Armadillo: Robust Secure Aggregation for Federated Learning with Input Validation

Conference Paper · June 2025

CITATIONS
0

READS
5

4 authors, including:

Harish Karthikeyan
Jpmorgan Chase & Co.
10 PUBLICATIONS **46** CITATIONS
SEE PROFILE

Antigoni Polychroniadou
Aarhus University
51 PUBLICATIONS **677** CITATIONS
SEE PROFILE

Armadillo: Robust Secure Aggregation for Federated Learning with Input Validation

Yiping Ma, Yue Guo, Harish Karthikeyan, Antigoni Polychroniadou
University of Pennsylvania, JP Morgan AI Research and AlgoCRYPT CoE
yipingma@seas.upenn.edu
yue.guo@jpmchase.com
harish.karthikeyan@jpmorgan.com
antigoni.polychroniadou@jpmorgan.com

Abstract

This paper presents a secure aggregation system Armadillo that has disruptive resistance against adversarial clients, such that any coalition of malicious clients can affect the aggregation result only by misreporting their private inputs in a pre-defined legitimate range. Armadillo is designed for federated learning environments, where a single powerful server interacts with numerous weak clients iteratively to train models on client's private data. While a few prior works consider disruption resistance under such setting, they either require high client costs that is quadratic in n (Chowdhury et al. CCS '22) or concretely high number of rounds that is logarithmic in n (Bell et al. USENIX Security '23) for an aggregation on n clients. Although disruption resistance can be achieved generically with zero-knowledge proof techniques (which we also use in this paper), we realize an efficient system with two new designs: 1) a simple two-layer aggregation protocol that requires only simple arithmetic computation; 2) an agreement protocol that removes the effect of malicious clients from the computation with low round complexity. With these techniques, Armadillo has only 3 rounds (our round complexity is independent of n) with lightweight server and clients.

1 Introduction

Can a server aggregate private data from many clients without learning any individual's data? And can this be done when clients *arbitrarily disrupt* the aggregation? In this work, we provide an affirmative answer to these questions with a fast aggregation system with strong privacy and robustness guarantees, even when we only have a single untrusted server coordinating the aggregation.

While the secure aggregation problem is decades old, there is a renewed interest motivated by the emergence of federated learning [39]: such a setting has a particular communication model and system constraints—*a central powerful server interacts with many weak clients*. Here, "powerful" vs. "weak" is with respect to computation power, communication bandwidth, and availability. Each client has a locally trained model (which can be viewed as a high-dimensional vector), and the server should learn the sum of these models (the sum of the vectors component-wise) but nothing else.

This setting makes secure aggregation challenging due to the likelihood of disruptive behavior from adversarial clients in a large participant pool. Many existing aggregation systems [1,17] that tolerates disruption work under a strictly weaker trust model than the setting we consider here. They require two or more non-colluding servers with at least one being trusted to achieve their goals, yet, in real-world federated learning deployment the single-server architecture seems to be the only choice: an organization that runs the training tasks either internally operate their own servers while ensuring isolation among the servers, which is rarely realistic; or they set up external servers elsewhere, which

Submitted to 38th Conference on Neural Information Processing Systems (NeurIPS 2024). Do not distribute.

introduces significant engineering overhead and operational risk. Indeed, industry precedent has consistently relied on the single-server model [39,9]. The few single-server aggregation solutions that tolerate disruption unfortunately have high costs: Chowdhury et al. [15] has a per-client computational workload quadratic in the number of clients, and Bell et al. [5] despite its modest client cost has too many rounds to be efficient—a single aggregation takes logarithmic in the number of clients and concretely 10–20 rounds. Some other works [5,37] settle for a relatively weak guarantee where the aggregation has to abort once disruption is detected. That is, there was essentially no affordable disruptive-resistant secure aggregation schemes under practical adversarial models.

This gap drives our work: our view is that ensuring privacy for clients while resisting disruption can be achieved in only 3 rounds, even in the presence of an adversary controlling the server and a subset of clients (up to some threshold). Our system Armadillo guarantees the following properties: 1) privacy, i.e., the server at most *learns the sum of inputs* from clients but nothing else, 2) robustness, i.e., the server, if following our protocol, is *ensured to get the sum* regardless how clients passively drop out or actively disrupt the aggregation. (A malicious client can also use invalid input to disrupt the result, but we show that our system can seamlessly integerate with existing efficient input validation techniques, resulting in a complete disruption-resistant system.) Armadillo's round complexity reduction does not come with a price on computational time: even when integrated with input validation techniques, the concrete computation at the clients and server is on par with ACORN-robust. Varying the number of clients and the fraction of adversarial clients, Armadillo outperforms ACORN-robust by $3\times$ to $7\times$ in rounds.

To achieve disruption resistance, Armadillo uses a generic paradigm: take a secure aggregation protocol, the clients prove that every step of their execution has been done correctly and the server verifies the proof. The core challenge is to make this efficient because cryptographic proofs are expensive; in fact, this is more challenging than it may seem—most of the prior works [37,5,15] making strides toward robustness do not follow this paradigm. Two key design ideas help us achieve lightweight clients and server. First, we design a secure aggregation protocol in which the bulk of computation is a simple linear computation, and importantly, it is sufficient to get robustness as long as the clients prove the correctness of the linear part (which is computationally efficient). Then, we structure all these proof statements (together with the input validation) as a single inner-product relation, so that with existing proof systems [11], the server can batch verify n proofs at a logarithmic cost in n.

Armadillo additionally has other beneficial properties in federated learning: many aggregations followed by a one-time setup, stateless participation of clients. We detail them in Section 1.3. Below, we give the technical contributions of our work and a more detailed comparison with prior works.

1.1 Our technical contributions

Efficiency. In Figure 1, we give the asymptotic cost of Armadillo compared with prior works with similar properties. The only prior work that has reasonable client cost and achieves the same property as ours is ACORN-robust [5], in which an aggregation on n clients takes $6+O(\log n)$ communication rounds between server and all clients. Armadillo has three rounds regardless of the number of clients, keeping the asymptotic computation and communication cost on par with the best prior result. As evidenced in prior works [38], round complexity is critical for end-to-end run time in the federated learning model.[1] In various circumstances, we have 3–$7\times$ concrete improvement on communication rounds, translating to up to $6\times$ improvement on run time for computing a sum. Our competitive advantage over ACORN-robust becomes more significant with more clients or a higher corruption rate among clients. These concrete improvements are detailed in Section 6.

Technical novelty. We utilize a key-and-message homomorphic encryption scheme to build a simple two-layer secure aggregation protocol: the clients first send encryption of their inputs, and the server sums up these ciphertexts (the *outer* layer). Then, it runs another aggregation on the keys to decrypt the sum of the ciphertexts (the *inner* layer). Note that the keys are much smaller than the inputs in federated learning; this two-layer paradigm reduces our original problem to a secure aggregation with smaller inputs. While the tool of key-and-message homomorphism has appeared

[1] In each communication round, the server has to wait until it receives a desired proportion of the client's responses. If a protocol has many rounds, the time spent on waiting may dominate over the actual computation cost.

	Client comm.	Client comp.	Server comm.	Server comp.	Rounds	Robustness	Input Val.
Effiel [15]	ℓn^2	ℓn^2	ℓn^3	ℓn^3	4	✓	Generic
RoFL [37]	$\ell + \log n$	$\ell \log n$	$\ell n + n \log n$	ℓn	6	×	L_2, L_∞
ACORN-detect [5]	$\ell + \log n$	$\ell \log n$	$\ell n + n \log n$	ℓn	7	×	L_2, L_∞
ACORN-robust [5]	$\ell + \log^2 n$	$\ell \log n + \log^2 n$	$\ell n + n \log^2 n$	$\ell n + n \log^2 n$	$6 + \log n$	✓	L_2, L_∞
Flamingo [38]	Regular: $\ell + C$ Decryptor: $n + C$	Regular: $\ell + C$ Decryptor: $n + C$	$\ell n + Cn$	$\ell n + Cn$	3	×	N/A
Armadillo (this work)	Regular: $\ell + C$ Decryptor: $n + C$	Regular: $\ell + C$ Decryptor: $n + C$	$\ell n + Cn$	$\ell n + Cn$	3	✓	L_2, L_∞

Figure 1: Asymptotic communication and computation cost for one aggregation for input vector length ℓ and n clients. For simplicity, we omit the asymptotic notation $O(\cdot)$ in the table. In practice we have $n < \ell$ (§1.2). The round complexity excludes any setup that is one-time. The round complexity of ACORN-detect are counted using the fixed version (Appendix C.2). For the protocols using the ideas of sub-sampling clients ("decryptors"), we denote the number of sampled clients as C where $C = o(n)$. In Flamingo, decryptor has asymptotic cost slightly larger than n when dropouts happen. Eiffel has a different communication model from all the other works: it assumes a public bulletin board and all clients in the protocol post messages on the board. Note that this table shows only asymptotics; protocols with matching asymptotics can exhibit markedly different concrete performance depending on the specifics of their cryptographic design.

in the secure aggregation context [36,35,5,49], our design is concretely different from prior works, and we address the robustness challenge (that almost none of these works have) by adding only lightweight components to the aggregation protocol.

At a high level Armadillo works as follows. We instantiate both the outer and inner layers with primarily linear computation; this allows the clients to generate low-cost proof of computation. We use commitment to bind the two layers, ensuring that the key underlying the ciphertext in the outer layer is indeed the input to the inner layer. Crucially, all these must be done in a concretely feasible way for computationally weak clients. Our technical overview is provided in Section 3.

Provable security. We prove simulation-based privacy under our threat model (§1.2) where an honest-but-curious server colludes with a subset of clients; this is the same threat model as ACORN-robust [5]. We show that Armadillo has robustness (Def.3) against any coalition of adversarial clients, and combined with existing input validation techniques, we get disruption resistance (Def.4). While some existing protocols [5,37] claimed privacy against a malicious server, these protocols are not provably secure when adding in their input validation techniques; some of them indeed do not include a security proof. We argue that any secure aggregation protocol with a security proof relying on the programmability of the random oracle model cannot incorporate input validation techniques that require proving the correct computation of a later programmed value. We discuss this in detail in Section 4.

1.2 Problem statement

In this section, we formally describe our problem setting. A training process consists of T iterations, running between the server and in total N clients. Each iteration has the same procedure: n clients (indexed from 1 to n) are selected from the N clients[2], where client i holds vector $\mathbf{x}_i$, and the goal is to let the server only learns the sum $\sum_{i=1}^{n} \mathbf{x}_i$ and what can be implied by the sum.

In practice, a sum of all the n clients is hard to guarantee as some clients (even if they are honest) may stop responding to the server during protocol execution (e.g., due to power failure or unstable connection); this is usually referred as *passive dropouts* [9]. The server must continue without waiting for them to return; otherwise, the training might be blocked for an unacceptable duration. Also, a malicious client can *actively deviate* from the protocol (e.g., sending incorrect messages, using invalid inputs). Therefore, a more precise goal in this paper is to compute the sum of the input vectors from the largest possible set of honest and online clients. We first introduce our setting below and then formally describe the desired properties.

[2]How to select the clients per iteration depends on the training design and is orthogonal to secure aggregation problem (details in §4). Typically, the same number of clients are randomly selected in each iteration.

Functionality $\mathcal{F}$

Parties: A set of n clients $P_1, \ldots, P_n$ and a server S.
Notation: Let corruption rate be η and dropout rate be δ, both among $\mathcal{P} = \{P_1, \ldots, P_n\}$. Let Adv be the adversary corrupting S and the set of clients of size ηn, Cor.

- $\mathcal{F}$ and Adv receive a set of dropout clients $\mathcal{O} \subset \mathcal{P}$ where $|\mathcal{O}|/|\mathcal{P}| \leq \delta$. $\mathcal{F}$ receives $\mathbf{x}_i$ of client $P_i \in \mathcal{P} \backslash \mathcal{O}$.
- $\mathcal{F}$ asks Adv for a set $\mathcal{E}$ with the requirements that: $|\mathcal{E} \cup \mathcal{O}|/n \leq \delta$.
- If Adv replies $\mathcal{F}$ with a set $\mathcal{E}$ that satisfies the requirement, then $\mathcal{F}$ outputs $\mathbf{z} = \sum_{i \in \mathcal{P} \backslash (\mathcal{E} \cup \mathcal{O} \cup \mathsf{Cor})} \mathbf{x}_i$ to Adv. Otherwise, $\mathcal{F}$ send `terminate` to all parties.

Figure 2: Ideal functionality for one aggregation. We follow the definition in prior works [9,6] assuming an oracle gives a dropout set to $\mathcal{F}$ and adversary Adv can also query the oracle.

Threat model. We assume a static adversary that corrupts the server and up to η^* fraction of the N clients. We assume the server is honest-but-curious: it follows the protocol but tries to learn client's inputs. For each aggregation on n clients, we assume η fraction of them are corrupted, and up to δ fraction of may passively drop out (excluding the adversarial clients who on purpose drop out).

Armadillo requires sub-sampling a set $\mathcal{C}$ from N clients during setup to assist secure aggregation. Let $\eta_{\mathcal{C}}$ and $\delta_{\mathcal{C}}$ denote the corruption and dropout rates in $\mathcal{C}$. The protocol ensures correctness under the conditions $\delta + \eta < 1/3$ for n clients and $\delta_{\mathcal{C}} + \eta_{\mathcal{C}} < 1/3$ for $\mathcal{C}$. The sampling algorithm guarantees the latter if η^* remains within specific bounds (§3.5). For our protocol specifically, the efficiency varies with parameter values; for example, when $\eta = 1\%$, the protocol is more efficient than at $\eta = 5\%$ (§4).

Communication model. Clients are heterogeneous devices with varying reliability (e.g., cellphones, laptops) and may stop responding due to device or network failures. We assume an implicit distribution for client response times.

Each client communicates with the server through a private, authenticated channel. Private messages between clients are relayed via the server and encrypted under the recipient's public key (assuming a public key infrastructure, PKI, as in prior works [9,6,38]). Public messages are signed with the sender's signing key (derived from the PKI).

Our protocol proceeds in *rounds*, starting with the server. A round trip involves the server sending messages to clients, waiting for a fixed time to collect responses in a message pool, processing them, and proceeding to the next round.

1.3 Properties

We formally give the properties that we aim to achieve in Armadillo for a single aggregation (computing one sum over n clients). Discussion for computing multiple sums over different sets of clients is given in Section 4.

Definition 1 (Privacy)**.** We say an aggregation protocol has privacy against a semi-honest adversary if the protocol realizes the ideal functionality in Figure 2.

Definition 2 (Dropout resilience)**.** We say an aggregation protocol on n clients with inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$ has dropout resilience if, when all clients follow the protocol and a set $\mathcal{X} \subseteq [n]$ of clients remains online throughout the aggregation, the server should output $\sum_{i \in \mathcal{I}} \mathbf{x}_i$ where $\mathcal{X} \subseteq \mathcal{I} \subseteq [n]$.

The salient aspect of Armadillo is disruption resistance: the coalition of malicious clients can affect the aggregation result only by misreporting their private inputs. This is formalized in Definition 3 and 4 below. A pre-condition for disruption resistance is that this protocol can be completed regardless how adversarial clients act.

Definition 3 (Robustness)**.** Let f be an aggregation function that takes in n inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Let $\mathcal{Z}$ be the set of all possible inputs for f. We say that an n-client aggregation protocol has *robustness* if, when the server follows the protocol, for every number of m adversarial clients (with $0 \leq m \leq n$)

and for every choice of honest client's inputs $\mathcal{I}_{honest} \in \mathcal{Z}^{n-m}$, the protocol always outputs to the server a value in the set $\{f(\mathcal{I}_{honest}, \mathcal{I}_{adv}) \mid \mathcal{I}_{adv} \in \mathcal{Z}^{n-m}\}$.

Definition 4 (Disruption resistance [17])**.** Let f be an aggregation function that takes in n inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n$. We say a protocol has *disruption resistance* if we restrict $\mathcal{Z}$ in Definition 4 to be a user-defined set of valid inputs ($\mathbf{x}$ is a valid input for f if $\mathbf{x} \in \mathcal{Z}$) and the protocol still satisfies Definition 4.

In this work, we consider the aggregation function f to be a simple sum function, and $\mathcal{Z}$ to be all possible vectors with application-defined L_2, L_∞ norms.

2 Preliminaries

Notation. Let $[z]$ denote the set $\{1, 2, \ldots, z\}$. We use $[a, b]$ to denote the set $\{x \in \mathbb{N} : a \leq x \leq b\}$. We use bold lowercase letters (e.g. $\mathbf{u}$) to denote vectors and bold upper case letters (e.g., $\mathbf{A}$) to denote matrices. Unless specified, vectors are column vectors. Given a value α and a vector $\mathbf{v}$, we use $\alpha\mathbf{v}$ to denote multiplying α to every coordinate of $\mathbf{v}$. For distribution $\mathcal{D}$, we use $a \leftarrow \mathcal{D}$ to denote sampling a from $\mathcal{D}$. For a vector $\mathbf{v}$, we use $\lfloor \mathbf{v} \rceil_c$ to denote rounding each entry of $\mathbf{v}$ to nearest multiples of c. For two vectors $\mathbf{v}_1$ of length ℓ_1, $\mathbf{v}_2$ of length ℓ_2, we use $\mathbf{v}_1|\mathbf{v}_2$ to denote the concatenation of them which is a vector of length $\ell_1 + \ell_2$. We use $\|\mathbf{v}\|_2$ to denote L_2 norm of $\mathbf{v}$ and use $\|\mathbf{v}\|_\infty$ to denote the largest entry in $\mathbf{v}$. We use $\mathbb{F}$ to denote a field.

Regev's encryption. Our construction utilizes the key-and-message homomorphism Regev's encryption [45]; we give the details below. The Regev's scheme is parameterized by a security parameter λ, a plaintext modulus p, and a ciphertext modulus q, and number of LWE samples m. Given a secret key $\mathbf{s} \xleftarrow{\$} \mathbb{Z}_q^\lambda$, the encryption of a vector $\mathbf{x} \in \mathbb{Z}_p^m$ is

$$(\mathbf{A}, \mathbf{c}) := (\mathbf{A}, \mathbf{A}\mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{x}),$$

where $\mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m \times \lambda}$ is a random matrix ($m > \lambda$), $\mathbf{e} \xleftarrow{\$} \chi^m$ is an error vector and χ is a discrete Gaussian distribution, and $\Delta := \lfloor q/p \rfloor$. Decryption is computed as $(\mathbf{c} - \mathbf{A}\mathbf{s}) \bmod q$ and rounding each entry to the nearest multiples of Δ, and then divide the rounding result by Δ. The decrypted result is correct if entries in $\mathbf{e}$ are less than $\Delta/2$.

