used We conduct experiments on benchmark MNIST and CIFAR-10 image classification dataset. MNIST and CIFAR-10 consists of 50,000 images for training and 10,000 images for testing. We use the cross-entropy loss function on the training set and accuracy on the testing set as our main evaluation metrics.

Neural Network Used We use convolutional neural networks (CNN) architectures. Specifically, we use a modified LeNet [34] (431,080 parameters) for the MNIST benchmark and Resnet-18 for CIFAR-10 benchmark. In each experiment, we launch 10 worker processes with $\{0, 1, 2\}$ Byzantine nodes, and repeat each experiment 10 times and take the average. In all the experiments, we take the learning rate $\gamma = 0.1$ and a batch size of 128.

Convergence Rates

Since our goal is to ensure that our algorithm converges at a similar rate as without using sketching, we compare our methods against the oracle, where we do not use sketching (converges to normal Krum). We first demonstrate the results and convergence rates for various sketches and using no sketching. We see that using no sketching produces the best results, as expected. However, SRHT and Count-Sketch also produces very decent results, even though it does not converge as quickly. In the graphs below, our sketch size $d' = 10000$ (for Count-Sketch, we use a table of 20×500).



(a) Iteration and Accuracy for No Attacks

(b) Iteration and Loss for No Attacks

Figure 2.1: Test Accuracy and Loss Against Number of Iterations for Using No Attacks

We can expand this further until we see a convergence in the accuracy, and run longer iterations. When we consider no workers, we find the following number of iterations until convergence.

Sketching Method	Number of Iterations to Convergence	Accuracy After 100 Iterations
No Sketching	81	0.94
SRHT	113	0.867
Count-Sketch	118	0.825
Subsampling	176	0.700

Table 2.2: Sketching Methods and Number of Iterations to Convergence using LeNet on MNIST Dataset

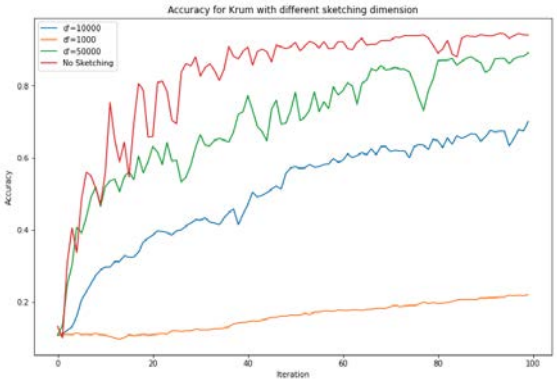
We also consider the same setup using Resnet-18 on CIFAR-10 dataset. When our sketch size is 50% of the original number of parameters, we find that to achieve a loss of 2.25, we have the setup as in Table 2.3.

Sketching Method	Number of Iterations to Convergence
No Sketching	15
SRHT	74
Count-Sketch	62
Subsampling	234

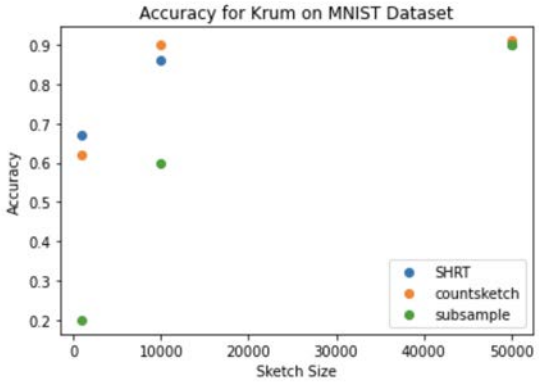
Table 2.3: Sketching Methods and Number of Iterations to Convergence using Resnet-18 on CIFAR-10 Dataset

Changing the Size of the Sketch

We consider changing the size of the sketch to see how the size of the sketch affects the convergence rates under different mechanisms. Specifically, we consider using $d' = \{1000, 10000, 50000\}$. As expected, increasing the size of the sketches allow for quicker convergence, but beyond 10000, the number of iterations to convergence changes minimally, and the savings in cost of running Krum on a lower-dimensional vector will likely compensate. We include No Sketch as a reference point.



(a) Iteration and Accuracy Under Different Sketching Dimensions



(b) Summary of Iteration and Accuracy Under Different Sketching Dimensions

Figure 2.2: Effect of Changing Sketching Dimensions on Test Accuracy using SketchedRobustAgg

Chapter 3

Fast Secure Aggregation for Federated Learning

3.1 Preliminaries

Problem Setup

In Federated Learning, we consider the setup in which we have a fixed set of M clients, each with their own local dataset. Importantly, these local datasets come from non-i.i.d. distributions, and the i-th client's data is sampled from a distribution $\mathcal{D}_i$. These workers then coordinate with the server to jointly train a model.

Then, the federated learning problem can be formalized as minimizing a sum of stochastic functions defined as

$$\arg\min_{\mathbf{w}\in\mathbb{R}^d} \left\{ \ell(\mathbf{w}) = \frac{1}{M}\sum_{i=1}^{M} \ell_i(\mathbf{w}) \right\}, \tag{3.1}$$

where $\ell_i(\mathbf{w}) = \mathbb{E}_{\zeta\sim\mathcal{D}_i}\left[\ell_i(\mathbf{w};\zeta)\right]$ is the expected loss of the prediction on the i-th client's data made with model parameters $\mathbf{w}$.

We assume that each client can compute $g_i(\mathbf{w}) = \nabla f_i(\mathbf{w};\zeta)$, which is an unbiased stochastic gradient of f_i with variance bounded by σ^2.

Federated Averaging

Federated Averaging (FedAvg) is a synchronous update scheme that proceeds in rounds of communication [37]. At the beginning of each round (also referred to as the iteration), the server selects a subset $\mathcal{C}$ of N clients (for some $N \leq M$).

Each of these clients $i \in \mathcal{C}$ copies the current model parameters $\mathbf{w}_{i,0}^t = \mathbf{w}^t$, and performs T steps of (mini-batch) stochastic gradient descent steps to obtain its local model $\mathbf{w}_{i,T}^t$. Indeed, each local step k is,

$$\mathbf{w}_{i,k}^t \leftarrow \mathbf{w}_{i,k-1}^t - \eta_l g_{i,k-1}^t$$

where $g_{i,k-1}^t$ is an unbiased stochastic gradient of ℓ_i at $\mathbf{w}_{i,k-1}^t$, and η_l is the local step-size. In practice, clients can make multiple training passes (called epochs) over its local dataset with a given step size. Further, typically client datasets are of different size, and the server takes a weighted average with weight of a client proportional to the size of the dataset [37]. Then, each client $i \in \mathcal{C}$ sends their update as

$$\Delta \mathbf{w}_i^t = \mathbf{w}_{i,L}^t - \mathbf{w}^t$$

At the server, the clients' updates $\Delta \mathbf{w}_i^t$ are aggregated to form the new server model as

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \frac{1}{|\mathcal{C}|} \sum_{i \in \mathcal{C}} \Delta \mathbf{w}_i^t$$

Our focus is on one iteration of FedAvg and we omit the explicit dependence on the iteration t hereafter. We assume that each client potentially compresses and suitably quantizes its model update $\Delta \mathbf{w}_i^t \in \mathbb{R}^d$ to obtain $\mathbf{u}_i \in \mathbb{Z}_R^L$, where $L \leq d$. In our work, our goal is to design a protocol that enables the server to securely compute $\sum_{i \in \mathcal{C}} \mathbf{u}_i$.

Threat Model

Federated learning can be considered as a multi-party computation consisting of N clients, each having their own private dataset, and an aggregator (or the server), with the goal of learning a model using all of the local private datasets. However, there are numerous threat models that we must consider. In the larger picture, we aim to ensure the privacy of model updates during the training process under the following threat models. Our objective, then, is to design a protocol to securely aggregate clients' model updates such that the joint view of the server and *any* set of up to T clients must not leak any information about the other clients' model updates, besides what can be inferred from the output of the summation. In addition, even if a *random* set of up to D clients drop out during an iteration, the server with high probability should be able to compute the sum of model updates (of the surviving clients) while maintaining the privacy.

Honest-but-curious Model The parties honestly follow the protocol, but attempt to learn about the model updates from other parties by using the messages exchanged during the execution of the protocol. The honest-but-curious (semi-honest) adversarial model is commonly used in the field of secure MPC, including prior works on secure federated learning [11, 55, 50].

Colluding parties In every iteration of federated averaging, the server first samples a set $\mathcal{C}$ of clients. The server may collude with any set of up to T clients from $\mathcal{C}$. We let T be the *privacy threshold*. The server can view the internal state and all the messages received/sent by the clients with whom it colludes. We refer to the internal state along with the messages received/sent by colluding parties (including the server) as their *joint view*. (see Sec. 3.4 for details)

Dropouts A random subset of up to D clients may drop out at any point of time during the execution of secure aggregation. We let D be the *dropout tolerance*.

Cryptographic Primitives

Key Agreement

A key agreement protocol consists of three algorithms (KA.PARAM, KA.GEN, KA.AGREE). Given a security parameter λ, the parameter generation algorithm $pp \leftarrow \text{KA.PARAM}(\lambda)$ generates some public parameters, over which the protocol will be parameterized. The key generation algorithm allows a client i to generate a private-public key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \text{KA.GEN}(pp)$. The key agreement procedure allows clients i and j to obtain a private shared key $k_{i,j} \leftarrow \text{KA.AGREE}(\mathsf{sk}_i, \mathsf{pk}_j)$. Correctness requires that, for any key pairs generated by clients i and j (using KA.GEN with the same parameters pp), $\text{KA.AGREE}(\mathsf{sk}_i, \mathsf{pk}_j) = \text{KA.AGREE}(\mathsf{sk}_j, \mathsf{pk}_i)$. Security requires that there exists a simulator Sim_{KA}, which takes as input an output key sampled uniformly at random and the public key of the other client, and simulates the messages of the key agreement execution such that the simulated messages are computationally indistinguishable from the protocol transcript.

Authenticated Encryption

An authenticated encryption allows two parties to communicate with data confidentially and data integrity. It consists of an encryption algorithm AE.ENC that takes as input a key and a message and outputs a ciphertext, and a decryption algorithm AE.DEC that takes as input a ciphertext and a key and outputs the original plaintext, or a special error symbol $\perp$. For correctness, we require that for all keys $k_i \in \{0,1\}^\lambda$ and all messages m, $\text{AE.DEC}\left((k_i, \text{AE.ENC}(k_i, m)\right) = m$. For security, we require semantic security under a chosen plaintext attack (IND-CPA) and ciphertext integrity (IND-CTXT) [6].

3.2 FastShare: FFT Based Secret Sharing

We present a novel, computationally efficient multi-secret sharing scheme FastShare which forms the core of our proposed secure aggregation protocol FastSecAgg.

A multi-secret sharing scheme splits a set of secrets into shares (with one share per client) such that coalitions of clients up to certain size have no information on the secrets, and a random[1] set of clients of large enough size can jointly reconstruct the secrets from their shares. In particular, we consider information-theoretic (perfect) security. A secret sharing scheme is said to be linear if any linear combination of valid share-vectors result in a valid share-vector of the linear combination applied to the respective secret-vectors. In particular, the scheme consists of two algorithms, one to generate shares and the other to recover the secrets, described further in Definition 3.2.1.

Definition 3.2.1 (Multi-secret Sharing)**.** *Let $\mathbb{F}_q$ be a finite field, and let S, T, D and N be positive integers such that $S + T + D < N \leq q$. A linear multi-secret sharing scheme over $\mathbb{F}_q$ consists of two algorithms* Share *and* Recon*. The sharing algorithm $\{(i, [\mathbf{s}]_i)\}_{i\in\mathcal{C}} \leftarrow \textsc{Share}(\mathbf{s}, \mathcal{C})$ is a probabilistic algorithm that takes as input a set of secrets $\mathbf{s} \in \mathbb{F}_q^S$ and a set $\mathcal{C}$ of N clients (client-IDs), and produces a set of N shares, each in $\mathbb{F}_q$, where share $[\mathbf{s}]_i$ is assigned to client i in $\mathcal{C}$. For a set $\mathcal{D} \subseteq \mathcal{C}$, the reconstruction algorithm $\{\mathbf{s}, \perp\} \leftarrow \textsc{Recon}\left(\{(i, [\mathbf{s}]_i)\}_{i\in\mathcal{C}\setminus\mathcal{D}}\right)$ takes as input the shares corresponding to $\mathcal{C} \setminus \mathcal{D}$, and outputs either a set of S field elements $\mathbf{s}$ or a special symbol $\perp$. The scheme should satisfy the following requirements.*

1. *T-Privacy: For all $\mathbf{s}, \mathbf{s}' \in \mathbb{F}_q^S$ and every $\mathcal{P} \subset \mathcal{C}$ of size at most T, the shares of $\mathbf{s}$ and $\mathbf{s}'$ restricted to $\mathcal{P}$ are identically distributed.*
2. *D-Dropout-Resilience: For every $\mathbf{s} \in \mathbb{F}_q^S$ and any random set $\mathcal{D} \subset \mathcal{C}$ of size at most D, $\textsc{Recon}\left(\{(i, [\mathbf{s}]_i)\}_{i\in\mathcal{C}\setminus\mathcal{D}}\right) = \mathbf{s}$ with probability at least $1 - poly(N)$.*
3. *Linearity: If $\{(i, [\mathbf{s}_1]_i)\}_{i\in\mathcal{C}}$ and $\{(i, [\mathbf{s}_2]_i)\}_{i\in\mathcal{C}}$ are sharings of $\mathbf{s}_1$ and $\mathbf{s}_2$, then $\{(i, a[\mathbf{s}_1]_i + b[\mathbf{s}_2]_i)\}_{i\in\mathcal{C}}$ is a sharing of $a\mathbf{s}_1 + b\mathbf{s}_2$ for any $a, b \in \mathbb{F}_q$.*

Next, we describe FastShare which can achieve a trade-off between S, T, and D for a given N. We begin with setting up the necessary notation. FastShare leverages the finite-field Fourier transform and Chinese Remainder Theorem. Towards this end, let $\{n_0, n_1\}$ be co-prime positive integers of the same order, such that $n_0 n_1$ divides $(q-1)$. Without loss of generality, assume that $n_0 < n_1$. As a running example, we use $n_0 = 10$, $n_1 = 13$, and $q = 131$. We consider N of the form $N = n_0 n_1$, and choose the field size q as a power of a prime such that N divides $q - 1$. By the co-primeness of n_0 and n_1, applying the Chinese Remainder Theorem, any number $j \in \{0, \cdots, n-1\}$ can be uniquely represented in a 2-dimensional (2D) grid as a tuple (a, b) where $a = j \mod n_0$ and $b = j \mod n_1$.

[1]Secret sharing [4] and multi-secret sharing [22, 10] schemes conventionally consider *worst-case* dropouts. We consider *random* dropouts to exploit the FL setup.

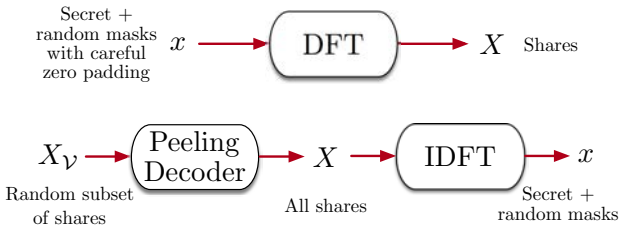


Figure 3.1: FastShare to generate shares, and FastRecon to recover the secrets from a subset of shares.

