SealAgg: A Lightweight Auditing Framework for One-Shot Secure Aggregation against Malicious Servers

Ge Gao[1], Haining Yu[1⋆], Yue Sun[1], and Zhongyun Hua[2]

[1] Harbin Institute of Technology, Harbin, China
gaoge92021@163.com, yuhaining@hit.edu.cn, alittleyellowduck@163.com
[2] Harbin Institute of Technology, Shenzhen, China
huazhongyun@hit.edu.cn

Abstract. Trustworthy federated learning requires both update privacy and aggregate integrity. While secure aggregation protects the confidentiality of client updates, it does not prevent a malicious server from tampering with the final aggregate. Existing verifiable schemes typically address this limitation by introducing separate integrity-verification layers, such as zero-knowledge proofs, homomorphic hashes, and commitment-based mechanisms. However, these approaches either require clients to remain online after uploading their updates for additional verification or recovery procedures, or incur substantial computation overhead that scales linearly with the model dimension. To address these limitations, we propose SealAgg, a one-shot verifiable secure aggregation framework that simultaneously guarantees update confidentiality and aggregate integrity against a malicious server. SealAgg allows clients to disconnect immediately after uploading their updates and introduces an *aggregate-native* auditing mechanism that enables lightweight integrity verification without a separate costly verification pipeline. Specifically, each client embeds dual hidden finite-field linear projections and a context-binding heartbeat value into its update before encryption, so that the audit material is co-aggregated with the gradients along the same path. Moreover, we design an asymmetric three-server architecture that confines integrity validation entirely to the server side, thereby fully supporting one-shot client participation. We provide formal analyses of the correctness and security of SealAgg. Extensive experiments show that SealAgg achieves practical end-to-end efficiency, incurs only a small auditing overhead relative to the total cost, and outperforms state-of-the-art schemes.

Keywords: Federated Learning · Secure Aggregation · Verifiable Aggregation · Malicious Server

1 Introduction

Federated learning (FL) enables collaborative model training over decentralized data. In a typical FL workflow, each client trains a model locally and submits its

⋆ Corresponding author.

update to a server for aggregation. Since uploaded updates may leak sensitive information about local data, secure aggregation techniques allow the server to compute the aggregate while preserving the confidentiality of individual updates. However, recent studies have shown that a malicious server may manipulate uploaded updates to bias the final aggregate and thereby alter the resulting global model by dropping, replacing, duplicating, or perturbing them [7,25,14]. As a result, a trustworthy federated learning framework should achieve both update confidentiality and aggregate integrity.

To address this issue, many studies have extended secure aggregation with a separate integrity-verification layer [26,11,7,2,21,31,14], which relies on cryptographic primitives such as zero-knowledge proofs (ZK) [25], message authentication codes (MAC) [14], homomorphic hashes (HH) [16,21,31], and commitments [8,10,7]. For example, representative masking-based schemes [26,11,7,2] employ homomorphic hashes or homomorphic commitments to provide aggregate integrity in addition to update privacy. At a high level, clients in these schemes upload not only their masked updates but also update-dependent verification material, such as hashes or commitments, which is aggregated separately by the server and later returned to the clients for integrity checking.

However, in cross-device FL deployments, clients are highly dynamic and frequently drop out [5]. Moreover, existing techniques tightly couple integrity verification with the high-dimensional update path. Particularly, processing a d-dimensional gradient vector using HH or commitments typically requires $O(d)$ expensive group operations, such as exponentiations or elliptic-curve multiplications [11,7]. When combined with the costly bilinear pairings required by other designs [26,12], this leads to a serious scalability bottleneck. As a result, the verification overhead grows linearly with the model dimension d, the number of clients n, or even both. Therefore, existing verifiable schemes suffer from two major limitations: (1) they require clients to remain online for subsequent checking, which is unsuitable for real deployments, and (2) their reliance on heavy cryptographic operations incurs substantial overhead.

To mitigate these limitations, recent "one-shot" aggregation systems [15,3] allow clients to upload a single message and disconnect immediately. To preserve integrity under this setting, they typically rely on committee-assisted architectures. For example, the Willow protocol requires each client to submit a ZK proof together with its update, while the server constructs a Merkle-like aggregation tree for verification by a committee. By generating ZK proofs only for constant-sized ciphertexts rather than high-dimensional updates, the scheme reduces the client-side proof cost from $O(d)$ to $O(1)$. However, its overall overhead remains substantial, since the committee must verify each client's proof individually, resulting in cost that scales linearly with the number of clients (e.g., 327 ms per client for $n = 1000$). In contrast, although OPA [15] attempts to avoid this heavy verification pipeline, it shifts the burden to larger subcommittees and increases client communication. Therefore, achieving computationally lightweight integrity auditing remains an open challenge.