Packed secret sharing. In standard Shamir secret sharing [46], a secret $\rho \in \mathbb{F}$ is hidden as the constant term of a polynomial $p(x) = a_0 + a_1 x + \cdots + a_t x^d$ where $a_0 = \rho$ and $a_1, \ldots, a_d$ are randomly sampled from $\mathbb{F}$. Given n parties, the share for party $i \in [n]$ is $p(i)$, and any subset of at least $d + 1$ parties can reconstruct ρ and any subset of d shares are independently random.

In packed secret sharing [24], one can hide multiple secrets using a single polynomial. Specifically, let $\mathbb{F}$ be a field of size at least $2n$ and k be the number of secrets packed in one sharing. Packed Shamir secret sharing of $(v_1, \ldots, v_k) \in \mathbb{F}^k$ first chooses a random polynomial $p(\cdot) \in \mathbb{F}[X]$ of degree at most $d + k - 1$ subject to $p(0) = v_1, \ldots, p(-k + 1) = v_k$, and then sets the share ρ_i for party i to be $\rho_i = p(i)$ for all $i \in [n]$. Reconstruction of a degree-$(d + k - 1)$ sharing requires at least $d + k$ shares from $\rho_1, \ldots, \rho_n$. Note that the corruption threshold is now d even if the degree is $d + k - 1$, i.e., any d shares are independently random, but any $d + 1$ shares are not.

Shamir sharing testing. Looking ahead, we will also use a probabilistic test for Shamir's secret shares, called SCRAPE test [12]. To check if $(\rho_1, \ldots, \rho_n) \in \mathbb{F}^n$ is a Shamir sharing over $\mathbb{F}$ of degree d (namely there exists a polynomial p of degree $\leq d$ such that $p(i) = \rho_i$ for $i = 1, \ldots, n$), one can sample $w_1, \ldots, w_n$ uniformly from the dual code to the Reed-Solomon code and check if $w_1\rho_1 + \cdots + w_n\rho_n = 0$ in $\mathbb{F}$.

To elaborate, let $c_i := \prod_{j \in [n] \setminus \{i\}} (i - j)^{-1}$ and $m^*(X) := \sum_{i=0}^{n-d-2} m_i \cdot X^i \leftarrow_\$ \mathbb{F}[X]_{\leq n-d-2}$ (a random polynomial over $\mathbb{F}$ of degree at most $n - d - 2$). Now, let $\mathbf{w} := (c_1 \cdot m^*(1), \ldots, c_n \cdot m^*(n))$ and $\boldsymbol{\rho} := (\rho_1, \ldots, \rho_n)$. Then,

- If there exists $p \in \mathbb{F}[X]_{\leq d}$ such that $\rho_i = p(i)$ for all $i \in [n]$, then $\langle \mathbf{w}, \boldsymbol{\rho} \rangle = 0$.
- Otherwise, $\Pr[\langle \mathbf{w}, \boldsymbol{\rho} \rangle = 0] = 1/|\mathbb{F}|$.

In other words, if $(\rho_1, \ldots, \rho_n)$ is not a Shamir sharing of degree d then the test will only pass with probability $1/|\mathbb{F}|$.

Pedersen and vector commitment. Let $\mathbb{G}$ be a group of order q, and G, H be two generators in $\mathbb{G}$. A Pedersen commitment to a value $v \in \mathbb{Z}_q$ is computed as $\mathsf{com}_G(v) := G^v H^r$, where the commitment randomness r is uniformly chosen from $\mathbb{Z}_q$. We use $\mathsf{com}_G(\cdot)$ notation because later in our protocol we compute commitments with different generators.

We can also commit to a vector $\mathbf{v} = (v_1, \ldots, v_L) \in \mathbb{Z}_q^L$ as follows: let $\mathbf{G} = (G_1, \ldots, G_L)$ be a list of L random generators in $\mathbb{G}$, define $\mathsf{com}_{\mathbf{G}}(\mathbf{v}) := G_1^{v_1} \cdots G_L^{v_L} \cdot H^r$, where r is randomly chosen from $\mathbb{Z}_q$; our notation $\mathsf{com}_{\mathbf{G}}(\cdot)$ implictly assumes a public H and a private r are included. In a special case that we will get to in Section 3.2 and 3.3, we do not include randomness in the commitment.

Inner-product proof. The inner product argument is an efficient proof system for the following relation: given two vector commitments $\mathsf{com}(\mathbf{a}), \mathsf{com}(\mathbf{b})$ known to both prover and verifier and a public value c, the prover can convince the verifier that $\langle \mathbf{a}, \mathbf{b} \rangle = c$. Bulletproof [11] gives non-interactive inner-product proof using Fiat-Shamir, with proof size $O(\log L)$ and prover/verifier cost $O(L)$.

For ease of presentation later, we introduce the following notations for proof. A proof system Π consists of a tuple of algorithms $(\mathcal{P}, \mathcal{V})$ run between a prover and verifier. An argument to prove can be described with public inputs/outputs io, a statement to be proved st, and a private witness wt. Given a proof system Π, the prover can generate a proof $\pi \leftarrow \Pi.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$ and the verifier checks the proof by $b \leftarrow \Pi.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi)$ where $b \in \{0, 1\}$ indicates rejecting or accepting π. For example, for proving inner product of $\mathbf{a}$ and $\mathbf{b}$, we set the constraint system to be

$$\{\mathsf{io} : (\mathsf{com}(\mathbf{a}), \mathsf{com}(\mathbf{b}), c),\ \mathsf{st} : \langle \mathbf{a}, \mathbf{b} \rangle = c,\ \mathsf{wt} : (\mathbf{a}, \mathbf{b})\}.$$

Denote the inner product proof system as Π_{ip}, the prover runs $\pi \leftarrow \Pi_{\text{ip}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$ and the verifier runs $b \leftarrow \Pi_{\text{ip}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$. The algorithms $\Pi_{\text{ip}}.\mathcal{P}$ and $\Pi_{\text{ip}}.\mathcal{V}$ both have complexity linear to the length of $\mathbf{a}$ (or $\mathbf{b}$) and π has logarithmic length of $\mathbf{a}$ (or $\mathbf{b}$). We will also prove *linear-relation*, and we denote the proof system as Π_{linear} and the constraint system will be

$$\{\mathsf{io} : (\mathsf{com}(\mathbf{b}), c),\ \mathsf{st} : \langle \mathbf{a}, \mathbf{b} \rangle = c,\ \mathsf{wt} : \mathbf{b}\}.$$

To differentiate the two proof systems, we call the former (that needs to commit to both vectors in the inner product) as quadratic proof and the later (that only needs to commit to one vector in the inner product) as linear proof.

3 Protocol design

Now we describe our construction for computing a single sum (one iteration in the training). Our full protocol is given in Figures 9 and 10 in Appendix C; below we describe our main technical ideas.

3.1 A two-layer secure aggregation

The key idea is to reduce an aggregation for long vectors to an aggregation for short vectors. To substantiate this idea, we utilize the key-and-message homomorphism of Regev's encryption.

Given two Regev ciphertexts $(\mathbf{A}, \mathbf{c}_1), (\mathbf{A}, \mathbf{c}_2)$ of vectors $\mathbf{x}_1, \mathbf{x}_2$ under the key $\mathbf{s}_1, \mathbf{s}_2$ with noise $\mathbf{e}_1, \mathbf{e}_2$, the tuple $(\mathbf{A}, \mathbf{c}_1 + \mathbf{c}_2)$ is an encryption of $\mathbf{x}_1 + \mathbf{x}_2$ under the key $\mathbf{s}_1 + \mathbf{s}_2$. The ciphertext $(\mathbf{A}, \mathbf{c}_1 + \mathbf{c}_2)$ can be properly decrypted if $\mathbf{e}_1 + \mathbf{e}_2$ is small. Note that computing $\mathbf{c}_1 + \mathbf{c}_2$ is very efficient—it is simply vector addition.

For ease of presentation later, we define a tuple of algorithms $(\mathsf{Enc}, \mathsf{Dec})$ parameterized by $(p, q, \lambda, m, \mathbf{A} \in \mathbb{Z}_q^{m \times \lambda})$ as follows:

- $\mathsf{Enc}(\mathbf{s}, \mathbf{x}) \to \mathbf{y}$: on input a secret key $\mathbf{s} \in \mathbb{Z}_q^\lambda$ and a message $\mathbf{x} \in \mathbb{Z}_p^m$, output $\mathbf{y} := \mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{x}$, where $\Delta = \lfloor q/p \rfloor$.
- $\mathsf{Dec}(\mathbf{s}, \mathbf{y}) \to \mathbf{x}'$: on input a secret key $\mathbf{s} \in \mathbb{Z}_q^\lambda$ and a ciphertext $\mathbf{y} \in \mathbb{Z}_q^m$, output $\mathbf{x}' := \lfloor \mathbf{y} - \mathbf{A}\mathbf{s} \rceil_\Delta$.

Now we decribe how our protocol work at a high level. Each client $i \in [n]$, holding an input vector $\mathbf{x}_i \in \mathbb{Z}_p^\ell$, samples a Regev encryption key $\mathbf{s}_i \in \mathbb{Z}_q^\lambda$ and sends the encrypted vector $\mathbf{y}_i = \mathsf{Enc}(\mathbf{s}_i, \mathbf{x}_i) := \mathbf{A}\mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i$ to the server. Note that $\lambda \ll \ell$. The server computes the sum of $\mathbf{y}_i$'s as

$$\mathbf{y} := \sum_{i\in[n]} \mathbf{y}_i = \sum_{i\in[n]} \mathbf{A}\mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i = \mathbf{A} \sum_{i\in[n]} \mathbf{s}_i + \Delta \sum_{i\in[n]} \mathbf{x}_i + \sum_{i\in[n]} \mathbf{e}_i,$$

To reconstruct $\sum_{i\in[n]} \mathbf{x}_i$, the server needs $\mathbf{s} := \sum_{i=1}^n \mathbf{s}_i$ to decrypt $\mathbf{y}$, and the decryption succeeds if $\sum_{i\in[n]} \mathbf{e}_i < \Delta/2$. We call the sum of Regev's ciphertexts $\mathbf{y}_i$'s as *outer aggregation*, and next we discuss *inner aggregation* where the server gets sum of $\mathbf{s}_i$'s.

The inner aggregation could be instantiated with a naive secure multi-party computation over the n clients: each client i secret shares $\mathbf{s}_i$ coordinate-wise to all the other clients (the shares are encrypted using public keys of the recipient clients and sent through the server), and each client adds up the shares which is then sent to the server for reconstruction of $\mathbf{s}$. However, this naive approach has per client communication $O(n\lambda)$ and server communication $O(n^2\lambda)$. The former is too much for a client given λ is a security parameter and is typically from a few hundreds to a thousand; and the latter is too much for the server because it is quadratic in n.

We reduce the communication complexity with two techniques: 1) let $C \ll n$, sample C clients from the whole population as *decryptors* to assist with unmasking the aggregation result[3]; 2) each client i uses packed secret sharing (§2) to share its secret vector $\mathbf{s}_i$ to the decryptors, so that each decryptor receives one share from a client. The combination of these two techniques result in communication complexity per regular client $O(C)$ and per decryptor $O(n)$, and the server communication complexity is $O(Cn)$. Although the decryptor cost is linear in n, we show in Section 6 that the cost is modest for the n needed in federated learning.

This inner-outer paradigm has a key advantage in handling dropouts, unlike the pairwise masking approach used in prior works [9,6,38] which incurs extra rounds. Specifically, if a client drops out during the outer aggregation (for sending $\mathbf{y}_i$), the server can safely ignore the client without affecting subsequent steps. If a decryptor client drops out during the inner aggregation, the server can still reconstruct $\mathbf{s}$ due to the threshold nature of secret sharing.

In the next few sections, we discuss how to make this simple protocol robust against adversarial clients (which may include some of the decryptors) in two parts: 1) proof of linear computation in outer aggregation and 2) an agreement protocol for inner aggregation.

Remark 1. As observed in a few works in orthogonal areas [27,20], Regev's encryption remains secure even if $\mathbf{A}$ is made public and the same matrix $\mathbf{A}$ is used to encrypt polynomially many messages, as long as the secret key $\mathbf{s}$ and the noise $\mathbf{e}$ are independently chosen in each instance of encryption. Therefore for our protocol, $\mathbf{A}$ can be generated by a trusted setup [19,16] (to reduce the cost of randomness generation, the trusted entity can generate a short random string and expand it to $\mathbf{A}$ using PRG). Since $\mathbf{A}$ can be reused, this only runs once. Also, a client i can pre-compute $\mathbf{A}\mathbf{s}_i$ prior to knowing their input $\mathbf{x}_i$.

3.2 Proof of client computation

Our high-level idea is "commit-and-proof": each client sends commitments to its private values (e.g., commitment to $\mathbf{s}_i$) together with a proof of the following relations. Let $\mathbf{F}, \mathbf{G}, \mathbf{H}$ be vectors of group generators in $\mathbb{G}$ of length λ, ℓ, ℓ respectively. Suppose client i sends to the server $\mathsf{com}_{\mathbf{F}}(\mathbf{s}_i), \mathsf{com}_{\mathbf{G}}(\mathbf{e}_i), \mathsf{com}_{\mathbf{H}}(\mathbf{x}_i)$, in addition to $\mathbf{y}_i$ as specified in the outer aggregation. The client proves to the server that:

1. For the outer aggregation, $\mathbf{y}_i := \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i \bmod q$, with $\mathbf{e}_i$ having small L_∞ norm.
2. For the inner aggregation, the client secret-shares $\mathbf{s}_i$ to the decryptors using a polynomial of degree d (the degree d is fixed by the threat model parameters, see §4).

Next, we express these requirements (except the norm condition which is non-linear) as inner-product relations. We will address proving the norms in Section 3.4. We set LWE modulus q to match the field size of the commit-and-proof system.

[3]We show that C can be polylogarithmic in n to have this work (§4).

Proving the first statement. At the first glance, we need to prove that each coordinate of $\mathbf{y}_i$ equals the corresponding coordinate of the RHS computation result. This would require ℓ proofs, one for each coordinate. We instead use a polynomial checking technique from Schwartz-Zippel Lemma to compress the proofs to a single one. In particular, if we want to check if two vectors of length ℓ over $\mathbb{Z}_q$ are equal, we view each vector (e.g., $\mathbf{y}_i$) as coefficients of a degree-ℓ polynomial and check if the evaluation of the two polynomials on a random point are equal. If they are indeed not equal, then the evaluation will be different except probability ℓ/q. Formally, let $r \in \mathbb{Z}_q$ be a random challenge value picked by the server (who is the verifier), and let $\mathbf{r} = (r^0, r^1, \ldots, r^{\ell-1})$. Let $c = \langle \mathbf{y}_i, \mathbf{r} \rangle$, and c is a public value since $\mathbf{y}_i$ and $\mathbf{r}$ are both public (known to both the client and the server). If the client can prove to the server that

$$c = \langle \mathbf{A}^\top \mathbf{r} \mid \mathbf{r} \mid \Delta \mathbf{r},\ \mathbf{s}_i \mid \mathbf{e}_i \mid \mathbf{x}_i \rangle \text{ in } \mathbb{Z}_q,$$

then the server will be convinced that $\mathbf{y}_i := \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i$, and there will only be ℓ/q probability that the client is dishonest but the server is convinced. The inner product argument comes from the following:

$$\begin{aligned}
\langle \mathbf{y}_i, \mathbf{r} \rangle =& \langle \mathbf{A}\mathbf{s}_i, \mathbf{r} \rangle + \langle \mathbf{e}_i, \mathbf{r} \rangle + \Delta \langle \mathbf{x}_i, \mathbf{r} \rangle \\
=& \langle \mathbf{A}^\top \mathbf{r}, \mathbf{s}_i \rangle + \langle \mathbf{r}, \mathbf{e}_i \rangle + \langle \Delta \mathbf{r}, \mathbf{x}_i \rangle \\
=& \langle \mathbf{A}^\top \mathbf{r} \mid \mathbf{r} \mid \Delta \mathbf{r},\ \mathbf{s}_i \mid \mathbf{e}_i \mid \mathbf{x}_i \rangle.
\end{aligned}$$

Also, note that $\mathbf{A}^\top \mathbf{r} \mid \mathbf{r} \mid \Delta \mathbf{r}$ is public, the client only needs to do a linear proof, where the witness is under the commitment $\mathsf{com}_{\mathbf{F}|\mathbf{G}|\mathbf{H}}(\mathbf{s}_i|\mathbf{e}_i|\mathbf{x}_i)$.

Proving the second statement. Recall that the client sends $\mathsf{com}_{\mathbf{F}}(\mathbf{s}_i)$ to the server in the outer aggregation, and now we want to ensure that the shares that the decryptors received (for the inner aggregation) are indeed the Shamir shares of this committed $\mathbf{s}_i$. Here we can exactly use the SCRAPE test (§2) to express this constraint as an inner product relation; this test seamlessly works with packed secret sharing.

Formally, supppose client i has a packed Shamir sharing of $\mathbf{s}_i$ as a vector of length C (recall that there are C decryptors)

$$\boldsymbol{\rho}_i = (\rho_i^{(1)}, \ldots, \rho_i^{(C)}),$$

which the client claims is a sharing of degree d over $\mathbb{Z}_q$. We observe that checking if $\boldsymbol{\rho}_i$ is a packed sharing of $\mathbf{s}_i$ is equivalent to checking if $(\boldsymbol{\rho}_i \mid \mathbf{s}_i)$ is a sharing of length $C+\lambda$ of a degree-d polynomial. Therefore, we let the client commit to $\boldsymbol{\rho}_i$ under a new generator vector $\mathbf{K} = (K_1, \ldots, K_C) \in \mathbb{G}^C$, and sends $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i)$ to the server in the outer aggregation as well. Then the client invokes a linear-relation proof that

$$\langle \boldsymbol{\rho}_i \mid \mathbf{s}_i,\ \mathbf{w} \rangle = 0 \text{ in } \mathbb{Z}_q,$$

where $\mathbf{w} := (w^{(1)}, \ldots, w^{(C+\lambda)})$ is sampled uniformly random from some code space (details in §2) and is public (known to both the server and client). In our setting, we cannot let the client choose $\mathbf{w}$ since the client may be malicious, so we apply the Fiat-Shamir transform and have client i derive $\mathbf{w}$ by hashing $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i) \cdot \mathsf{com}_{\mathbf{F}}(\mathbf{s}_i)$.