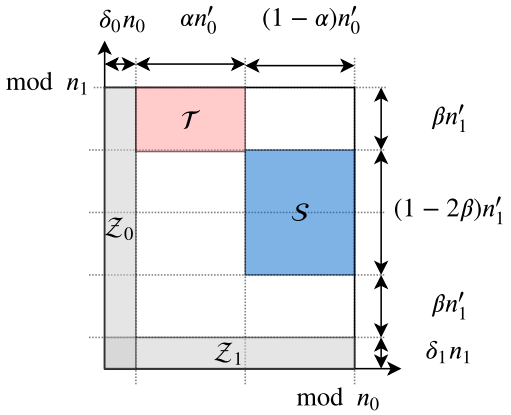


Figure 3.2: Indices assigned to a grid using the Chinese remainder theorem, and sets $\mathcal{Z}_0$, $\mathcal{Z}_1$, $\mathcal{S}$, and $\mathcal{T}$. Here, we define $n_i' = (1-\delta_i)n_i$ for $i \in \{0,1\}$. Set $\mathcal{S}$ is used for secrets, and sets $\mathcal{Z}_0$ and $\mathcal{Z}_1$ are used for zeros, and the remaining locations are used for random masks. Set $\mathcal{T}$ is used in our privacy proof.

FastShare to Generate Shares

The sharing algorithm $\{(i, [\mathbf{s}]_i)\}_{i\in\mathcal{C}} \leftarrow \textsc{FastShare}(\mathbf{s}, \mathcal{C})$ is a probabilistic algorithm that takes as input a set of secrets $\mathbf{s} \in \mathbb{F}_q^S$, and a set $\mathcal{C}$ of N clients (in the form of client ID), and produces a set of N shares, each in $\mathbb{F}_q$, where share $[\mathbf{s}]_i$ is assigned to client i.

At a high level, FastShare consists of two stages. First, it constructs a length-N signal consisting of the secrets and random masks with zeros placed at carefully chosen locations. Second, it takes the fast Fourier transform of the signal to generate the shares (see Fig. 3.1). The shares can be considered as the spectrum of the signal constructed in the first stage.

In particular, the signal is constructed as follows. Given fractions $\delta_0, \delta_1, \alpha \in (0,1)$ and $\beta \in (0, 1/2)$, we define the sets of tuples $\mathcal{Z}_0$, $\mathcal{Z}_1$, $\mathcal{S}$, and $\mathcal{T}$ using the grid representation as depicted in Fig. 3.2. We round all real numbers to the largest integer no larger than the

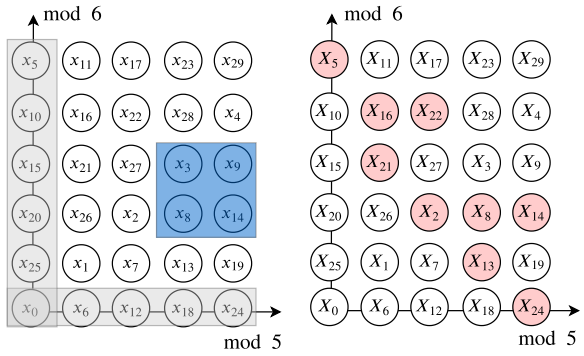


Figure 3.3: Example with $\ell = 4$ secrets, and co-prime integers $n_0 = 5$, $n_1 = 6$, yielding $N = n_0 n_1 = 30$. Zeros are placed at gray locations, and the secrets at blue locations. Careful zero padding induces parity checks on the shares in each row and column due to *aliasing*, i.e., $[X_0 + X_5 + \cdots + X_{25}; X_1 + \cdots + X_{26}; \cdots; X_4 + \cdots + X_{29}] = [0, \cdots, 0]$ and $[X_0 + X_6 + \cdots + X_{24}; \cdots; X_5 + \cdots + X_{29}] = [0, \cdots, 0]$. Any one missing share in a row or column can be recovered using the parity-check structure. E.g., dropped out red shares can be recovered by *iteratively decoding* the missing shares in rows and columns.

number itself. In other words, for some real number $x \in \mathbb{R}$, we round x to $\lfloor x \rfloor$, and omit the floor sign for the sake of brevity.

$$\mathcal{Z}_0 = \{(a,b) : 0 \le a \le \delta_0 n_0 - 1\}, \tag{3.2}$$

$$\mathcal{Z}_1 = \{(a,b) : 0 \le b \le \delta_1 n_1 - 1\}, \tag{3.3}$$

$$\begin{aligned}\mathcal{S} = \{(a,b) : {} & \delta_0 n_0 + \ \alpha(1-\delta_0)n_0 \le a \le n_0 - 1, \\ & \delta_1 n_1 + \beta(1-\delta_1)n_1 \le b \le \delta_1 n_1 + (1-\beta)(1-\delta_1)n_1\},\end{aligned} \tag{3.4}$$

$$\begin{aligned}\mathcal{T} = \{(a,b) : {} & \delta_0 n_0 \le a \le \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1, \\ & \delta_1 n_1 + (1-\beta)(1-\delta_1)n_1 \le b \le n_1 - 1\}.\end{aligned} \tag{3.5}$$

We place zeros at indices in $\mathcal{Z}_0$ and $\mathcal{Z}_1$, and secrets at indices $\mathcal{S}$. Each of the remaining indices is assigned a uniform random mask from $\mathbb{F}_q$ (independent of other masks and the secrets). We use $\mathcal{T}$ in our privacy proof. Let $\mathbf{x}$ denote the resulting length-N vector, which we refer to as the signal. (See Fig. 3.3 for a toy example.)

Let ω be a primitive N-th root of unity in $\mathbb{F}_q$. FastShare computes the finite field Fourier transform $\mathbf{X}$ of the signal $\mathbf{x}$ generated by ω. We have included Appendix A for a brief overview of the finite field Fourier transform. The coefficients of $\mathbf{X}$ represent shares of $\mathbf{s}$, i.e., $[\mathbf{s}]_i = \mathbf{X}_i$. The details are given in Algorithm 2.

FastRecon to Reconstruct the Secrets

Let $\mathcal{D} \subseteq \mathcal{C}$ denote a random subset of size at most D. The reconstruction algorithm $\{\mathbf{s}, \perp\} \leftarrow \textsc{FastRecon}\left(\{(i, [\mathbf{s}]_i)\}_{i \in \mathcal{C} \setminus \mathcal{D}}\right)$ takes as input the shares corresponding to $\mathcal{C} \setminus \mathcal{D}$, and outputs either a set of S field elements $\mathbf{s}$ or a special symbol $\perp$. At a high level, FastRecon consists of two stages. First, it *iteratively* recovers all the missing shares by leveraging a linear-relationship between the shares (as described next). Second, it takes the inverse Fourier transform of the shares (i.e., the "spectrum") to obtain the signal, which contains the secrets at appropriate locations (see Fig. 3.1).

In particular, if we suppose there is a way to recover the missing shares to obtain $\mathbf{X}$, then, we can take the inverse Fourier transform of $\mathbf{X}$ to obtain $\mathbf{x}$, and read off the secrets from the indices in $\mathcal{S}$. Indeed, the key ingredient of FastRecon is a computationally efficient iterative algorithm, rooted in the field of coding theory, to recover the missing shares. Towards this end, we show that the shares satisfy certain parity-check constraints, which are induced by the careful placement of zeros in $\mathbf{x}$. (See figure 3.3 for a toy example.)

Lemma 3.2.1. *Let $\{X_j\}_{j=0}^{N-1}$ denote the shares produced by the* FastShare *scheme for an arbitrary secret $\mathbf{s}$. Then, for each $i \in \{0,1\}$, for every $c \in \{0, 1, \ldots, \frac{N}{n_i}\}$ and $v \in \{0, 1, \ldots, \delta_i n_i - 1\}$, it holds that*

$$\sum_{u=0}^{n_i - 1} \omega^{-uv\frac{N}{n_i}} X_{u\frac{N}{n_i}+c} = 0. \tag{3.6}$$

Proof. For $i \in \{0,1\}$, for $v \in \{0, 1, \ldots, \delta_i n_i - 1\}$, let us denote $\mathbf{x}^{(v)(\downarrow n_i)}$ as $\mathbf{x}$ circularly shifted (in advance) by v and then subsampled by n_i. That is, $\mathbf{x}_j^{(v)(\downarrow n_i)} = \mathbf{x}_{(jn_i+v) \mod n_i}$ for $j = 0, 1, \ldots, N/n_i - 1$. Let $\mathbf{X}^{(v)(\downarrow n_i)}$ denote the DFT of $\mathbf{x}^{(v)(\downarrow n_i)}$ generated by ω^{n_i}. Now, from the aliasing property (A.3), it holds that

$$\mathbf{X}_c^{(v)(\downarrow n_i)} = \frac{1}{n_i} \sum_{\substack{j=0 \\ j \mod (N/n_i) = c}}^{N-1} \mathbf{X}_j^{(v)}, \tag{3.7}$$

for $v = 0, 1, \ldots, \delta_i n_i - 1$ and $c = 0, 1, \ldots, N/n_i - 1$. However, from the circular shift property (A.4), we have $\mathbf{X}_j^{(v)} = \omega^{-vj}\mathbf{X}_j$ for $j = 0, 1, \ldots, N-1$. Thus, we have

$$\mathbf{X}_c^{(v)(\downarrow n_i)} = \frac{1}{n_i} \sum_{\substack{j=0 \\ j \mod (N/n_i) = c}}^{N-1} \omega^{-vj}\mathbf{X}_j, \tag{3.8}$$

for $v = 0, 1, \ldots, \delta_i n_i - 1$, and $c = 0, 1, \ldots, N/n_i - 1$. Note that, by construction, $\mathbf{x}^{(v)(\downarrow n_i)}$ is a length-(N/n_i) zero vector for $v = 0, 1, \ldots, \delta_i n_i - 1$ for each $i \in \{0,1\}$. Therefore, $X^{(v)(\downarrow n_i)}$

is also a length-(N/n_i) zero vector for $v = 0, 1, \ldots, \delta_i n_i - 1$ for each $i \in \{0, 1\}$. Hence, from (3.8), we get

$$\sum_{\substack{j=0 \\ j \mod (N/n_i)=c}}^{N-1} \omega^{-vj} X_j = \sum_{u=0}^{n_i-1} \omega^{-v\left(u\frac{N}{n_i}+c\right)} X_{u\frac{N}{n_i}+c} = 0, \tag{3.9}$$

for $v = 0, 1, \ldots, \delta_i n_i - 1$, and $c = 0, 1, \ldots, N/n_i - 1$.

Simplifying the above, we get

$$\sum_{u=0}^{n_i-1} \left(\omega^{-uv\frac{N}{n_i}}\right) X_{u\frac{N}{n_i}+c} = 0 \tag{3.10}$$

for $u = 0, 1, \ldots, n_i - 1$, $v = 0, 1, \ldots, \delta_i n_i - 1$, and $c = 0, 1, , \ldots, N/n_i - 1$ □

When translated in coding theory parlance, the above lemma essentially states that the shares form a codeword of a *product code* with Reed-Solomon component codes [36]. In particular, when the shares are represented on a 2D-grid using the Chinese remainder theorem, each row (resp. column) forms a *codeword* of a *Reed-Solomon code* with block-length n_0 (resp. n_1) and dimension $(1 - \delta_0)n_0$ (resp. $(1 - \delta_1)n_1$). In other words, $\bar{X}_c = \begin{bmatrix} X_c & X_{\frac{N}{n_i}+c} & \cdots & X_{(n_i-1)\frac{N}{n_i}+c} \end{bmatrix}$ is a codeword of an $(n_i, (1 - \delta_i)n_i)$ Reed-Solomon code for every $c = 0, 1, \ldots, N/n_i - 1$. To see this, observe that when the constraints in (3.6), for a fixed c, are written in a matrix form, the resulting matrix is a Vandermonde matrix, and it is straightforward to show that $\bar{X}_c$ corresponds to the evaluations of a polynomial of degree $(1 - \delta_i)n_i - 1$ at ω^{-uN/n_i} for $u = 0, 1, \ldots, n_i - 1$.

These parity-check constraints on the shares make it possible to iteratively recover missing shares from each row and column until all the missing shares can be recovered. We present a toy example for this in Fig. 3.3. It is worth noting that this way of recovering the missing symbols of a codeword is known in coding theory as an *iterative (bounded distance) decoder* [44, 30].

In particular, codewords of a Reed-Solomon code with block-length n_i and dimension $(1-\delta_i)n_i$ are evaluations of a polynomial of degree at most $(1 - \delta_i)n_i - 1$. Therefore, any δ_i fraction of erasures can be recovered via polynomial interepolation. Therefore, if a row (resp. column) has less than $\delta_0 n_0$ (resp. $\delta_1 n_1$) missing shares, then they can be recovered. This process is repeated for a fixed number if iterations, or until all the missing shares are recovered.

Putting things together, FastRecon first uses an iterative decoder to obtain the missing shares. If the peeling decoder fails, it outputs $\perp$, and declares failure. Otherwise, it takes the inverse (fast) Fourier transform of $\mathbf{X}$ (generated by ω) to obtain $\mathbf{x}$. Finally, it output the

coordinates of $\mathbf{x}$ indexed by $\mathcal{S}$ (in the 2D-grid representation) as the secret. The details are given in Algorithm 2.

Analysis of FastShare

First, we analyze the security and correctness of FastShare in the honest-but-curious setting. We defer the proofs to B.

Theorem 3.2.2. *Given fractions $\delta_0, \delta_1, \alpha \in (0,1)$ and $\beta \in (0, 1/2)$,* FastShare *generates N shares from $S = (1-\alpha)(1-2\beta)(1-\delta_0)(1-\delta_1)N$ secrets such that it satisfies*

1. *T-privacy for $T = \alpha\beta(1-\delta_0)(1-\delta_1)N$,*
2. *D-dropout resilience for $D = (1-(1-\delta_0)(1-\delta_1))\frac{N}{2}$, and*
3. *linearity.*

For example, choosing $\alpha = 1/2$, $\beta = 1/4$, $\delta_0 = \delta_1 = 1/10$, yields $S = 0.2N$, $T = 0.1N$, and $D = 0.095N$.

Lemma 3.2.3. FastShare *runs in $\mathcal{O}(N \log N)$ time*

Proof. Constructing the signal takes $\mathcal{O}(N)$ time and the Fourier transform can be computed in $\mathcal{O}(N \log N)$ time using a Fast Fourier Transform (FFT). FastRecon also runs in $O(N \log N)$ time, when computations are parallelized. Specifically, recovering the missing shares in a row or column of shares (when arranged in the 2D-grid) can be done in $\mathcal{O}(n_i^2) = \mathcal{O}(N)$ complexity by leveraging that rows and columns are Reed-Solomon codewords, which are evaluations of a degree-$(n_i - \delta_i n_i - 1)$ polynomial. Since all rows and columns can be decoded in parallel, and decoding is carried out for a constant number of iterations, the iterative decoder runs in $\mathcal{O}(N)$ time. The second step is the inverse Fourier transform, which can be computed in $\mathcal{O}(N \log N)$ time using an FFT. □

Note that in practical federated learning setups, the number of clients typically scales up to ten thousand [12]. Therefore, in order to gain full parallelism, one needs to have $\sqrt{N} \approx 100$ processors/cores. In the next section, we present a variant of FastShare that removes the requirement of parallelism for sufficiently large N. Moreover, this variant also allows us to achieve a more favorable trade-off between the number of shares, privacy threshold, and dropout tolerance.

