FastSecAgg: Scalable Secure Aggregation for Privacy-Preserving Federated Learning

Swanand Kadhe, Nived Rajaraman, O. Ozan Koyluoglu, and Kannan Ramchandran
Department of Electrical Engineering and Computer Sciences
University of California Berkeley
{swanand.kadhe,nived,ozan.koyluoglu,kannanr}@berkeley.edu

ABSTRACT

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. A *secure aggregation* protocol enables the server to aggregate clients' models in a privacy-preserving manner. However, existing secure aggregation protocols incur high computation/communication costs, especially when the number of model parameters is larger than the number of clients participating in an iteration – a typical scenario in federated learning.

In this paper, 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. FastShare is information-theoretically secure, and achieves a trade-off between the number of secrets, privacy threshold, and dropout tolerance. Riding on the capabilities of FastShare, we prove that FastSecAgg is (i) secure against the server colluding with *any* subset of some constant fraction (e.g. $\sim 10\%$) of the clients in the honest-but-curious setting; and (ii) tolerates dropouts of a *random* subset of some constant fraction (e.g. $\sim 10\%$) of the clients. FastSecAgg achieves significantly smaller computation cost than existing schemes while achieving the same (orderwise) communication cost. In addition, it guarantees security against adaptive adversaries, which can perform client corruptions dynamically during the execution of the protocol.

CCS CONCEPTS

•**Security and privacy → Privacy-preserving protocols;**

KEYWORDS

secret sharing, secure aggregation, federated learning, machine learning, data sketching, privacy-preserving protocols

1 INTRODUCTION

Federated Learning (FL) is 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. In particular, training a neural net in a federated manner is an iterative process. In each iteration of FL, the server selects a subset of clients, and sends them the current global model. Each client runs several steps of mini-batch stochastic gradient descent, and communicates its model update—the difference between the local model and the received global model—to the server. The server aggregates the model updates from the clients to obtain an improved global model, and moves to the next iteration.

FL ensures that every client keeps their local data on their own device, and sends only model updates that are functions of their dataset. Even though any update cannot contain more information than the client's local dataset, the updates may still leak significant information making them vulnerable to inference and inversion attacks [17, 19, 33, 40]. Even well-generalized deep models such as DenseNet can leak a significant amount of information about their training data [33]. In fact, certain neural networks (e.g., generative text models) trained on sensitive data (e.g., private text messages) can memorize the training data [13].

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 [9]. Secure aggregation can be efficiently composed with differential privacy to further enhance privacy guarantees [9, 22, 45].

The secure aggregation problem for securely computing the summation has received significant research attention in the past few years, see [22] for a survey. However, the FL setup presents unique *challenges* for a secure aggregation protocol [8, 28]:

(1) *Massive scale:* Up to 10,000 users can be selected to participate in an iteration.
(2) *Communication costs:* Model updates can be high-dimensional vectors, e.g., the ResNet model consists of hundreds of millions of parameters [23].
(3) *Dropouts:* Clients are mobile devices which can drop out at any point. The percentage of dropouts can go up to 10%.
(4) *Privacy:* It is crucial to provide strongest possible privacy guarantees since malicious servers can launch powerful attacks by using *sybils* [14], wherein the adversary simulates a large number of fake client devices [18].

Due to these challenges, existing secure aggregation protocols for FL [3, 9, 43, 45, 47] either incur heavy computation and communication costs or provide only weaker forms of privacy guarantees, which limits their scalability (see Table 1).

Secret sharing [2] is an elegant primitive that is used as a building block in a large number of secure summation (and in general secure MPC) protocols, see, e.g., [15, 22]. Secret sharing based secure aggregation protocols are often more efficient than other approaches such as homomorphic encryption and Yao's garbled

A shorter version of this paper has been accepted in ICML Workshop on Federated Learning for User Privacy and Data Confidentiality, July 2020, and CCS Workshop on Privacy-Preserving Machine Learning in Practice, November 2020.

Protocol	Computation (Server)	Communication (Server)	Computation (Client)	Communication (Client)	Adversary	Dropouts
SecAgg [9]	$O\left(LN^2\right)$	$O\left(LN + N^2\right)$	$O\left(LN + N^2\right)$	$O\left(L + N\right)$	Adaptive	Worst-case
TurboAgg [43]	$O\left(L \log N \log^2 \log N\right)$	$O\left(LN \log N\right)$	$O\left(L \log N \log^2 \log N\right)$	$O\left(L \log N\right)$	Non-adaptive	Average-case
SecAgg+ [3]	$O\left(LN \log N + N \log^2 N\right)$	$O\left(LN + N \log N\right)$	$O\left(L \log N + \log^2 N\right)$	$O\left(L + \log N\right)$	Non-adaptive	Average-case
FastSecAgg	$O\left(L \log N\right)$	$O\left(LN + N^2\right)$	$O\left(L \log N\right)$	$O\left(L + N\right)$	Adaptive	Average-case

Table 1: Comparison of the proposed FastSecAgg with SecAgg [9], TurboAgg [43], and SecAgg+ [3]. Here N is the total number of clients and L is the length of model updates (i.e., client inputs). An adaptive adversary can choose which clients to corrupt during the protocol execution, whereas the corruptions happen before the protocol execution starts for a non-adaptive adversary. Worst-case (resp. average-case) dropout guarantee ensures that any (resp. a random) subset of clients of a bounded size may drop out without affecting the correctness and security. TurboAgg requires at least $\log N$ rounds, which increases the overheads. The protocols in [45, 47] assume a trusted third party for key distribution (see Sec. 5.3 for details).

circuits [1, 11, 22]. However, naïvely using the popular Shamir's secret sharing scheme [39] for aggregating length-L updates from N clients incurs $O\left(LN^2\right)$ communication and computation cost, which is intractable at the envisioned scale of FL.

In this paper, we propose FastShare—a novel secret sharing scheme based on a finite-field version of the Fast Fourier Transform (FFT)—that is designed to leverage the FL setup. FastShare is a *multi-secret* sharing scheme, which can *simultaneously* share S secrets to N clients such that (i) it is information-theoretically secure against a collusion of *any* subset of some constant fraction (e.g. $\sim 10\%$) of the clients; and (ii) with overwhelming probability, it is able to recover the secrets with $O\left(N \log N\right)$ complexity even if a *random* subset of some constant fraction (e.g. $\sim 10\%$) of the clients drop out. In general, FashShare can achieve a trade-off between the security and dropout parameters (see Theorem 3.3); thresholds of 10% are chosen to be well-suited in the FL setup. We note that a larger number of *arbitrary* (or even adversarial) dropouts does not affect security, and may only result in a failure to recover the secrets (i.e, affects only correctness). FastShare uses ideas from sparse-graph codes [27, 29, 38] and crucially exploits *insights derived from a spectral view of codes* [6, 34], which enables us to prove strong privacy guarantees.

Using FastShare as a building block, we propose FastSecAgg – a scalable secure aggregation protocol for FL that has low computation and communication costs and strong privacy guarantees (see Table 1 for a comparison of costs). FastSecAgg achieves low costs by riding on the capabilities of FastShare and leveraging the following unique attribute of FL. Privacy breaches in FL are inflicted by malicious/adversarial actors (e.g., a server using sybils), requiring *worst-case* guarantees; whereas dropouts occur due to non-adversarial/natural causes (e.g., system dynamics and wireless outages) [8], making *statistical or average-case* guarantees sufficient. FastShare achieves $O\left(N \log N\right)$ computation cost by relaxing the dropout constraint from worst-case to random.

For N clients, each having a length-L update vector, FastSecAgg requires $O\left(L \log N\right)$ computation and $O\left(LN + N^2\right)$ communication at the server, and $O\left(L \log N\right)$ computation and $O\left(L + N\right)$ communication per client. We compare the costs and setups with existing protocols in Table 1 (see Sec. 5.3 for details). We point out that the computation cost at the server was observed to be the key barrier to scalability in [8, 9]. FastSecAgg significantly reduces the computation cost at the server as compared to prior works. When the length of model updates L is larger than the number of clients N, FastSecAgg achieves the same (order-wise) communication cost as the prior works. We note that, in typical FL applications, the number of model parameters (L) is in millions, whereas the number of clients (N) is up to tens of thousands [8, 32]. In addition, FastSecAgg guarantees security against an *adaptive* adversary, which can corrupt clients adaptively during the execution of the protocol. This is in contrast with the protocols in [3, 43], which are secure only against non-adaptive (or static) adversaries, wherein client corruptions happen before the protocol executions begins.