Up to this point, we have not guaranteed the shares received by the decryptors are consistent with the commitment $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i)$. The reason is that the client could in fact send to a decryptor a share (under the encryption) that is different from what was committed to. Therefore, instead of having the client send $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i)$ to the server, we have the client send commitments to each coordinate of $\boldsymbol{\rho}_i$, namely $\mathsf{com}_{K_1}(\rho_i^{(1)}), \ldots, \mathsf{com}_{K_C}(\rho_i^{(C)})$. Since the shares are random and in a sufficiently large space (§6), we will compute the commitment to a share ρ simply as K_j^ρ. The server can still verify the proof for packed sharing, as it can compute the vector commitment $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i)$ from the individual C commitments as

$$\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i) = \mathsf{com}_{K_1}(\rho_i^{(1)}) \cdots \mathsf{com}_{K_C}(\rho_i^{(C)}).$$

For those clients whose proofs are valid (for both Enc computation and SCRAPE test), the server forwards their commitments to the corresponding decryptors where $\rho_i^{(j)}$ is intended for the j-th decryptor. Then each decryptor j verifies if the received share (after decryption) is consistent with the commitment $\mathsf{com}_{K_j}(\rho_i^{(j)})$.

3.3 Agreement protocol for decryptors

So far, each decryptor can identify a set of *valid* shares, which are consistent with the vector commitments. However, this alone is not enough for the server to obtain the correct sum, as illustrated in the following example. Suppose there are three clients P_1, P_2, and P_3, with P_3 being malicious, and three decryptors H_1, H_2, and H_3, with H_3 being malicious. Each client shares a secret using 2-out-of-3 Shamir sharing. If P_3 sends a valid share to H_1 but an invalid share to H_2, the decryptors will form sets of clients with valid shares:

- H_1 forms $\{P_1, P_2, P_3\}$,
- H_2 forms $\{P_1, P_2\}$,
- H_3 can form any arbitrary set, e.g., $\{P_3\}$.

If each decryptor adds up the valid shares locally, the resulting values will not reconstruct the sum of the secrets from any set of clients. The server can only reconstruct a meaningful sum if the decryptors sum shares from the *same set* of clients, i.e., the sum of the secrets from that specific set.

Our goal is to ensure that all honest decryptors agree on the same set of clients with valid shares. Although this seems like a consensus problem that may require many rounds, we can utilize the server as the central coordinator where any decryptor can complain to the server about an invalid share from a malicious client. However, the server does not know whether this is a fake complaint from a malicious decryptor. To help the server distinguish between valid complaints and fake complaints, we use a simple proof of decryption to make complaints *verifiable*.

We let the clients in the first round send their shares to decryptors encrypted using public key encryption $\mathsf{AsymEnc}$, namely client i sends to decryptors j a ciphertext $\mathsf{AsymEnc}(PK_j, \rho_i^{(j)})$ via the server where PK is j's public key and $\rho_i^{(j)}$ is the share intended for j. If a decryptor j finds that the decryption result ρ is not consistent with the commitment then it sends a *verifiable complaint* to the server consisting of: the purported invalid share ρ (in the clear) and a zero-knowledge proof of decryption (proving that it knows the secret key that derives ρ). The server can verify the complaint: it knows the ciphertext $\mathsf{AsymEnc}(PK_j, \rho_i^{(j)})$ (from the first round), and the purported ρ, it just needs to be convinced that the client knows a secret key SK_j that decrypts this ciphertext to ρ. The server then informs the decryptors of the lying clients to remove them from the sets. At this point, all the honest decryptors agree on a same set. Now they can add up the shares of this set. Finally with the error correction of the Shamir shares from the decryptors, the server can reconstruct the sum even if there are bogus shares from malicious decryptors.

Now we describe how to instantiate this public key encryption and proof of decryption in a efficient way. Concretely, we can use RSA as follows. We choose too large primes p, q and let $N = pq$, and we choose an integer e such that e is coprime to $p-1$ and $q-1$. The public key is (n, e). The private key (n, d) where $d = e^{-1} \bmod N$. An integer $0 \le m < n$ is encrypted as $c = m^e \bmod n$ and the decryption is computed as $m = c^d \bmod n$. Then proof of decryption is exactly a proof of knowledge of discrete log: the decryptor proves that it knows d, i.e., $\mathrm{DL}_c(m) = d$.

To prevent the decryptor from lying about d, we let the decryptor commit to d by sending $a^d \bmod n$ to the server before the aggregation starts, where a is a generator in $\mathbb{Z}_n^*$. In addition to proof of decryption, the decryptor proves that $\mathrm{DL}_a(a^d) = \mathrm{DL}_c(m) \bmod n$.

3.4 Integrating with proof of norms

Our protocol can be seamlessly integrated with existing norm validation techniques [5,25]. Now we describe how a client proves a vector $\mathbf{x}_i$ has bounded L_2, L_∞ norms; this also applies to proving norms of the LWE error vector $\mathbf{e}_i$ that we mentioned earlier. We push the complete integrated proof to Appendix A.

Proof of L_∞ norm. Given a vector $\mathbf{a}$ of length ℓ, we want to prove that $\|\mathbf{a}\|_\infty < B$. The starting point in [25] to build an efficient proof is bypassing the decomposition of each coordinate (as in prior work [37]). There, to prove $v \in [0, 2^B - 1]$, the prover decomposes v into B binary values, denoted as $\mathbf{a} \in \mathbb{Z}_2^B$; and let $\mathbf{b} = (2^0, 2^1, \ldots, 2^{B-1})$ be a public vector. Then the prover proves that $\langle \mathbf{a}, \mathbf{b} \rangle = v$, and proves that every entry of $\mathbf{a}$ is in $\{0, 1\}$. This decomposition approach has two

drawbacks: 1) for communication, the client sends ℓ proofs; 2) for computation, the client computes ℓ vector commitments of length B, and since B is typically at least 16 and ℓ is large (see concrete examples in §1.2), this incurs a high cost.

We instead use a technique in [25,5] which builds proof of L_∞ norm on a length-ℓ vector with range B at a cost independent of B and proof size sublinear in ℓ, as long as $B \ll q$ where $\mathbb{Z}_q$ is the field that the proof system supports. The idea is to reduce this proof into a fast *approximate* proof on vectors where the bound intended to be proven is much looser than B. To start with, $\|\mathbf{a}\|_\infty < B$ is equivalent to $\mathbf{a}(B-\mathbf{a}) \geq 0$, and any non-negative integer can be written as a sum of four squares [4]. In $\mathbb{Z}_q$ a negative value means the value is between $q/2$ and q (treated as $-q/2$ to 0). The prover performs two steps:

1. Finds four vectors such that they add up to $\mathbf{a}(B-\mathbf{a})$, which is equivalent to finding $\mathbf{u}, \mathbf{v}, \mathbf{w}$ such that $\mathbf{u}\circ\mathbf{u} + \mathbf{v}\circ\mathbf{v} + \mathbf{w}\circ\mathbf{w} + \mathbf{a}'\circ\mathbf{a}' = -(B^2-2B+2)\mathbf{1}$ where $\mathbf{a}' = 2\mathbf{a} - (B-1)\mathbf{1}$.
2. Proves $\|\mathbf{a}'|\mathbf{u}|\mathbf{v}|\mathbf{w}\|_\infty < \sqrt{q}/4$ (i.e., the entries in the vectors are small that the step 1 computation does not wrap around in $\mathbb{Z}_q$).

The first relation can be reduced into an inner product using Schwartz-Zippel Lemma. Let r be a random value chosen after the witness $\mathbf{u}, \mathbf{v}, \mathbf{w}, \mathbf{a}'$ are committed, and let $\mathbf{r} := (r^0, r^1, \ldots, r^{\ell-1})$. If the prover can prove

$$\langle \mathbf{a}', \mathbf{a}'\circ\mathbf{r}\rangle + \langle \mathbf{u}, \mathbf{u}\circ\mathbf{r}\rangle + \langle \mathbf{v}, \mathbf{v}\circ\mathbf{r}\rangle + \langle \mathbf{w}, \mathbf{w}\circ\mathbf{r}\rangle = \langle \mathbf{c}, \mathbf{r}\rangle$$

where $\mathbf{c} = -(B^2-2B+2)\mathbf{1}$, then the relation in step 1 holds except probability ℓ/q. This relation can be further reduced to a single quadratic proof $\langle \mathbf{z}_1, \mathbf{z}_2\rangle = c$ for some $\mathbf{z}_1, \mathbf{z}_2$ of length 4ℓ and a public value c [5,25].

The second relation requires again a proof of L_∞ norm, but the essence is that the actual entries in $\mathbf{a}'$ (similarly $\mathbf{u}, \mathbf{v}, \mathbf{w}$) are much smaller than the bound $\sqrt{q}/4$ that we want to prove. This is exactly the *approximate proof* introduced by Gentry et al. [25]: given a vector $\mathbf{b}$ of length $\hat{\ell}$ where $\|\mathbf{b}\|_\infty < B$, and $B' \gg B$, proving $\|\mathbf{b}\|_\infty < B'$ can be much easier than proving $\|\mathbf{b}\|_\infty < B$. They give a protocol that proves $\|\mathbf{b}\|_\infty < B'$ using only a single linear proof of length $\hat{\ell}+\sigma$ where σ is a security parameter (App. A.0.1). We can use it as a black box and set $\mathbf{b} = \mathbf{a}'|\mathbf{u}|\mathbf{v}|\mathbf{w}$ and correspondingly $\hat{\ell} = 4\ell$. (One may wonder why an approximate proof by itself is not enough since we have a large q. We explain the reason in App. A.0.1.)

In sum, proving L_∞ norm of a length-ℓ vector requires a length-4ℓ quadratic proof (step 1), and a length-$(4\ell+\sigma)$ linear proof (step 2) where σ is a security parameter typically taken as 256.

Proving L_2 norm. Suppose the prover has a length-ℓ vector $\mathbf{a}$ and wishes to prove $\|\mathbf{a}\|_2 < B$. The prover finds four non-negative integers $\alpha_1, \alpha_2, \alpha_3, \alpha_4$ such that $(\alpha_1^2+\alpha_2^2+\alpha_3^2+\alpha_4^2) + \|\mathbf{a}\|_2 = B^2$. Let $\mathbf{u} = (\alpha_1, \alpha_2, \alpha_3, \alpha_4)$ and $\mathbf{v} = (\mathbf{a}|\mathbf{u})$. The prover does an inner product proof that $\langle \mathbf{v}, \mathbf{v}\rangle = B^2$. Also, the prover does an approximate proof that $\|\mathbf{v}\|_\infty < \sqrt{q/(\ell+4)}$. The bottleneck of this approach is the quadratic proof $\langle \mathbf{v}, \mathbf{v}\rangle = B^2$ which is more expensive than a linear proof. Below we describe a technique from Gentry et al. [25] that significantly reduces the quadratic proof cost.

The idea is to reduce proving L_2 norm on a long vector to proving L_2 norm on a short vector. Specifically, Gentry et al. [25] proposed a matrix projection technique: given a vector $\mathbf{a}$ of length ℓ, sample a matrix $\mathbf{R} \leftarrow \mathcal{D}^{256\times\ell}$ from a special distribution[5] $\mathcal{D}$, if $\mathbf{b} := \mathbf{R}\mathbf{a}$ has small L_2 norm, then with high probability $\mathbf{a}$ also has small L_2 norm. Therefore, we just need to invoke the above L_2 proof on $\mathbf{b}$ which is of length 256.

The above projection technique is correct when we work over integers, but if we work over $\mathbb{Z}_q$, $\mathbf{a}$ may have a large L_2 norm but $\mathbf{b}$ has a small L_2 norm. But this event can only occur when the entry of $\mathbf{a}$ is large enough so that when multiplied with $\mathbf{R}$, the values get wrapped around in $\mathbb{Z}_q$. Since $\mathbf{R}$

[4] This is also known as Lagrange's four-square theorem. Rabin and Shallit proposed randomized algorithms for computing a single representation for a given integer a as $a = \alpha_1^2+\alpha_2^2+\alpha_3^2+\alpha_4^2$ in $O(\log^2 a)$ time [43].

[5] The distribution $\mathcal{D}$ is: $\mathcal{D}(0) = 1/2$ and $\mathcal{D}(\pm 1) = 1/4$. Since a sample from $\mathcal{D}$ is binary, transmitting matrices $\mathbf{R}$ and $\mathbf{R}'$ incur very small communication costs. The row dimension 256 is chosen by Johnson-Lindenstrauss lemma [29] to ensure checking on the projected (short) vector is sufficient except with negligible probability.

consists of entries only from $\{-1, 0, 1\}$, we just still need an approximate proof for $\mathbf{a}$ to show that wrapping around does not happen. Recall that this approximate proof can be instantiated with a linear proof of length $\ell + \sigma$.

We push the details of the complete proof protocol Π_{enc} to Appendix A and summarize the cost in Lemma 1.

Lemma 1 (Cost of proof of encryption and input validity)**.** Given a set of parameters (λ, ℓ, q, C) and let $\mathbf{s} \in \mathbb{Z}_q^\lambda, \mathbf{s} \in \mathbb{Z}_q^C, \mathbf{M} \in \mathbb{Z}_q^{\lambda \times C}, \mathbf{x} \in \mathbb{Z}_q^\ell$. Let $\mathbb{G}$ be a group of size q. Let Δ be a constant. Let

$$\begin{aligned}\mathbb{CS}_{\text{enc}} : \{&\mathsf{io} : (\mathsf{com}(\mathbf{s}), \mathsf{com}(\mathbf{x}), \mathsf{com}(\mathbf{e})),\\ &\mathsf{st} : \mathbf{y} = \mathbf{A} \cdot \mathbf{s} + \mathbf{e} + \Delta \cdot \mathbf{x}, \|\mathbf{x}\|_2 < B_{\mathbf{x}}(L_2),\\ &\|\mathbf{x}\|_\infty < B_{\mathbf{x}}(L_\infty), \|\mathbf{e}\|_\infty < B_{\mathbf{e}}(L_\infty),\\ &\mathsf{wt} : (\mathbf{s}, \mathbf{x}, \mathbf{e})\}.\end{aligned}$$

There exist a commit-and-proof protocol Π_{enc} (Appendix 3.4) with group $\mathbb{G}$ of order q for proving the above statement with the following cost, dominated by the inner-product proof (either linear proof or quadratic proof) invocations:

- 1 length-σ quadratic proof,
- 2 length-$(\ell + \sigma)$ linear proof,
- 1 length-$(\ell + \lambda + 2\sigma)$ linear proof,
- 2 length-(4ℓ) quadratic proof,
- 2 length-$(4\ell + \sigma)$ linear proof,

where we omit the lower order terms and write e.g., $\sigma + 4$ as σ. The last two proofs are L_∞ proofs for $\mathbf{x}$ and $\mathbf{e}$.

3.5 Electing decryptors

The focus of previous sections is the aggregation protocol and we therefore assume the existence of a set $\mathcal{C}$ of decryptors with corruption rate $\eta_{\mathcal{C}}$ before the aggregation starts. In fact, $\eta_{\mathcal{C}}$ depends on the exogenous parameter η^* and how we sample $\mathcal{C}$. In this section, we describe the sampling and the relation between $\eta_{\mathcal{C}}$ and η^*. We will leave the analysis of how large the set $\mathcal{C}$ need to be in Section 4.

We use a sampling protocol by Alon et al. [3] which does not require additional assumptions like random beacon used in a prior work Flamingo [38]. Their core building block is Feige's election protocol [22]: suppose for now we have a public bulletin board, and we want to sample (roughly) X out of N clients. We initialize N/X bins on the bulletin board. Each client jumps into a bin independently at random (malicious clients may not do it randomly). Then we take the smallest bin as the decryptor set. This simple sampling actually ensures that when the total population has a corruption rate η, the sampled set (with size at most X) also has a corruption rate bounded by $\frac{\eta^*}{1-2\epsilon}$, except probability $N \cdot e^{\Omega(-\epsilon^4 X)}$ [22,33]. The protocol of Alon et al. [3] eliminates the need for the bulletin board and extends Feige to work exactly under our communication model (§1.2); their protocol is more sophisticated but the key take away is that: if we have corruption rate η^* over all clients, then their protocol will elect a set with corruption rate $\eta_{\mathcal{C}} \leq 2\eta^*$ except with negligible probability, as long as the target X is at least polylogarithmic of N.

4 Security analysis

In this section, we discuss how to select proper parameters for our protocol, and formally state the properties of Armadillo.

Parameters. The system Armadillo has a set of parameters listed below. First, n is the number of clients per round. (λ, ℓ, p, q) are LWE parameters (for the outer aggregation), (C, d, λ) are secret-sharing parameters (for the inner aggregation), where C is the number of shares (which equals to the number of decryptors), d is the degree of the secret-sharing polynomial and λ is the number of secrets. $B_{\mathbf{x}}(L_\infty), B_{\mathbf{x}}(L_2), B_{\mathbf{e}}(L_\infty)$ are bounds on norms. The parameters n and the norm bounds depends on the machine learning setting which is orthogonal to security analysis. For (λ, ℓ, p, q), we can choose any secure instance of LWE [2,18,13].

The choice of C and d is shown below. Recall that in our protocol (§3), each client secret-shares a vector of length λ using a polynomial of degree d. We must have

$$\begin{aligned} &d - \lambda > C \cdot \eta_{\mathcal{C}} && \text{by security of packed secret sharing,} \\ &d < C(1 - \delta_{\mathcal{C}}) && \text{necessary condition to reconstruct the secret.} \\ &C \cdot \eta_{\mathcal{C}} < \frac{(1-\delta_{\mathcal{C}})C - d + 1}{2} && \text{in order to do error correction.} \end{aligned}$$

We combine the equations and get

$$C\eta_{\mathcal{C}} + \lambda < d < C(1 - \delta_{\mathcal{C}} - 2\eta_{\mathcal{C}}). \tag{1}$$

we choose $C > \lambda/(1 - \delta_{\mathcal{C}} - 3\eta_{\mathcal{C}})$ and set d accordingly. We need to ensure C is at least polylogarithmic in the total population so that $\eta_{\mathcal{C}} \leq 2\eta$ (§3.5).

Theorem 1 formally states the properties of Armadillo.

Theorem 1. Let Φ be the protocol in Figures 9 and 10. Let $(\delta, \eta, \delta_{\mathcal{C}}, \eta_{\mathcal{C}})$ be threat model parameters defined in Section 1.2 and $\mathcal{C}$ is a randomly sampled set of clients prior to the aggregation. Let B_2, B_∞ be norm bounds we want to impose on client's inputs. Let (λ, ℓ, p, q) be LWE parameters. If (λ, ℓ, p, q) is a secure LWE instance, and $|\mathcal{C}| > \lambda/(1 - \delta_{\mathcal{C}} - 3\eta_{\mathcal{C}})$, then under the communication model defined in Section 1.2,

- Φ realizes ideal functionality $\mathcal{F}_{\text{sum}}$ (Fig.2) in the presence of a semi-honest adversary controlling the server, η fraction of n clients for each aggregation and $\eta_{\mathcal{C}}$ for $\mathcal{C}$.
- Φ satisfies dropout resilience in Definition 2.
- Φ satisfies disruption resistance in Definition 3, where the aggregation function f is a sum function and the set $\mathcal{Z}$ consists of all vectors with L_2 and L_∞ norms bounded by B_2 and B_∞ respectively.