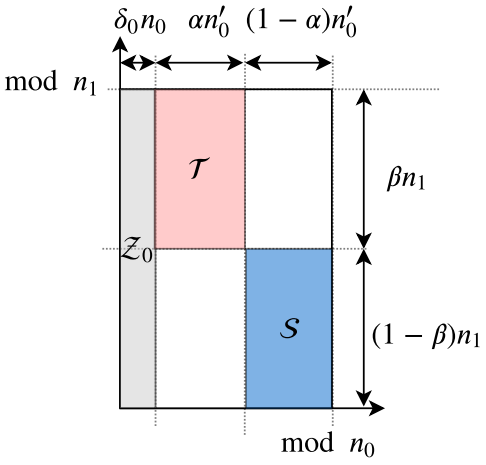


Figure 3.4: Indices assigned to a grid, and sets $\mathcal{Z}$, $\mathcal{S}$, and $\mathcal{T}$. Here, we define $n_0' = (1-\delta_0)n_0$. Set $\mathcal{S}$ is used for secrets, $\mathcal{Z}_0$ is used for zeros, and the remaining locations are used for random masks. Set $\mathcal{T}$ is used in our privacy proof.

Improving the Performance for Large Number of Clients

FastShare ensures that when the shares are arranged in a 2D-grid using the Chinese remainder theorem, each row and column satisfies certain parity-check constraints. As we show next, when N is sufficiently large, it suffices to ensure that only rows (or columns) satisfy the parity-check constraints. In this case, the FastShare algorithm remains the same as before except for the placement of secrets, zeros, and random masks changes slightly.

Let q be a power of a prime, and let N be a positive integer such that N divides $q-1$. Let $c \geq 1$ be a constant. We consider N of the form $N = n_0 n_1$ such that $n_0 = c \log N$. Given fractions $\delta_0, \alpha, \beta \in (0,1)$, we define the sets of tuples $\mathcal{Z}_0$, $\mathcal{S}$, and $\mathcal{T}$ using the grid representation determined by the Chinese remainder theorem, as depicted in Fig. 3.4. We give the formal definitions below. Note that we round any rational number x to $\lfloor x \rfloor$, and omit the floor sign for the sake of brevity.

$$\mathcal{Z}_0 = \{(a,b) : 0 \leq a \leq \delta_0 n_0 - 1\}, \tag{3.11}$$

$$\mathcal{S} = \{(a,b) : \delta_0 n_0 + \alpha(1-\delta_0)n_0 \leq a \leq n_0 - 1, \\ 0 \leq b \leq (1-\beta)n_1\}, \tag{3.12}$$

$$\mathcal{T} = \{(a,b) : \delta_0 n_0 \leq a \leq \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1, \\ (1-\beta)n_1 \leq b \leq n_1 - 1\}. \tag{3.13}$$

We construct the signal by placing zeros at indices in $\mathcal{Z}_0$, and secrets at indices in $\mathcal{S}$. Each of the remaining indices is assigned a uniform random mask from $\mathbb{F}_q$ (independent of other

masks and the secrets). We then take the Fourier transform (generated by a primitive N-th root of unity in $\mathbb{F}_q$) of the signal to obtain the shares.

Using the same arguments as in the proof of Lemma 3.2.1, it is straightforward to show that, for every $c \in \{0, \ldots, n_1 - 1\}$ and $v \in \{0, 1, \ldots, \delta_0 n_0 - 1\}$, it holds that

$$\sum_{u=0}^{n_0-1} \omega^{-uvn_1} X_{un_1+c} = 0. \tag{3.14}$$

When translated in coding theory parlance, the above condition states that when the shares are represented in a 2D-grid using the Chinese remainder theorem, then each row forms a codeword of a Reed-Solomon code with block-length n_0 and dimension $(1 - \delta_0)n_0$.[2]

FastRecon first recovers the missing shares by decoding each row. If every row has less than δ_0 fraction of erasures, then it recovers all the missing shares. Otherwise, it outputs $\perp$, and declares failure. Next, it takes the inverse (fast) Fourier transform of the shares to obtain the signal. The coordinates of the signal indexed by $\mathcal{S}$ (in the 2D-grid representation) gives the secrets.

Security and Correctness Given fractions $\delta_0, \alpha, \beta \in (0, 1)$, this variant of FastShare generates N shares from

$$S = (1 - \alpha)(1 - \beta)(1 - \delta_0)N \tag{3.15}$$

secrets such that it satisfies T-privacy for

$$T = \alpha\beta(1 - \delta_0)N, \tag{3.16}$$

D-dropout-resilience for

$$D = \delta_0 N \tag{3.17}$$

and linearity. (The proof is similar to Theorem 3.2.2, and is omitted.)

Observe that this variant achieves a better trade-off between S, T, and D. For instance, choosing $\delta_0 = 1/10$, $\alpha = 1/2$, and $\beta = 1/2$, yields $S = 0.225N$, $T = 0.225$, and $D = 0.1$.

Computation Cost In this case, FastShare also has $\mathcal{O}(N \log N)$ computational cost, since constructing the signal takes $\mathcal{O}(N)$ time and the Fourier transform can be computed in $\mathcal{O}(N \log N)$ time using an FFT. FastRecon has $\mathcal{O}(N \log N)$ computational cost without requiring any parallelism. In particular, recovering the missing shares in a row (when arranged in the 2D-grid) can be done in $\mathcal{O}(n_0^2) = \mathcal{O}\left(\log^2 N\right)$ complexity by leveraging that the rows are Reed-Solomon codewords, which are evaluations of a degree-$(n_0 - \delta_0 n_0 - 1)$ polynomial. Since there are $\mathcal{O}(N/\log N)$ rows, the decoder to recover the missing shares has $\mathcal{O}(N \log N)$ complexity. The second step is the inverse FFT, which also has $\mathcal{O}(N \log N)$ complexity.

[2]We note that this variant of FastShare is related to a class of erasure codes called Locally Recoverable Codes (LRCs) (see [25, 42, 54], and references therein).

3.3 FastSecAgg Based on FastShare

We present FastSecAgg, a procedure that allows the server to securely compute the summation of clients' model updates. We begin with a high-level overview of FastSecAgg (see Fig. 3.5). Each client generates shares for its length-L input (by breaking it into a sequence of $\lceil L/S \rceil$ vectors, each of length at most S, and treating each vector as S secrets) using FastShare, and distributes the shares to N clients. Since all the communication in FL is mediated by the server, clients encrypt their shares before sending it to the server to prevent the server from reconstructing the secrets. Each client then decrypts and sums the shares it receives from other clients (via the server). The linearity property of FastShare ensures that the sum of shares is a share of the sum of secret vectors (i.e., client inputs). Each client then sends the *sum-share* to the server (as a plaintext). The server can then recover the sum of secrets (i.e., client inputs) with high probability as long as it receives the shares from a random set of clients of sufficient size. We note that FastSecAgg uses secret sharing as a primitive in a standard manner, similar to several secure aggregation protocols [7, 14, 26].

FastSecAgg is a three round interactive protocol. See Fig. 3.5 for a high-level overview, and Algorithm 3 for the detailed protocol. Recall that the model update for client $i \in \mathcal{C}$ is assumed to be $\mathbf{u}_i \in \mathbb{Z}_R^L$, for some $R \leq q$. In practice, this can be achieved by appropriately quantizing the updates.

Round 0 consists of generating and advertising encryption keys. Specifically, each client i uses the key agreement protocol to generate a public-private key pair $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \text{KA.GEN}(pp)$, and sends their public key pk_i to the server. The server waits for at least $N - D$ clients (denoted as $\mathcal{C}_0 \subseteq \mathcal{C}$), and forwards the received public keys to clients in $\mathcal{C}_0$.

Round 1 consists of generating secret shares for the updates. Each client i partitions their update $\mathbf{u}_i$ into $\lceil L/S \rceil$ vectors, $\mathbf{u}_i^1$, $\mathbf{u}_i^2$, ..., $\mathbf{u}_i^{\lceil L/S \rceil}$, such that the last vector has length at most S and all others have length S. Treating each $\mathbf{u}_i^\ell$ as S secrets, the client computes N shares as $\{(j, \left[\mathbf{u}_i^\ell\right]_j)\}_{j \in \mathcal{C}} \leftarrow \text{FastShare}(\mathbf{u}_i^\ell, \mathcal{C})$ for $1 \leq \ell \leq \lceil L/S \rceil$. For simplicity, we denote client i's share for client j as $\mathsf{sh}_{i \to j} = ([\mathbf{u}_i^1]_j \,||\, [\mathbf{u}_i^2]_j \,||\, \cdots \,||\, [\mathbf{u}_i^{\lceil L/S \rceil}]_j)$.

In addition, every client receives the list of public keys from the server. Client i generates a shared key $k_{i,j} \leftarrow \text{KA.AGREE}(\mathsf{sk}_i, \mathsf{pk}_j)$ for each $j \in \mathcal{C}_0 \setminus \{i\}$, and encrypts $\mathsf{sh}_{i \to j}$ using the shared key $k_{i,j}$ as $c_{i \to j} \leftarrow \text{AE.ENC}\,(k_{i,j}, \mathsf{sh}_{i \to j})$. The client then sends all the encrypted shares $\{c_{i \to j}\}_{j \in \mathcal{C}_0 \setminus \{i\}}$ to the server.

The server waits for at least $N - D$ clients to respond (denoted as $\mathcal{C}_1 \subseteq \mathcal{C}_0$).[3] Then, the

[3]For simplicity, we assume that a client does not drop out after initiating communication with the server, i.e., while sending or receiving messages from the server. The same assumption is also made in [11, 50, 5]. This is not a critical assumption, and the protocol and the analysis can be easily adapted if it does not hold.

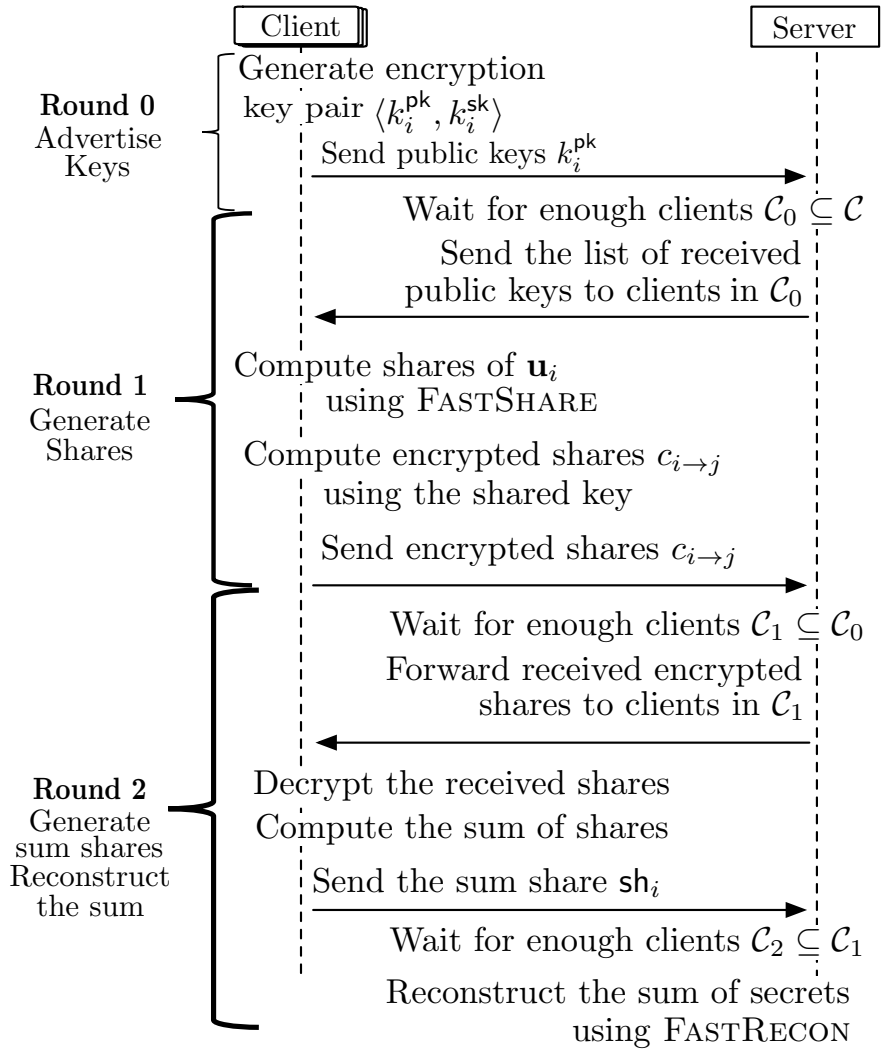


Figure 3.5: High-level overview of FastSecAgg protocol.

server sends to each client $i \in \mathcal{C}_1$, all ciphertexts encrypted for it: $\{c_{j\to i}\}_{j\in\mathcal{C}_1\setminus\{i\}}$.

Round 2 consists of generating sum-shares and reconstructing the approximate sum. Every surviving client receives the list of encrypted shares from the server. Each client i then decrypts the ciphertexts $c_{j\to i}$ using the shared key $k_{j,i}$ to obtain the shares $\mathsf{sh}_{j\to i}$, i.e., $\mathsf{sh}_{j\to i} \leftarrow \text{AE.DEC}(k_{j,i}, c_{j\to i})$ where $k_{j,i} \leftarrow \text{KA.AGREE}(\mathsf{sk}_j, \mathsf{pk}_i)$. Then, each client i computes the sum (over $\mathbb{F}_q$) of all the shares including its own share as: $\mathsf{sh}_i = \sum_{j\in\mathcal{C}_1} \mathsf{sh}_{j\to i}$. Each client i sends the sum-share sh_i to the server (as a plaintext).

The server waits for at least $N - D$ clients to respond (denoted as $\mathcal{C}_2 \subseteq \mathcal{C}_1$). Let sh_i^ℓ denote the ℓ-th coefficient of sh_i for $1 \le \ell \le \lceil L/S \rceil$. The server computes $\{\mathbf{z}^\ell, \perp\} \leftarrow$

FastRecon $\left(\{(i, \mathsf{sh}_i^\ell)\}_{i\in\mathcal{C}_2}\right)$, for $1 \le \ell \le \lceil L/S \rceil$. If the reconstruction fails (i.e., outputs $\perp$) for any ℓ, then the server aborts the current FL iteration and moves to the next one. Otherwise, it outputs $\mathbf{z} = [\mathbf{z}^1 \ \mathbf{z}^2 \ \cdots \ \mathbf{z}^{\lceil L/S \rceil}]$.

3.4 Analysis

Correctness and Security

First we state the correctness of FastSecAgg, which essentially follows from the linearity and D-dropout tolerance of FastShare.

Theorem 3.4.1 (Correctness)**.** *Let $\{\mathbf{u}_i\}_{i\in\mathcal{C}}$ denote the client inputs for* FastSecAgg*. If a random set of at most D clients drop out during the execution of* FastSecAgg*, i.e., $|\mathcal{C}_2| \ge N - D$, then the server does not abort and obtains $\mathbf{z} = \sum_{i\in\mathcal{C}_1} \mathbf{u}_i$ with probability at least $1 - 1/\mathit{poly}\, N$, where the probability is over the randomness in dropouts.*

Next, we show that FastSecAgg is secure against the server colluding with up to T clients in the honest-but-curious setting, irrespective of how and when clients drop out. Specifically, we prove that the joint *view* of the server and any set of clients of bounded size does not reveal any information about the updates of the honest clients, besides what can be inferred from the output of the summation.

We will consider executions of FastSecAgg where FastShare has privacy threshold T, and the underlying cryptographic primitives are instantiated with security parameter λ. We denote the server (i.e., the aggregator) as A, and the set of of N clients as $\mathcal{C}$. Clients may drop out (or, abort) at any point during the execution, and we denote with $\mathcal{C}_i$ the subset of the clients that correctly sent their message to the server in round i. Therefore, we have $\mathcal{C} \supseteq \mathcal{C}_0 \supseteq \mathcal{C}_1 \supseteq \mathcal{C}_2$. For example, the set $\mathcal{C}_0 \setminus \mathcal{C}_1$ are the clients that abort before sending the message to the server in Round 1, but after sending the message in Round 0. Let $\mathbf{u}_i \in \mathbb{Z}_R^L$ denote the model update of client i (i.e., $\mathbf{u}_i$ the i-th client's input to the secure aggregation protocol), and for any subset $\mathcal{C}' \subseteq \mathcal{C}$, let $\mathbf{u}_{\mathcal{C}'} = \{\mathbf{u}_i\}_{i\in\mathcal{C}'}$.