We note that, although FastShare secret sharing scheme is inspired from the federated learning setup, it may be of independent interest. In fact, FastShare is a cryptographic primitive that can be used for secure aggregation (and secure multi-party computation in general) in wide applications including smart meters [26], medical data collection [12], and syndromic surveillance [46].

2 PROBLEM SETUP AND PRELIMINARIES

2.1 Federated Learning

We consider the setup in which a fixed set of M clients, each having their local dataset, coordinate with the server to jointly train a model. The i-th client's data is sampled from a distribution $\mathcal{D}_i$. The federated learning problem can be formalized as minimizing a sum of stochastic functions defined as

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

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 $\mathbf{w}$.

Federated Averaging (FedAvg) is a synchronous update scheme that proceeds in rounds of communication [32]. At the beginning of each round (called iteration), the server selects a subset C of N clients (for some $N \leq M$). Each of these clients $i \in C$ 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 $\mathbf{w}_{i,T}^t$; each local step k is of the form $\mathbf{w}_{i,k}^t \leftarrow \mathbf{w}_{i,k-1}^t - \eta_l g_{i,k-1}^t$, 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.[1] Then, each client $i \in C$ 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 $\mathbf{w}^{t+1} = \mathbf{w}^t + \frac{1}{|C|} \sum_{i \in C} \Delta \mathbf{w}_i^t$.

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 their model update $\Delta \mathbf{w}_i^t \in \mathbb{R}^d$ to obtain $\mathbf{u}_i \in \mathbb{Z}_R^L$, where $L \leq d$. Our goal is to design a protocol that enables the server to securely compute $\sum_{i \in C} \mathbf{u}_i$. We describe the threat model and the objectives in the next section.

2.2 Threat Model

Federated learning can be considered as a multi-party computation consisting of N parties (i.e., the clients), each having their own private dataset, and an aggregator (i.e., the server), with the goal of learning a model using all of the datasets.

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 (aka semi-honest) adversarial model is commonly used in the field of secure MPC, including prior works on secure federated learning [9, 43, 45].

Colluding parties: In every iteration of federated averaging, the server first samples a set C of clients. The server may collude with any set of up to T clients from C. 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. 5.1 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.

Objective: Our objective 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. We refer to T as the *privacy threshold* and D as the *dropout tolerance*.

2.3 Cryptographic Primitives

2.3.1 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 $(\text{pk}_i, \text{sk}_i) \leftarrow \text{KA.gen}(pp)$. The key agreement procedure allows clients i and j to obtain a private shared key $k_{i,j} \leftarrow \text{KA.agree}(\text{sk}_i, \text{pk}_j)$. Correctness requires that, for any key pairs generated by clients i and j (using KA.gen with the same parameters pp), $\text{KA.agree}(\text{sk}_i, \text{pk}_j) = \text{KA.agree}(\text{sk}_j, \text{pk}_i)$. 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.

2.3.2 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, $\text{AE.dec}((k_i, \text{AE.enc}(k_i, m)) = m$. For security, we require semantic security under a chosen plaintext attack (IND-CPA) and ciphertext integrity (IND-CTXT) [4].

3 FASTSHARE: FFT BASED SECRET SHARING

In this section, we present a novel, computationally efficient multi-secret sharing scheme FastShare which forms the core of our proposed secure aggregation protocol FastSecAgg (described in the next section).

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[2] 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. We summarize this in the following definition.

Definition 3.1 (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 $\{(i, [\mathbf{s}]_i)\}_{i \in C} \leftarrow \text{Share}(\mathbf{s}, C)$ is a probabilistic algorithm that takes as input a set of secrets $\mathbf{s} \in \mathbb{F}_q^S$ and a set 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 C. For a set $\mathcal{D} \subseteq C$, the reconstruction algorithm $\{\mathbf{s}, \perp\} \leftarrow \text{Recon}\left(\{(i, [\mathbf{s}]_i)\}_{i \in C \setminus \mathcal{D}}\right)$ takes as input the shares corresponding to $C \setminus \mathcal{D}$, 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 $\mathcal{P} \subset C$ 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 $\mathcal{D} \subset C$ of size at most D, $\text{Recon}\left(\{(i, [\mathbf{s}]_i)\}_{i \in C \setminus \mathcal{D}}\right) = \mathbf{s}$ with probability at least $1 - \text{poly}(N)$.

[1] 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 its dataset. See [32] for details.

[2] Secret sharing [2] and multi-secret sharing [7, 16] schemes conventionally consider *worst-case* dropouts. We consider *random* dropouts to exploit the FL setup.

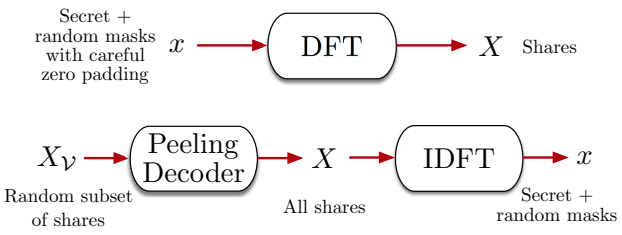

Figure 1: FastShare to generate shares, and FastRecon to recover the secrets from a subset of shares.

(3) *Linearity:* If $\{(i, [\mathbf{s}_1]_i)\}_{i\in C}$ and $\{(i, [\mathbf{s}_2]_i)\}_{i\in C}$ are sharings of $\mathbf{s}_1$ and $\mathbf{s}_2$, then $\{(i, a[\mathbf{s}_1]_i + b[\mathbf{s}_2]_i)\}_{i\in C}$ 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 $n_0 < n_1$. E.g., $n_0 = 10$, $n_1 = 13$, 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 $b = j \mod n_1$.

3.1 FastShare to Generate Shares

The sharing algorithm $\{(i, [\mathbf{s}]_i)\}_{i\in C} \leftarrow \text{FastShare}(\mathbf{s}, C)$ is a probabilistic algorithm that takes as input a set of secrets $\mathbf{s} \in \mathbb{F}_q^S$, and a set 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. 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 judiciously chosen locations. Second, it takes the fast Fourier transform of the signal to generate the shares (see Fig. 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. 2. We give the formal definitions below. Here, we round any real number to the largest integer no larger than it, i.e., round x to $\lfloor x \rfloor$, and omit the floor sign for the sake of brevity.

$$\mathcal{Z}_0 = \{(a,b) : 0 \le a \le \delta_0 n_0 - 1\}, \tag{2}$$

$$\mathcal{Z}_1 = \{(a,b) : 0 \le b \le \delta_1 n_1 - 1\}, \tag{3}$$

$$\mathcal{S} = \{(a,b) : \delta_0 n_0 + \alpha(1-\delta_0)n_0 \le a \le n_0 - 1, \\ \delta_1 n_1 + \beta(1-\delta_1)n_1 \le b \le \delta_1 n_1 + (1-\beta)(1-\delta_1)n_1\}, \tag{4}$$

$$\mathcal{T} = \{(a,b) : \delta_0 n_0 \le a \le \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1, \\ \delta_1 n_1 + (1-\beta)(1-\delta_1)n_1 \le b \le n_1 - 1\}. \tag{5}$$

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 for a toy example.)

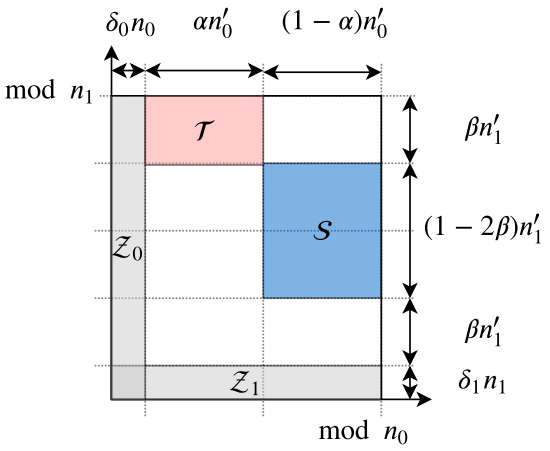

Figure 2: Indices assigned to a grid using the Chinese remainder theorem, and sets $\mathcal{Z}_0$, $\mathcal{Z}_1$, $\mathcal{S}$, and $\mathcal{T}$. Here, we define $n_i' = (1-\delta_i)n_i$ for $i \in \{0,1\}$. Set $\mathcal{S}$ is used for secrets, and sets $\mathcal{Z}_0$ and $\mathcal{Z}_1$ are used for zeros, and the remaining locations are used for random masks. Set $\mathcal{T}$ is used in our privacy proof.