In this paper, we present SealAgg, a verifiable secure aggregation FL framework featuring one-shot clients and lightweight integrity checking. Our design builds on the core paradigm of one-shot encrypted aggregation, where high-dimensional updates are protected by a key-and-message homomorphic scheme, while the corresponding keys are securely aggregated through a dedicated subprotocol [29,21,17]. Since directly applying a multi-committee design to cross-device FL leads to practical deployment challenges and requires strong non-collusion assumptions, SealAgg instead adopts an asymmetric three-server architecture consisting of one leader and two auditors. The leader aggregates the high-dimensional payload ciphertexts, whereas all three servers receive compact encrypted session keys that form a separate control path. This control path gates aggregate ciphertext decryption, determines the accepted client set, and enables the auditors to verify aggregate integrity independently.

To avoid the use of expensive cryptographic primitives whose cost scales with the model dimension, SealAgg introduces an *aggregate-native* auditing mechanism. Instead of appending a separate proof-generation and verification pipeline, each client embeds compact audit evidence directly into its update before encryption. Inspired by the linear-algebraic properties in [22], clients inject two hidden finite-field linear projections and a context-binding heartbeat value into their updates. After the global aggregate is decrypted, the auditors evaluate these invariants directly on the plaintext result. This embedded auditing mechanism simultaneously verifies mathematical correctness and context integrity (i.e., consistency with the intended round and accepted client set), thereby eliminating the need for a separate costly verification layer.

The contributions of this paper are summarized as follows.

- We present SealAgg, a one-shot verifiable secure aggregation framework that preserves update privacy and guarantees aggregate integrity against a malicious server, without requiring clients to remain online after upload.
- We design an asymmetric three-server architecture with an *aggregate-native* auditing mechanism. By embedding two hidden finite-field projections and a context-binding heartbeat value into each encrypted update, the framework enables direct verification of the decrypted aggregate while avoiding expensive standalone verification pipelines.
- We formalize the security guarantees of SealAgg, and extensive evaluations show that it scales efficiently with both the number of clients and the model dimension, while incurring only marginal auditing overhead per round.

2 Related Work

A large body of prior work achieves verifiability by adding a separate verification pipeline on top of secure aggregation. VerifyNetv [26] is an early representative, combining pairwise masking with pairing-based homomorphic hashing, but its pairing cost is very high. VeriFL [11] and LightVeriFL [7] replace this with linear homomorphic hashing and equivocal commitments, optimizing the recovery structure to reduce server-side overhead under dropouts from

Table 1. Complexity of integrity mechanisms in verifiable secure aggregation schemes.

Scheme	Main Privacy Backbone	Main Integrity Technique	Client Added Cost	Server Added Cost
VerifyNet [26]	Pairwise masking	Pairing-based HH	$O(d)$	$O(nd)$
VeriFL [11]	Pairwise masking	LHH + EqCom	$O\left(n^2 + \frac{\beta+1}{\beta}d\right)$	$O(n^2)^\dagger$
LightVeriFL [7]	Pairwise masking	LHH + Com	$O\left(n\log n + \frac{\beta+1}{\beta}d\right)$	$O(n\log n)^\dagger$
e-SeaFL [2]	Masking with assisting nodes	HVC	$O(d)$	$O(n+k)$
RLVP-FL [31]	Non-pairwise masking	LHH + Com	$O(d)$	$O(n)$
VPFLI [21]	Paillier HE	LHH	$O(d+n)$	$O(1)$
PROV-FL [9]	AHE + DP	MLHA	$O(d)$	$O(n+d)$
AlphaFL [14]	MPC	InCom + IT-MAC	$O(d)$	$O(nd)$
Camel [27]	Shuffle-based DP	Blind-MAC	$O(s_{\mathsf{cam}})$	$O(N_{\mathsf{sh}} + r_{\mathsf{sh}}d)$
Willow [3]	AHE + KAHE	ZKPoK + nonce + Aggregation-Tree	$O(1)$	$O(n\log n)$
SealAgg	AHE + KAHE	Hidden projection	$O(d)$	$O(d+n)$