Orthogonal security goals. In federated learning, the sum is computed multiple times on randomly sampled clients from the total population. Several works discuss attacks that are orthogonal to cryptographic design. So et al. [47] demonstrate that the server can infer client data by observing sums over many training iterations if the subset of clients in each iteration are not carefully chosen. They propose a partitioning strategy to mitigate this risk.

Pasquini et al.[41] show an attack where a malicious server can circumvent secure aggregation by sending inconsistent models to clients. To mitigate this, some works [41,38] suggest binding the model hash to pairwise masks, which cancel out if all clients share the same hash. Since our secure aggregation protocol does not rely on pairwise masking, we adopt a different approach: each client hashes the received model and sends the hash to the decryptors. The decryptors then perform a majority vote on the hashes and exclude shares from clients whose hashes do not match the majority.

Remark 2 (Privacy against a malicious server when input validation is required)**.** In a simulation-based proof, to prove privacy against the server, we need to simulate the server's view throughout the protocol execution. When the server is malicious, a challenge in the simulation is that the set of honest online clients is only determined after the corrupted server has seen the ciphertexts (crafted by the simulator). Consequently, the simulation could rely on the Programmable Random Oracle Model (PROM) to go through, where certain outputs of cryptographic primitives are re-programmed after this set has been determined. This is widely used in several prior works [6,38,31,7] to prove privacy.[6]

Their proof requires the honest clients to compute $\mathcal{H}(x) = y$ for a secret x. Later, the simulator programs the choice of this value, changing it from y to y'. However, this programming is incompatible with input validation where the clients prove the correct computation of $\mathcal{H}(x)$. This impossibility stems from the fact that if there exists a zero-knowledge proof system to prove that for secret witnesses x, y and some cryptographic primitive $\mathcal{H}$ (usually $\mathcal{H}$ is a PRG or hash function in prior works), then there needs to be a commitment to y which would enable the verifier to verify that the computation was indeed successful. However, if one were to program the value of $\mathcal{H}(x) = y'$ after the generation of the proof, the commitment to y now needs to commit to y', violating the binding property of commitment schemes.

[6] One could conceivably avoid this by relying simply on secret-sharing of the entire vector. However, this increases the communication to the decryptors clients, which is unfavorable.

While prior works with input validation such as RoFL [37] and ACORN-detect [5] claim to guarantee privacy against a malicious server, these protocols fall into this impossibility and hence are not provably secure; and indeed no formal proof was shown in these works.

5 Optimizations

Sparse LWE The bulk of server-side decryption lies in computing the matrix-vector product $\mathbf{A} \cdot \mathbf{s}$. When $\mathbf{A}$ is sparse, that is, most of its entries are zero, this computation can be significantly accelerated. To realize this, we can replace the standard LWE assumption used in Regev's encryption scheme (§2) with Sparse LWE assumption; by increasing the length of the secret (λ) to $1.5\times$ of its size in the LWE instance, we maintain the same security level with the LWE instance [28].

Multi-exponentiation Naively computing commitments to a length-m vector requires $m+1$ group exponentiations and m group multiplications (§2). We can reduce the number of group exponentiations to sublinear in m using the Pippenger algorithm, given in Lemma 2. In short, the Pippenger algorithm requires only a small number of expensive group exponentiation (e.g., σ is typically no larger than 256) by increasing the cheap group multiplication by a factor of σ. Therefore, we can use Pippenger for Pedersen vector commitment in any of our inner-product proofs.

Lemma 2 (Complexity of Pippenger algorithm [42,10,23])**.** Let $\mathbb{G}$ be a group of order $q \approx 2^\sigma$, and $G_1, \ldots, G_m$ be m generators of $\mathbb{G}$. Given $v_1, \ldots, v_m \in \mathbb{Z}_q$, Pippenger algorithm can compute $G_1^{v_1} \cdots G_n^{v_m}$ using $\frac{2\sigma m}{\log m}$ group multiplications and σ group exponentiations.

Optimistic batch verification There are two verification steps we can optimize. First, each decryptor verifies that each online client i's share ρ_i matches the commitment $\mathsf{com}_K(\rho_i) = K^{\rho_i}$ where K is the generator of $\mathbb{G}$. Doing this individually for each share requires up to n group exponentiations; exponentiation is an expensive operation. We can apply batch verification technique [8] to reduce the number of exponentiations to sublinear in n. Say the helper has shares $\rho_1, \ldots, \rho_n$ and it wants to batch verify if they match with $\mathsf{com}_K(\rho_1), \ldots, \mathsf{com}_K(\rho_n)$ respectively. It samples random values $\alpha_1, \ldots, \alpha_n$ in $\mathbb{Z}_q$ where q is group order and checks if $\alpha_1\rho_1 + \cdots + \alpha_n\rho_n$ matches $\mathsf{com}_K(\rho_1)^{\alpha_1} \cdots \mathsf{com}_K(\rho_n)^{\alpha_n}$. The batch verification only needs a length-n inner product on $\mathbb{Z}_q$ and a single length-n multi-exponentiation.

The batching technique can also be applied to proof verification at the server. For this we open the black box of the inner-product proof system and utilize a property of Bulletproof [11] that allows much faster batch verfication than verifying each proof individually. To verify a quadratic proof of length ℓ, the verifier computation can be abstracted as[7]

$$\mathbf{G}^{\mathbf{z}} \stackrel{?}{=} \mathbf{P}^{\mathbf{r}}, \tag{2}$$

where $\mathbf{G}$ is a vector of group generators of length-2ℓ and $\mathbf{P}$ is the proof transcript which consists of $2\log\ell$ group elements. The exponents $\mathbf{z}, \mathbf{r}$ only depend on the verifier challenges. Verifying n quadratic proofs naively would require the server to compute n multi-exponentiation of length 2ℓ (LHS of 2), and n multi-exponentiation of length $2\log\ell$ (RHS of 2).

The key idea is that the LHS of 2 can be computed for a batch of n proofs using a single multi-exponentiation instead of n multi-exponentiations; the insight is that all the proofs share the same generators $\mathbf{G}$. To batch verify n proofs with exponents $\mathbf{z}_1, \ldots, \mathbf{z}_n$, the verifier (server) can sample random $\alpha_1, \ldots, \alpha_n$ and computes $\mathbf{z}' = \alpha_1\mathbf{z}_1 + \cdots + \alpha_n\mathbf{z}_n$ and checks if

$$\mathbf{G}^{\mathbf{z}'} \stackrel{?}{=} \mathbf{P}_1^{\alpha_1\mathbf{r}_1} \cdots \mathbf{P}_n^{\alpha_n\mathbf{r}_n}, \tag{3}$$

where $\mathbf{P}_i$ and $\mathbf{r}_i$ are from the RHS of equation 2 for client i's proof. For LHS of 3, the server computes a single multi-exponentiation of length 2ℓ. For RHS of 3, the server computes a single multi-exponentiation of length $2n\log\ell$.

In both the above cases, if there exists malicious clients such that the batch verification returns false, then the decryptor (or the server) can divide the shares (or the proofs) into smaller groups and recursively apply batch verification until it identifies the malicious client.

[7] The linear proof can be abstracted the same way with length-ℓ vector $\mathbf{G}$.

6 Evaluation

In this section, we provide benchmarks to answer the following questions:

- What are Armadillo's concrete costs of the client and the server, for aggregation and proofs, respectively?
- What is Armadillo's overall cost including input validation?
- What is the cost of the decryptors and how does it compare to the cost of regular clients?
- How does Armadillo compare to prior robust secure aggregation protocols in terms of costs?

Selecting a proper baseline. The most relevant work is ACORN-robust [5]: they provide the same input validation but achieve robustness in a very different way. We provide details in Appendix B.2 but roughly, ACORN-robust follows the pairwise masking approach in Bell et al. [6], but they additionally have a dispute protocol to iteratively find cheating clients and remove their inputs from the aggregation result. The other prior work with disruption resistance is Eiffel [15], but their per-client work is $O(n^2\ell)$ which is not feasible for computationally restricted clients. Other works like RoFL [37], ACORN-detect [5] have strictly weaker property: the server has to abort the aggregation once a malicious client has been detected. Therefore, we identify ACORN-robust as the only valid baseline.

Libraries and testbed. We implement our protocol using Rust. We instantiate the proof using Bulletproof [11,21] since it is designed for inner-product relation. We use the dalek library [21] for elliptic curve cryptography. We run our experiments with a 2.4GHz CPU.

Concrete parameter selection. We use a proof system based on Ristretto group and set the LWE modulus q equal to the group order which is a 253-bit prime.[8] To control the noise growth in ciphertext computation, we require $n \cdot B_{\mathbf{e}} < \Delta/2$, i.e., $2pnB_{\mathbf{e}} < q$. According to the LWE estimator [2], we can set λ to be 256, $\mathbf{s}$ uniform in $\mathbb{Z}_q$, and the error uniform in $\mathbb{Z}_{2^{214}}$, which gives 132 bits of security; this supports plaintext modulus $p = 2^{16}$ with summation up to 5K clients, sufficient for federated learning applications [30, Table 2].

Following prior works we use power-of-2 input length for benchmarking. Since the inner-product proof system works efficiently for vectors of length power-of-2, our input length is set slightly smaller than powers of 2 such that the vector input to the inner product has length exactly or almost power of 2 (nearly no waste for padding).

Central theme in evaluation. Experiments in the following sections will substantiate a central argument we make: while Armadillo has similar computational cost as ACORN-robust (§6.1), the reduction in round complexity plays a crucial role in lowering the end-to-end runtime. The key to this improvement lies in the interplay between the allowed dropout rate and server waiting time (§6.2).

6.1 Computation and communication

Regular client costs. We present the breakdown of a regular client's computational cost in Figure 3, varying the input length. The cost for decryptors is discussed separately later. Since our baseline ACORN-robust shares similar types of computation with Armadillo, we present the costs for two systems jointly, dividing the process into four steps: input-independent secret sharing, input masking, commitment generation, and proof generation.[9] In each phase, we indicate whether ACORN-robust and Armadillo perform the same or different operations: shared operations are shown in black text, while differences are highlighted using distinct colors. Since the authors of ACORN-robust did not implement this protocol, we estimate their client's costs by extracting the types of operations and counting the number of operations for each type (e.g., how many scalar multiplication on elliptic curve), and simulating them with Rust (the same language used for implementing ours).

[8] One could use a LWE modulus that is much smaller than the group order of the proof system, but this requires additional L_∞ proofs and non-trivial changes for the Schwartz-Zippel compressing technique.

[9] In ACORN-robust, each client establishes input-independent pairwise secrets with $O(\log n)$ neighbors, expands them to $O(\log n)$ masks of input length, and performs Feldman secret sharing of the secrets. Microbenchmarks for ACORN-robust assume 40 neighbors per client (from [6]).

Input length ℓ approx.	2^{10}	2^{11}	2^{12}	2^{13}	2^{14}
Packed sharing in Arma. (ms)	29.67	29.67	29.67	29.67	29.67
Feldman in ACORN (ms)	90.32	90.32	90.32	90.32	90.32
Masking in Arma. (ms)	0.26	0.49	1.13	2.01	4.07
Masking in ACORN (ms)	1.69	3.81	9.41	15.53	31.45
Commitment (ms)	16.98	18.28	21.35	27.18	39.02
- commit to $\mathbf{s}$	1.31	1.31	1.31	1.31	1.31
- commit to $\mathbf{e}$	1.99	2.93	5.25	9.72	18.53
- commit to $\mathbf{x}$	1.19	1.55	2.30	3.66	6.69
- commit to shares in Arma.	12.49	12.49	12.49	12.49	12.49
- commit to masks in ACORN	164.80	288.85	535.58	1038.54	2042.97
Proofs (sec)					
- L_∞ norm (§3.4)	1.74	2.40	4.78	7.04	14.12
- L_2 norm (§3.4)	0.09	0.17	0.33	0.64	1.26
- enc linear (§3.2)	0.08	0.16	0.56	0.61	1.22
- scrape test (§3.2)	0.80	0.80	0.80	0.80	0.80

Figure 3: Computation cost per client in Armadillo and ACORN-robust [5] varying input vector lengths and fixing C (We set C to be 512 which is more than needed in most cases: with $\lambda = 256$, this can tolerate $\delta_{\mathcal{C}} + \eta_{\mathcal{C}} < 1/2$).

For both systems, the bulk of the computation time is spent on proof generation; most other phases are only on the order of milliseconds. The L_2 and L_∞ proofs (for input validation) are the same in both systems. Armadillo has a 1–2 second additional work for the proof required for robustness, but as we show later (§6.2), this is a tradeoff that yields significant gains: while our clients spend slightly more time (), the number of rounds for us is that for them. Since round complexity is substantially important for run time in a setting where clients may drop out arbitrarily (evidenced in §6.2), this reduction leads to much overall performance gains that is far outweigh the effect of increased client computation.

A final complication is that ACORN-robust has an additional cheater identification phase to remove the effects of malicious clients from the aggregation result. Since this phase involves only sending small stored messages to the server (Algorithm 4 in [5]) and there is no cryptographic computation at clients, we do not depict them in Figure 3.

Decryptor costs. Per-decryptor cost is independent of the input length ℓ, so we vary the number of clients n and show the corresponding costs. For RSA decryption, per-decryptor computation time is linear to the number of clients: for 500, 1K, 1.5K and 2K clients, the time is 0.6, 1.2, 1.8, 2.4 seconds respectively. For verifying a batch of 500, 1K, 1.5K, 2K shares, the computation time is 2.5, 4.1, 5.5, 7.4 ms respectively. For proof of decryption, per-proof takes 1ms and this only needs to be done for at most ηn clients. Importantly, these numbers show that per-decryptor computation is cheaper than that of the regular client even if excluding the proof of norm computation time.

Server costs. Armadillo's server computation depends on both input length ℓ and the number of clients n. Figure 4 breaks down the computation into aggregation (summing masked vectors and decoding using $\mathbf{s}$) and proof verification. As the number of clients increases, the proportion of time spent on proof verification decreases due to batching optimizations.

For ACORN-robust, the server performs the same proof verification and vector additions as in Armadillo, and additionally verifying Feldman commitments for every pairwise seed. Without optimization, Feldman verification could become a bottleneck: for 1K clients with 40 neighbors each, the server computes $40^2 \cdot 1000$ group exponentiations, taking 40 seconds. We can use the batching technique (§5) to reduce this to 25 ms in the above setting.

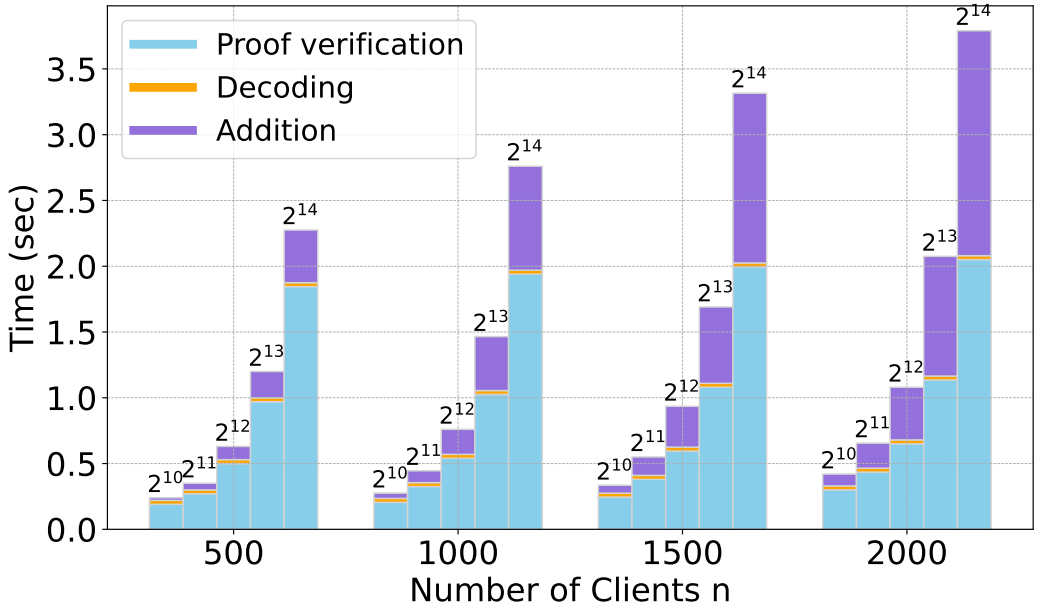
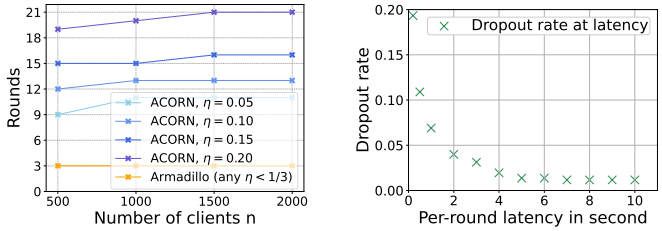


Figure 4: Server computation in Armadillo for different number of clients (indicated via x-axis) and different lengths of inputs (indicated on the top of bars).



(a) Number of rounds for Armadillo and ACORN-robust under different settings of n and η.

(b) The server waiting time determined by target $\hat{\delta}$, fixing a message arrival distribution.

Figure 5: Plots studying: (a) number of rounds for Armadillo and ACORN-robust as a function of n, η, (b) The server waiting time under different target $\hat{\delta}$.

6.2 Simulating communication rounds

Why rounds matter When executing an interactive protocol in the server-client setting, fixing the server's waiting time per round and dropping any clients who are late is a common strategy. This exacerbates the impact of round complexity: say a protocol is theoretically designed to tolerate 15% dropouts over all rounds of an aggregation, if this protocol has many rounds, e.g., 20 rounds, it may fail in practice because even 1% dropout per round can accumulate to 20% dropouts in total in which case the protocol just fails (the server does not get any aggregation result). This effect is depicted in the first two graphs in Figure 6: we set the waiting time per round to be 5 seconds and assume the message arrival time follows an exponential distribution, and we plot the arrival times of all messages and count the percentage of late messages (dropouts). In this case, a 3-round protocol that can tolerate up to 15% overall dropouts can successfully output the aggregation result, but a 7-round protocol that can tolerate up to 15% overall dropouts will fail. To ensure the latter protocol succeeds, one needs to increase the server waiting time per round, thereby reducing the dropouts per round and making the overall dropouts within the acceptable threshold. This is shown in the third graph in Figure 6. As a result, more rounds in fact amplifies the overall run time increase by increasing the per-round waiting time.