In such a protocol execution, the view of a participant consists of their internal state (including their update, encryption keys, and randomness) and all messages they received from other parties. Note that the messages sent by the participant are not included in the view, as they can be determined using the other elements of their view. If a client drops out (or, aborts), it stops receiving messages and the view is not extended past the last message received.

Given any subset $\mathcal{M} \subset \mathcal{C}$, let $\mathsf{REAL}_{\mathcal{M}}^{\mathcal{C},T,\lambda}(\mathbf{u}_{\mathcal{C}}, \mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2)$ be a random variable representing the combined views of all parties in $\mathcal{M} \cup \{A\}$ in an execution of FastSecAgg, where the randomness is over the internal randomness of all parties, and the randomness in the setup phase. We show that for any such set $\mathcal{M}$ of honest-but-curious clients of size up to T, the

joint view of $\mathcal{M} \cup \{A\}$ can be simulated given the inputs of the clients in $\mathcal{M}$, and only the sum of the values of the remaining clients.

Theorem 3.4.2 (Security). *There exists a probabilistic polynomial time (PPT) simulator* SIM *such that for all* $\mathcal{C}$, $\mathbf{u}_{\mathcal{C}}$, $\mathcal{C}_0$, $\mathcal{C}_1$, $\mathcal{C}_2$, *and* $\mathcal{M}$ *such that* $M \subset \mathcal{C}$, $|\mathcal{M}| \leq T$, $\mathcal{C} \supseteq \mathcal{C}_0 \supseteq \mathcal{C}_1 \supseteq \mathcal{C}_2$, *the output of* SIM *is computationally indistiguishable from the joint view* $\mathsf{REAL}_{\mathcal{M}}^{\mathcal{C},T,\lambda}$ *of the server and the corrupted clients* $\mathcal{M}$, *i.e.,*

$$\mathsf{REAL}_{\mathcal{M}}^{\mathcal{C},T,\lambda}(\mathbf{u}_{\mathcal{C}}, \mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2) \approx \mathsf{SIM}_{\mathcal{M}}^{\mathcal{C},T,\lambda}(\mathbf{u}_{\mathcal{M}}, \mathbf{z}, \mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2), \tag{3.18}$$

where

$$\mathbf{z} = \begin{cases} \sum_{i \in \mathcal{C}_1 \setminus \mathcal{M}} \mathbf{u}_i & \text{if } |\mathcal{C}_1| \geq N - D, \\ \perp & \text{otherwise.} \end{cases} \tag{3.19}$$

The security and correctness of FastSecAgg critically relies on the guarantees provided by FastShare proved in Theorem 3.2.2. We defer proof of the above theorem to [31].

Computation and Communication Costs

We assume that the addition and multiplication operations in $\mathbb{F}_q$ are $\mathcal{O}(1)$ each.

Computation Cost at a Client $\mathcal{O}(\max\{L, N\} \log N)$. Each client's computation cost can be broken as (1) computing shares using FastShare which is

$$\mathcal{O}(\lceil L/S \rceil N \log N) = \mathcal{O}(\max\{L, N\} \log N)$$

(2) encrypting and decrypting shares, which is

$$\mathcal{O}(\lceil L/S \rceil N)$$

(3) adding the received shares, which is

$$\mathcal{O}(\lceil L/S \rceil N) = \mathcal{O}(\max\{L, N\})$$

Communication Cost at a Client $\mathcal{O}(\max\{L, N\})$. Each client sends and receives $N-1$ shares (each having $\lceil L/S \rceil$ elements in $\mathbb{F}_q$), resulting in $\mathcal{O}(\lceil L/S \rceil N) = \mathcal{O}(\max\{L, N\})$ communication. In addition, each client sends the sum-share consisting of $\lceil L/S \rceil$ elements in $\mathbb{F}_q$.

Computation Cost at the Server $\mathcal{O}(\max\{L, N\} \log N)$. The server first recovers missing sum-shares using FastRecon, each has $\lceil L/S \rceil$ elements in $\mathbb{F}_q$. This results in $\mathcal{O}(\lceil L/S \rceil N \log N)$ complexity. The second step is the inverse FFT on N shares, each has of $\lceil L/S \rceil$ elements in $\mathbb{F}_q$, resulting in $\mathcal{O}(\lceil L/S \rceil N \log N)$ complexity.

Communication Cost at the Server $\mathcal{O}(N \max\{L, N\})$. The server communicates N times of what each client communicates.

1. **Computation cost at server** is $O(N \log N)$ which is the cost of reconstructing the sum-share (Chapter 3.2.2).
2. **Communication cost at server** is $\mathcal{O}(N^2)$ to broadcast each client's shares to N other clients.
3. **Computation cost at client** is $\mathcal{O}(N \log N)$ which is the cost of generating the shares (Chapter 3.2.2).
4. **Communication cost at client** is $\mathcal{O}(N)$ to transmit shares and sum-shares.

3.5 FastSecAgg Meets Federated Learning

Using FastSecAgg in the federated learning setup requires extra setup. With FastSecAgg, after computing the gradient update $\Delta \mathbf{w}_i^t \in \mathbb{R}^d$, we clip the coordinates of the gradient, and restrict it to $[-t, t]$ for some parameter t, and quantize each coordinate to $\{0, \ldots, q-1\}$ since we work in $\mathbb{F}_q$. For more practical purposes, we also scale our gradient updates by s (scale parameter) since most gradient update values are very small, and are not reflected when we quantize the coordinates to the nearest integer.

However, similarly to our work on Distributed Learning presented in Chapter 2, this setup still suffers from poor communication efficiency. We note that sketching techniques have been used to significantly reduce communication costs in Federated Learning setup [2, 16, 51, 29]. Indeed, in practical machine learning problems, accuracy decreases only slightly even if the model updates are computed approximately. Compressing model updates via sketching enable us to effectively share the sketched updates. In particular, we have chosen to use Subsampled Randomized Hadamard Transform, because using SRHT concentrates values around 0 (see figure 3.6) as a Gaussian, and SRHT helps us with restricting our values to $[-t, t]$. Then, the SRHT sketching algorithm computes $\mathbf{s} = SHD\mathbf{w}$ for some gradient $\mathbf{w}$, where we let $\ell = \mathcal{O}(\text{poly } \log d)$. Since we can choose other sketching algorithms, we let this operation be Sketch.

We estimate (or unsketch) the sketched vector at the server node by obtaining a d-dimensional vector from $\mathbf{s}$ by substituting zeros at locations not sampled by S, then multiplying it by $(HD)^{-1}$ to obtain an estimate of $\mathbf{v}$. We let this operation be Estimate moving forward.

We also note that beyond theoretical guarantees from Johnson-Lindenstrauss Lemma, the result of unsketching the vectors produce good estimates (see figure 3.7).

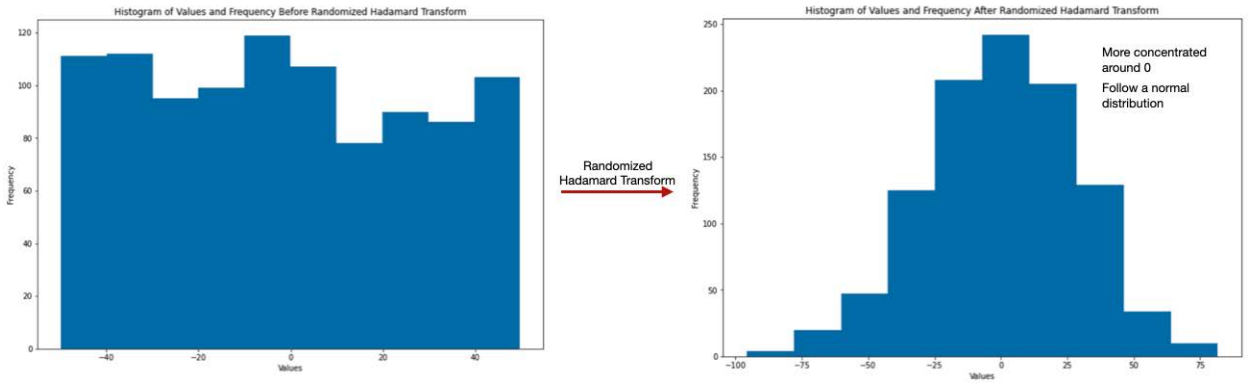


Figure 3.6: Effect of Applying Randomized Hadamard Transform on a Set of Uniform Vectors

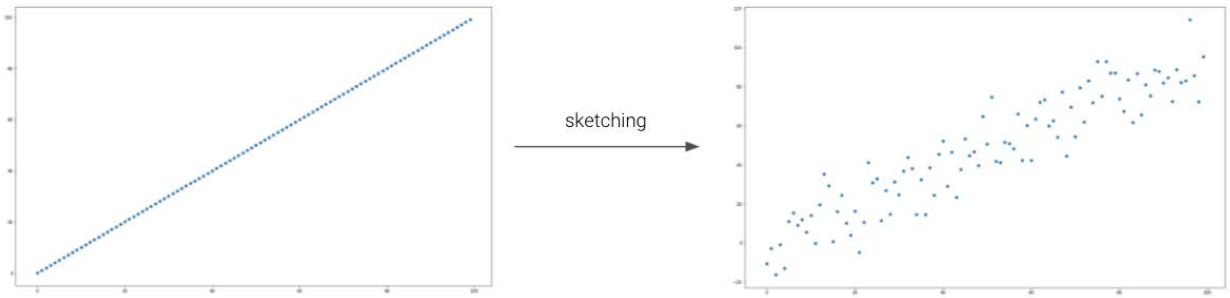


Figure 3.7: Effect of Estimating Vector Coordinates After Applying SRHT (80% Sketching). Left-hand side contains coordinates prior to SRHT, and right-hand side is the result of applying ESTIMATE on the Sketched vectors

We note that it is indeed possible to use other sketching techniques such as COUNT-SKETCH, but concentration around zero after Randomized Hadamard Transform "uniformizes" the energy of each update makes SRHT a good starting point for applying Sketching techniques on FASTSHARE.

In addition to Sketching, we also clip gradients and quantize the gradients to the nearest integer in $\{0, \ldots, q-1\}$. The second step is a requirement to run FASTSHARE effectively on the sketched gradients, since FASTSHARE takes as inputs secrets in $\mathbb{F}_q$. We take the first step of clipping the gradients to $[-t, t]$ to avoid modular wraparounds. Since we operate in $\mathbb{F}_q$, when the sum of shares become larger than q, then we will have modulo wraparounds, where some individual coordinates of the gradients will be incorrectly shared. Works around parameter pruning [52] suggests that there are only a few parameters in a neural network that contribute greatly to learning, and these parameters are often the parameters with the

largest coordinates. When we combine this with that the coordinates most greatly affected by modulo wraparounds will be the parameters with the largest coordinates, it suggests a need to clip the gradients to avoid modulo wraparounds. That said, t is a hyperparameter in our model, because we gain greater precision when t is large, at the cost of wrapping some coordinates around. We tune t to achieve the best performances possible.

We can then update our high-level overview of FastSecAgg protocol in 3.5 to include Sketching in 3.8. In the figure we note that sketching has been added in Round 1, where we generate the shares, and then the unsketching (or estimation) protocol has been added in the final step after reconstructing the shares.

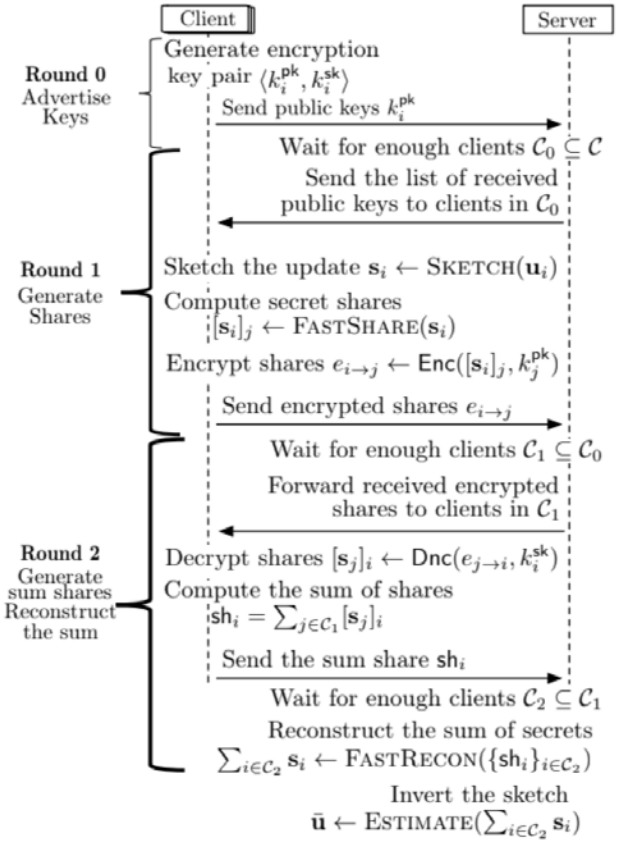


Figure 3.8: High Level Overview of FastSecAgg Protocol with Sketching.

3.6 Experiments for FastSecAgg

Datasets Used

In our work, we have used MNIST dataset (sampled non-i.i.d.) and LEAF datasets [17]. In constructing non-i.i.d. partition of the MNIST dataset, we sort the data by digit label and divided it into 2000 shards of size 30, and assigned each clients 2 shards of the dataset. LEAF Datasets a suite of open-source benchmark federated datasets for testing machine learning problems in a practical federated environment. Of the LEAF datasets, we have used EMNIST and Shakespeare Dataset. EMNIST [20] extends MNIST Dataset to handwritten letters, and the EMNIST Dataset from LEAF splits the dataset into non-i.i.d. partitions based on the writer of the digit/character (mean of 226.83 data points per writer). The Shakespeare Dataset is built from *The Complete Works of William Shakespeare*, and each speaking role in each play is considered a separate client. In the Shakespeare Dataset, we perform next character prediction, and with MNIST and EMNIST datasets, we perform image classification.

Experimental Setup for FastSecAgg

To train the MNIST dataset, we used a modified LeNet [33]. To train the EMNIST dataset, we used FLNet [38] using a CNN with two 5×5 convolutional layers (each followed by MaxPool layer), a fully connected layer with 512 units and ReLU activation, and a final softmax output layer. In training for the Shakespeare Dataset, we used the same LSTM model used in [17] to replicate their results (two layers with 256 units each, and emits an output embedding that is scored against all items of the vocabulary via dot product, followed by a final softmax output layer). We use a sequence length of 80 for the LSTM model.

In the first experiment using EMNIST dataset, we simulated, again $N = 2244$ clients with non-i.i.d. partition of the training data across devices, as provided by the LEAF dataset partition. We selected 50% of the clients per round of communication, and were able to tolerate up to 10% dropout. To compute a client model update, we used a batch size of 128 training examples for 3 epochs through the client's data with a learning rate of 0.01, and momentum parameter 0.9. Each client computed the model update, then we sketched the number of parameters by a factor of 50%, before quantizing each coordinate to $[0, 4]$ after scaling it by 1000.

In the second experiment using Shakespeare dataset, we simulated $N = 1129$ clients with non-i.i.d. partition of the training data across devices, as provided by the LEAF dataset partition. We selected 10% of the clients per round of communication, and were able to tolerate up to 10% dropout. To compute a client model update, we used a batch size of 128 training examples for 3 epochs through the client's data with a learning rate of 0.01, and momentum parameter 0.9. Each client computed the model update, then we sketched the

number of parameters by a factor of 50%, before quantizing each coordinate to $[0, 4]$ after scaling it by 1000.