Note. n is the number of clients, d is the model dimension, k is the number of assisting nodes, s_{cam} is the number of sampled gradients per client in Camel, $N_{\mathsf{sh}} = ns_{\mathsf{cam}}$ is the number of shuffled gradients, r_{sh} is the number of post-shuffle sampled gradients, and β is the amortization batch size. Abbreviations: HH = homomorphic hash; LHH = linear homomorphic hash; Com = commitment; EqCom = equivocal commitment; HVC = homomorphic vector commitment; InCom = input commitment; MLHA = Multi-key linearly homomorphic authenticator; IT-MAC = information-theoretic message authentication code; Blind-MAC = blind message authentication code; ZKPoK = zero-knowledge proof of knowledge. † denotes recovery overhead under user dropouts during verification.

$O(n^2)$ to $O(n\log n)$. However, these designs still require pairwise key establishment among clients, which scales poorly in cross-device FL. To relax this, e-SeaFL [2] introduces assisting nodes and authenticated homomorphic vector commitments, while RLVP-FL [31] adopts non-pairwise masking. Yet, both still return verification objects to users after upload, and compute homomorphic hashes or commitments over the d-dimensional gradient path. Consequently, their dominant costs remain tied to heavy exponentiation-style operations. A different line changes the integrity primitive itself. PROV-FL [9] uses multi-key linearly homomorphic authenticators but assumes a semi-honest helper setting. AlphaFL [14] and Camel [27] replace exponentiation-heavy checks with information-theoretic or blind MACs over finite fields and shuffles. Still, these do not achieve lightweight server-side integrity alongside one-shot secure aggregation for intermittent clients. As summarized in Table 1, most prior schemes either require post-upload client participation or add verification pipelines whose costs scale with the model dimension d, client population n, or both.

Among the closest baselines, several HE-based verifiable schemes still attach integrity to heavy encrypted-domain operations. VCSA [28] combines symmetric homomorphic encryption with verifiable multi-secret sharing and generalized Pedersen commitments. Other systems [24,21,17] rely on large-integer homomorphic arithmetic over $\mathbb{Z}_{N^2}^*$-style ciphertext domains, such as Paillier-type constructions. Their main bottleneck is that integrity verification remains tied to dimension-scaled public-key operations. The client performs d-dependent exponentiation-level work, and the server incurs $O(nd)$ encrypted-domain overhead. Since $\mathbb{Z}_{N^2}^*$ exponentiation is highly expensive, binding it directly to a

high-dimensional gradient path causes severe latency at deep-learning scales. Yang [29] mitigates this by moving the homomorphic path to LWE-based lattice operations, but its integrity component still requires elliptic-curve operations. Song [23] uses homomorphic proxy re-encryption, yet integrity remains tied to an auxiliary model-dependent cryptographic path. In short, these schemes realize integrity as a separate verification pipeline rather than a lightweight check on the final plaintext aggregate. Appendix A details this operation-level comparison: while prior schemes report $O(d)$ or $O(nd)$ complexities, these are dominated by exponentiations, pairings, or elliptic-curve arithmetic. In contrast, SealAgg confines lattice homomorphic work to $O(1)$ and uses fast finite-field arithmetic for the remaining $O(d)$ term.

Finally, we discuss one-shot secure aggregation systems closest to our deployment target. Flamingo [19] optimizes multi-round FL via reusable setups and lightweight helper work, but assumes a PKI for all clients and a fixed participant set. LERNA [18] amortizes initial setup across repeated aggregations, but incurs extremely high communication and requires a stateful committee. OPA [15] and Willow [3] advance further toward cross-device settings by supporting one-shot clients. OPA enables dynamic participation and a stateless committee, but reducing its helper cost requires larger subcommittee structures, inflating client communication. Willow is our closest privacy backbone: it supports one-shot dynamic participation and separates high-dimensional payloads from compact key material. It reduces the client-side integrity-proof cost from $O(d)$ to $O(1)$ by proving only constant-sized ciphertext relations. However, it still relies on separate decryptor and verifier committees and a committee-driven verification pipeline. SealAgg adopts this one-shot encrypted backbone but replaces the heavy multi-committee verification with an asymmetric three-server architecture and an aggregate-native audit path. Consequently, SealAgg extends one-shot secure aggregation to provide lightweight, server-side aggregate integrity for weak, intermittent clients.