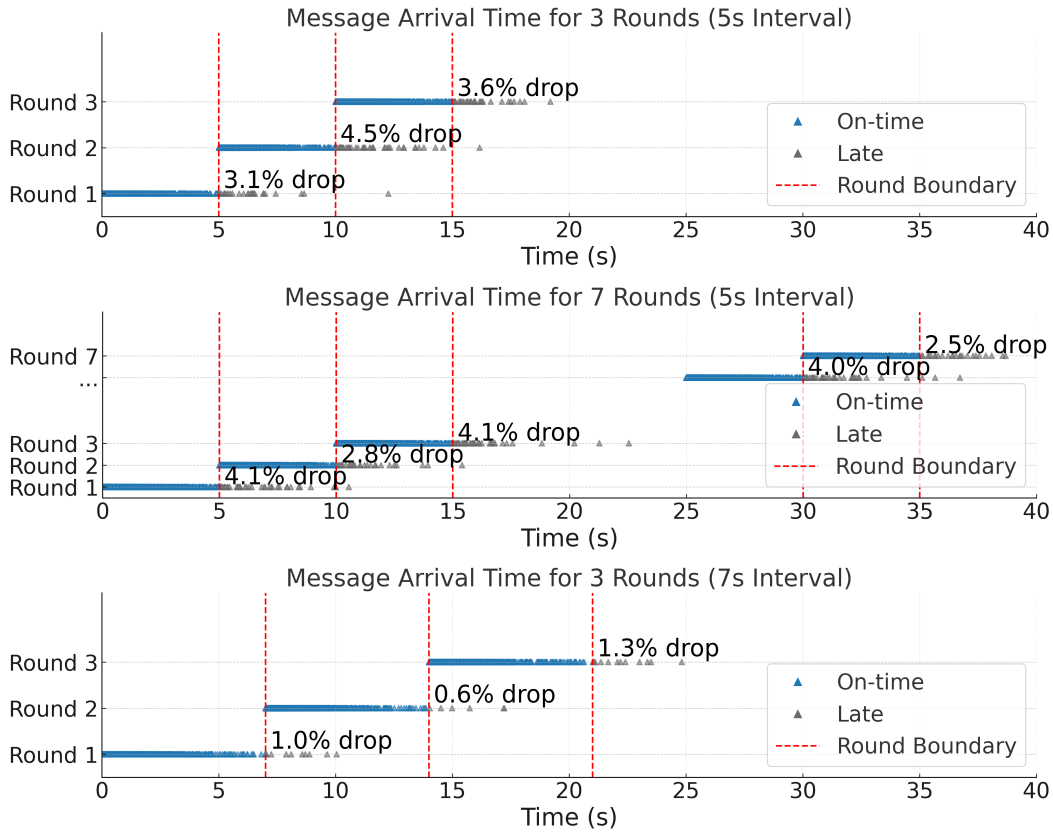


Figure 6: The relation between waiting time and dropout percentage. Fixing the waiting time per round (5 or 8 seconds), the dots in blue are messages that arrive on time in a round, and the dots in grey are messages that arrive late for that round; the server will only process the blue messages and the clients who sent grey messages are considered dropouts. The message arrival time follows exponential distribution.

Simulation methodology. For a fixed dropout rate δ that a R-round protocol can tolerate, we assume per round the server can handle $\hat{\delta} = \delta/R$ dropout rate. We use a standard message arrival distribution depicted derived from a pairwise network simulator in Figure 5b. Next, for Armadillo and our baseline ACORN-robust, given δ and R, we calculate the average per-round dropout rate $\hat{\delta}$ and determine server waiting time T using Figure 5b. The total round trip time is estimated as $T \cdot R$, and the total runtime includes this plus the computation time.

Rounds and run time comparison. Figure 5a shows the round complexity for ACORN-robust under different n and η based on their probabilistic analysis [5, Theorem 4.1]. For a 0.9 success probability, ACORN-robust requires up to 21 rounds ($7\times$ of Armadillo) in the worst setting ($n = 2000$, $\eta = 0.2$), compared to 3 rounds for Armadillo. In the best setting when $n = 500$ and $\eta = 0.05$, ACORN-robust still has 9 rounds ($3\times$ of Armadillo). Also, ACORN communicates with all clients in every round, while Armadillo involves all clients in the first round and only decryptors in the remaining two.

Figure 7 compares total runtime for 1K clients with 2^{14} input length. Computation time is similar for Armadillo and ACORN-robust (22 seconds), but Armadillo achieves up to $6\times$ improvement in total runtime due to reduced round trip time. Since L_∞ proof dominates computation (14 seconds, Fig. 3), if there exists faster proof of norms (especially proof of L_∞) in the future, it would amplify Armadillo's advantage as the portion of round trip time becomes more significant.

	η	δ	R	$\hat{\delta} = \delta/R$	$\hat{T}$ (sec)	RTs (sec)	Total (sec)
Armadillo	0.1	0.1	3	0.0333	4	12	33
	0.2	0.1	3	0.0333	4	12	33
	0.1	0.2	3	0.0667	2	6	27
ACORN	0.1	0.1	12	0.0083	10	120	144
	0.2	0.1	19	0.0053	10	190	212
	0.1	0.2	12	0.0167	10	120	144

Figure 7: Estimated time spent on round trips (RTs) and the total run time for Armadillo and ACORN-robust.

7 Related work and discussion

Single-server setting. Bonawitz et al. [9] introduced the first dropout-resilient secure aggregation protocol for federated learning. Subsequent works [6,48,49,38,26,35,31,7] focus on improving its efficiency. Recently, there has been increased interest in ensuring input validity within secure aggregation protocols, with various techniques proposed in prior work.

The first set of works [15,37,5] ensures the server either obtains a valid sum or the protocol aborts. Eiffel [15] uses SNIP [17], which allows proving arbitrary predicates on inputs but incurs high communication costs, with client communication scaling as $O(n^2\ell)$ where n is the number of clients and ℓ is the input length. RoFL [37] adopts Bonawitz et al.'s protocol [9] and uses range proofs for each input coordinate, resulting in lower communication costs: each client sends ℓ Pedersen commitments. However, both works do not guarantee that the honest server always gets the correct sum when some clients act maliciously.

The most relevant work to ours is the ACORN family [5], specifically ACORN-detect and ACORN-robust. ACORN-detect reduces proof size significantly compared to RoFL. ACORN-robust extends this to a more robust protocol with a probabilistic cheater identification mechanism that requires $O(\log n)$ rounds and $\mathsf{polylog}(n)$ work per client.

A recent work by Alon et al. [3] shows that any function can be securely computed with $\mathsf{polylog}(n)$ work per client and $\mathsf{poly}(n)$ work at the server, with $\mathsf{polylog}(n)$ rounds, assuming fully homomorphic encryption exists. However, it remains unclear whether this protocol is practical for tasks like computing sums.

Multiple non-colluding servers. Some works distribute trust across multiple servers, such as the two-server solutions Elsa [44] and SuperFL [50], or the generic multi-server solution Flag [4]. These approaches involve clients secret-sharing their inputs with multiple servers, which then communicate to validate inputs. Or protocols like Mario [40] use threshold additively homomorphic encryption to encrypt inputs for a set of servers. While some of these solutions are faster in run time compared to single-server protocols, they face criticism for relying on the non-collusion assumption among servers, which is difficult to ensure when the same organization owns the servers.

Our model (and [5,38]) differs fundamentally from the multi-server approach. In multi-server setups, powerful servers can handle heavy computation, such as the $n\ell$ work required by secret-sharing solutions [44,4], where n is the number of clients and ℓ is the input length. In single-server models, computational and communication overhead must be minimized for the resource-constrained clients. In other words, we must push the heavy computation to the server—we cannot have a protocol with $n\ell$ work at any client.

Our protocol (and [5,38]) works by each client sending its masked input of length ℓ to the server; then, in the rest of the steps, the clients do computation *independent* of ℓ to help the server unmask the sum. For ACORN family protocols [6,5], each client communicates with $\mathsf{polylog}(n)$ other clients where each of them doing $\mathsf{polylog}(n)$ work. For Armadillo (and [38]), each client communicates to a set of $\mathsf{polylog}(n)$ clients, where the latter does linear work in n. In practice, n is smaller than the input size ℓ so that the input size will dominate the client cost.

Secure aggregation for training iteration t

Server and clients agree on public parameters: LWE parameters $(\lambda, m, p, q, \mathbf{A} \in \mathbb{Z}_q^{\lambda \times m})$, the group $\mathbb{G}$ (of order q) for the commit-and-proof system, the norm bound $B_{\mathbf{x}}(L_\infty), B_{\mathbf{x}}(L_2), B_{\mathbf{e}})$. Let $\Delta = \lfloor q/p \rfloor$. The dropout rate is δ and malicious rate over n clients is η across all rounds in each iteration. The set of C helpers is determined via a random beacon or Feige election (Appendix **??**), with threshold being d.

Round 1 (Server → Clients)
Server notifies n clients (indexed by numbers in $[n]$) to start iteration $t \in [T]$.

Round 1 (Clients → Server)
Client $i \in [n]$ on input $\mathbf{x}_i \in \mathbb{Z}_q^m$, computes the following:

1. $\mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^\lambda$, $\mathbf{e}_i \leftarrow \chi^m$,
2. $\mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + \mathbf{e}_i + \Delta \mathbf{x}_i$, where $\mathbf{y}_i \in \mathbb{Z}_q^m$,
3. Compute degree-d packed secret sharing of $\mathbf{k}_i$ as $\mathbf{s}_i = (s_i^{(1)}, \ldots, s_i^{(C)})$.
4. Computes commitment to vector $\mathbf{k}_i$ as $\mathsf{com}(\mathbf{k}_i)$ and to vector $\mathbf{x}_i$ as $\mathsf{com}(\mathbf{x}_i)$, // two elements in $\mathbb{G}$
5. Computes commitment to shares $s_i^{(j)}$ as $\mathsf{com}_{G_j}(s_i^{(j)})$ for $j \in [C]$, where $\{G_j\}_{j\in[C]}$ are a set of generators in $\mathbb{G}$. // C elements in $\mathbb{G}$
6. Set constraint system $\{\mathsf{io} : (\mathsf{com}(\mathbf{s}_i), \mathbf{w}), \quad \mathsf{st} : \langle \mathbf{s}_i, \mathbf{w} \rangle = 0, \quad \mathsf{wt} : \mathbf{s}_i\}$, and computes $\pi_{\text{Shamir}} \leftarrow \Pi_{\text{ip}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$, where $\mathbf{w}$ is computed as: $m^*(X) \leftarrow_{\$} \mathbb{F}[X]_{\leq C-d-2}$ and let $\mathbf{w} := (v_1 \cdot m^*(1), \ldots, v_n \cdot m^*(C))$.
7. Set constraint system $\{\mathsf{io} : (\mathsf{com}(\mathbf{s}_i), \mathsf{com}(\mathbf{k}_i), \mathbf{M}), \quad \mathsf{st} : \mathbf{k}_i = \mathbf{M} \cdot \mathbf{s}_i, \quad \mathsf{wt} : (\mathbf{s}_i, \mathbf{k}_i)\}$, and compute $\pi_{\text{bind}} \leftarrow \Pi_{\text{linear}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$, // $O(\log C + \log \lambda)$ elements in $\mathbb{G}$, see algorithm in Figure **??**
8. Set constraint system $\mathbb{CS}_{\text{enc}} : \{\mathsf{io} : (\mathsf{com}(\mathbf{k}), \mathsf{com}(\mathbf{x}), \mathsf{com}(\mathbf{e})), \mathsf{st} : \mathbf{y} = \mathbf{A} \cdot \mathbf{k} + \mathbf{e} + \Delta \cdot \mathbf{x}, \|\mathbf{e}\|_2 < B_{\mathbf{e}}(L_2), \|\mathbf{x}\|_\infty < B_{\mathbf{x}}(L_\infty), \|\mathbf{x}\|_2 < B_{\mathbf{x}}(L_2), \mathsf{wt} : (\mathbf{k}, \mathbf{x}, \mathbf{e})\}$, and compute $\pi_{\text{enc}} \leftarrow \Pi_{\text{enc}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$. // $O(\log m)$ elements in $\mathbb{G}$, see algorithm in Section 3

Client $i \in [n]$ sends a tuple to the server:

$\{$"server" : $(\mathbf{y}_i, \mathsf{com}(\mathbf{k}_i), \mathsf{com}(\mathbf{x}_i), \mathsf{com}(\mathbf{e}_i), \{\mathsf{com}_{G_j}(s_i^{(j)})\}_{j\in[C]}$, $\pi_{\text{Shamir}}, \pi_{\text{bind}}, \pi_{\text{input}}, \pi_{\text{enc}})$;
"helper $j \in \mathcal{C}$" : $(s_i^{(j)}, r_i^{(j)})\}$
where $r_i^{(j)}$ is the randomness for $\mathsf{com}_{G_j}(s_i^{(j)})$.

// Note that every message from clients intended for clients/helpers is symmetrically encrypted.

Round 2 (Server → Helpers)
Let $\mathcal{S}_1 \subset [n]$ be the clients who sent the prescribed messages in Round 1.
The server does the following computation for each client $i \in \mathcal{S}_1$:

1. Compute $\mathsf{com}(\mathbf{s}_i) := \prod_{j\in[C]} \mathsf{com}_{G_j}(s_i^{(j)})$,
2. Run $\Pi_{\text{ip}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{Shamir}})$, $\Pi_{\text{linear}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{bind}})$, $\Pi_{\text{enc}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{enc}})$.
3. If all the proofs are valid, then send to each helper j the share commitment $\mathsf{com}_{G_i}(s_i^{(j)})$.

Round 2 (Helpers → Server) Each helper $j \in [C]$:

1. If received less than $(1 - \delta - \eta)n$ shares $s_i^{(j)}$, abort. Otherwise, it verifies $(s_i^{(j)}, r_i^{(j)})$ against the commitment $\mathsf{com}_{G_j}(s_i^{(j)})$, denote the set of clients whose commitments are valid as $\mathcal{S}_2$.
2. Sign the set $\mathcal{S}_2$ and sends the signature to all the other helpers via the server.

Round 3 (Server → Helpers) Server forwards the signatures to helpers.

Round 3 (Helpers → Server) Each helper $j \in [C]$: if received more than $2C/3$ valid signatures on the same set (including its own signature), then continues. Otherwise abort. Computes $s^{(j)} := \sum_{i\in\mathcal{S}_2} s_i^{(j)}$ and sends it to the server.
Server reconstruct the shares $\{s^{(j)}\}_{j\in\mathcal{C}}$ to $\mathbf{k}$, and computes $\mathbf{y} := \sum_{i\in\mathcal{S}_3} \mathbf{y}_i$.
Server computes $\lfloor \mathbf{y} - \mathbf{A} \cdot \mathbf{k} \mod q \rceil_\Delta$.

Figure 8: A secure aggregation protocol with dropout resilience, robustness, and input validity.

Discussion on efficiency/guarantee tradeoff If one's application does not require disruption resistance, some prior protocols based on pairwise masking may offer better performance than our two-layer construction. However, using zero-knowledge proofs to enforce correct computation in these protocols is computationally expensive. In contrast, our design is purpose-built to enable fast proofs for achieving disruption resistance.

Discussion on adaptive adversary Unlike some prior works [38], clients in Armadillo do not maintain any state across different aggregation executions. As a result, when the adversary changes the corrupted set after each aggregation (each training iteration) completes, the protocol still remains privacy against the server. Like all the prior works [5,6,38], Armadillo does not provide privacy if the adversary changes corrupted set within an aggregation execution.

financial product or service, or to be used in any way for evaluating the merits of participating in any transaction, and shall not constitute a solicitation under any jurisdiction or to any person, if such solicitation under such jurisdiction or to such person would be unlawful.

References

[1] S. Addanki, K. Garbe, E. Jaffe, R. Ostrovsky, and A. Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In C. Galdi and S. Jarecki, editors, *Proceedings of the International Conference on Security and Cryptography for Networks(SCN)*, volume 13409 of *LNCS*, pages 516–539, Amalfi, Italy, Sept. 12–14, 2022. Springer, Cham, Switzerland.

[2] M. R. Albrecht, R. Player, and S. Scott. On the concrete hardness of learning with errors. *Journal of Mathematical Cryptology*, 9(3):169–203, 2015.

[3] B. Alon, M. Naor, E. Omri, and U. Stemmer. MPC for tech giants (GMPC): Enabling gulliver and the lilliputians to cooperate amicably, 2022. https://eprint.iacr.org/2022/902.

[4] L. Bangalore, M. H. F. Sereshgi, C. Hazay, and M. Venkitasubramaniam. Flag: A framework for lightweight robust secure aggregation. In J. K. Liu, Y. Xiang, S. Nepal, and G. Tsudik, editors, *Flag: A Framework for Lightweight Robust Secure Aggregation*, pages 14–28, Melbourne, VIC, Australia, July 10–14, 2023. ACM Press.

[5] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun. ACORN: input validation for secure aggregation. In J. A. Calandrino and C. Troncoso, editors, *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*, pages 4805–4822, Anaheim, CA, USA, Aug. 9–11, 2023. USENIX Association.

[6] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In J. Ligatti, X. Ou, J. Katz, and G. Vigna, editors, *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pages 1253–1269, Virtual Event, USA, Nov. 9–13, 2020. ACM.

[7] J. Bell-Clark, A. Gascón, B. Li, M. Raykova, and P. Schoppmann. Willow: Secure aggregation with one-shot clients. Cryptology ePrint Archive, Report 2024/936, 2024.

[8] M. Bellare, J. A. Garay, and T. Rabin. Fast batch verification for modular exponentiation and digital signatures. In K. Nyberg, editor, *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, volume 1403 of *LNCS*, pages 236–250, Espoo, Finland, May 31 – June 4, 1998. Springer Berlin Heidelberg, Germany.

[9] K. A. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In B. Thuraisingham, D. Evans, T. Malkin, and D. Xu, editors, *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security, CCS 2017, Dallas, TX, USA, October 30 - November 03, 2017*, pages 1175–1191, Dallas, TX, USA, Oct. 31 – Nov. 2, 2017. ACM.

[10] J. Bootle. Efficient multi-exponentiation, 2017. https://jbootle.github.io/Misc/pippenger.pdf.

[11] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy, SP 2018, Proceedings, 21-23 May 2018, San Francisco, California, USA*, pages 315–334, San Francisco, CA, USA, May 21–23, 2018. IEEE Computer Society.

[12] I. Cascudo and B. David. SCRAPE: Scalable randomness attested by public entities. In D. Gollmann, A. Miyaji, and H. Kikuchi, editors, *ACNS 17International Conference on Applied Cryptography and Network Security*, volume 10355 of *LNCS*, pages 537–556, Kanazawa, Japan, July 10–12, 2017. Springer, Cham, Switzerland.