In our proof-of-concept experiment with MNIST dataset, we simulated $N = 224$ clients for federated learning with a non-i.i.d. partition of the training data across devices. In particular, we selected 50% of the clients per round of communication, and were able to tolerate up to 10% dropout. To compute a client model update, we used a batch size of 10 training examples for 3 epochs through the client's data with a learning rate of 0.01, and momentum parameter 0.9. Each client first computes the model update, then sketched number of parameters (either 50% or no sketching). We furthermore used $q = 1001947$, clipped to $[0, 20]$ and scaled by 2000.

In all experiments, all clients use the same S and D matrices provided by the server (via a random seed). This aggressive quantization is chosen to avoid overflows since we chose FastSecAgg to work over $\mathbb{F}_{1123}$, and computes the addition modulo 1123. We assume a 10% dropout rate with each client dropping out independently. If FastRecon fails, the server simply moves to the next iteration with the same global model.

3.7 Results

Running FastSecAgg on MNIST, EMNIST and Shakespeare Datasets Against Oracle

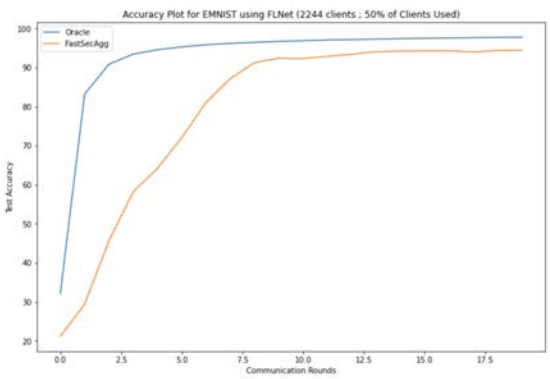
We plot test accuracy of classifying EMNIST against the number of communication rounds in and predicting the next character using Shakespeare Dataset in Fig. 3.9.

Observe that, even though FastSecAgg uses aggressive subsampling and quantization, and computes the sum modulo 1123, the drop in accuracy is controllable. We note that the accuracy achieved by SecAgg is similar, since it also computes the sum over a finite field requiring aggressive quantization. On the other hand, the computation cost of SecAgg at the server for 1122 users is larger by several orders of magnitude, making every iteration significantly slower.

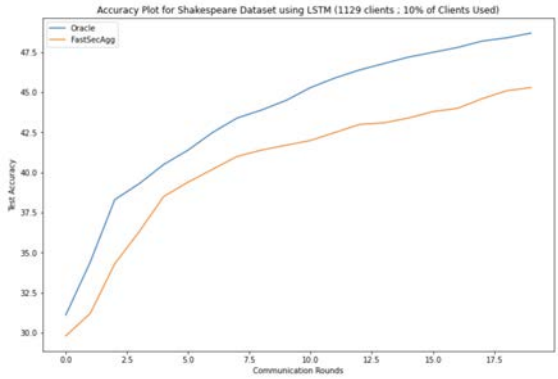
Effect of Changing Sketch Sizes (Proof-of-Concept Using MNIST)

In the plots above, we noticed that there is a divergence between the oracle (not using FastSecAgg), and using FastSecAgg. We attributed the error to three causes: (1) Sketching and (2) Quantization and (3) Modulo Wraparounds. To investigate further, we lowered our aggresive subsampling and quantization thresholds to more reasonable levels, and tested the effect of sketching.

Below, we have three plots: (1) Oracle - which does not use FastSecAgg, (2) 100% - which



(a) Plot of Communication Rounds and Test Accuracy for EMNIST Dataset using FastSecAgg and Oracle for 2244 clients with 50% selection



(b) Plot of Communication Rounds and Test Accuracy for Shakespeare Dataset using FastSecAgg and Oracle for 1129 clients with 10% selection

Figure 3.9: Plot of Communication Rounds and Test Accuracy for EMNIST and Shakespeare Dataset

uses FastSecAgg without Sketching and (3) 50% - which uses FastSecAgg with 50% Sketching

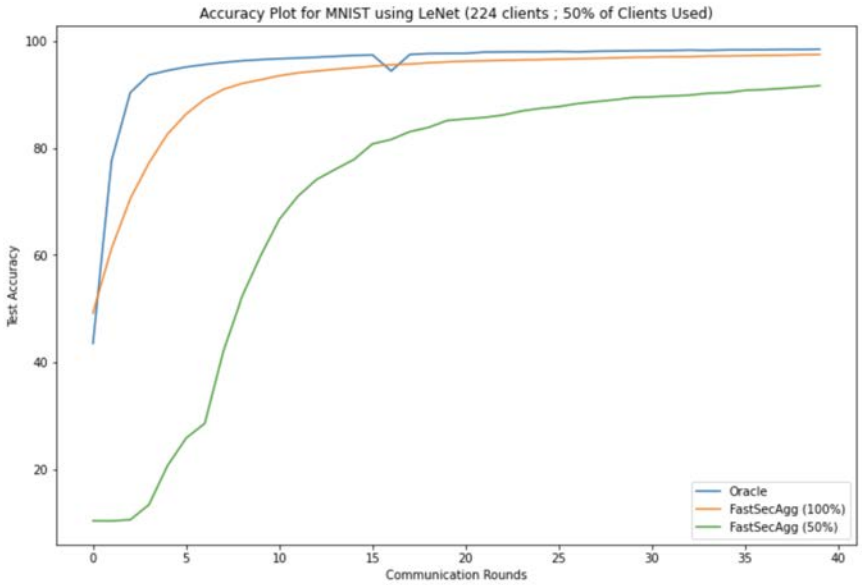


Figure 3.10: Plot of Communication Rounds and Test Accuracy for MNIST Dataset using FastSecAgg and Oracle on different sized sketches

In figure 3.10, notice that while there still exists a gap between FastSecAgg and the oracle, even without sketching, we believe the difference can be attributed to quantization and stochasticity in training. This result shows that we can achieve good tradeoffs between computation and communication cost, and accuracy.

Algorithm 2 FastShare Secret Sharing

procedure FastShare($\mathbf{s}, \mathcal{C}$)
parameters: finite field $\mathbb{F}_q$ of size q, a primitive root of unity ω in $\mathbb{F}_q$, privacy threshold T, dropout resilience D
inputs: secret $\mathbf{s} \in \mathbb{F}_q^L$, set of N clients $\mathcal{C}$ (client-IDs)
initialize: sets $\mathcal{Z}_0$, $\mathcal{Z}_1$, $\mathcal{S}$ as per (3.2), (3.2), (3.4), respectively; an arbitrary bijection $\sigma : \mathcal{S} \to \{0, 1, \ldots, S-1\}$
for $j = 0$ to $n-1$ **do**
if $j \in \mathcal{Z}_0 \cup \mathcal{Z}_1$ **then**
$x_j \leftarrow 0$
else if $j \in \mathcal{S}$ **then**
$x_j \leftarrow \mathbf{s}_{\sigma(j)}$ ▷ $\mathbf{s}_i$ is the i-th coordinate of $\mathbf{s}$
else
$x_j \xleftarrow{\$} \mathbb{F}_q$
end if
end for
$X \leftarrow FFT_\omega(x)$
Output: $\{(i, [\mathbf{s}]_i)\}_{i \in \mathcal{C}} \leftarrow \{(i, X_i)\}_{i=0}^{N-1}$
end procedure

procedure FastRecon($\{(i, [\mathbf{s}]_i)\}_{i \in \mathcal{R}}$)
parameters: finite field $\mathbb{F}_q$ of size q, a primitive root of unity ω in $\mathbb{F}_q$, privacy threshold T, dropout resilience D, number of iterations J, bijection σ and set $\mathcal{S}$ used in FastShare
input: subset of shares with client-IDs $\{(i, [\mathbf{s}]_i)\}_{i \in \mathcal{R}}$
for iterations $j = 1$ to J **do**
for rows $r = 0$ to $n_1 - 1$ in parallel **do**
if r has fewer than $\delta_0 n_0$ missing shares **then**
Decode the missing share values by polynomial interpolation
end if
end for
for columns $c = 0$ to $n_0 - 1$ in parallel **do**
if c has fewer than $\delta_1 n_1$ missing shares **then**
Decode the missing share values by polynomial interpolation
end if
end for
end for
if any missing share **then**
Output: $\perp$
else
$\mathbf{x} \leftarrow IFFT_\omega(\mathbf{X})$
Output: $\mathbf{s} \leftarrow \mathbf{X}(\sigma(\mathcal{S}))$
end if
end procedure=0

Algorithm 3 FastSecAgg Protocol

- **Parties:** Clients $1, 2, \ldots, N$ and the server
- **Public Parameters:** Update length L, input domain $\mathbb{Z}_R$, key agreement parameter $pp \leftarrow \text{KA.param}(\lambda)$, finite field $\mathbb{F}_q$ for FastShare secret sharing with primitive N-th root of unity ω
- **Input:** $\mathbf{u}_i \in \mathbb{Z}_R^L$ (for each client i)
- **Output:** $\mathbf{z} \in \mathbb{Z}_R^L$ (for the server)
- **Round 0 (Advertise Keys)**
  Client i:
  - Generate key pairs $(\mathsf{pk}_i, \mathsf{sk}_i) \leftarrow \text{KA.gen}(pp)$
  - Send pk_i to the server and move to the next round

  Server:
  - Wait for at least $N - D$ clients to respond (denote this set as $\mathcal{C}_0 \subseteq \mathcal{C}$); otherwise, abort
  - Send to all clients in $\mathcal{C}_0$ the list $\{(i, \mathsf{pk}_i)\}_{i \in \mathcal{C}_0}$, and move to the next round
- **Round 1** (Generate shares)
  Client i:
  - Receive the list $\{(j, \mathsf{pk}_j)\}_{j \in \mathcal{C}_0}$ from the server; Assert that $|\mathcal{C}_0| \geq N - D$, otherwise abort
  - Partition the input $\mathbf{u}_i \in \mathbb{Z}_R^L$ into $\lceil L/S \rceil$ vectors, $\mathbf{u}_i^1$, $\mathbf{u}_i^2$, ..., $\mathbf{u}_i^{\lceil L/S \rceil}$, such that $\mathbf{u}_i^{\lceil L/S \rceil}$ has length at most S and all others have length S
  - Compute N shares by treating each $\mathbf{u}_i^\ell$ as S secrets as $\{(j, \left[\mathbf{u}_i^\ell\right]_j)\}_{j \in \mathcal{C}} \leftarrow \text{FastShare}(\mathbf{u}_i^\ell, \mathcal{C})$ for $1 \leq \ell \leq \lceil L/S \rceil$ (by using independent private randomness for each ℓ); Denote client i's share for client j as $\mathsf{sh}_{i \to j} = \left(\left[\mathbf{u}_i^1\right]_j \,||\, \left[\mathbf{u}_i^2\right]_j \,||\, \cdots \,||\, \left[\mathbf{u}_i^{\lceil L/S \rceil}\right]_j \right)$
  - For each client $j \in \mathcal{C}_0 \setminus \{i\}$, compute encrypted share: $c_{i \to j} \leftarrow \text{AE.enc}(k_{i,j}, i \,||\, j \,||\, \mathsf{sh}_{i \to j})$, where $k_{i,j} = \text{KA.agree}(\mathsf{sk}_i, \mathsf{pk}_j)$
  - Send all the ciphertexts $\{c_{i \to j}\}_{j \in \mathcal{C}_0 \setminus \{i\}}$ to the server by adding the addressing information i, j as metadata
  - Store all the messages received and values generated in this round and move to the next round

  Server:
  - Wait for at least $N - D$ clients to respond (denote this set as $\mathcal{C}_1 \subseteq \mathcal{C}_0$)
  - Send to each client $i \in \mathcal{C}_1$, all ciphertexts encrypted for it: $\{c_{j \to i}\}_{j \in \mathcal{C}_1 \setminus \{i\}}$, and move to the next round
- **Round 2** (Recover the aggregate update)
  Client i:
  - Receive from the server the list of ciphertexts $\{c_{j \to i}\}_{j \in \mathcal{C}_1 \setminus \{i\}}$
  - Decrypt the ciphertext $(i' \,||\, j' \,||\, \mathsf{sh}_{j \to i}) \leftarrow \mathsf{Dec}(\mathsf{sk}_i, c_{j \to i})$ for each client $j \in \mathcal{C}_1 \setminus \{i\}$, and assert that $(i = i') \wedge (j = j')$
  - Compute the sum of shares over $\mathbb{F}_q$ as $\mathsf{sh}_i = \sum_{j \in \mathcal{C}_1} \mathsf{sh}_{j \to i}$
  - Send the share sh_i to the server

  Server:
  - Wait for at least $N - D$ clients to respond (denote this set as $\mathcal{C}_2 \subseteq \mathcal{C}_1$)
  - Run the reconstruction algorithm to obtain $\left\{\mathbf{z}^\ell, \perp\right\} \leftarrow \text{FastRecon}(\left\{(i, \mathsf{sh}_i^\ell)\right\}_{i \in \mathcal{C}_2})$ for $1 \leq \ell \leq \lceil L/S \rceil$, where sh_i^ℓ is the ℓ-th coefficient of sh_i; Denote $\mathbf{z} = [\mathbf{z}^1 \;\; \mathbf{z}^2 \;\; \cdots \;\; \mathbf{z}^{\lceil L/S \rceil}]$
  - If the reconstruction algorithm returns $\perp$ for any ℓ, then abort
  - Output the aggregate result $\mathbf{z}$

Chapter 4

Conclusion

Our work spans lands of distributed and federated learning, both of which are critical developments. In these realms, there has been work around security and robustness considerations, but there is an inevitable tradeoff between runtime and security/robustness. In this work, we propose SketchedRobustAgg and FastSecAgg as solutions to these problems.

First, we introduce SketchedRobustAgg, a novel algorithm that tackles both the issues of Byzantine workers and heavy communication complexity. We show that, as long as conditions in [9] are satisfied, we can achieve Byzantine robustness, while improving our communication complexity from $\mathcal{O}(nd)$ to $\mathcal{O}(ns)$ where $s \ll d$, and also improving our time complexity at the server side. On top of these gains, we show empirically that we can compress to less than 50% of original model parameters while preserving convergence rates to without using sketching.

Second, we introduce FastSecAgg with additional optimization for sketching. We showed that FastSecAgg can achieve lower communication and computation costs, compared to current state-of-the-art secure aggregation schemes, against an adaptive adversary and client dropouts. FastSecAgg uses FastShare, a novel multi-secret sharing scheme based on Fast Fourier Transform, which is information-theoretically secure and achieves a trade-off between the number of secrets, privacy threshold, and dropout tolerance. We additionally use Sketching to further reduce communication and computation costs, at a small accuracy tradeoff. Finally, we show that our algorithm performs well against benchmark Federated Learning datasets, and show empirically that we can compress to 50% of the original model parameters, while preserving convergence rates. We furthermore show that, if we increase q and scale our values higher, we can achieve higher accuracies.

As future work, we notice a large connection between the second line of work and differential privacy [21]. In particular, gradient clipping mechanism we propose has also been used to construct Differentially-Private Stochastic Gradient Descent (DP-SGD) [1], and by jittering our gradients with Gaussian noise, we will be able to achieve (ϵ, δ)-differential privacy.

Bibliography

[1] Martin Abadi et al. "Deep learning with differential privacy". In: *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 2016, pp. 308–318.

[2] Amirali Aghazadeh et al. "MISSION: Ultra Large-Scale Feature Selection using Count-Sketches". In: *Proceedings of the 35th International Conference on Machine Learning*. 2018, pp. 80–88.

[3] Dan Alistarh et al. "QSGD: Communication-efficient SGD via gradient quantization and encoding". In: *Advances in Neural Information Processing Systems*. 2017, pp. 1709–1720.