Let ω be a primitive N-th root of unity in $\mathbb{F}_q$. FastShare computes the (fast) Fourier transform $\mathbf{X}$ of the signal $\mathbf{x}$ generated by ω.[3] The coefficients of $\mathbf{X}$ represent shares of $\mathbf{s}$, i.e., $[\mathbf{s}]_i = \mathbf{X}_i$. The details are given in Algorithm 1.

3.2 FastRecon to Reconstruct the Secrets

Let $\mathcal{D} \subseteq C$ denote a random subset of size at most D. The reconstruction algorithm $\{\mathbf{s}, \perp\} \leftarrow \text{FastRecon}\left(\{(i, [\mathbf{s}]_i)\}_{i\in C\setminus\mathcal{D}}\right)$ takes as input the shares corresponding to $C \setminus \mathcal{D}$, 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. 1).

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 Fig. 3 for a toy example.)

Lemma 3.2. *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*

$$\sum_{u=0}^{n_i-1} \omega^{-uv\frac{N}{n_i}} X_{u\frac{N}{n_i}+c} = 0. \tag{6}$$

[3] See Appendix A for a brief overview of the finite field Fourier transform.

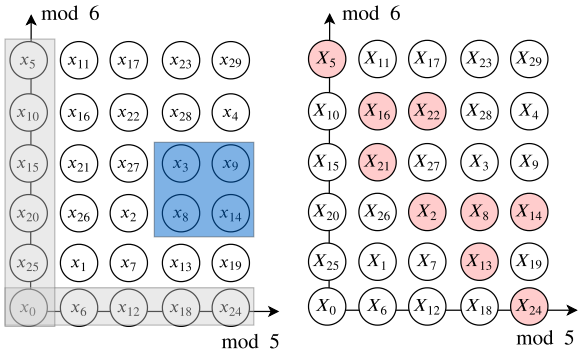

Figure 3: Example with $\ell = 4$ secrets, and co-prime integers $n_0 = 5$, $n_1 = 6$, yielding $N = n_0 n_1 = 30$. Zeros are placed at gray locations, and the secrets at blue locations. Careful zero padding induces parity checks on the shares in each row and column due to *aliasing*, i.e., $[X_0 + X_5 + \cdots + X_{25}; X_1 + \cdots + X_{26}; \cdots; X_4 + \cdots + X_{29}] = [0, \cdots, 0]$ 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.

The proof essentially follows from the subsampling and aliasing properties of the Fourier transform and is deferred to Appendix B.

Remark 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 [31]. 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 (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 $u = 0, 1, \ldots, n_i - 1$. We present a comparison with the Shamir's scheme in Appendix F.*

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. 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* [27, 38].

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.

Algorithm 1 FastShare Secret Sharing

procedure FastShare($\mathbf{s}, C$)
  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 C (client-IDs)
  initialize: sets $\mathcal{Z}_0$, $\mathcal{Z}_1$, $\mathcal{S}$ as per (2), (2), (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: $\{(i, [\mathbf{s}]_i)\}_{i \in C} \leftarrow \{(i, X_i)\}_{i=0}^{N-1}$
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

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 1.

3.3 Analysis of FastShare

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

Theorem 3.3. *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 $D = 0.095N$.

Next, we present the computation cost in terms of the number of (basic) arithmetic operations in $\mathbb{F}_q$.

Computation Cost: FastShare runs in $O(N \log N)$ time, since constructing the signal takes $O(N)$ time and the Fourier transform can be computed in $O(N \log N)$ 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 $O\left(n_i^2\right) = O(N)$ 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 $O(N)$ time. The second step is the inverse Fourier transform, which can be computed in $O(N \log N)$ time using an FFT.

Note that in practical FL scenarios, the number of clients typically scales up to ten thousand [8]. 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.

3.4 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 (cf. Remark 1). 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. 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.

$$\mathcal{Z}_0 = \{(a,b) : 0 \leq a \leq \delta_0 n_0 - 1\}, \tag{7}$$

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

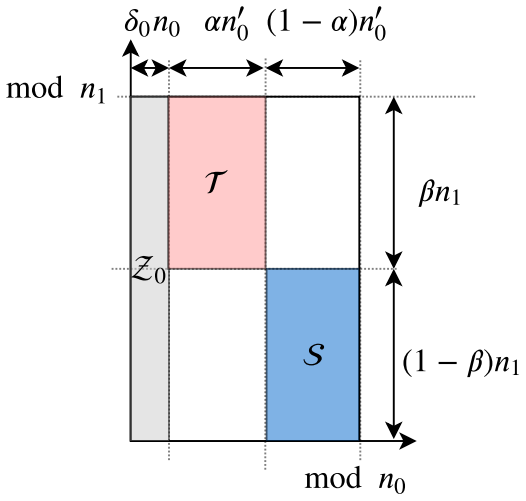

Figure 4: Indices assigned to a grid, and sets $\mathcal{Z}$, $\mathcal{S}$, and $\mathcal{T}$. Here, we define $n_0' = (1-\delta_0)n_0$. Set $\mathcal{S}$ is used for secrets, $\mathcal{Z}_0$ is used for zeros, and the remaining locations are used for random masks. Set $\mathcal{T}$ is used in our privacy proof.

$$0 \leq b \leq (1-\beta)n_1\}, \tag{8}$$

$$\mathcal{T} = \{(a,b) : \delta_0 n_0 \leq a \leq \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1,$$
$$(1-\beta)n_1 \leq b \leq n_1 - 1\}. \tag{9}$$

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, 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

$$\sum_{u=0}^{n_0-1} \omega^{-uvn_1} X_{un_1+c} = 0. \tag{10}$$

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 $(1-\delta_0)n_0$.[4]

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

$$S = (1-\alpha)(1-\beta)(1-\delta_0)N \tag{11}$$

[4]We note that this variant of FastShare is related to a class of erasure codes called Locally Recoverable Codes (LRCs) (see [21, 37, 44], and references therein). See Appendix F for details.

secrets such that it satisfies T-privacy for

$$T = \alpha\beta(1-\delta_0)N, \tag{12}$$

D-dropout-resilience for

$$D = \delta_0 N \tag{13}$$

and linearity. (The proof is similar to Theorem 3.3, 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 $O(N \log N)$ computational cost, since constructing the signal takes $O(N)$ time and the Fourier transform can be computed in $O(N \log N)$ time using an FFT. FastRecon has $O(N \log N)$ 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 $O\left(n_0^2\right) = O\left(\log^2 N\right)$ 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 $O(N/\log N)$ rows, the decoder to recover the missing shares has $O(N \log N)$ complexity. The second step is the inverse FFT, which also has $O(N \log N)$ complexity.

4 FASTSECAGG BASED ON FASTSHARE

In this section, we present our proposed protocol FastSecAgg, which allows the server to securely compute the summation of clients' model updates. We begin with a high-level overview of FastSecAgg (see Fig. 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 [5, 11, 22].

FastSecAgg is a three round interactive protocol. See Fig. 5 for a high-level overview, and Algorithm 2 for the detailed protocol. Recall that the model update for client $i \in C$ 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 $C_0 \subseteq C$), and forwards the received public keys to clients in 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, $\mathbf{u}_i^1, \mathbf{u}_i^2, \ldots, \mathbf{u}_i^{\lceil L/S \rceil}$, 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 $\{(j, [\mathbf{u}_i^\ell]_j)\}_{j \in C} \leftarrow \text{FastShare}(\mathbf{u}_i^\ell, C)$ for $1 \leq \ell \leq \lceil L/S \rceil$. For simplicity, we denote client i's share for client j as $\mathsf{sh}_{i\to j} = ([\mathbf{u}_i^1]_j \,||\, [\mathbf{u}_i^2]_j \,||\, \cdots \,||\, [\mathbf{u}_i^{\lceil L/S \rceil}]_j)$.