[13] H. Chen, L. Chua, K. Lauter, and Y. Song. On the concrete security of LWE with small secret, 2020. https://eprint.iacr.org/2020/539.pdf.

[14] J. H. Cheon, D. Kim, D. Kim, J. Lee, J. Shin, and Y. Song. Lattice-based secure biometric authentication for hamming distance. In J. Baek and S. Ruj, editors, *Information Security and Privacy*, volume 13083 of *LNCS*, pages 653–672, Virtual Event, Dec. 1–3, 2021. Springer, Cham, Switzerland.

[15] A. R. Chowdhury, C. Guo, S. Jha, and L. van der Maaten. EIFFeL: Ensuring integrity for federated learning. In H. Yin, A. Stavrou, C. Cremers, and E. Shi, editors, *ACM CCS 2022*, pages 2535–2549, Los Angeles, CA, USA, Nov. 7–11, 2022. ACM Press.

[16] Cloudflare. Cloudflare randomness beacon. https://developers.cloudflare.com/randomness-beacon/, 2025.

[17] H. Corrigan-Gibbs and D. Boneh. Prio: private, robust, and scalable computation of aggregate statistics. In *Proceedings of the 14th USENIX Conference on Networked Systems Design and Implementation*, NSDI'17, page 259–282, USA, 2017. USENIX Association.

[18] D. Dachman-Soled, L. Ducas, H. Gong, and M. Rossi. Lwe with side information: Attacks and concrete security estimation. In D. Micciancio and T. Ristenpart, editors, *Proceedings of the International Cryptology Conference (CRYPTO)*, volume 12171 of *LNCS*, pages 329–358, Santa Barbara, CA, USA, Aug. 17–21, 2020. Springer, Cham, Switzerland.

[19] S. Das, V. Krishnan, I. M. Isaac, and L. Ren. Spurt: Scalable distributed randomness beacon with transparent setup. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 2502–2517, San Francisco, CA, USA, May 22–26, 2021. IEEE Computer Society Press.

[20] A. Davidson, G. Pestana, and S. Celi. Frodopir: Simple, scalable, single-server private information retrieval. In *Proceedings of the Privacy Enhancing Technologies Symposium (PETS)*, volume 2023, pages 365–383. Sciendo, Jan. 2023.

[21] H. de Valence, C. Yun, and O. Andreev. Bulletproof implementation based on delak-cryptography, 2018. https://github.com/dalek-cryptography/bulletproofs.

[22] U. Feige. Noncryptographic selection protocols. In *40th FOCS*, pages 142–153, New York, NY, USA, Oct. 17–19, 1999. IEEE Computer Society Press.

[23] B. Feng. Multi-scalar multiplication (MSM), 2023. https://hackmd.io/@tazAymRSQCGXTUKkbh1BAg/Sk27liTW9.

[24] M. K. Franklin and M. Yung. Communication complexity of secure computation (extended abstract). In *24th ACM STOC*, pages 699–710, Victoria, BC, Canada, May 4–6, 1992. ACM Press.

[25] C. Gentry, S. Halevi, and V. Lyubashevsky. Practical Non-interactive Publicly Verifiable Secret Sharing with Thousands of Parties. In O. Dunkelman and S. Dziembowski, editors, *Proceedings of the International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, volume 13275 of *LNCS*, pages 458–487, Trondheim, Norway, May 30 – June 3, 2022. Springer, Cham, Switzerland.

[26] Y. Guo, A. Polychroniadou, E. Shi, D. Byrd, and T. Balch. Microsecagg: Streamlined single-server secure aggregation. *Proc. Priv. Enhancing Technol.*, 2024(3):246–275, 2024.

[27] A. Henzinger, M. M. Hong, H. Corrigan-Gibbs, S. Meiklejohn, and V. Vaikuntanathan. One Server for the Price of Two: Simple and Fast Single-Server Private Information Retrieval. In J. A. Calandrino and C. Troncoso, editors, *Proceedings of the USENIX Security Symposium*, pages 3889–3905, Anaheim, CA, USA, Aug. 9–11, 2023. USENIX Association.

[28] A. Jain, H. Lin, and S. Saha. A systematic study of sparse lwe. In L. Reyzin and D. Stebila, editors, *Proceedings of the International Cryptology Conference (CRYPTO)*, volume 14922 of *LNCS*, pages 210–245, Santa Barbara, CA, USA, Aug. 18–22, 2024. Springer, Cham, Switzerland.

[29] W. B. Johnson and J. Lindenstrauss. Extensions of lipschitz mappings into a Hilbert space, 1984.

[30] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. Nitin Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings, R. G. L. D'Oliveira, H. Eichner, S. El Rouayheb, D. Evans, J. Gardner, Z. Garrett, A. Gascón, B. Ghazi, P. B. Gibbons, M. Gruteser, Z. Harchaoui, C. He, L. He, Z. Huo, B. Hutchinson, J. Hsu, M. Jaggi, T. Javidi, G. Joshi, M. Khodak, J. Konecný, A. Korolova, F. Koushanfar, S. Koyejo, T. Lepoint, Y. Liu, P. Mittal, M. Mohri, R. Nock, A. Özgür, R. Pagh, H. Qi, D. Ramage, R. Raskar, M. Raykova, D. Song, W. Song, S. U. Stich, Z. Sun, A. T. Suresh, F. Tramèr, P. Vepakomma, J. Wang, L. Xiong, Z. Xu, Q. Yang, F. X. Yu, H. Yu, and S. Zhao. Advances and open problems in federated learning. *Found. Trends Mach. Learn.*, 14(1–2):1–210, June 2021.

[31] H. Karthikeyan and A. Polychroniadou. OPA: One-shot private aggregation with single client interaction and its applications to federated learning, 2024. https://eprint.iacr.org/2024/723.

[32] D. Kim, D. Lee, J. Seo, and Y. Song. Toward practical lattice-based proof of knowledge from hint-MLWE. In H. Handschuh and A. Lysyanskaya, editors, *CRYPTO 2023, Part V*, volume 14085 of *LNCS*, pages 549–580, Santa Barbara, CA, USA, Aug. 20–24, 2023. Springer, Cham, Switzerland.

[33] I. Komargodski, S. Matsuo, E. Shi, and K. Wu. $\log^*$-round game-theoretically-fair leader election. In Y. Dodis and T. Shrimpton, editors, *Proceedings of the International Cryptology Conference (CRYPTO)*, volume 13509 of *LNCS*, pages 409–438, Santa Barbara, CA, USA, Aug. 15–18, 2022. Springer, Cham, Switzerland.

[34] J. Lee, D. Kim, D. Kim, Y. Song, J. Shin, and J. H. Cheon. Instant privacy-preserving biometric authentication for hamming distance, 2018. https://eprint.iacr.org/2018/1214.

[35] H. Li, H. Lin, A. Polychroniadou, and S. Tessaro. Lerna: Secure single-server aggregation via key-homomorphic masking. In J. Guo and R. Steinfeld, editors, *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, volume 14438 of *LNCS*, pages 302–334, Guangzhou, China, Dec. 4–8, 2023. Springer, Singapore, Singapore.

[36] Z. Liu, S. Chen, J. Ye, J. Fan, H. Li, and X. Li. SASH: Efficient secure aggregation based on shprg for federated learning. In *Proceedings of the 38th Conference on Uncertainty in Artificial Intelligence (UAI)*, Eindhoven, The Netherlands, 2022. Association for Uncertainty in Artificial Intelligence (AUAI).

[37] H. Lycklama, L. Burkhalter, A. Viand, N. Küchler, and A. Hithnawi. Rofl: Robustness of secure federated learning. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 453–476, San Francisco, CA, USA, May 21–25, 2023. IEEE.

[38] Y. Ma, J. Woods, S. Angel, A. Polychroniadou, and T. Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*, pages 477–496, San Francisco, CA, USA, May 21–25, 2023. IEEE.

[39] H. B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Proceedings of the Artificial Intelligence and Statistics Conference (AISTATS)*, 2017.

[40] T. S. Nguyen, T. Lepoint, and N. Trieu. Mario: Multi-round multiple-aggregator secure aggregation with robustness against malicious actors. Cryptology ePrint Archive, Paper 2024/1428, 2024.

[41] D. Pasquini, D. Francati, and G. Ateniese. Eluding secure aggregation in federated learning via model inconsistency. In H. Yin, A. Stavrou, C. Cremers, and E. Shi, editors, *Proceedings of the ACM Conference on Computer and Communications Security (CCS)*, pages 2429–2443, Los Angeles, CA, USA, Nov. 7–11, 2022. ACM Press.

[42] N. Pippenger. On the evaluation of powers and monomials. *SIAM Journal on Computing*, 9(2):230–250, 1980.

[43] M. O. Rabin and J. O. Shallit. Randomized algorithms in number theory, 1986.

[44] M. Rathee, C. Shen, S. Wagh, and R. A. Popa. Elsa: Secure aggregation for federated learning with malicious actors. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, pages 1961–1979, San Francisco, CA, USA, May 21–25, 2023. IEEE Computer Society Press.

[45] O. Regev. On lattices, learning with errors, random linear codes, and cryptograpy. In H. N. Gabow and R. Fagin, editors, *Proceedings of the ACM Symposium on Theory of Computing (STOC)*, pages 84–93, Baltimore, MA, USA, May 2005. ACM Press.

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

[47] J. So, R. E. Ali, B. Güler, J. Jiao, and A. S. Avestimehr. Securing secure aggregation: mitigating multi-round privacy leakage in federated learning. In *Proceedings of the Thirty-Seventh AAAI Conference on Artificial Intelligence and Thirty-Fifth Conference on Innovative Applications of Artificial Intelligence and Thirteenth Symposium on Educational Advances in Artificial Intelligence*, AAAI'23/IAAI'23/EAAI'23, Washington DC, USA, 2023. AAAI Press.

[48] J. So, C. J. Nolet, C. Yang, S. Li, Q. Yu, R. E. Ali, B. Guler, and S. Avestimehr. Lightsecagg: a lightweight and versatile design for secure aggregation in federated learning. In D. Marculescu, Y. Chi, and C. Wu, editors, *Proceedings of the Fifth Conference on Machine Learning and Systems, MLSys 2022, Santa Clara, CA, USA, August 29 - September 1, 2022*, Santa Clara, CA, USA, 2022. mlsys.org.

[49] T. Stevens, C. Skalka, C. Vincent, J. Ring, S. Clark, and J. Near. Efficient differentially private secure aggregation for federated learning via hardness of learning with errors. In K. R. B. Butler and K. Thomas, editors, *Proceedings of the USENIX Security Symposium*, pages 1379–1395, Boston, MA, USA, Aug. 10–12, 2022. USENIX Association.

[50] Y. Zhao, H. Zhou, and Z. Wan. Superfl: Privacy-preserving federated learning with efficiency and robustness. Cryptology ePrint Archive, Paper 2024/081, 2024. https://eprint.iacr.org/2024/081.

A Deferred material for proof of norms

A.0.1 Approximate proof

We describe the protocol in Gentry, Halevi, and Lyubashevsky [25] below. Let the security parameter be σ. The prover has a vector $\mathbf{a}$ of length m where $\|\mathbf{a}\|_\infty < B$. Let B' be the bound that the prover can prove with the following protocol. For security, the gap $\gamma := B'/B$ should be larger than $19.5\sigma\sqrt{m}$.

1. The prover first sends $\mathsf{com}(\mathbf{a})$ to the verifier.
2. The prover chooses a uniform length-σ vector $\mathbf{y} \xleftarrow{\$} [\pm\lceil b/2(1+1/\sigma)\rceil]^\sigma$, and sends $\mathsf{com}(\mathbf{y})$ to the verifier.
3. The verifier chooses $\mathbf{R} \leftarrow \mathcal{D}^{\sigma\times m}$ and sends it to the prover.
4. The prover computes $\mathbf{u} := \mathbf{R}\cdot\mathbf{a}$ and $\mathbf{z} = \mathbf{u} + \mathbf{y}$. It restarts the protocol from Step 2 if either $\|\mathbf{u}\|_\infty > b/2\lambda$ or $\|\mathbf{z}\| > b/2$.
5. The prover sends $\mathbf{z}$ to the verifier.
6. The verifier chooses a random r and sends r to the prover.
7. The prover and the verifier run an inner-product proof that

$$\langle \mathbf{R}^\top\mathbf{r}, \mathbf{a}\rangle + \langle \mathbf{r}, \mathbf{y}\rangle = \langle \mathbf{R}^\top\mathbf{r} | \mathbf{r}, \mathbf{a}|\mathbf{y}\rangle = \langle \mathbf{z}, \mathbf{r}\rangle,$$

   where $\mathbf{r} = (r^0, r^1, \ldots, r^{\sigma-1})$.

Note that $\langle \mathbf{z}, \mathbf{r}\rangle$ is a public value. The last step is essentially a length-$(m+\sigma)$ inner product proof.

There are two properties of this approximate proof: correctness and soundness. Correctness means that if a client has a vector of L_∞ norm smaller than B then it should fail to prove it with negligible probability. Soundness means that if a client has a vector with L_∞ norm larger than B', then it fails the approximate proof with overwhelming probability. Note that this soundness does not mean that a client will fail the proof when the vector has L_∞ norm between B and B'; and this is exactly why we cannot merely use approximate proof for our proof of L_∞ norm.

A.0.2 Complete proof of encryption

We now describe proof of Regev's encryption ([25, Lemma 3.7]). We denote the proof as Π_{enc}. Recall that the constraints that client i wishes to prove is

$$\begin{aligned}\mathbb{CS}_{\text{enc}} : \{&\mathsf{io} : (\mathsf{com}(\mathbf{s}_i), \mathsf{com}(\mathbf{x}_i), \mathsf{com}(\mathbf{e}_i)),\\ &\mathsf{st} : \mathbf{y}_i = \mathbf{A}\mathbf{s}_i + \mathbf{e}_i + \Delta\mathbf{x}_i,\\ &\quad \|\mathbf{x}_i\|_2 < B_{\mathbf{X}}(L_2),\\ &\quad \|\mathbf{x}_i\|_\infty < B_{\mathbf{X}}(L_\infty), \|\mathbf{e}_i\|_\infty < B_{\mathbf{e}}(L_\infty),\\ &\mathsf{wt} : (\mathbf{s}_i, \mathbf{x}_i, \mathbf{e}_i)\}\end{aligned}$$

Note that $\|\mathbf{x}_i\|_\infty < B_{\mathbf{X}}(L_\infty)$ and $\|\mathbf{e}_i\| \le B_{\mathbf{X}}(L_\infty)$ can be proven using the techinque described in Section 3.4, independent of the rest of the steps we will present. So below we focus on proving L_2 norms. For simplicity, until the end of this section we omit the subscript i. Protocol Π_{enc} works as follows:

1. The prover sets $\mathbf{y} = \mathbf{A}\mathbf{s} + \mathbf{e} + \Delta\mathbf{x} \mod q$, sends to the verifier $\mathbf{y}$ and the commitment to $\mathbf{s}, \mathbf{x}, \mathbf{e}$. Recall that $\mathbf{A} \in \mathbb{Z}_q^{\ell\times\lambda}, \mathbf{e} \in \chi^\ell$.
2. The verifier chooses projection matrices $\mathbf{R} \leftarrow \mathcal{D}^{256\times\lambda}$ and $\mathbf{R}' \leftarrow \mathcal{D}^{256\times\ell}$, and sends them to the prover.
3. The prover computes $\mathbf{u} := \mathbf{R}'\cdot\mathbf{e}$, and $\mathbf{v} = \mathbf{R}'\cdot\mathbf{x}$. The prover aborts if $\|\mathbf{u}\|_2 > B_{\mathbf{u}}$ or $\|\mathbf{v}\|_2 > B_{\mathbf{v}}$, otherwise it sends to the verifier the commitment to $\mathbf{u}, \mathbf{v}$. The bound $B_{\mathbf{u}}, B_{\mathbf{v}}$ are determined by the LWE parameters and the bound $B_{\mathbf{X}}, B_{\mathbf{e}}$. Note that vectors $\mathbf{u}, \mathbf{v}$ are only of length 256.
4. The prover and the verifier run the following sub-protocols:
   (a) Proof of L_2 norm that

$$\|\mathbf{u}\|_2 < B_{\mathbf{u}},\ \|\mathbf{v}\|_2 < B_{\mathbf{v}}$$

(b) Proof of linear relation that:

$$\mathbf{R}' \cdot \mathbf{e} = \mathbf{u}, \quad \mathbf{R}' \cdot \mathbf{x} = \mathbf{v} \mod q.$$

(c) Proof of linear relation that:

$$\mathbf{R}' \cdot \mathbf{y} = (\mathbf{R}'\mathbf{A}) \cdot \mathbf{s} + \mathbf{u} + \Delta\mathbf{v} \mod q.$$

(d) Approximate proof that:

$$\|\mathbf{x}\|_\infty, \|\mathbf{e}\|_\infty < \sqrt{q/(\ell+4)}$$

5. The verifier accepts if all the above proofs are valid.

For step 4(a), we can directly prove L_2 norm as we described before for length-256 vectors $\mathbf{u}, \mathbf{v}$. For step 4(b) and 4(c), we can use Schwartz-Zippel techinque as described in Section 3.2. For step 4(d), we use the approximate proof described before (see details in Appendix A.0.1).

B Deferred material for baseline

B.1 Cost overview of ACORN-detect

To understand how ACORN-robust works we first present ACORN-detect. In ACORN-detect protocol, the server can detect if a client cheats but the protocol does not have guaranteed output delivery. We outline the protocol below and briefly analyze its cost.

We start with the protocol (without input validation) in Bell et al. [6] which is also a base protocol for ACORN. Initially, the server establishes a public graph on all n clients where each client has $k = O(\log n)$ neighbors; let $N(i) \subset [n]$ denote the neighbors of i. Each pair of clients establish pairwise secrets p_{ij}. Each client i generates a random PRG seed z_i and masks the input $\mathbf{x}_i$ as

$$\mathbf{y}_i = \mathbf{x}_i + \mathbf{r}_i,$$

where the mask $\mathbf{r}_i$ is defined as

$$\mathbf{r}_i = \sum_{i<j, j\in N(i)} \mathtt{PRG}(p_{ij}) - \sum_{i>j, j\in N(i)} \mathtt{PRG}(p_{ij}) + \mathtt{PRG}(z_i).$$

The client sends $\mathbf{y}_i$ to the server. Note that z_i is for ensuring privacy when handling dropouts; see more details in Bell et al. [6]. We skip the rest of details of the protocol here, but their key feature is that for any online set $\mathcal{O} \subset [n]$, the server eventually gets $\mathbf{r} := \sum_{i\in\mathcal{O}} \mathbf{r}_i$ so that it can remove $\mathbf{r}$ from $\mathbf{y} := \sum_{i\in\mathcal{O}} \mathbf{y}_i$ and obtain the desired output $\sum_{i\in\mathcal{O}} \mathbf{x}_i$.