[4] Amos Beimel. "Secret-Sharing Schemes: A Survey". In: *Coding and Cryptology*. Ed. by Yeow Meng Chee et al. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 11–46.

[5] James Bell et al. *Secure Single-Server Aggregation with (Poly)Logarithmic Overhead*. Cryptology ePrint Archive, Report 2020/704. `https://eprint.iacr.org/2020/704`. 2020.

[6] Mihir Bellare and Chanathip Namprempre. "Authenticated Encryption: Relations among Notions and Analysis of the Generic Composition Paradigm". In: *Advances in Cryptology — ASIACRYPT 2000*. Ed. by Tatsuaki Okamoto. 2000, pp. 531–545.

[7] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. "Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation". In: *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*. STOC '88. Chicago, Illinois, USA: Association for Computing Machinery, 1988, pp. 1–10. ISBN: 0897912640. DOI: `10.1145/62212.62213`. URL: `https://doi.org/10.1145/62212.62213`.

[8] Jeremy Bernstein et al. "signSGD: Compressed optimisation for non-convex problems". In: *arXiv preprint arXiv:1802.04434* (2018).

[9] Peva Blanchard, Rachid Guerraoui, Julien Stainer, et al. "Machine learning with adversaries: Byzantine tolerant gradient descent". In: *Advances in Neural Information Processing Systems*. 2017, pp. 119–129.

[10] Carlo Blundo et al. "Multi-Secret Sharing Schemes". In: *Advances in Cryptology — CRYPTO '94*. Ed. by Yvo G. Desmedt. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 150–163. ISBN: 978-3-540-48658-9.

[11] Keith Bonawitz et al. "Practical Secure Aggregation for Privacy-Preserving Machine Learning". In: *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. CCS '17. 2017, pp. 1175–1191.

[12] Keith Bonawitz et al. "Towards Federated Learning at Scale: System Design". In: *Proceedings of the 2nd SysML Conference*. 2019.

[13] Léon Bottou. "Online learning and stochastic approximations". In: *On-line learning in neural networks* 17.9 (), p. 142.

[14] Martin Burkhart et al. "SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics". In: *Proceedings of the 19th USENIX Conference on Security*. USENIX Security'10. Washington, DC: USENIX Association, 2010, p. 15. ISBN: 8887666655554.

[15] V.R. Cadambe and A. Mazumdar. "Bounds on the Size of Locally Recoverable Codes". In: *Information Theory, IEEE Transactions on* 61.11 (Nov. 2015), pp. 5787–5794.

[16] Sebastian Caldas et al. "Expanding the Reach of Federated Learning by Reducing Client Resource Requirements". In: *ArXiv* abs/1812.07210 (2018).

[17] Sebastian Caldas et al. "LEAF: A benchmark for federated settings". In: *arXiv preprint arXiv:1812.01097* (2018).

[18] Nicholas Carlini et al. "The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks". In: *28th USENIX Security Symposium (USENIX Security 19)*. 2019, pp. 267–284.

[19] Lingjiao Chen et al. "Draco: Byzantine-resilient distributed training via redundant gradients". In: *arXiv preprint arXiv:1803.09877* (2018).

[20] Gregory Cohen et al. "EMNIST: Extending MNIST to handwritten letters". In: *2017 International Joint Conference on Neural Networks (IJCNN)*. IEEE. 2017, pp. 2921–2926.

[21] Cynthia Dwork. "Differential privacy: A survey of results". In: *International conference on theory and applications of models of computation*. Springer. 2008, pp. 1–19.

[22] Matthew Franklin and Moti Yung. "Communication Complexity of Secure Computation (Extended Abstract)". In: *Proceedings of the Twenty-Fourth Annual ACM Symposium on Theory of Computing*. 1992, pp. 699–710.

[23] 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*. 2015, pp. 1322–1333.

[24] Karan Ganju et al. "Property Inference Attacks on Fully Connected Neural Networks Using Permutation Invariant Representations". In: *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. 2018, pp. 619–633.

[25] P. Gopalan et al. "On the Locality of Codeword Symbols". In: *Information Theory, IEEE Transactions on* 58.11 (Nov. 2012), pp. 6925–6934.

[26] S. Goryczka and L. Xiong. "A Comprehensive Comparison of Multiparty Secure Additions with Differential Privacy". In: *IEEE Transactions on Dependable and Secure Computing* 14.5 (2017), pp. 463–477.

[27] Lie He, Sai Praneeth Karimireddy, and Martin Jaggi. *Secure Byzantine-Robust Machine Learning*. 2020. arXiv: `2006.04747 [cs.LG]`.

[28] Nikita Ivkin et al. "Communication-efficient distributed sgd with sketching". In: *Advances in Neural Information Processing Systems*. 2019, pp. 13144–13154.

[29] Nikita Ivkin et al. "Communication-efficient Distributed SGD with Sketching". In: *Advances in Neural Information Processing Systems 32*. 2019, pp. 13144–13154.

[30] J. Justesen. "Performance of Product Codes and Related Structures with Iterated Decoding". In: *IEEE Transactions on Communications* 59.2 (2011), pp. 407–415.

[31] Swanand Kadhe et al. "FastSecAgg: Scalable Secure Aggregation for Privacy-Preserving Federated Learning". In: *arXiv preprint arXiv:2009.11248* (2020).

[32] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Communications of the ACM* 60.6 (2017), pp. 84–90.

[33] Y. Lecun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[34] Yann LeCun et al. "Gradient-based learning applied to document recognition". In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324.

[35] Mu Li et al. "Scaling distributed machine learning with the parameter server". In: *11th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 14)*. 2014, pp. 583–598.

[36] F.J. MacWilliams and N.J.A. Sloane. *The Theory of Error-Correcting Codes*. 2nd. North-holland Publishing Company, 1978.

[37] Brendan McMahan et al. "Communication-Efficient Learning of Deep Networks from Decentralized Data". In: *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. Apr. 2017, pp. 1273–1282.

[38] Brendan McMahan et al. "Communication-efficient learning of deep networks from decentralized data". In: *Artificial Intelligence and Statistics*. PMLR. 2017, pp. 1273–1282.

[39] S. Pawar and K. Ramchandran. "FFAST: An Algorithm for Computing an Exactly k -Sparse DFT in $O(k \log k)$ Time". In: *IEEE Transactions on Information Theory* 64.1 (2018), pp. 429–450.

[40] Krishna Pillutla, Sham M. Kakade, and Zaid Harchaoui. *Robust Aggregation for Federated Learning*. 2019. arXiv: `1912.13445 [stat.ML]`.

[41] John M. Pollard. "The fast Fourier transform in a finite field". In: *Mathematics of Computation* 25 (1971), pp. 365–374.

[42] N. Prakash et al. "Optimal linear codes with a local-error-correction property". In: *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*. July 2012, pp. 2776–2780.

[43] Shashank Rajput et al. "DETOX: A redundancy-based framework for faster and more robust gradient aggregation". In: *Advances in Neural Information Processing Systems*. 2019, pp. 10320–10330.

[44] T. J. Richardson and R. L. Urbanke. "The capacity of low-density parity-check codes under message-passing decoding". In: *IEEE Transactions on Information Theory* 47.2 (2001), pp. 599–618.

[45] R. Shokri et al. "Membership Inference Attacks Against Machine Learning Models". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 3–18.

[46] D. Silva and F. R. Kschischang. "Universal Secure Network Coding via Rank-Metric Codes". In: *IEEE Transactions on Information Theory* 57.2 (Feb. 2011), pp. 1124–1135.

[47] David Silver et al. "Mastering the game of Go with deep neural networks and tree search". In: *nature* 529.7587 (2016), pp. 484–489.

[48] David Silver et al. "Mastering the game of go without human knowledge". In: *nature* 550.7676 (2017), pp. 354–359.

[49] Jinhyun So, Basak Guler, and A. Salman Avestimehr. *Byzantine-Resilient Secure Federated Learning*. 2020. arXiv: `2007.11115 [cs.CR]`.

[50] Jinhyun So, Basak Guler, and Amir Salman Avestimehr. "Turbo-Aggregate: Breaking the Quadratic Aggregation Barrier in Secure Federated Learning". In: *CoRR* abs/2002.04156 (2020). URL: `https://arxiv.org/abs/2002.04156`.

[51] Ryan Spring et al. "Compressing Gradient Optimizers via Count-Sketches". In: *Proceedings of the 36th International Conference on Machine Learning*. Sept. 2019, pp. 5946–5955.

[52] Suraj Srinivas and R Venkatesh Babu. "Data-free parameter pruning for deep neural networks". In: *arXiv preprint arXiv:1507.06149* (2015).

[53] Lili Su and Nitin H Vaidya. "Fault-tolerant multi-agent optimization: optimal iterative distributed algorithms". In: *Proceedings of the 2016 ACM symposium on principles of distributed computing*. 2016, pp. 425–434.

[54] I. Tamo and A. Barg. "A Family of Optimal Locally Recoverable Codes". In: *Information Theory, IEEE Transactions on* 60.8 (Aug. 2014), pp. 4661–4676.

[55] Stacey Truex et al. "A Hybrid Approach to Privacy-Preserving Federated Learning". In: *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. 2019, pp. 1–11.

[56] David P. Woodruff. "Sketching As a Tool for Numerical Linear Algebra". In: *Found. Trends Theor. Comput. Sci.* 10 (2014), pp. 1–157.

[57] Cong Xie, Sanmi Koyejo, and Indranil Gupta. "Zeno: Distributed stochastic gradient descent with suspicion-based fault-tolerance". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6893–6901.

[58] Runhua Xu et al. "HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning". In: *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. AISec'19. London, United Kingdom: Association for Computing Machinery, 2019, pp. 13–23. ISBN: 9781450368339. DOI: 10.1145/3338501.3357371. URL: https://doi.org/10.1145/3338501.3357371.

[59] Dong Yin et al. "Byzantine-robust distributed learning: Towards optimal statistical rates". In: *arXiv preprint arXiv:1803.01498* (2018).

Appendix A

Finite Field Fourier Transform

Let q be a power of a prime, and consider a finite field $\mathbb{F}_q$ of order q. Let N be a positive integer such that N divides $(q-1)$ and ω be a primitive N-th root of unity in $\mathbb{F}_q$.

The discrete Fourier transform (DFT) of length N generated by ω is a mapping from $\mathbb{F}_q^N$ to $\mathbb{F}_q^N$. Let $x = [x_0\, x_1\, \ldots x_{N-1}]$ be a vector over $\mathbb{F}_q$. Then, the DFT of x generated by ω, denoted as $DFT_\omega(x)$, is the vector over $\mathbb{F}_q$, $X = [X_0\, X_1\, \ldots X_{N-1}]$, given by

$$X_j = \sum_{i=0}^{N-1} \omega^{ij} x_i, \quad j = 0, 1, \ldots, N-1. \tag{A.1}$$

The inverse DFT (IDFT), denoted as $IDFT_\omega(X)$, is given by

$$x_i = \frac{1}{N} \sum_{j=0}^{N-1} \omega^{-ij} X_j, \quad i = 0, 1, \ldots, N-1, \tag{A.2}$$

where $\frac{1}{N}$ denotes the reciprocal of the sum of N ones in the field $\mathbb{F}_q$. We refer to x as the "time-domain signal" and index i as time, and X as the "frequency-domain signal" or the "spectrum" and j as frequency.

If a signal is subsampled in the time-domain, its frequency components mix together, i.e., alias, in a pattern that depends on the sampling procedure. In particular, let n be a positive integer that divides N. Let $x^{(\downarrow n)}$ denote the subsampled version of x with period n, i.e., $x^{(\downarrow n)} = \{x_{ni} : 0 \le i \le N/n - 1\}$. Then, (N/n)-length DFT of $x^{(\downarrow n)}$, $X^{(\downarrow n)}$, (generated by ω^n) is related to the N-length DFT of x, X, (generated by ω) as

$$X_j^{(\downarrow n)} = \frac{1}{n} \sum_{\substack{i=0 \\ i \mod (N/n) = j}}^{N-1} X_i, \quad j = 0, 1, \ldots, \frac{N}{n} - 1, \tag{A.3}$$

where $1/n$ is the reciprocal of the sum of n ones in $\mathbb{F}_q$.

A circular shift in the time-domain results in a phase shift in the frequency-domain. Given a signal x, consider its circularly shifted version $x^{(1)}$ defined as $x_i^{(1)} = x_{(i+1) \mod N}$. Then, the DFTs of $x^{(1)}$ and x (both generated by ω) are related as $X_j^{(1)} = \omega^{-j} X_j$. In general, a circular shift of t results in

$$X_j^{(t)} = \omega^{-tj} X_j, \quad j = 0, 1, \ldots, N-1. \tag{A.4}$$

For more details, we invite the readers to reference [41].

Appendix B

Analysis of FastShare

We first focus on the correctness, i.e., dropout tolerance. To prove that FastShare has a dropout tolerance of D, we need to show that FastRecon can recover the secrets from a random subset of $N-D$ shares. This is equivalent to showing that the iterative peeling decoder of FastRecon can recover all the shares from a random subset of $N-D$ shares. Note that the shares generated by FastShare form a codeword of a product code. From the *density evolution* analysis of the iterative peeling decoder of product codes in [30, 39], it follows that, when $D \leq (1-(1-\delta_0)(1-\delta_1))\frac{N}{2}$, the decoder recovers all the shares from a random subset of $N-D$ shares with probability at least $1-\text{poly}N$. Therefore, FastShare has the dropout tolerance of $D = (1-(1-\delta_0)(1-\delta_1))\frac{N}{2}$.

To prove the privacy threshold of T, we consider the information-theoretic equivalent of the security definition [10]. In particular, we need to show the following: for any $\mathcal{P} \subset C$ such that $|\mathcal{P}| \leq T$, it holds that $H\left(\mathbf{s} \mid \{[\mathbf{s}]_i\}_{i\in\mathcal{P}}\right) = H\left(\mathbf{s}\right)$, where H denotes the Shannon entropy. For simplicity, let us denote $[\mathbf{s}]_i = \mathbf{X}_i$ for all i. In the remainder of the proof, we denote random variables by boldface letters.

First, we show that the information-theoretic security condition is equivalent to a specific linear algebraic condition. To this end, we first observe that the vector of N shares can be written as,

$$\mathbf{X} = G\begin{bmatrix}\mathbf{s}\\ \mathbf{m}\end{bmatrix}, \tag{B.1}$$

where $\mathbf{s} \in \mathbb{F}_q^{\ell}$ is the vector of secrets, $\mathbf{m} \in \mathbb{F}_q^{K-\ell}$ is a vector with each element chosen independently and uniformly from $\mathbb{F}_q$, and G is a particular submatrix of the DFT matrix whose formal definition we defer later on. We then show that information theoretic security is equivalent to a particular linear algebraic condition on submatrices of G. To prove this condition we leverage the fact that G is derived from a DFT matrix and consider an alternate representation based on the Chinese Remainder theorem (CRT). We furnish more details while formally discussing the lemmas.

Recall that n_0 and n_1 are co-prime. By the CRT, we may conclude that any number $j \in \{0, \cdots, N-1\}$ can be uniquely represented in a 2D-grid as a tuple (a, b) where $a = j \mod n_0$ and $b = j \mod n_1$.

First, define the set of indices

$$\begin{aligned}\mathcal{S} = \{(p_0, p_1) : \\ &\delta_0 n_0 + \alpha(1-\delta_0)n_0 \le p_0 \le n_0 - 1, \\ &\delta_1 n_1 + \beta(1-\delta_1)n_1 + 1 \le p_1 \le n_1 - \beta n_1(1-\delta_1)\}. \end{aligned} \tag{B.2}$$

Similarly, define

$$\mathcal{Z}_0 = \{(p_0, p_1) : 0 \le p_0 \le \delta_0 n_0 - 1,\ 0 \le p_1 \le n_1 - 1\}, \tag{B.3}$$

$$\mathcal{Z}_1 = \{(p_0, p_1) : 0 \le p_0 \le n_0 - 1,\ 0 \le p_1 \le \delta_1 n_1 - 1\}. \tag{B.4}$$

For a pictorial representation of these indices, refer to Fig. 3.2. These sets correspond to points in the grid, and by the CRT map back to integers in the range $\{0, \cdots, N-1\}$.