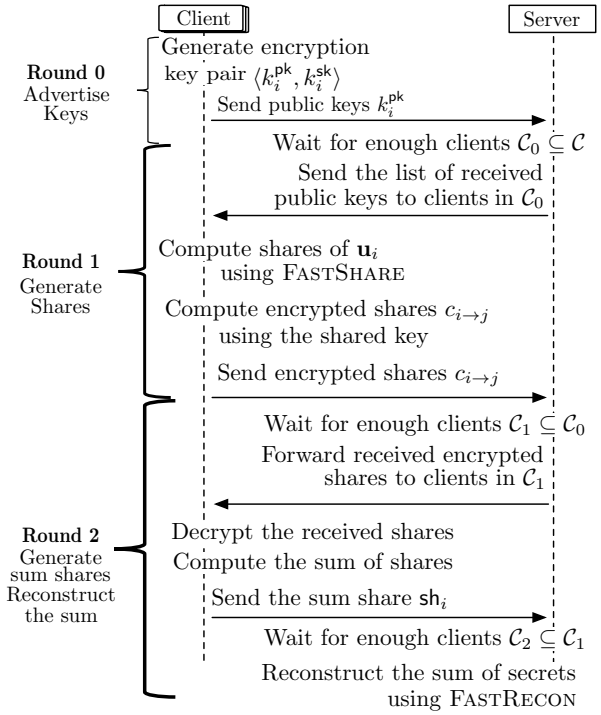

Figure 5: High-lever overview of FastSecAgg protocol.

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 $j \in C_0 \setminus \{i\}$, and encrypts $\mathsf{sh}_{i\to j}$ using the shared key $k_{i,j}$ as $c_{i\to j} \leftarrow \text{AE.enc}\left(k_{i,j}, \mathsf{sh}_{i\to j}\right)$. The client then sends all the encrypted shares $\{c_{i\to j}\}_{j \in C_0 \setminus \{i\}}$ to the server.

The server waits for at least $N-D$ clients to respond (denoted as $C_1 \subseteq C_0$).[5] Then, the server sends to each client $i \in C_1$, all ciphertexts encrypted for it: $\{c_{j\to i}\}_{j \in C_1 \setminus \{i\}}$.

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: $\mathsf{sh}_i = \sum_{j \in C_1} \mathsf{sh}_{j\to i}$. 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 $C_2 \subseteq C_1$). Let sh_i^ℓ denote the ℓ-th coefficient of sh_i for $1 \leq \ell \leq \lceil L/S \rceil$. The server computes $\{\mathbf{z}^\ell, \perp\} \leftarrow \text{FastRecon}\left(\{(i, \mathsf{sh}_i^\ell)\}_{i \in C_2}\right)$,

[5] 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 [3, 9, 43]. This is not a critical assumption, and the protocol and the analysis can be easily adapted if it does not hold.

Algorithm 2 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 $C_0 \subseteq C$); otherwise, abort
  - Send to all clients in C_0 the list $\{(i, \mathsf{pk}_i)\}_{i \in C_0}$, and move to the next round
- **Round 1** (Generate shares)

  Client i:
  - Receive the list $\{(j, \mathsf{pk}_j)\}_{j \in C_0}$ from the server; Assert that $|C_0| \geq N - D$, otherwise abort
  - Partition the input $\mathbf{u}_i \in \mathbb{Z}_R^L$ into $\lceil L/S \rceil$ vectors, $\mathbf{u}_i^1, \mathbf{u}_i^2, \ldots, \mathbf{u}_i^{\lceil L/S \rceil}$, 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 $\{(j, \left[\mathbf{u}_i^\ell\right]_j)\}_{j \in C} \leftarrow \text{FastShare}(\mathbf{u}_i^\ell, C)$ 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 $j \in C_0 \setminus \{i\}$, 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 $\{c_{i \to j}\}_{j \in C_0 \setminus \{i\}}$ 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 $C_1 \subseteq C_0$)
  - Send to each client $i \in C_1$, all ciphertexts encrypted for it: $\left\{c_{j \to i}\right\}_{j \in C_1 \setminus \{i\}}$, and move to the next round
- **Round 2** (Recover the aggregate update)

  Client i:
  - Receive from the server the list of ciphertexts $\left\{c_{j \to i}\right\}_{j \in C_1 \setminus \{i\}}$
  - Decrypt the ciphertext $(i' \,||\, j' \,||\, \mathsf{sh}_{j \to i}) \leftarrow \text{Dec}(\mathsf{sk}_i, c_{j \to i})$ for each client $j \in C_1 \setminus \{i\}$, and assert that $(i = i') \wedge (j = j')$
  - Compute the sum of shares over $\mathbb{F}_q$ as $\mathsf{sh}_i = \sum_{j \in C_1} \mathsf{sh}_{j \to i}$
  - Send the share sh_i to the server

  Server:
  - Wait for at least $N - D$ clients to respond (denote this set as $C_2 \subseteq C_1$)
  - Run the reconstruction algorithm to obtain $\left\{\mathbf{z}^\ell, \perp\right\} \leftarrow \text{FastRecon}(\left\{(i, \mathsf{sh}_i^\ell)\right\}_{i \in C_2})$ 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 $\mathbf{z}$

for $1 \leq \ell \leq \lceil L/S \rceil$. 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 $\mathbf{z} = [\mathbf{z}^1 \;\; \mathbf{z}^2 \;\; \cdots \;\; \mathbf{z}^{\lceil L/S \rceil}]$.

5 ANALYSIS

5.1 Correctness and Security

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

Theorem 5.1 (Correctness). *Let $\{\mathbf{u}_i\}_{i \in C}$ denote the client inputs for FastSecAgg. If a random set of at most D clients drop out during the execution of FastSecAgg, i.e., $|C_2| \geq N - D$, then the server does not abort and obtains $\mathbf{z} = \sum_{i \in C_1} \mathbf{u}_i$ with probability at least $1 - 1/poly\, N$, 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 C.

Clients may drop out (or, abort) at any point during the execution, and we denote with C_i the subset of the clients that correctly sent their message to the server in round i. Therefore, we have $C \supseteq C_0 \supseteq C_1 \supseteq C_2$. For example, the set $C_0 \setminus C_1$ 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 $C' \subseteq C$, let $\mathbf{u}_{C'} = \{\mathbf{u}_i\}_{i \in C'}$.

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 $\mathcal{M} \subset C$, let $\mathsf{REAL}_{\mathcal{M}}^{C,T,\lambda}(\mathbf{u}_C, C_0, C_1, C_2)$ 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 5.2 (Security). *There exists a probabilistic polynomial time (PPT) simulator* SIM *such that for all* C, $\mathbf{u}_C$, C_0, C_1, C_2, *and* $\mathcal{M}$ *such that* $M \subset C$, $|\mathcal{M}| \leq T$, $C \supseteq C_0 \supseteq C_1 \supseteq C_2$, *the output of* SIM *is computationally indistiguishable from the joint view* $\mathsf{REAL}_{\mathcal{M}}^{C,T,\lambda}$ *of the server and the corrupted clients* $\mathcal{M}$, *i.e.,*

$$\mathsf{REAL}_{\mathcal{M}}^{C,T,\lambda}(\mathbf{u}_C, C_0, C_1, C_2) \approx \mathsf{SIM}_{\mathcal{M}}^{C,T,\lambda}(\mathbf{u}_{\mathcal{M}}, \mathbf{z}, C_0, C_1, C_2), \quad (14)$$

where

$$\mathbf{z} = \begin{cases} \sum_{i \in C_1 \setminus \mathcal{M}} \mathbf{u}_i & \text{if } |C_1| \geq N - D, \\ \perp & \text{otherwise.} \end{cases} \quad (15)$$

The security and correctness of FastSecAgg critically relies on the guarantees provided by FastShare proved in Theorem 3.3. We present the proofs of the above theorem in Appendices E.

5.2 Computation and Communication Costs

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

Computation Cost at a Client: $O(\max\{L, N\} \log N)$. Each client's computation cost can be broken as computing shares using FastShare which is $O(\lceil L/S \rceil N \log N) = O(\max\{L, N\} \log N)$; encrypting and decrypting shares, which is $O(\lceil L/S \rceil N)$; and adding the received shares, which is $O(\lceil L/S \rceil N) = O(\max\{L, N\})$.

Communication Cost at a Client: $O(\max\{L, N\})$. Each client sends and receives $N-1$ shares (each having $\lceil L/S \rceil$ elements in $\mathbb{F}_q$), resulting in $O(\lceil L/S \rceil N) = O(\max\{L, N\})$ 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: $O(\max\{L, N\} \log N)$. The server first recovers missing sum-shares using FastRecon, each has $\lceil L/S \rceil$ elements in $\mathbb{F}_q$. This results in $O(\lceil L/S \rceil N \log N)$ 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 $O(\lceil L/S \rceil N \log N)$ complexity.

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

5.3 Comparison with Prior Works

Bonawitz et al. [9] 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, i.e., it can tolerate any D users dropping out. However, SecAgg incurs significant computation cost of $O(LN^2)$ at the server. This limits its scalability to several hundred clients as observed in [8].