To achieve input validity, they added the following steps to the above protocol. Each client i computes the commitment to $\mathbf{x}_i$ and the commitment to the aggregated mask $\mathbf{r}_i$ and sends them together with the masked vector $\mathbf{y}_i = \mathbf{x}_i + \mathbf{r}_i$. Then the client proves that

- $\mathbf{x}_i$ has valid L_2, L_∞ norm (same as Section 3.4);
- It added $\mathbf{r}_i$ to $\mathbf{x}_i$ correctly (which can be done using a linear proof).

Recall that the server learns $\mathbf{r} := \sum_{i\in\mathcal{O}} \mathbf{r}_i$. Next, the clients and the server run a distributed key correctness (DKC) protocol to check if the server obtains $\mathbf{r} := \sum_{i\in\mathcal{O}} \mathbf{r}_i$ where the $\mathbf{r}_i$'s are indeed consistent with the commitments that the clients sent in the first place.

Remark 3. When the PRG is instantiated with homomorphic PRG (e.g., RLWE-based PRG), the client can optimize its computation by first computing the sum of the seeds and then expanding the aggregated seeds with PRG. A trade-off is that the masking here is not simply $\mathbf{x}_i + \mathbf{r}_i$: since the PRG output is defined over polynomial rings, the input $\mathbf{x}_i$ should be interpreted as polynomials when added to $\mathbf{r}_i$ and this requires non-trivial encoding of $\mathbf{x}_i$ (see Equation 5 in ACORN [5]). As a result, the client also needs to prove it performs the encoding correctly.

Cost. Each client computes two vector commitments to length ℓ vectors $\mathbf{x}_i, \mathbf{r}_i$. For the DKC protocol, the client performs a constant number of elliptic curve scalar multiplications, and the server performs $3n$ of them.

B.2 Cost overview of ACORN-robust

ACORN-robust is similar to ACORN-detect but with the following differences:

- The pairwise secrets are established differently (see details below);
- When the server fails verification in the DKC protocol, it invokes an $O(\log n)$-round bad message resolution protocol with all the clients to remove the malicious clients' contribution from the sum.

Suppose the server establishes a public graph on all n clients where each client has $k = O(\log n)$ neighbors. First, each client i generates k seeds $s_{i,j}$ for neighbor j, and sends them to the neighbors; client i additionally generates (deterministic) commitments to the seeds, namely $s_{i,j} \cdot G$, which are sent to the server. Next, clients exchange the seeds with their neighbors: a client i neighboring with client j will send $s_{i,j}$ and receive $s_{j,i}$, and vice versa. Client i and j then establish pairwise secret $p_{ij} = s_{i,j} + s_{j,i}$; this p_{ij} will be used for pairwise-masking the input vector.

Each client i then Shamir-shares $s_{i,j}$ and sends the Feldman commitments to the sharing of $s_{i,j}$ (commitments to the coefficients of the sharing polynomial) to the server. The server checks if the Feldman commitments match the commitment $s_{i,j} \cdot G$. If not, the server disqualifies client i; if it matches, the server computes $s_{i,j}^{(k)} \cdot G$ from Feldman commitments, where $s_{i,j}^{(k)}$ is the share meant for the k-th neighbor of client i. Then the server sends $s_{i,j}^{(k)} \cdot G$ to the corresponding client. The recipient client checks if the decrypted share $s_{i,j}^{(k)}$ matches the commitment $s_{i,j}^{(k)} \cdot G$. Then the server and clients invoke an $O(\log n)$-round bad message resolution protocol to form a set of clients whose pairwise masks can be canceled out.

There are two costly parts of ACORN-robust: 1) the obvious complexity of the logarithmic number of rounds between the server and all the clients; 2) the server needs to verify $O(n \log n)$ Feldman commitments of sharing of degree $\log n$. Concretely, using the parameters estimated by Bell et al., with $n = 1000$ clients and $\delta = \eta = 0.05$, the neighbors required (for security) is roughly 30, meaning that here the server needs to perform $2 \cdot 30^2 \cdot 1000 = 1,800,000$ elliptic curve scalar multiplications and this takes roughly 10 minutes; note that this cannot be trivially optimized with multi-exponentiation because the server needs to identify the malicious clients.

C Security proof of Armadillo

We give our full protocol description in Figures 9 and 10.

C.1 Proof of Theorem 1

We follow the proof of security similar to that of ACORN-robust [5]. However, there are key differences. Their protocol guarantees the privacy of honest clients only with a semi-honest server. This is an artifact of their protocol where the server is empowered to recover the masks—both the self-masks and pairwise masks—for misbehaving clients to then remove their inputs. In other words, the server is capable of recovering the actual inputs of malicious clients. Consequently, a malicious server could claim honest clients to be malicious and thereby recover the inputs of these clients. In contrast, our protocol works by using a single mask, and these masks are never revealed to the server, even for those misbehaving clients.

Our proof methodology relies on the standard simulation-based proof, where we show that every adversary attacking our protocol can be simulated by an adversary Sim in an ideal world where the functionality $\mathcal{F}$ (Fig.2). In the following, we first prove privacy against any adversary corrupting ηn clients and the server; then we prove robustness assuming the adversary corrupting ηn clients but not the server (recall our threat model in §1.2).

The challenge in the simulation is the ability of Sim to generate a valid distribution for the honest clients' inputs, even without knowing their keys. To this end, we will show that Sim, when only given the sum of the user inputs $\mathbf{X} = \sum_{i=1}^{n} \mathbf{x}_i$, can simulate the expected leakage for the server which includes n ciphertexts, the sum of the n keys $\mathbf{K} = \sum_{i=1}^{n} \mathbf{k}_i$, and such that the sum of the n ciphertexts, when decrypted with $\mathbf{K}$, correctly decrypts to $\mathbf{X}$.

Secure aggregation for training iteration t

Server and clients agree on public parameters:

- LWE parameters $(\lambda, \ell, p, q, \mathbf{A} \in \mathbb{Z}_q^{\ell \times \lambda})$ and $\Delta = \lfloor q/p \rfloor$.
- Proof parameters: Let $\mathbb{G}$ be the group of order q for the commit-and-proof system. Let $\mathbf{F}, \mathbf{G}, \mathbf{H}, \mathbf{K}$ be vectors of generators in $\mathbb{G}$ of length λ, ℓ, ℓ, C. The norm bounds are $B_{\mathbf{x}}(L_\infty), B_{\mathbf{x}}(L_2), B_{\mathbf{e}}(L_\infty)$.
- System model parameters: dropout rate is δ and malicious rate over this iteration of selected clients is η.

Setup. The set $\mathcal{C}$ of helpers is determined independently from the aggregation as described in Section 3.5, with threshold being d. Let the secret key and public key for $j \in \mathcal{C}$ be (SK_j, PK_j). Let $k_{i,j}$ be MAC key shared between client i and helper $j \in \mathcal{C}$; such key can be derived from PKI.

Round 1 (Server $\rightarrow$ Clients)
Server notifies a set S_t of n clients (indexed by numbers in $[n]$) to start iteration $t \in [T]$. It also tells the helpers the IDs of the n clients. Each helper $j \in \mathcal{C}$ derives the MAC key $k_{i,j}$ for each $i \in S_t$.

Round 1 (Clients $\rightarrow$ Server)
Client $i \in S_t$ on input $\mathbf{x}_i \in \mathbb{Z}_q^m$, computes the following:

1. Compute $\mathbf{y}_i = \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i \bmod q$, where $\mathbf{s}_i \xleftarrow{\$} \mathbb{Z}_q^\lambda$, $\mathbf{e}_i \leftarrow \chi^m$. // For outer aggregation
2. Compute degree-d packed secret sharing of $\mathbf{s}_i$ as $\boldsymbol{\rho}_i = (\rho_i^{(1)}, \ldots, \rho_i^{(D)})$. // For inner aggregation
3. Compute commitments $\mathsf{com}_{\mathbf{F}}(\mathbf{s}_i), \mathsf{com}_{\mathbf{G}}(\mathbf{e}_i), \mathsf{com}_{\mathbf{H}}(\mathbf{x}_i)$; and $\mathsf{com}_{K_j}(\rho_i^{(j)})$ for $j \in \mathcal{C}$, where $\mathbf{K}$ is parsed as $\{K_j\}_{j \in \mathcal{C}}$.
4. Set constraint system $\mathbb{CS}_{\text{shares}}$:

$$\{\mathsf{io} : (\mathsf{com}(\boldsymbol{\rho}_i), \mathsf{com}(\mathbf{s}_i), \mathbf{w}), \quad \mathsf{st} : \langle \boldsymbol{\rho}_i | \mathbf{s}_i, \mathbf{w} \rangle = 0, \quad \mathsf{wt} : (\boldsymbol{\rho}_i, \mathbf{s}_i)\},$$

   and compute $\pi_{\text{shares}} \leftarrow \Pi_{\text{ip}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$, where $m^*(X) \leftarrow_{\$} \mathbb{F}[X]_{\leq D+\lambda-d-2}$ and $\mathbf{w} := (v_1 \cdot m^*(1), \ldots, v_n \cdot m^*(D))$.
5. Set constraint system $\mathbb{CS}_{\text{enc}}$:

$$\{\mathsf{io} : (\mathsf{com}(\mathbf{s}_i), \mathsf{com}(\mathbf{x}_i), \mathsf{com}(\mathbf{e}_i)),$$
$$\mathsf{st} : \mathbf{y}_i = \mathbf{A} \cdot \mathbf{s}_i + \mathbf{e}_i + \Delta \cdot \mathbf{x}_i, \|\mathbf{x}\|_2 < B_{\mathbf{x}}(L_2), \|\mathbf{e}\|_\infty < B_{\mathbf{e}}(L_\infty), \|\mathbf{x}\|_\infty < B_{\mathbf{x}}(L_\infty),$$
$$\mathsf{wt} : (\mathbf{s}_i, \mathbf{x}_i, \mathbf{e}_i)\},$$

   and compute $\pi_{\text{enc}} \leftarrow \Pi_{\text{enc}}.\mathcal{P}(\mathsf{io}, \mathsf{st}, \mathsf{wt})$.
6. Send a tuple to the server:

$$\{\text{“server”} : (\mathbf{y}_i, \mathsf{com}(\mathbf{s}_i), \mathsf{com}(\mathbf{x}_i), \mathsf{com}(\mathbf{e}_i), \pi_{\text{shares}}, \pi_{\text{enc}});$$
$$\text{“helper } j \in \mathcal{C}\text{”} : \mathsf{ct}_j := \mathsf{AsymEnc}(PK_j, \rho_i^{(j)}), \mathsf{com}_{K_j}(\rho_i^{(j)}) \text{ and a MAC tag } \sigma_{i,j} \leftarrow \mathrm{MAC}(k_{i,j}, \mathsf{ct}_{i,j})\}$$

Figure 9: Armadillo protocol description for computing a single sum privately (Part I).

Before we detail the definition of Sim and prove its security, we present an assumption that we will use later.

Definition 5 (A variant of Hint-LWE [34,14])**.** Consider integers λ, m, q and a probability distribution χ' on $\mathbb{Z}_q$, typically taken to be a normal distribution that has been discretized. Then, the Hint-LWE

Secure aggregation for training iteration t contd.

Round 2 (Server → Helpers)
Let $\mathcal{X}_1$ be the clients who sent the prescribed messages in Round 1.
The server for each client $i \in \mathcal{X}_1$ computes:

1. Compute $\mathsf{com}_{\mathbf{K}}(\boldsymbol{\rho}_i) := \prod_{j\in[C]} \mathsf{com}_{K_j}(\rho_i^{(j)})$.
2. Run $\Pi_{\text{linear}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{shares}})$ and $\Pi_{\text{enc}}.\mathcal{V}(\mathsf{io}, \mathsf{st}, \pi_{\text{enc}})$.
3. Remove all clients with invalid proof from $\mathcal{X}_1$. Call this set $\mathcal{X}_2$.
4. If all the proofs are valid, forward messages intended for $j \in \mathcal{C}$.

Round 2 (Helpers → Server)
Each helper $j \in \mathcal{C}$: for every i,

1. Check if $\sigma_{i,j}$ is valid. If there are fewer than $(1-\delta-\eta)n$ valid messages, abort. Otherwise continue.
2. It computes $\rho_i^{(j)} := \mathsf{AsymDec}(SK_j, \mathsf{ct}_j)$, and checks if it is consistent with $\mathsf{com}_{K_j}(\rho_i^{(j)})$. If not, create a verifiable complaint that consists of $\rho_i^{(j)}$ and the proof of decryption of ct_j; denote this proof as π_{dec}.
3. It formed a set $\mathcal{V}_j$ that consists of all the clients whose shares are valid.

Round 3 (Server → Helpers)
Server tells all the helpers a set of clients who were complained about, denoted as $\mathcal{B}$. Set $\mathcal{S}_3 := \mathcal{S}_2 \setminus \mathcal{B}$.

Round 3 (Helpers → Server)
Each helper $j \in \mathcal{C}$:

1. Remove clients in $\mathcal{B}$ from $\mathcal{V}_j$.
2. Compute $\rho^{(j)} := \sum_{i\in\mathcal{V}_j} \rho_i^{(j)}$ and send it to the server.

Server reconstructs the shares $\{\rho^{(j)}\}_{j\in\mathcal{C}}$ to $\mathbf{s}$, and computes $\mathbf{y} := \sum_{i\in\mathcal{X}_2\setminus\mathcal{B}} \mathbf{y}_i$ and computes $\lfloor \mathbf{y} - \mathbf{A}\cdot\mathbf{s} \mod q \rceil_\Delta$.

Figure 10: Armadillo protocol description for computing a single sum privately (Part II).

assumption states that for all PPT adversaries $\mathcal{A}$, there exists a negligible function negl such that:

$$\Pr\left[b = b' \;\middle|\; \begin{array}{c} \mathbf{A} \xleftarrow{\$} \mathbb{Z}_q^{m\times\lambda}, \mathbf{k} \xleftarrow{\$} \mathbb{Z}_q^{\lambda}, \mathbf{e} \xleftarrow{\$} \chi'^m \\ \mathbf{r} \xleftarrow{\$} \mathbb{Z}_q^{\lambda}, \mathbf{f} \xleftarrow{\$} \chi'^m \\ \mathbf{y}_0 := \mathbf{A}\mathbf{k} + \mathbf{e}, \mathbf{y}_1 \xleftarrow{\$} \mathbb{Z}_q^m, b \xleftarrow{\$} \{0,1\} \\ b' \xleftarrow{\$} \mathcal{A}(\mathbf{A}, (\mathbf{y}_b, \mathbf{k}+\mathbf{r}, \mathbf{e}+\mathbf{f})) \end{array} \right] = \frac{1}{2} + \mathsf{negl}(\kappa)$$

where κ is the security parameter.

Intuitively, Hint-LWE assumption says that $\mathbf{y}_0$ looks pseudorandom to an adversary, even when given some randomized leakage on the secret and the error vectors. Kim et al. [32] show that solving Hint-LWE is no easier than solving LWE problem. For a secure LWE instance (λ, m, q, χ) where χ is a discrete Gaussian distribution with standard deviation σ, the corresponding Hint-LWE instance (λ, m, q, χ'), where χ' is a discrete Gaussian distribution with standard deviation σ', is secure when $\sigma' = \sigma/\sqrt{2}$. Consequently, any $\mathbf{e} \in \chi$ can be written as $\mathbf{e}_1 + \mathbf{e}_2$ where $\mathbf{e}_1, \mathbf{e}_2 \in \chi'$. This gives us the real distribution $\mathcal{D}_R$, with the error term re-written and the last ciphertext modified.

$$\left\{ \begin{array}{c} \mathbf{K} = \sum_{i=1}^n \mathbf{k}_i \bmod q \\ \mathbf{y}_1, \ldots, \mathbf{y}_n \end{array} \;\middle|\; \begin{array}{c} \forall i \in [n], \mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^{\lambda}, \mathbf{e}_i, \mathbf{f}_i \xleftarrow{\$} \chi'^m \\ \forall i \in [n-1], \mathbf{y}_i = \mathbf{A}\cdot\mathbf{k}_i + \mathbf{e}_i + \Delta\mathbf{x}_i \\ \mathbf{y}_n = \mathbf{A}\mathbf{K} - \sum_{i=1}^{n-1}\mathbf{y}_i + \sum_{i=1}^n(\mathbf{e}_i + \mathbf{f}_i) + \Delta\mathbf{X} \end{array} \right\}$$

We now define $\mathsf{Sim}(\mathbf{A}, \mathbf{X})$:

$\underline{\mathsf{Sim}(\mathbf{A}, \mathbf{X})}$

Sample $\mathbf{u}_1, \ldots, \mathbf{u}_{n-1} \xleftarrow{\$} \mathbb{Z}_q^m$

Sample $\mathbf{k}_1, \ldots, \mathbf{k}_n \xleftarrow{\$} \mathbb{Z}_q^\lambda$

Sample $\mathbf{e}_1, \ldots, \mathbf{e}_n \xleftarrow{\$} \chi'^m$

Sample $\mathbf{f}_1, \ldots, \mathbf{f}_n \xleftarrow{\$} \chi'^m$

Set $\mathbf{K} := \sum_{i=1}^n \mathbf{k}_i \bmod q$

Set $\mathbf{u}_n = \mathbf{A} \cdot \mathbf{K} - \sum_{i=1}^{n-1} \mathbf{u}_i + \sum_{i=1}^n (\mathbf{e}_i + \mathbf{f}_i) + \Delta \cdot \mathbf{X}$

Return $\mathbf{K}, \mathbf{u}_1, \ldots, \mathbf{u}_n$

In other words, the simulated distribution, $\mathcal{D}_{\mathsf{Sim}}$, is:

$$\left\{ \begin{array}{c|c} \mathbf{K} = \sum_{i=1}^n \mathbf{k}_i \bmod q & \forall\, i \in [n]\ \mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^\lambda, \mathbf{e}_i, \mathbf{f}_i \xleftarrow{\$} \chi'^m \\ \mathbf{u}_1, \ldots, \mathbf{u}_n & \forall\, i \in [n-1]\ \mathbf{u}_i \xleftarrow{\$} \mathbb{Z}_q^m \\ & \mathbf{u}_n = \mathbf{AK} - \sum_{i=1}^{n-1} \mathbf{u}_i + \sum_{i=1}^n (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{X} \end{array} \right\}$$

We will now prove that $\mathcal{D}_R$ is indistinguishable from $\mathcal{D}_{\mathsf{Sim}}$ through a sequence of hybrids.