In Chapter 3.2.2, the number of secrets ℓ, is chosen to be equal to $(1-\alpha)(1-2\beta)(1-\delta_0)(1-\delta_1)n_0 n_1$. By design this coincides with the size of $\mathcal{S}$.

With these definition, we next describe the construction of the matrix G is obtained by starting with the $N \times N$ DFT matrix, removing the columns corresponding to $\mathcal{Z}_0 \cup \mathcal{Z}_1$, and permuting the remaining columns so that the columns corresponding to $\mathcal{S}$ are ordered as the first ℓ columns in $\mathcal{S}$ (in arbitrary sequence).

Having defined the matrix G, in the following lemma we show that information theoretic security can be guaranteed by a particular rank condition satisfied by submatrices of G. In particular, In addition, define $K = N - |\mathcal{Z}_0 \cup \mathcal{Z}_1|$.

The proof is similar to Lemma 6 in [46], and thus omitted.

Lemma B.0.1. *Let $\mathbf{s} \in \mathbb{F}_q^{\ell}$, $\mathbf{m} \in \mathbb{F}_q^{k-\ell}$, and $\mathbf{X} = G[\mathbf{s}\ \mathbf{m}]^T$ be random variables representing the secrets, random masks, and shares, respectively. Let $\mathbf{X}_{\mathcal{P}}$ be an arbitrary set of shares corresponding to the indices in $\mathcal{P} \subset \{1, 2, \ldots, N\}$. Let $G_{\mathcal{P}}$ denote the sub-matrix of G corresponding to the rows indexed by $\mathcal{P}$. Let $G_{\mathcal{P}} = [G_1\ G_2]$, where G_1 consists of the first ℓ columns of $G_{\mathcal{P}}$ and G_2 consists of the last $K - \ell$ columns of $G_{\mathcal{P}}$. If $\mathbf{m}$ is uniform over $\mathbb{F}_q^{K-\ell}$, then it holds that*

$$H(\mathbf{s}) - H(\mathbf{s} \mid \mathbf{X}_{\mathcal{P}}) \le \mathit{rank}(G_{\mathcal{P}}) - \mathit{rank}(G_2). \tag{B.5}$$

Now, to prove the information-theoretic security, it suffices to prove that for any $\mathcal{P}$ of size at most $T = \alpha\beta(1-\delta_0)(1-\delta_1)N$, $\mathrm{rank}(G_2) = \mathrm{rank}(G_{\mathcal{P}})$. We prove this by showing that, for any $\mathcal{D}$ with $|\mathcal{P}| \le T$, the columns of G_1 lie in the span of the columns of G_2. Note that columns of G_1 are the columns of the DFT matrix indexed by $\mathcal{S}$.

Proof of rank$(G_{\mathcal{P}})$ = **rank**(G_2): Assume that $\{\omega_\partial : \partial \in \mathcal{P}\}$ are the set of primitive elements generating the rows of $G_{\mathcal{P}}$. For the rest of the proof we are guided by the grid representation of fig:2D-secrets - the Chinese remainder theorem (CRT) furnishes a representation of the columns of $G_{\mathcal{P}}$, indexed by $\{0, \cdots, n-1\}$ as points on a 2D-grid, $\{(p_0, p_1) : p_0 \in \{0, \cdots, n_0 - 1\}, p_1 \in \{0, \cdots, n_1 - 1\}\}$.

Notation: In the grid representation, consider the set of points $\mathcal{T}$ in fig:2D-secrets. Formally,

$$\begin{aligned}\mathcal{T} = \{(p_0, p_1) : \delta_0 n_0 \leq\ & p_0\ \leq \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1, \\ & n_1 - \beta(1-\delta_1)n_1 \leq\ p_1\ \leq n_1 - 1\}. \end{aligned} \tag{B.6}$$

Define $G_2(\mathcal{T})$ as the matrix G_2 only populated by the columns whose indices belong to $\mathcal{T}$. Henceforth, we use the terminology "points" to refer to a column of $G_{\mathcal{P}}$ in its representation as a tuple in the 2D-grid. Moreover we use the terminology "span" and "rank" of the points to indicate the span and rank of the corresponding columns. Consider a column index $p \in \{0, \cdots, n-1\}$ and $\Delta \in \{0, \cdots, n-1\}$ where $p \mapsto (p_0, p_1)$ and $\Delta \mapsto (\Delta_0, \Delta_1)$ under the CRT bijection. Then, the notation $(p_0, p_1) + (\Delta_0, \Delta_1)$ returns the point $((p_0 + \Delta_0) \mod n_0, (p_1 + \Delta_1) \mod n_1)$.

Since $G_{\mathcal{P}}$ is derived from a DFT matrix, this in fact corresponds to multiplying the column corresponding to p by the matrix $\mathrm{diag}((\omega_\partial^\Delta : \partial \in \mathcal{P}))$. The notations (i) $p+\Delta$, (ii) $p+(\Delta_0, \Delta_1)$ are defined analogously. We also use the terminology "shift p horizontally by Δ_0" and "shift p vertically by Δ_1" to respectively denote $p + (\Delta_0, 0)$ and $p + (0, \Delta_1)$. Given a set of points $\mathcal{P}$, we overload notation and use $\mathcal{P} + \Delta$ as the set of points $\{p + \Delta : p \in \mathcal{P}\}$.

Observe that $\mathcal{P} + \Delta$ corresponds to multiplying the columns indexed by $\mathcal{P}$ by a full-rank matrix. Then, using the structure of $G_{\mathcal{P}}$ inherited from the DFT (Vandermonde) matrix structure, we get the following result immediately.

Lemma B.0.2. *If $p \in$ span$(\mathcal{P})$ for some set of points $\mathcal{P}$, then for any $\Delta \in \{0, \cdots, n-1\}$, $p + \Delta \in$ span$(\mathcal{P} + \Delta)$.*

Proof. If $p \in \mathrm{span}(\mathcal{P})$ this implies that there exists weights $\{\alpha_q : q \in \mathcal{P}\}$ such that for each $\partial \in \mathcal{P}$, $\omega_\partial^p = \sum_{q \in \mathcal{P}} \alpha_q \omega_\partial^q$. Fixing any $\partial \in \mathcal{P}$, and multiplying both sides by ω_∂^Δ,

$$\omega_\partial^{p+\Delta} = \sum\nolimits_{q \in \mathcal{P}} \alpha_q \omega_\partial^{q+\Delta} = \sum\nolimits_{q \in \mathcal{P}+\Delta} \alpha'_q \omega_\partial^q. \tag{B.7}$$

Since this is true for each $\partial \in \mathcal{P}$ the proof follows immediately. □

By construction of the set, observe that the number of points in $\mathcal{T}$ equals $\alpha\beta(1-\delta_0)(1-\delta_1)n_0n_1$. Moreover, recalling the choice of the privacy threshold T in Chapter 3.2.2, observe that $|\mathcal{T}| = T$. By this fact, we have that $\mathrm{rank}(G_{\mathcal{P}}) \leq T$. This implies two possibilities:

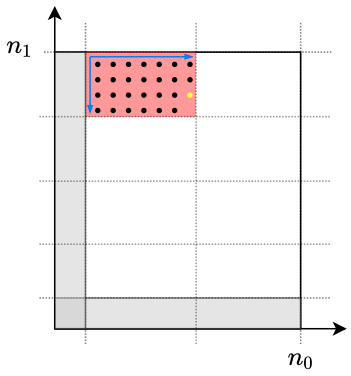


Figure B.1: Enumerating the columns in the bottom-right sequence until the first column (yellow) is reached which lies in the span of the previous columns

Case I. If $\text{rank}(G_2(\mathcal{T})) = \text{rank}(G_{\mathcal{P}})$, then the remaining columns in G_1 must lie in the span of $G_2(\mathcal{T})$, and the proof concludes.

Case II. If not, then the columns of $G_2(\mathcal{T})$ must have at least one column dependent on the others.

We can identify one such column by the following procedure: starting at the top left corner of $\mathcal{T}$ in the grid, sequentially collect the points in $\mathcal{T}$ in a top-to-bottom, left-to-right sequence. Stop at the first point $y = (y_0, y_1)$ which lies in the span of the previously collected points (by assumption, such a point must exist as we are in Case II). Chapter B.1 illustrates this procedure where we collect the black points sequentially until the yellow point is reached which is the first point that lies in the span of the previously collected points. Let $\mathcal{B}$ be defined as the set of black points which are enumerated until y is found. By definition, this implies that there exist (not necessarily unique) weights $\{\alpha_b\}_{b\in\mathcal{B}}$ such that,

$$\omega^y = \sum\nolimits_{b\in\mathcal{B}} \alpha_b \omega^b \tag{B.8}$$

This implies that $y - (0,1) \in \text{span}(\mathcal{B} - (0,1))$. Define $\mathcal{Y}$ as the set of points,

$$\mathcal{Y} = \{(y_0, \theta) : y_1 - n_1(1-\delta_1)(1-\beta) + 1 \le \theta \le y_1\}. \tag{B.9}$$

Note that the set of points $\mathcal{Y}$ implicitly depends on the location of point y. Next, we define $\mathcal{L}$ and $\mathcal{C}$ to be the set of points,

$$\mathcal{L} = \cup_{k=0}^{n_0-y_0-1} (\mathcal{Y} + (k, 0)) \tag{B.10}$$

$$\mathcal{C} = \{(p_0, p_1) : n_0\delta_0 \le p_0 \le y_0, n_1\delta_1 \le p_1 \le n_1 - 1\} \setminus \mathcal{Y}. \tag{B.11}$$

In addition, we define $\mathcal{Q}$ to be the set of points,

$$\begin{aligned}\mathcal{Q} =&\{(p_0, p_1) : \\ &n_0\delta_0 \leq p_0 \leq n_0 - 1, n_1\delta_1 \leq p_1 \leq n_1 - 1\} \setminus \mathcal{L}.\end{aligned} \tag{B.12}$$

Pictorially these points are represented in Figs B.2 and B.3. $\mathcal{Y} \subseteq \text{span}(\mathcal{C})$.

Proof. We prove this result by induction sequentially iterating over all the points from top to bottom in $\mathcal{Y}$. Clearly $\mathcal{B} \subseteq \mathcal{C}$, therefore the induction holds for the first point in $\mathcal{Y}$, which is $y = (y_0, y_1)$.

Suppose for some $k \geq 1$ that for each $t \leq k - 1$ the point $y - (0, t)$ lies in $\text{span}(\mathcal{C})$. Then we show that $y - (0, k) \in \text{span}(\mathcal{C})$ as long as $k \geq y_1 - n_1(1 - \delta_1)(1 - \beta) + 1$. Indeed, observe first that,

$$y - (0, k) \in \text{span}(\mathcal{B} - (0, k)). \tag{B.13}$$

The key observation is that $\mathcal{B} - (0, k)$ is always $\subseteq \mathcal{C} \cup \{y - (0, t) : 0 \leq t \leq k - 1\}$ (unless $k > y_1 - n_1(1 - \delta_1)(1 - \beta) + 1$, in which case $\mathcal{B} - (0, k)$ shifts far enough down that it includes points in $\mathcal{Z}_1$ - this has null intersection with $\mathcal{C}$ and $\mathcal{Y}$ so the assertion is clearly false). For a pictorial presentation of this fact, refer to Fig. B.4. By the induction hypothesis, for each $t \leq k - 1$, $y - (0, t) \in \text{span}(\mathcal{C})$. Therefore, $\text{span}(\mathcal{B} - (0, k)) \subseteq \text{span}(\mathcal{C})$. Plugging into eq:1203102 completes the proof of the claim. □

$\mathcal{L} \subseteq \text{span}(\mathcal{Q})$.

Proof. Recall that $\mathcal{L} = \cup_{k=0}^{n_0 - y_0 - 1}(\mathcal{Y} + (k, 0))$. We follow a similar proof by induction strategy as Chapter B to result in the assertion. In particular, for $k = 0$ the statement follows directly from the fact that $\mathcal{C} \subseteq \mathcal{Q}$ and using Chapter B to claim that $\mathcal{Y} \subseteq \text{span}(\mathcal{C})$. Assuming the induction hypothesis that for all $0 \leq t \leq k - 1$, $\mathcal{Y} + (t, 0) \subseteq \text{span}(\mathcal{Q})$ we show that $\mathcal{Y} + (k, 0) \subseteq \text{span}(\mathcal{Q})$ as long as $k \leq n_0 - y_0 - 1$. In particular, observe that from Chapter B,

$$\mathcal{Y} + (0, k) \subseteq \text{span}(\mathcal{C} + (0, k)). \tag{B.14}$$

Next observe that unless $k > n_0 - y_0 - 1$,

$$\mathcal{C} + (k, 0) \subseteq \mathcal{Q} \cup \{\mathcal{Y} + (t, 0) : 0 \leq t \leq k - 1\} \tag{B.15}$$

if $k > n_0 - y_0 - 1$, then $\mathcal{C} + (k, 0)$ shifts far enough that it wraps around and include points in $\mathcal{Z}_0$ - this has null intersection with $\mathcal{Q}$ and $\mathcal{Y} + (t, 0)$ for any $t \leq n_0 - y_0 - 1$ and the assertion becomes false. A pictorial representation of this fact is provided in Fig. B.5. By the induction hypothesis for each $0 \leq t \leq k - 1$, $\mathcal{Y} + (t, 0) \subseteq \text{span}(\mathcal{Q})$. Combining this fact with eq:12312111,eq:10211 implies that $\mathcal{Y} + (0, k) \subseteq \text{span}(\mathcal{Q})$ and completes the induction step for k. □

Recall from def:L that $\mathcal{L}$ is defined as the set of points, $\cup_{k=0}^{n_0-y_0-1}(\mathcal{Y}+(k,0))$. More explicitly, noting that the set of points $\mathcal{Y}$ is defined as $\{(y_0,\theta) : y_1 - n_1(1-\delta_1)(1-\beta)+1 \le \theta \le y_1\}$,

$$\begin{aligned}\mathcal{L} = \{(p_0,p_1) :\ & y_0 \le p_0 \le n_0 - 1, \\ & y_1 - n_1(1-\delta_1)(1-\beta)+1 \le p_1 \le y_1\}.\end{aligned} \tag{B.16}$$

The set of points $\mathcal{S} \subseteq \mathcal{L}$ irrespective of the location of the point $y = (y_0, y_1)$ (note that the set of points $\mathcal{L}$ is a function of y_0 and y_1).