Truex et al. [45] uses threshold homomorphic encryption and Xu et al. [47] uses functional encryptionto 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. [43], 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 $O(L \log N)$. 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 $O(L)$ communication per client, runs in 3 rounds, and is robust against an adaptive adversary which can corrupt clients during the protocol execution.

Recently, Bell et al. [3] 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.

The performance comparison between FastSecAgg and SecAgg [9], TurboAgg [43], and SecAgg+ [3] is summarized in Table 1.

There are several recent works [24, 35, 42] 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.

ACKNOWLEDGEMENTS

Swanand Kadhe was introduced to the secure aggregation problem in the Decentralized Security course by Raluca Ada Popa in Fall 2019. He would like to thank Raluca for an excellent course. Swanand would like to thank Mukul Kulkarni for immensely helpful discussions and for providing feedback on drafts of this paper.

This work is supported in part by the National Science Foundation grant CNS-1748692.

REFERENCES

[1] Gergely Ács and Claude Castelluccia. 2011. I Have a DREAM! Differentially Private Smart Metering. In *Proceedings of the 13th International Conference on Information Hiding (IH'11)*. Springer-Verlag, Berlin, Heidelberg, 118–132.

[2] Amos Beimel. 2011. Secret-Sharing Schemes: A Survey. In *Coding and Cryptology*, Yeow Meng Chee, Zhenbo Guo, San Ling, Fengjing Shao, Yuansheng Tang, Huaxiong Wang, and Chaoping Xing (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 11–46.

[3] James Bell, K. A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure Single-Server Aggregation with (Poly)Logarithmic Overhead. Cryptology ePrint Archive, Report 2020/704. (2020). https://eprint.iacr.org/2020/704.

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

[5] Michael Ben-Or, Shafi Goldwasser, and Avi Wigderson. 1988. Completeness Theorems for Non-Cryptographic Fault-Tolerant Distributed Computation. In *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing (STOC '88)*. Association for Computing Machinery, New York, NY, USA, 1–fi?!10. https://doi.org/10.1145/62212.62213

[6] R. E. Blahut. 1979. Transform Techniques for Error Control Codes. *IBM Journal of Research and Development* 23, 3 (1979), 299–315.

[7] Carlo Blundo, Alfredo De Santis, Giovanni Di Crescenzo, Antonio Giorgio Gaggia, and Ugo Vaccaro. 1994. Multi-Secret Sharing Schemes. In *Advances in Cryptology — CRYPTO '94*, Yvo G. Desmedt (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 150–163.

[8] Keith Bonawitz, Hubert Eichner, Wolfgang Grieskamp, Dzmitry Huba, Alex Ingerman, Vladimir Ivanov, Chlo M Kiddon, Jakub Koneffhn, Stefano Mazzocchi, Brendan McMahan, Timon Van Overveldt, David Petrou, Daniel Ramage, and Jason Roselander. 2019. Towards Federated Learning at Scale: System Design. In *Proceedings of the 2nd SysML Conference*.

[9] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS '17)*. 1175–1191.

[10] A. Borodin and R. Moenck. 1974. Fast modular transforms. *J. Comput. System Sci.* 8, 3 (1974), 366 – 386. https://doi.org/10.1016/S0022-0000(74)80029-2

[11] Martin Burkhart, Mario Strasser, Dilip Many, and Xenofontas Dimitropoulos. 2010. SEPIA: Privacy-Preserving Aggregation of Multi-Domain Network Events and Statistics. In *Proceedings of the 19th USENIX Conference on Security (USENIX Security'10)*. USENIX Association, USA, 15.

[12] B. Cakici, K. Hebing, M Grunewald, P. Saretok, and A. Hulth. 2010. CASE: a framework for computer supported outbreak detection. In *BMC Medical Information and Decision Making*.

[13] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. 2019. The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks. In *28th USENIX Security Symposium (USENIX Security 19)*. 267–284.

[14] John R. Douceur. 2002. The Sybil Attack. In *Peer-to-Peer Systems*. Springer Berlin Heidelberg, Berlin, Heidelberg, 251–260.

[15] David Evans, Vladimir Kolesnikov, and Mike Rosulek. 2018. A Pragmatic Introduction to Secure Multi-Party Computation. *Foundations and Trends in Privacy and Security* 2 (2018), 70–246.

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

[17] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1322–1333.

[18] Clement Fung, Chris J. M. Yoon, and Ivan Beschastnikh. 2018. Mitigating Sybils in Federated Learning Poisoning. *CoRR* abs/1808.04866 (2018). arXiv:1808.04866 http://arxiv.org/abs/1808.04866

[19] Karan Ganju, Qi Wang, Wei Yang, Carl A. Gunter, and Nikita Borisov. 2018. 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*. 619–633.

[20] Joachim Von Zur Gathen and Jurgen Gerhard. 2003. *Modern Computer Algebra* (2 ed.). Cambridge University Press, USA.

[21] P. Gopalan, Cheng Huang, H. Simitci, and S. Yekhanin. 2012. On the Locality of Codeword Symbols. *Information Theory, IEEE Transactions on* 58, 11 (Nov 2012), 6925–6934.

[22] S. Goryczka and L. Xiong. 2017. A Comprehensive Comparison of Multiparty Secure Additions with Differential Privacy. *IEEE Transactions on Dependable and Secure Computing* 14, 5 (2017), 463–477.

[23] K. He, X. Zhang, S. Ren, and J. Sun. 2016. Deep Residual Learning for Image Recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 770–778.

[24] Lie He, Sai Praneeth Karimireddy, and Martin Jaggi. 2020. Secure Byzantine-Robust Machine Learning. (2020). arXiv:cs.LG/2006.04747

[25] Ellis Horowitz. 1972. A fast method for interpolation using preconditioning. *Inform. Process. Lett.* 1, 4 (1972), 157 – 163. https://doi.org/10.1016/0020-0190(72)90050-6

[26] Marek Jawurek, Martin Johns, and Florian Kerschbaum. 2011. Plug-In Privacy for Smart Metering Billing. In *Privacy Enhancing Technologies*, Simone Fischer-Hübner and Nicholas Hopper (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 192–210.

[27] J. Justesen. 2011. Performance of Product Codes and Related Structures with Iterated Decoding. *IEEE Transactions on Communications* 59, 2 (2011), 407–415.

[28] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary A. Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konecný, Aleksandra Korolova, Farinaz Koushanfar, Oluwasanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Xiaodong Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2019. Advances and Open Problems in Federated Learning. *ArXiv* abs/1912.04977 (2019).

[29] M. G. Luby, M. Amin Shokrolloahi, M. Mizenmacher, and D. A. Spielman. 1998. Improved low-density parity-check codes using irregular graphs and belief propagation. In *Proceedings. 1998 IEEE International Symposium on Information Theory*. 117–121.

[30] M. G. Luby, M. Mitzenmacher, M. A. Shokrollahi, and D. A. Spielman. 2001. Efficient erasure correcting codes. *IEEE Transactions on Information Theory* 47, 2 (2001), 569–584.

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

[32] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*. 1273–1282.

[33] M. Nasr, R. Shokri, and A. Houmansadr. 2019. Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning. In *2019 IEEE Symposium on Security and Privacy (SP)*. 739–753.

[34] S. Pawar and K. Ramchandran. 2018. FFAST: An Algorithm for Computing an Exactly k -Sparse DFT in $O(k \log k)$ Time. *IEEE Transactions on Information Theory* 64, 1 (2018), 429–450.

[35] Krishna Pillutla, Sham M. Kakade, and Zaid Harchaoui. 2019. Robust Aggregation for Federated Learning. (2019). arXiv:stat.ML/1912.13445

[36] John M. Pollard. 1971. The fast Fourier transform in a finite field. *Math. Comp.* 25 (1971), 365–374.

[37] N. Prakash, G.M. Kamath, V. Lalitha, and P.V. Kumar. 2012. Optimal linear codes with a local-error-correction property. In *Information Theory Proceedings (ISIT), 2012 IEEE International Symposium on*. 2776–2780.

[38] T. J. Richardson and R. L. Urbanke. 2001. The capacity of low-density parity-check codes under message-passing decoding. *IEEE Transactions on Information Theory* 47, 2 (2001), 599–618.

[39] Adi Shamir. 1979. How to Share a Secret. *Commun. ACM* 22, 11 (Nov. 1979), 612–613.

[40] R. Shokri, M. Stronati, C. Song, and V. Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*. 3–18.