- Hybrid 0: This is $\mathcal{D}_R$.
- Hybrid 1: In this hybrid, we will replace the real ciphertext $\mathbf{y}_1$ with a modified one. In other words, we set:

$$\left\{ \begin{array}{c|c} \mathbf{K} & \forall\, i \in [n]\ \mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^\lambda, \mathbf{e}_i, \mathbf{f}_i \xleftarrow{\$} \chi'^m, \mathbf{u}_1' \xleftarrow{\$} \mathbb{Z}_q^m \\ \mathbf{y}_1 = \mathbf{u}_1' + \mathbf{f}_1 + \Delta \mathbf{x}_1 & \forall\, i \in [2, n-1]\ \mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{x}_i \\ \{\mathbf{y}_i\}_{i=2}^n & \mathbf{y}_n = \mathbf{AK} - \sum_{i=1}^{n-1} \mathbf{y}_i + \sum_{i=1}^n (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{X} \end{array} \right\}$$

  Now, we will show that if there exists an adversary $\mathcal{B}$ that can distinguish between Hybrid 0 and 1, then we can define an adversary $\mathcal{A}$ who can distinguish the two ensembles in the Hint-LWE Assumption. Let us define $\mathcal{A}$ now.

  $\underline{\mathcal{A}(\mathbf{A}, \mathbf{y}^*, \mathbf{k}^* = \mathbf{k} + \mathbf{r} \bmod q, \mathbf{e}^* = \mathbf{e} + \mathbf{f})}$

  Sample $\mathbf{k}_2, \ldots, \mathbf{k}_{n-1} \xleftarrow{\$} \mathbb{Z}_q^\lambda$

  Sample $\mathbf{e}_2, \ldots, \mathbf{e}_n \xleftarrow{\$} \chi'^m$

  Sample $\mathbf{f}_2, \ldots, \mathbf{f}_n \xleftarrow{\$} \chi'^m$

  Set $\mathbf{K} = \sum_{i=2}^{n-1} \mathbf{k}_i + \mathbf{k}^* \bmod q$ // implicitly, $\mathbf{k}_n := \mathbf{r}$

  $\forall\, i \in \{2, \ldots, n-1\}$, $\mathbf{y}_i = \mathbf{A}\mathbf{k}_i + \mathbf{e}_i + \mathbf{f}_i + \Delta \mathbf{x}_i$

  Set $\mathbf{y}_1 = \mathbf{y}^* + \mathbf{f}_n + \Delta \mathbf{x}_1$

  Set $\mathbf{y}_n := \mathbf{AK} - \sum_{i=1}^{n-1} \mathbf{y}_i + \mathbf{e}^* + \sum_{i=2}^n (\mathbf{e}_i + \mathbf{f}_i) + \Delta \cdot \mathbf{X}$

  Run $b' \xleftarrow{\$} \mathcal{B}(\mathbf{K}, \mathbf{y}_1, \ldots, \mathbf{y}_n)$

  return b'

  We need to argue that the reduction correctly simulates the two hybrids, based on the choice of y^*.

  - If $\mathbf{y}^* = \mathbf{Ak} + \mathbf{e}$, then $\mathbf{y}_1$ is a valid encryption of $\mathbf{x}_1$ with key $\mathbf{k}$ and error $(\mathbf{e} + \mathbf{f}_n)$. Further, it is easy to verify that $\mathbf{y}_n$ satisfies the definition present in Hybrid 0.
  - If $\mathbf{y}^* = \mathbf{u}$ for some random $\mathbf{u}$. Then, we get that $\mathbf{y}_n$ is of the prescribed format, while also guaranteeing that $\mathbf{y}_1$ is generated as expected.
- Hybrid 2: In this hybrid, we will replace $\mathbf{y}_1$ with $\mathbf{y}_1$ that is sampled uniformly at random.

$$\left\{ \begin{array}{c|c} \mathbf{K} & \forall\, i \in [n]\ \mathbf{k}_i \xleftarrow{\$} \mathbb{Z}_q^\lambda, \mathbf{e}_i, \mathbf{f}_i \xleftarrow{\$} \chi'^m, \mathbf{u}_1 \xleftarrow{\$} \mathbb{Z}_q^m \\ \mathbf{u}_1 & \forall\, i \in [2, n-1]\ \mathbf{y}_i = \mathbf{A} \cdot \mathbf{k}_i + (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{x}_i \\ \{\mathbf{y}_i\}_{i=2}^n & \mathbf{y}_n = \mathbf{AK} - \mathbf{u}_1 - \sum_{i=2}^{n-1} \mathbf{y}_i + \sum_{i=1}^n (\mathbf{e}_i + \mathbf{f}_i) + \Delta \mathbf{X} \end{array} \right\}$$

  Hybrid 1, and Hybrid 2 are identically distributed $\mathbf{u}_1'$ is uniformly sampled and essentially mask the values in $\mathbf{y}_1$ of Hybrid 1.

In Hybrids 3 and 4, we replace $\mathbf{y}_2$ with a random element $\mathbf{u}_2$, by using a similar logic. Therefore, in Hybrid $2n-2$, the distribution will resemble $\mathcal{D}_{\mathsf{Sim}}$. This concludes the proof of simulatability.

Privacy against a semi-honest server. Here we prove privacy against an attacker corrupting the server and a set of ηn clients (some of them can be helpers). Denote the simulator as Sim_p. Here, the server acts semi-honestly. The formal proof proceeds through a sequence of hybrids. The sequence of hybrids is similar to the work of Bell et al. [6]. Let $\mathcal{H} = [n] \setminus \mathcal{C}$. Below, we detail the hybrids.

- Hybrid 0: This is the real execution of the protocol where the adversary is interacting with honest parties.
- Hybrid 1: This is where we introduce a simulator Sim which knows all the inputs and secret keys involved, i.e., it knows the keys and the shares of all the clients. Sim runs a full execution of the protocol with the adversary and programs the random oracle as needed. The view of the adversary in this hybrid is indistinguishable from the previous hybrid.
- Hybrid 2: Our next step is for the simulator Sim to rely on the Special Honest Verifier Zero Knowledge (SHVZK) property of all the proof systems to simulate the zero-knowledge proofs for each honest client. Any non-negligible distinguishing advantage between Hybrids 1 and 2 will violate the SHVZK property of the underlying proof systems.
- Hybrid 3: In the next step, we rely on the hiding property of Pedersen commitments. Recall that the hiding property guarantees that there is a negligible distinguishing advantage for an adversary between an actual Pedersen commitment and a random group element. Therefore, for all the honest clients, Sim can simply replace the commitments provided with a random group element. Any non-negligible distinguishing advantage between Hybrids 2 and 3 will violate the hiding property of the commitment scheme.
- Hybrid 4: In the next step, we rely on the privacy property of Shamir Secret Sharing. This guarantees that any insufficient number of shares does not leak the privacy of the secret. In this hybrid Sim uses this property to replace the shares of the honest user's keys meant for the corrupt helpers with random values. Recall that the number of corrupt helpers is strictly less than the reconstruction threshold. Therefore, any non-negligible advantage in distinguishing advantage between Hybrids 3 and 4 will imply that the statistical security of Shamir's Secret Sharing is broken.

  Thus far, for the honest clients' Sim has successfully generated all the contributions for the honest users, except for the ciphertexts themselves. However, Sim cannot simply rely on the semantic security of LWE encryption to replace with encryptions of random values. This is because the output might differ from the real world. Instead, Sim, which has control of the corrupted parties, simply instructs the corrupted parties to provide their inputs as $\mathbf{0}$. Then, the output of the functionality is simply the sum of the honest clients' inputs. Let us call it $\mathbf{x}_H$. With this knowledge, Sim can generate its own choices of individual inputs for honest clients, with the only constraint that the values necessarily need to sum up $\mathbf{x}_H$. This guarantees that the output is correct.
- Hybrid 5: Sim now relies on the semantic security of LWE encryption, under leakage resilience as argued earlier in this section, to instead encrypt these sampled values for honest clients. Any non-negligible distinguishing advantage between Hybrids 4 and 5 will imply that the LWE encryption is no longer semantically secure.

At Hybrid 5, it is clear that Sim can successfully simulate a valid distribution that does not rely on the honest party's inputs. This concludes the proof.

Remark 4 (On privacy of ACORN-robust)**.** A critical artifact of ACORN-robust in [5] is the loop-based resolution of malicious behavior. Specifically, the protocol relies on a looping process by which the server identifies some malicious clients in every round of communication. This is done by finding inconsistencies in the clients' communication. Unfortunately, once a misbehaving client is detected, the protocol must communicate with the parties to retrieve the self-mask *and* the pairwise masks along each edge of the neighborhood graph. Consequently, the server receives all the information necessary to unmask the inputs. Therefore, a malicious server could conceivably claim an honest client to be a misbehaving client, thereby compromising the privacy of the inputs. This is acknowledged by the authors of [5]. However, a simple fix would be for the server to attach necessary proofs of malicious behavior but the communication involved in this process is higher.

Robustness. Now we turn to proving robustness (and also showing privacy) when the adversary corrupts only a set of ηn clients (some can be helpers). Here, the server follows the protocol but can try to violate the privacy.

We denote the simulator here as Sim_r. Note that in the ideal world, Sim_r has to provide the inputs for both the honest and corrupted clients. Meanwhile, in the real world, the inputs for the corrupted clients come from the adversary, call it $\mathcal{B}$. Note that $\mathcal{B}$ can choose these inputs with any restrictions. Therefore, to ensure that it produces a valid set of inputs to the functionality in the ideal world, Sim_r does the following:

- It invokes $\mathcal{B}$ by internally running it. Sim_r honestly follows the protocol, fixing the inputs for the honest clients to be some valid vector $\mathbf{X}$. To $\mathcal{B}$, this is an expected run, and therefore, it behaves exactly like in the real-world execution.
- Sim_r records the set of corrupted parties $\mathcal{A}$ and the set of dropout clients $\mathcal{O}$ encountered in this internal execution.
- At some point, $\mathcal{B}$ provides the NIZK proofs to the server for adversarial clients. However, Sim_r controls the server with these proofs including proof of Shamir sharing, proof of correct encryption, range proofs, and the proof of binding of shares and the key.
- Using the Knowledge Soundness property of the NIZK proofs, Sim_r is able to extract the witnesses, specifically the inputs for the adversarial clients.
- Finally, Sim_r also records whatever $\mathcal{B}$ outputs in the internal execution.

With these steps in place, Sim_r can simulate the ideal world.

- It sends the recorded $\mathcal{O}, \mathcal{A}$ to the ideal functionality.
- It sends the extracted adversarial inputs for those clients, while sending the valid inputs for the non-dropout honest clients.
- Note that the inputs in both the real-world and ideal-world match. We need to show that the computed output matches too.
- Finally, Sim_r outputs whatever $\mathcal{B}$ had output in the internal execution.

It is clear that the output of Sim_r (in the ideal world) is indistinguishable from the output of $\mathcal{B}$ (in the real world). However, we now need to argue that the output sum cannot differ at all. Specifically, while it is guaranteed that the adversarial inputs are included in the sum in the real world (as it was done in the internal execution of $\mathcal{B}$). We need to show that the honest clients' inputs cannot be dropped from the computed sum.

To see this, observe that the server only removes a client if there is a proof of the client misbehaving. As a corollary, it implies that an honest party's input is never rejected by the honest server as it would not have proof of malicious behavior. This guarantees that any honest client's inputs, which hasn't dropped out, is always included in the computed sum in the real world. In other words, the computed sum in the real and ideal world have to match.

C.2 A fix to ACORN-detect

We clarify details related to counting rounds in experiments, point out an overlooked issue in ACORN-detect, and propose a patch.

ACORN-detect, as described in Figure 6 of [5], achieves input validation by integrating the distributed key correctness (DKC) protocol (as described in Figure 2 of [5]) and zero-knowledge proof into the main secure aggregation protocol. The DKC protocol is an interactive protocol which helps the server verify that the masks the server reconstructs is what the clients committed to when sending the masked inputs to the server. In the protocol description of ACORN-detect in Figure 6, communication of the distributed key correctness protocol is embedded into the main protocol, thus there is no additional communication round incurred. However, it seems that the authors overlooked the assumption that the clients can drop offline in any round in the protocol execution when plugging the DKC protocol into ACORN-detect. More specifically, the set of clients who participate in step 8 in ACORN-detect (which contains step 3 of DKC) in which each client sends both the masked input and the commitment to the mask the user might be a superset of the set of clients participating in Step 10 (which contains step 5 of DKC) in which each client sends the server the information needed to verify the commitment of the mask if some clients drop offline between these two rounds. Note that the set $\mathcal{O}$ of clients whose inputs are chosen to be included in the final result is determined when the server receives the masked input in step 9 of ACORN-detect and is not changed later. As a consequence, in the last step

of ACORN-detect (which contains step 6 of DKC), the server is not able to collect all the information needed for the key verification for the online set and the server will abort due to the verification failure even when all participants are honest, which breaks dropout resilience. This problem can be fixed by extracting step 4 and 5 of the DKC protocol from ACORN-detect as a separate round between steps 8 and 9 of ACORN-detect rather than embedded in step 9 and 10 of ACORN-detect and determining the online set $\mathcal{O}$ by who sends both the commitment of the masks and the information needed for the verification of the commitment. This fix introduces one extra round to ACORN-detect.

NeurIPS Paper Checklist

The checklist is designed to encourage best practices for responsible machine learning research, addressing issues of reproducibility, transparency, research ethics, and societal impact. Do not remove the checklist: **The papers not including the checklist will be desk rejected.** The checklist should follow the references and follow the (optional) supplemental material. The checklist does NOT count towards the page limit.

Please read the checklist guidelines carefully for information on how to answer these questions. For each question in the checklist:

- You should answer [Yes] , [No] , or [NA] .
- [NA] means either that the question is Not Applicable for that particular paper or the relevant information is Not Available.
- Please provide a short (1–2 sentence) justification right after your answer (even for NA).

The checklist answers are an integral part of your paper submission. They are visible to the reviewers, area chairs, senior area chairs, and ethics reviewers. You will be asked to also include it (after eventual revisions) with the final version of your paper, and its final version will be published with the paper.

The reviewers of your paper will be asked to use the checklist as one of the factors in their evaluation. While "[Yes] " is generally preferable to "[No] ", it is perfectly acceptable to answer "[No] " provided a proper justification is given (e.g., "error bars are not reported because it would be too computationally expensive" or "we were unable to find the license for the dataset we used"). In general, answering "[No] " or "[NA] " is not grounds for rejection. While the questions are phrased in a binary way, we acknowledge that the true answer is often more nuanced, so please just use your best judgment and write a justification to elaborate. All supporting evidence can appear either in the main paper or the supplemental material, provided in appendix. If you answer [Yes] to a question, in the justification please point to the section(s) where related material for the question can be found.

IMPORTANT, please:

- **Delete this instruction block, but keep the section heading "NeurIPS paper checklist"**,
- **Keep the checklist subsection headings, questions/answers and guidelines below.**
- **Do not modify the questions and only use the provided macros for your answers**.

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: The paper sets out to solve a critical problem in prior work on secure aggregation. In this work, we demonstrate how to guarantee robustness of model, in the face of malicious clients by identifying and removing these bad actors. We demonstrate experiments to show competitive performance over prior work.

   Guidelines:

   - The answer NA means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A No or NA answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

Justification: We have a conclusion paragraph that draws attention to some of the limitations while identifying how they can be remedied in future work.

Guidelines:

- The answer NA means that the paper has no limitation while the answer No means that the paper has limitations, but those are not discussed in the paper.
- The authors are encouraged to create a separate "Limitations" section in their paper.
- The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
- The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
- The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
- The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
- If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
- While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory Assumptions and Proofs**

Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [Yes]

Justification: The paper introduces all the necessary theoretical framework and assumptions for security of the construction. There's detailed proof deferred to the appendix.

Guidelines:

- The answer NA means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental Result Reproducibility**

Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

Answer: [Yes]

Justification: The protocols are well detailed, including the parameter settings for our classifier. We use publicly available ABIDES framework to simulate real-life networking situations.

Guidelines:

- The answer NA means that the paper does not include experiments.
- If the paper includes experiments, a No answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
- If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
- Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
- While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
  (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
  (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
  (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
  (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

   Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

   Answer: [NA]

   Justification: Unfortunately, there was no support for supplementary material upload. However, we are happy to furnish the anonymized code for interested reviewers.

   Guidelines:

   - The answer NA means that paper does not include experiments requiring code.
   - Please see the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - While we encourage the release of code and data, we understand that this might not be possible, so "No" is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
   - The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (https://nips.cc/public/guides/CodeSubmissionPolicy) for more details.
   - The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
   - The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.

- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental Setting/Details**

   Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer, etc.) necessary to understand the results?

   Answer: [NA]

   Justification: We do not perform any training or testing in this work.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
   - The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment Statistical Significance**

   Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

   Answer: [NA]

   Justification: Our running time experiments report the mean as specified.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The authors should answer "Yes" if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.
   - The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
   - The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
   - The assumptions made should be given (e.g., Normally distributed errors).
   - It should be clear whether the error bar is the standard deviation or the standard error of the mean.
   - It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
   - For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g. negative error rates).
   - If error bars are reported in tables or plots, The authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments Compute Resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: We detail the system settings of the device on which experiments are performed.

   Guidelines:

   - The answer NA means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.

- The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
- The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code Of Ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics https://neurips.cc/public/EthicsGuidelines?

   Answer: [Yes]

   Justification: The paper complies with the code of ethics.

   Guidelines:

   - The answer NA means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer No, they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader Impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: The work focuses on privacy of client-held data. This is surveyed in the introduction and motivates why privacy-preserving federated learning is important and its positive impact.

    Guidelines:

    - The answer NA means that there is no societal impact of the work performed.
    - If the authors answer NA or No, they should explain why their work has no societal impact or why the paper does not address societal impact.
    - Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
    - The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
    - The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
    - If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pretrained language models, image generators, or scraped datasets)?

    Answer: [NA]

    Justification: We do not use any of the stated models or data sources.

    Guidelines:

    - The answer NA means that the paper poses no such risks.

- Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
- Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
- We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [NA]

    Justification: This is not applicable for our work.

    Guidelines:

    - The answer NA means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.
    - The name of the license (e.g., CC-BY 4.0) should be included for each asset.
    - For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
    - If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
    - For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
    - If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New Assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [NA]

    Justification: We are not releasing any new assets.

    Guidelines:

    - The answer NA means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and Research with Human Subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [NA]

    Justification: There were no human subjects involved in this project.

    Guidelines:

- The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
- Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
- According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional Review Board (IRB) Approvals or Equivalent for Research with Human Subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [NA]

    Justification: We do not have any research with human subjects that forms a part of this work.

    Guidelines:

    - The answer NA means that the paper does not involve crowdsourcing nor research with human subjects.
    - Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
    - We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
    - For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.