Proof. Before furnishing the details of the proof, note that a pictorial representation of this statement is provided in Fig. B.5. First recall that $y = (y_0, y_1)$ is a point in the set $\mathcal{T}$. Therefore, $\delta_0 n_0 \le y_0 \le \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1$ and $n_1 - \beta(1-\delta_1)n_1 \le y_1 \le n_1 - 1$. In particular, this means that for any y,

$$\left\{\begin{array}{l}(p_0,p_1) : \\ \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1 \le p_0 \le n_0 - 1, \\ y_1 - n_1(1-\delta_1)(1-\beta)+1 \le p_1 \le y_1\end{array}\right\} \subseteq \mathcal{L} \tag{B.17}$$

Furthermore, we see that (i) $y_1 \ge n_1 - \beta(1-\delta_1)n_1$, and (ii) $y_1 - n_1(1-\delta_1)(1-\beta)+1 \le n_1 - n_1(1-\delta_1)(1-\beta) = n_1\delta_1 + \beta(1-\delta_1)n_1$. Therefore, we have that,

$$\left\{\begin{array}{l}(p_0,p_1) : \\ \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1 \le p_0 \le n_0 - 1, \\ n_1\delta_1 + \beta(1-\delta_1)n_1 \le p_1 \le n_1 - \beta(1-\delta_1)n_1\end{array}\right\} \subseteq \mathcal{L} \tag{B.18}$$

The LHS is exactly the definition of $\mathcal{S}$ in def:S. □

From Claims B and B and the definition of $\mathcal{L}$ in def:L, we see that $\mathcal{S} \subseteq \text{span}(\mathcal{Q})$. In particular this implies that the each column indexed by points in $\mathcal{S}$ can be expressed as a linear combination of some set of remaining columns that do not belong to $\mathcal{Z}_0$ or $\mathcal{Z}_1$. This implies that for any choice of $\mathcal{P}$ (note that we do not specify a particular choice of $\mathcal{P}$ in the proof as long as it has size $\le T$) that $\text{rank}(G_2) = \text{rank}(G_{\mathcal{P}})$.

Moreover, we do not specify the particular choice of $\mathcal{P}$ in the proof, so it holds for all sets of size $\le T$.

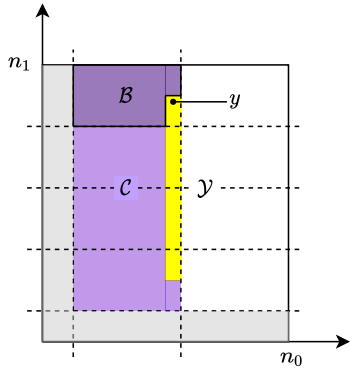


Figure B.2: Definition of $\mathcal{B}, \mathcal{Y}$ and $\mathcal{C}$

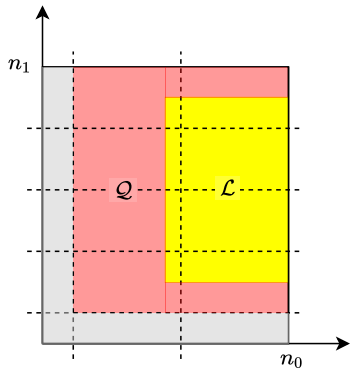


Figure B.3: Definition of $\mathcal{Q}$ and $\mathcal{L}$

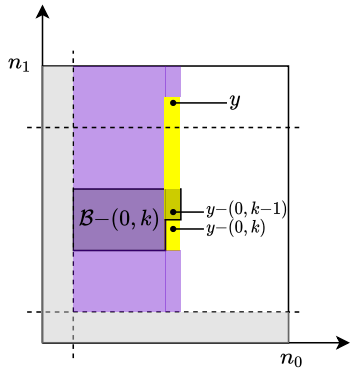


Figure B.4: Showing that $\mathcal{B} - (0, k)$ is a subset of the intersection of $\mathcal{C}$ and $\{y - (0, t) : 0 \leq t \leq k - 1\}$.

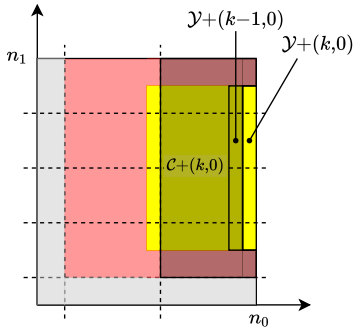


Figure B.5: Showing that $\mathcal{C} + (k, 0)$ is a subset of the intersection of $\mathcal{Q}$ and $\bigcup_{t=0}^{k-1}(\mathcal{Y} + (t, 0))$.

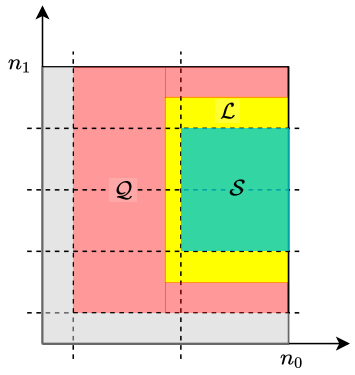


Figure B.6: The $\mathcal{S}$ is indeed a subset of $\mathcal{L}$. For the particular choice of y this figure shows that the inclusion is true. More generally we in construct $\mathcal{S}$ as the intersection of $\mathcal{L}$ over all the possible locations of $y \in \mathcal{T}$.

Appendix C

Correctness of FastSecAgg

Correctness essentially follows from the correctness of key agreement and authenticated encryption protocols together with the linearity and D-dropout tolerance of FastShare.

Specifically, using the correctness of key agreement and authenticated encryption, it is straightforward to show that, in Round 2, each client $i \in \mathcal{C}_1$ computes and sends to the server the sum of shares it receives in Round 1 as follows

$$\mathsf{sh}_i = \left(\sum_{j \in \mathcal{C}_1} [\mathbf{u}_j^1]_i \,||\, \cdots \,||\, \sum_{j \in \mathcal{C}_1} [\mathbf{u}_j^{\lceil L/S \rceil}]_i \right). \tag{C.1}$$

The linearity property of FastShare ensures that the sum of shares is a share of the sum of secret vectors (i.e., client inputs). In other words, by linearity of FastShare, it holds that

$$\mathsf{sh}_i = \left(\left[\sum_{j \in \mathcal{C}_1} \mathbf{u}_j^1 \right]_i \,||\, \cdots \,||\, \left[\sum_{j \in \mathcal{C}_1} \mathbf{u}_j^{\lceil L/S \rceil} \right]_i \right). \tag{C.2}$$

Recall that sh_i^ℓ denotes the ℓ-th coefficient of sh_i. Therefore, from (C.2), it holds, for $1 \le \ell \le \lceil L/S \rceil$, that

$$\mathsf{sh}_i^\ell = \left[\sum_{j \in \mathcal{C}_1} \mathbf{u}_j^\ell \right]_i. \tag{C.3}$$

Let $\mathcal{C}_2$ be a random subset of at least $N - D$ clients that survive in Round 2, i.e., clients in $\mathcal{C}_2$ send their sum-shares to the server. Now, from (C.3) and the D-dropout tolerance of FastShare, it holds that FastRecon $\left(\{(i, \mathsf{sh}_i^\ell)\}_{i \in \mathcal{C}_2} \right) = \sum_{j \in \mathcal{C}_1} \mathbf{u}_j^\ell$ with probability at least $1 - 1/\text{poly}\, N$ for all $1 \le \ell \le \lceil L/S \rceil$. Note that we do not need to use a union bound here because if FastRecon succeeds (i.e., does not output $\perp$) for $\ell = 1$, then it succeeds for each $1 \le \ell \le \lceil L/S \rceil$, since the locations of missing indices (among N) of shares for every $1 \le \ell \le \lceil L/S \rceil$ are the same.

Hence, we get

$$[\mathbf{z}^1 \ \mathbf{z}^2 \ \cdots \ \mathbf{Z}^{\lceil L/S \rceil}] = \left[\sum_{j \in \mathcal{C}_1} \mathbf{u}_j^1 \ \sum_{j \in \mathcal{C}_1} \mathbf{u}_j^2 \ \cdots \ \sum_{j \in \mathcal{C}_1} \mathbf{u}_j^{\lceil L/S \rceil} \right]$$

with probability at least $1 - 1/\text{poly}\, N$. In other words, $\mathbf{z} = \sum_{i \in \mathcal{C}_1} \mathbf{u}_i$ with probability at least $1 - 1/\text{poly}\, N$.

Appendix D

Security of FastSecAgg

In FastSecAgg, each client i first partitions their input $\mathbf{u}_i$ to $\lceil L/S \rceil$ vectors $\mathbf{u}_i^1$, $\mathbf{u}_i^2$, ..., $\mathbf{u}_i^{\lceil L/S \rceil}$, each of length at most S, and computes shares for each $\mathbf{u}_i^\ell$, $1 \leq \ell \leq \lceil L/S \rceil$. In other words, we have

$$\left\{ \left(j, \left[\mathbf{u}_i^\ell \right]_j \right) \right\}_{j \in \mathcal{C}} \leftarrow \textsc{FastShare}(\mathbf{u}_i^\ell, \mathcal{C}), \tag{D.1}$$

where independent private randomness is used for each $1 \leq \ell \leq \lceil L/S \rceil$. Let us denote the set of shares that client i generates for client j as $[\mathbf{u}_i]_j$, i.e., we have

$$[\mathbf{u}_i]_j = \left\{ \left[\mathbf{u}_i^1 \right]_j, \left[\mathbf{u}_i^2 \right]_j, \ldots, [\mathbf{u}_i^{\lceil L/S \rceil}]_j \right\}. \tag{D.2}$$

It is straightforward to show that for any set of up to T clients $\mathcal{P} \subset \mathcal{C}$, the shares $\{[\mathbf{u}_i]_j\}_{j \in \mathcal{P}}$ reveal no information about $\mathbf{u}_i$.

Lemma D.0.1. *For every $\mathbf{u}_i, \mathbf{v}_i \in \mathbb{F}_q^L$, for any $\mathcal{P} \subset \mathcal{C}$ such that $|\mathcal{P}| \leq T$, the distribution of $\{[\mathbf{u}_i]_j\}_{j \in \mathcal{P}}$ is identical to that of $\{[\mathbf{v}_i]_j\}_{j \in \mathcal{P}}$.*

Proof. Here we treat $\mathbf{u}_i$ to be a random variable (with arbitrary distribution), and $\{[\mathbf{u}_i]_j\}_{j \in \mathcal{P}}$ to be a conditional random variable given a realization of $\mathbf{u}_i$. By slightly abusing the notation for simplicity, denote $\mathbf{u}_i$ as a set $\mathbf{u}_i = \{\mathbf{u}_i^1, \mathbf{u}_i^2, \ldots, \mathbf{u}_i^{\lceil L/S \rceil}\}$ (instead of a vector). Further, for simplicity, define

$$\left[\mathbf{u}_i^\ell \right]_{\mathcal{P}} = \{[\mathbf{u}_i]_j\}_{j \in \mathcal{P}}; \quad \left[\mathbf{u}_i^\ell \right]_{\mathcal{P}} = \left\{ \left[\mathbf{u}_i^\ell \right]_j \right\}_{j \in \mathcal{P}}, \ 1 \leq \ell \leq \lceil L/S \rceil.$$

Now, observe that, given $\mathbf{u}_i^\ell$, the distribution of $\left[\mathbf{u}_i^\ell \right]_{\mathcal{P}}$ is conditionally independent of $\mathbf{u}_i \setminus \{\mathbf{u}_i^\ell\}$ and $[\mathbf{u}_i]_{\mathcal{P}} \setminus \left\{ \left[\mathbf{u}_i^\ell \right]_{\mathcal{P}} \right\}$. This is because, for the ℓ-th instantiation, $1 \leq \ell \leq \lceil L/S \rceil$, FastShare takes only $\mathbf{u}_i^\ell$ as its input and uses independent private randomness. Therefore, the distribution

of $[\mathbf{u}_i]_{\mathcal{P}}$ factors into the product of the distributions of $\left[\mathbf{u}_i^{\ell}\right]_{\mathcal{P}}$. The proof then follows by applying the T-privacy property for each instantiation of FastShare. □

We prove Theorem 3.4.2 by a standard hybrid argument. We will present a sequence of hybrids starting that start from the real execution and transition to the simulated execution where each two consecutive hybrids are computationally indistinguishable.

Hybrid$_0$: This random variable is $\mathsf{REAL}_{\mathcal{M}}^{\mathcal{C},T,\lambda}(\mathbf{u}_{\mathcal{C}}, \mathcal{C}_0, \mathcal{C}_1, \mathcal{C}_2)$, the joint view of the parites $\mathcal{M}$ in the real execution of the protocol.

Hybrid$_1$: In this hybrid, we change the behavior of honest clients in $\mathcal{C}_1 \setminus \mathcal{M}$ so that instead of using KA.agree($\mathsf{sk}_i, \mathsf{pk}_j$) to encrypt and decrypt messages, we run the key agreement simulator $\mathsf{Sim}_{KA}(s_{i,j}, \mathsf{pk}_j)$, where $s_{i,j}$ is chosen uniformly at random. The security of the key agreement protocol guarantees that this hybrid is indistinguishable from the previous one.

Hybrid$_2$: In this hybrid, for every client $i \in \mathcal{C}_1 \setminus \mathcal{M}$, we replace the shares of $\mathbf{u}_i$ sent to other honest clients in Round 1 with zeros, which the adversary observes encrypted as $c_{i\to j}$. Since only the contents of the ciphertexts are changed, the IND-CPA security of the encryption scheme guarantees that this hybrid is indistinguishable from the previous one.

Hybrid$_3$: In this hybrid, for every client $i \in \mathcal{C}_1 \setminus \mathcal{M}$, we replace the shares of $\mathbf{u}_i$ sent to the corrupt clients in $\mathcal{M}$ in Round 1 with shares of $\mathbf{v}_i$, which are chosen as follows depending on $\mathbf{z}$. If $\mathbf{z} = \perp$, then $\{\mathbf{v}_i\}_{i\in\mathcal{C}_1\setminus\mathcal{M}}$ are chosen uniformly at random. Otherwise, $\{\mathbf{v}_i\}_{i\in\mathcal{C}_1\setminus\mathcal{M}}$ are chosen uniformly at random subject to $\sum_{i\in\mathcal{C}_1\setminus\mathcal{M}} \mathbf{v}_i = \mathbf{z}\left(= \sum_{i\in\mathcal{C}_1\setminus\mathcal{M}} \mathbf{u}_i\right)$. The joint view of corrupt parties contains only $|\mathcal{M}| \le T$ shares of each $\mathbf{v}_i$. From Lemma D.0.1, it follows that this hybrid is identically distributed to the previous one.

If $\mathbf{z} = \perp$, then we do not need to consider the further hybrids, and let SIM is defined to sample from **Hybrid**$_3$. This distribution can be computed from the inputs $\mathbf{z}$, $\mathcal{C}_0$, and $\mathcal{C}_1$. In the following hybrids, we assume $\mathbf{z} \neq \perp$.

Hybrid$_4$: Partition $\mathbf{z}$ into $\lceil L/S \rceil$ vectors, $\mathbf{z}^1$, $\mathbf{z}^2$, …, $\mathbf{z}^{\lceil L/S \rceil}$, each of length at most S. In this hybrid, for every client $i \in \mathcal{C}_2 \setminus \mathcal{M}$ and each $1 \le \ell \le \lceil L/S \rceil$, we replace the share sh_i^{ℓ} with the i-th share of $\mathbf{z}^{\ell} + \sum_{j\in\mathcal{M}} \mathbf{u}_j^{\ell}$, i.e., $\left[\mathbf{z}^{\ell} + \sum_{j\in\mathcal{M}} \mathbf{u}_j^{\ell}\right]_i$. Since $\mathbf{z} = \sum_{i\in\mathcal{C}_1\setminus\mathcal{M}} \mathbf{u}_i$, from (C.2) and (C.3), it follows that the distribution of $\left\{\left[\mathbf{z} + \sum_{j\in\mathcal{M}} \mathbf{u}_j\right]_i\right\}_{i\in\mathcal{C}_2}$ is identical to that of $\{\mathsf{sh}_i\}_{i\in\mathcal{C}_2}$. Therefore, this hybrid is identically distributed to the previous one.

We define a PPT simulator SIM to sample from the distribution described in the last hybrid. This distribution can be computed from the inputs $\mathbf{z}$, $\mathbf{u}_{\mathcal{M}}$, $\mathcal{C}_0$, $\mathcal{C}_1$, and $\mathcal{C}_2$. The argument above proves that the output of the simulator is computationally indistinguishable from the output of REAL, which concludes the proof.