[41] D. Silva and F. R. Kschischang. 2011. Universal Secure Network Coding via Rank-Metric Codes. *IEEE Transactions on Information Theory* 57, 2 (Feb. 2011), 1124–1135.

[42] Jinhyun So, Basak Guler, and A. Salman Avestimehr. 2020. Byzantine-Resilient Secure Federated Learning. (2020). arXiv:cs.CR/2007.11115

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

[44] I. Tamo and A. Barg. 2014. A Family of Optimal Locally Recoverable Codes. *Information Theory, IEEE Transactions on* 60, 8 (Aug 2014), 4661–4676.

[45] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A Hybrid Approach to Privacy-Preserving Federated Learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and*

Security. 1–11.

[46] Michael M. Wagner. 2006. CHAPTER 1 - Introduction. In *Handbook of Biosurveillance*, Michael M. Wagner, Andrew W. Moore, and Ron M. Aryel (Eds.). Academic Press, Burlington, 3 – 12. https://doi.org/10.1016/B978-012369378-5/50003-X

[47] Runhua Xu, Nathalie Baracaldo, Yi Zhou, Ali Anwar, and Heiko Ludwig. 2019. HybridAlpha: An Efficient Approach for Privacy-Preserving Federated Learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security (AISec'19)*. Association for Computing Machinery, New York, NY, USA, 13–fi?!23. https://doi.org/10.1145/3338501.3357371

A FINITE FIELD FOURIER TRANSFORM

Here, we review the basics of the finite field Fourier transform, for details, see, e.g., [36]. 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 $\mathbb{F}_q^N$ defined as follows. Let $x = [x_0 x_1 \dots x_{N-1}]$ be a vector over $\mathbb{F}_q$. Then, the DFT of x generated by ω, denoted as $DFT_\omega(x)$, is the vector over $\mathbb{F}_q$, $X = [X_0\ X_1\ \dots X_{N-1}]$, given by

$$X_j = \sum_{i=0}^{N-1} \omega^{ij} x_i, \quad j = 0, 1, \dots, N-1. \tag{16}$$

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

$$x_i = \frac{1}{N} \sum_{j=0}^{N-1} \omega^{-ij} X_j, \quad i = 0, 1, \dots, N-1, \tag{17}$$

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

$$X_j^{(\downarrow n)} = \frac{1}{n} \sum_{\substack{i=0 \\ i \mod (N/n) = j}}^{N-1} X_i, \quad j = 0, 1, \dots, \frac{N}{n} - 1, \tag{18}$$

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

$$X_j^{(t)} = \omega^{-tj} X_j, \quad j = 0, 1, \dots, N-1. \tag{19}$$

B PROOF OF LEMMA

Proof of Lemma 3.2. For $i \in \{0, 1\}$, for $v \in \{0, 1, \dots, \delta_i n_i - 1\}$, 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 $j = 0, 1, \dots, N/n_i - 1$. 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 (18), it holds that

$$\mathbf{X}_c^{(v)(\downarrow n_i)} = \frac{1}{n_i} \sum_{\substack{j=0 \\ j \mod (N/n_i) = c}}^{N-1} \mathbf{X}_j^{(v)}, \tag{20}$$

for $v = 0, 1, \dots, \delta_i n_i - 1$ and $c = 0, 1, \dots, N/n_i - 1$. However, from the circular shift property (19), we have $\mathbf{X}_j^{(v)} = \omega^{-vj} \mathbf{X}_j$ for $j = 0, 1, \dots, N-1$. Thus, we have

$$\mathbf{X}_c^{(v)(\downarrow n_i)} = \frac{1}{n_i} \sum_{\substack{j=0 \\ j \mod (N/n_i) = c}}^{N-1} \omega^{-vj} \mathbf{X}_j, \tag{21}$$

for $v = 0, 1, \dots, \delta_i n_i - 1$, and $c = 0, 1, \dots, N/n_i - 1$. Note that, by construction, $\mathbf{x}^{(v)(\downarrow n_i)}$ is a length-(N/n_i) zero vector for $v = 0, 1, \dots, \delta_i n_i - 1$ for each $i \in \{0, 1\}$. Therefore, $X^{(v)(\downarrow n_i)}$ is also a length-(N/n_i) zero vector for $v = 0, 1, \dots, \delta_i n_i - 1$ for each $i \in \{0, 1\}$. Hence, from (21), we get

$$\sum_{\substack{j=0 \\ j \mod (N/n_i) = c}}^{N-1} \omega^{-vj} X_j = \sum_{u=0}^{n_i - 1} \omega^{-v\left(u\frac{N}{n_i} + c\right)} X_{u\frac{N}{n_i} + c} = 0, \tag{22}$$

for $v = 0, 1, \dots, \delta_i n_i - 1$, and $c = 0, 1, \dots, N/n_i - 1$. Simplifying the above, we get

$$\sum_{u=0}^{n_i - 1} \left(\omega^{-uv\frac{N}{n_i}}\right) X_{u\frac{N}{n_i} + c} = 0, \tag{23}$$

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

C 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 [27, 34], it follows that, when $D \le (1 - (1 - \delta_0)(1 - \delta_1))\frac{N}{2}$, 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 [7]. In particular, we need to show the following: for any $\mathcal{P} \subset C$ such that $|\mathcal{P}| \le T$, it holds that $H(\mathbf{s} \mid \{[\mathbf{s}]_i\}_{i \in \mathcal{P}}) = H(\mathbf{s})$, 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,

$$\mathbf{X} = G \begin{bmatrix} \mathbf{s} \\ \mathbf{m} \end{bmatrix}, \tag{24}$$

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

$$\begin{aligned}\mathcal{S} = \{(p_0, p_1) : \\ \delta_0 n_0 + \alpha(1-\delta_0)n_0 \le p_0 \le n_0 - 1, \\ \delta_1 n_1 + \beta(1-\delta_1)n_1 + 1 \le p_1 \le n_1 - \beta n_1(1-\delta_1)\}. \end{aligned} \tag{25}$$

Similarly, define

$$\mathcal{Z}_0 = \{(p_0, p_1) : 0 \le p_0 \le \delta_0 n_0 - 1,\ 0 \le p_1 \le n_1 - 1\}, \tag{26}$$

$$\mathcal{Z}_1 = \{(p_0, p_1) : 0 \le p_0 \le n_0 - 1,\ 0 \le p_1 \le \delta_1 n_1 - 1\}. \tag{27}$$

For a pictorial representation of these indices, refer to Fig. 2. These sets correspond to points in the grid, and by the CRT map back to integers in the range $\{0, \cdots, N-1\}$.

Remark 2. *In Theorem 3.3, the number of secrets ℓ, is chosen to be equal to $(1-\alpha)(1-2\beta)(1-\delta_0)(1-\delta_1)n_0n_1$. 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 [41], and thus omitted.

Lemma C.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*

$$H(\mathbf{s}) - H(\mathbf{s} \mid \mathbf{X}_{\mathcal{P}}) \le \mathit{rank}(G_{\mathcal{P}}) - \mathit{rank}(G_2). \tag{28}$$

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 $\mathrm{rank}(G_{\mathcal{P}}) = \mathrm{rank}(G_2)$: 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. 2 - 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. 2. Formally,

$$\begin{aligned}\mathcal{T} = \{(p_0, p_1) : \delta_0 n_0 \le\ & p_0\ \le \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1, \\ & n_1 - \beta(1-\delta_1)n_1 \le\ p_1\ \le n_1 - 1\}. \end{aligned} \tag{29}$$

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 C.2. *If $p \in \mathit{span}(\mathcal{P})$ for some set of points $\mathcal{P}$, then for any $\Delta \in \{0, \cdots, n-1\}$, $p + \Delta \in \mathit{span}(\mathcal{P} + \Delta)$.*

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 $\omega_{\partial}^{\Delta}$,

$$\omega_{\partial}^{p+\Delta} = \sum\nolimits_{q \in \mathcal{P}} \alpha_q \omega_{\partial}^{q+\Delta} = \sum\nolimits_{q \in \mathcal{P}+\Delta} \alpha'_q \omega_{\partial}^{q}. \tag{30}$$

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 Theorem 3.3, observe that $|\mathcal{T}| = T$. By this fact, we have that $\mathrm{rank}(G_{\mathcal{P}}) \le T$. This implies two possibilities:

Case I. If $\mathrm{rank}(G_2(\mathcal{T})) = \mathrm{rank}(G_{\mathcal{P}})$, 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). Figure 6 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

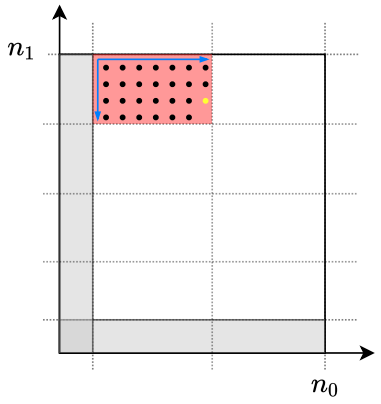

Figure 6: Enumerating the columns in the bottom-right sequence until the first column (yellow) is reached which lies in the span of the previous columns

until y is found. By definition, this implies that there exist (not necessarily unique) weights $\{\alpha_b\}_{b\in\mathcal{B}}$ such that,

$$\omega^y = \sum\nolimits_{b\in\mathcal{B}} \alpha_b \omega^b \tag{31}$$

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

$$\mathcal{Y} = \{(y_0,\theta) : y_1 - n_1(1-\delta_1)(1-\beta)+1 \le \theta \le y_1\}. \tag{32}$$

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

$$\mathcal{L} = \cup_{k=0}^{n_0-y_0-1} (\mathcal{Y}+(k,0)) \tag{33}$$

$$\begin{aligned} C =& \{(p_0,p_1) : \\ & n_0\delta_0 \le p_0 \le y_0, n_1\delta_1 \le p_1 \le n_1-1\} \setminus \mathcal{Y}. \end{aligned} \tag{34}$$

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

$$\begin{aligned} \mathcal{Q} =& \{(p_0,p_1) : \\ & n_0\delta_0 \le p_0 \le n_0-1, n_1\delta_1 \le p_1 \le n_1-1\} \setminus \mathcal{L}. \end{aligned} \tag{35}$$

Pictorially these points are represented in Figs 7 and 8.

Claim 1. $\mathcal{Y} \subseteq span(C)$.

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

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

$$y-(0,k) \in \text{span}(\mathcal{B}-(0,k)). \tag{36}$$

The key observation is that $\mathcal{B}-(0,k)$ is always $\subseteq C \cup \{y-(0,t) : 0 \le t \le k-1\}$ (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 C and $\mathcal{Y}$ so the assertion is clearly false). For a pictorial presentation of this fact, refer to Fig. 9. By the induction hypothesis, for each $t \le k-1$, $y-(0,t) \in$ span(C). Therefore, span$(\mathcal{B}-(0,k)) \subseteq$ span(C). Plugging into eq. (36) completes the proof of the claim. □

Claim 2. $\mathcal{L} \subseteq span(\mathcal{Q})$.

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 Claim 1 to result in the assertion. In particular, for $k=0$ the statement follows directly from the fact that $C \subseteq \mathcal{Q}$ and using Claim 1 to claim that $\mathcal{Y} \subseteq$ span(C). Assuming the induction hypothesis that for all $0 \le t \le k-1$, $\mathcal{Y}+(t,0) \subseteq$ span$(\mathcal{Q})$ we show that $\mathcal{Y}+(k,0) \subseteq$ span$(\mathcal{Q})$ as long as $k \le n_0-y_0-1$. In particular, observe that from Claim 1,

$$\mathcal{Y}+(0,k) \subseteq \text{span}(C+(0,k)). \tag{37}$$

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

$$C+(k,0) \subseteq \mathcal{Q} \cup \{\mathcal{Y}+(t,0) : 0 \le t \le k-1\} \tag{38}$$

if $k > n_0-y_0-1$, then $C+(k,0)$ 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 $t \le n_0-y_0-1$ and the assertion becomes false. A pictorial representation of this fact is provided in Fig. 10. By the induction hypothesis for each $0 \le t \le k-1$, $\mathcal{Y}+(t,0) \subseteq$ span$(\mathcal{Q})$. Combining this fact with eqs. (37) and (38) implies that $\mathcal{Y}+(0,k) \subseteq$ span$(\mathcal{Q})$ and completes the induction step for k. □

Recall from eq. (33) 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 $\{(y_0,\theta) : y_1 - n_1(1-\delta_1)(1-\beta)+1 \le \theta \le y_1\}$,

$$\begin{aligned} \mathcal{L} = \{(p_0,p_1) :\ & y_0 \le p_0 \le n_0-1, \\ & y_1 - n_1(1-\delta_1)(1-\beta)+1 \le p_1 \le y_1\}. \end{aligned} \tag{39}$$

Claim 3. *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. 10. 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,

$$\left\{ \begin{array}{l} (p_0,p_1) : \\ \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1 \le p_0 \le n_0-1, \\ y_1 - n_1(1-\delta_1)(1-\beta)+1 \le p_1 \le y_1 \end{array} \right\} \subseteq \mathcal{L} \tag{40}$$

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,

$$\left\{ \begin{array}{l} (p_0,p_1) : \\ \delta_0 n_0 + \alpha(1-\delta_0)n_0 - 1 \le p_0 \le n_0-1, \\ n_1\delta_1 + \beta(1-\delta_1)n_1 \le p_1 \le n_1 - \beta(1-\delta_1)n_1 \end{array} \right\} \subseteq \mathcal{L} \tag{41}$$

The LHS is exactly the definition of $\mathcal{S}$ in eq. (25). □

From Claims 2 and 3 and the definition of $\mathcal{L}$ in eq. (33), we see that $\mathcal{S} \subseteq$ 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 $\mathcal{Z}_1$.

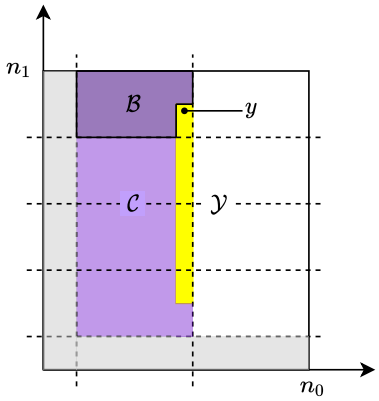

Figure 7: Definition of $\mathcal{B}, \mathcal{Y}$ and C

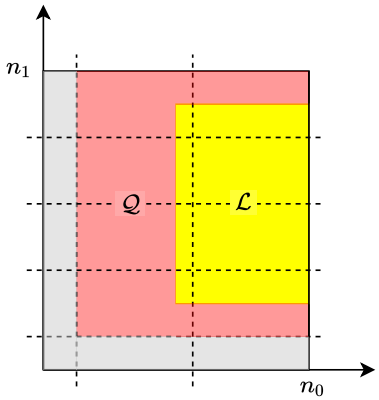

Figure 8: Definition of Q and $\mathcal{L}$

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 $\leq T$) 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 $\leq T$.

D 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 $i \in C_1$ computes and sends to the server the sum of shares it receives in Round 1 as follows

$$\mathsf{sh}_i = \left(\sum_{j \in C_1} [\mathbf{u}_j^1]_i \;||\; \cdots \;||\; \sum_{j \in C_1} [\mathbf{u}_j^{\lceil L/S \rceil}]_i \right). \tag{42}$$

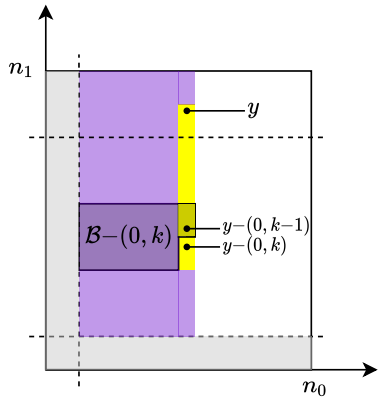

Figure 9: Showing that $\mathcal{B}-(0,k)$ is a subset of the intersection of C and $\{y-(0,t) : 0 \leq t \leq k-1\}$.

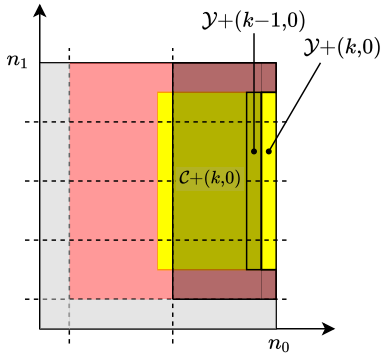

Figure 10: Showing that $C + (k,0)$ is a subset of the intersection of Q and $\bigcup_{t=0}^{k-1}(\mathcal{Y} + (t,0))$.

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

$$\mathsf{sh}_i = \left(\left[\sum_{j \in C_1} \mathbf{u}_j^1 \right]_i \;||\; \cdots \;||\; \left[\sum_{j \in C_1} \mathbf{u}_j^{\lceil L/S \rceil} \right]_i \right). \tag{43}$$

Recall that sh_i^ℓ denotes the ℓ-th coefficient of sh_i. Therefore, from (43), it holds, for $1 \leq \ell \leq \lceil L/S \rceil$, that

$$\mathsf{sh}_i^\ell = \left[\sum_{j \in C_1} \mathbf{u}_j^\ell \right]_i. \tag{44}$$

Let C_2 be a random subset of at least $N - D$ clients that survive in Round 2, i.e., clients in C_2 send their sum-shares to the server. Now, from (44) and the D-dropout tolerance of FastShare, it holds that $\textsc{FastRecon}\left(\{(i, \mathsf{sh}_i^\ell)\}_{i \in C_2}\right) = \sum_{j \in C_1} \mathbf{u}_j^\ell$ with probability at least $1 - 1/\text{poly}\, N$ for all $1 \leq \ell \leq \lceil L/S \rceil$. 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 $1 \leq \ell \leq \lceil L/S \rceil$,

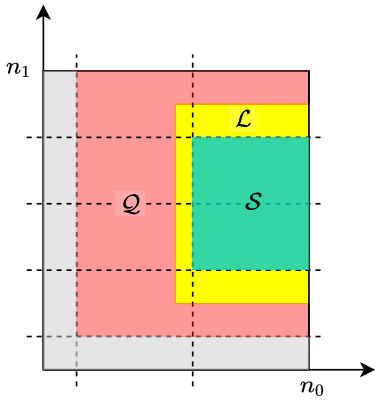

Figure 11: The $\mathcal{S}$ is indeed a subset of $\mathcal{L}$. For the particular choice of y this figure shows that the inclusion is true. More generally we in construct $\mathcal{S}$ as the intersection of $\mathcal{L}$ over all the possible locations of $y \in \mathcal{T}$.

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

$$[\mathbf{z}^1 \ \mathbf{z}^2 \ \cdots \ \mathbf{Z}^{\lceil L/S \rceil}] = \left[\sum_{j \in C_1} \mathbf{u}_j^1 \ \sum_{j \in C_1} \mathbf{u}_j^2 \ \cdots \ \sum_{j \in C_1} \mathbf{u}_j^{\lceil L/S \rceil} \right]$$

with probability at least $1 - 1/\text{poly}\, N$. In other words, $\mathbf{z} = \sum_{i \in C_1} \mathbf{u}_i$ with probability at least $1 - 1/\text{poly}\, N$. This concludes the proof.

E SECURITY OF FASTSECAGG

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

$$\left\{ \left(j, \left[\mathbf{u}_i^\ell \right]_j \right) \right\}_{j \in C} \leftarrow \text{FastShare}(\mathbf{u}_i^\ell, C), \tag{45}$$

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

$$[\mathbf{u}_i]_j = \left\{ \left[\mathbf{u}_i^1 \right]_j, \left[\mathbf{u}_i^2 \right]_j, \ldots, \left[\mathbf{u}_i^{\lceil L/S \rceil} \right]_j \right\}. \tag{46}$$

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

Lemma E.1. *For every $\mathbf{u}_i, \mathbf{v}_i \in \mathbb{F}_q^L$, for any $\mathcal{P} \subset C$ such that $|\mathcal{P}| \le T$, 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

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

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, $1 \le \ell \le \lceil L/S \rceil$, 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 5.2 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 $\text{REAL}_{\mathcal{M}}^{C,T,\lambda}(\mathbf{u}_C, C_0, C_1, C_2)$, 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 $C_1 \setminus \mathcal{M}$ so that instead of using KA.agree$(\text{sk}_i, \text{pk}_j)$ to encrypt and decrypt messages, we run the key agreement simulator $\text{Sim}_{KA}(s_{i,j}, \text{pk}_j)$, 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 $i \in C_1 \setminus \mathcal{M}$, 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 $i \in C_1 \setminus \mathcal{M}$, 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 $\{\mathbf{v}_i\}_{i \in C_1 \setminus \mathcal{M}}$ are chosen uniformly at random. Otherwise, $\{\mathbf{v}_i\}_{i \in C_1 \setminus \mathcal{M}}$ are chosen uniformly at random subject to $\sum_{i \in C_1 \setminus \mathcal{M}} \mathbf{v}_i = \mathbf{z} \left(= \sum_{i \in C_1 \setminus \mathcal{M}} \mathbf{u}_i \right)$. The joint view of corrupt parties contains only $|\mathcal{M}| \le T$ shares of each $\mathbf{v}_i$. From Lemma E.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 $\mathbf{z}$, C_0, and C_1. In the following hybrids, we assume $\mathbf{z} \neq \perp$.

Hybrid$_4$: Partition $\mathbf{z}$ into $\lceil L/S \rceil$ vectors, $\mathbf{z}^1, \mathbf{z}^2, \ldots, \mathbf{z}^{\lceil L/S \rceil}$, each of length at most S. In this hybrid, for every client $i \in C_2 \setminus \mathcal{M}$ 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 $\mathbf{z} = \sum_{i \in C_1 \setminus \mathcal{M}} \mathbf{u}_i$, from (43) and (44), it follows that the distribution of $\left\{ \left[\mathbf{z} + \sum_{j \in \mathcal{M}} \mathbf{u}_j \right]_i \right\}_{i \in C_2}$ is identical to that of $\{\text{sh}_i\}_{i \in C_2}$. 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 $\mathbf{z}$, $\mathbf{u}_{\mathcal{M}}$, C_0, C_1, and C_2. The argument above proves that

the output of the simulator is computationally indistinguishable from the output of REAL, which concludes the proof.

F COMPARISON OF FASTSHARE WITH SHAMIR'S SCHEME AND RELATION TO LOCALLY RECOVERABLE CODES

Comparison with the Shamir's scheme: A multi-secret sharing variant of the Shamir's scheme can be constructed as follows (see, e.g., [16]). We consider the case of generating N shares (each in $\mathbb{F}_q$, $q \geq N$) from S secrets (each in $\mathbb{F}_q$) such that the privacy threshold is T and dropout tolerance is $N - S - T$. Construct a polynomial of degree-$(S + T - 1)$ with the secrets and T random masks (each in $\mathbb{F}_q$) as coefficients. Its evaluations on N distinct non-zero points in $\mathbb{F}_q$ yield the N shares for the Shamir's scheme.

Note that it is possible to recover the secrets from any $S+T$ shares via polynomial interpolation. Although there exist fast polynomial interpolation algorithms that can interpolate a degree-n polynomial in $O\left(n \log^2 n\right)$ time (see, e.g., [10, 20, 25]), the actual complexity of these algorithms is typically large due to huge constants hidden by the big-Oh notation as noted in [10, 20, 30]. This limits the scalability of such fast interpolation algorithms.

Since the shares generated by Shamir are evaluations of a degree-$(S+T-1)$ polynomial on N distinct non-zero points, it is easy to see that the shares form a codeword of a Reed-Solomon code of block-length N and dimension $S + T$. On the other hand, FastShare constructs a polynomial of degree-$(N - 1)$, and evaluates it on primitive N-th roots of unity. The shares generated by FastShare form a codeword of a product code with Reed-Solomon codes as component codes (see Remark 1).

Suppose we choose primitive N-th roots of unity as the evaluation points for the Shamir's multi-secret sharing scheme. Then, the Shamir's scheme essentially computes the Fourier transform of the "signal" consisting of S secrets, concatenated with T random masks followed by $N - S - T$ zeros. In contrast, in case of FastShare, we construct the signal by judiciously placing zeros, which ensures the product code structure on the shares.

Relation to LRCs: The variant of FastShare described in 3.4 is related to a class of erasure codes called Locally Recoverable Codes (LRCs) (see [21, 37, 44], and references therein). Specifically, consider LRCs with (ℓ, r) locality [37] in which coordinates can be partitioned into several subsets of cardinality ℓ such that the coordinates in each subset form a maximum distance separable (MDS) code of length ℓ and dimension r. An efficient construction of such LRCs is presented in [44, Section V-C]. Note that the shares generated by FastShare (as described in 3.4) form codewods of an LRC with $(n_0, (1 - \delta_0)n_0)$ locality. Moreover, it is not difficult to show that FastShare as an *evaluation code* is isomorphic to the LRC constructed in [44, Section V-C] when the *good polynomial* is x^ℓ (see [44, Sections III-A] for the definition of a good polynomial) and the evaluation points are primitive N-th roots of unity.