3 Preliminaries

Notations. Let λ denote the security parameter. The symbol $\|$ denotes string concatenation. Let $\mathbb{F}_p$ denote the finite field of prime order p. We utilize a collision-resistant hash function H, a pseudorandom function PRF, and a domain-separated expansion function $\mathsf{Expand}(k, \mathsf{tag}\|m)$ that deterministically expands a seed k into a length-m vector.

Distributed Random Secret Sharing. We define a dealer-free (t, n)-threshold secret sharing scheme $\mathsf{VSS} = (\mathsf{RandShare}, \mathsf{Recover})$. The interactive $\mathsf{RandShare}$ protocol allows n parties to jointly sample a random secret s, outputting a public commitment Com and authenticated private shares s_i. We require standard *verifiability* (shares can be checked against Com), *unpredictability/secrecy* (the secret is perfectly hidden against up to $t-1$ collusions), and *robust reconstructability* from any t valid shares.

Threshold Additive Homomorphic Encryption. We define a public-key (t, n)-threshold additive homomorphic encryption scheme featuring distributed key generation and decryption. The scheme $\mathsf{AHE} = (\mathsf{Setup}, \mathsf{KeyGenShare}, \mathsf{KeyAgg}, \mathsf{Enc}, \mathsf{EvalAdd}, \mathsf{PartDec}, \mathsf{Recover})$ allows each party to independently generate a private and public key share pair $(sk_j, pk_j) \leftarrow \mathsf{KeyGenShare}()$. The global public key is aggregated as $pk \leftarrow \mathsf{KeyAgg}(\{pk_j\})$. During decryption, each secret-key holder computes a partial decryption $pd_j \leftarrow \mathsf{PartDec}(ct, sk_j)$. The plaintext is then reconstructed from any t valid partial decryptions via $\mathsf{Recover}$. Beyond standard *homomorphic correctness* and *IND-CPA* security against coalitions of up to $t-1$ parties, our protocol requires *decryption simulatability*: honest partial decryptions must be efficiently simulatable using only the public decryption context and the recovered aggregate witness. This ensures they reveal no additional information about honest secret shares beyond what is implied by an aggregate recovery. We instantiate AHE under the RLWE assumption as in [4,6,3].

Key-Additive Homomorphic Encryption. We define a symmetric key encryption scheme $\mathsf{KAHE} = (\mathsf{Setup}, \mathsf{KeyGen}, \mathsf{Enc}, \mathsf{EvalAdd}, \mathsf{Dec})$ with additive key- and message-homomorphisms: given any two ciphertexts c_1 and c_2 encrypting x_1 and x_2 under keys k_1 and k_2, respectively, $c_1 + c_2$ is a valid encryption of $x_1 + x_2$ under the key $k_1 + k_2$. KAHE must satisfy *aggregate-key leakage resilience*: given a number of ciphertexts encrypted under different KAHE keys, revealing the aggregate key only reveals the sum of the encrypted messages. We instantiate KAHE based on the RLWE assumption as in [3].

4 Problem Statement

4.1 System Model

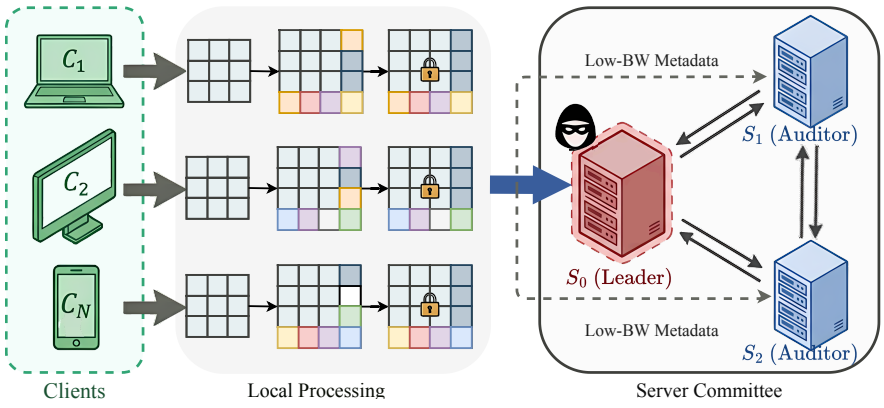


Fig. 1. System overview of SealAgg.

Entities. Figure 1 illustrates the system model of SealAgg. We consider an emerging distributed trust setting with three servers deployed in separate trust domains, denoted by $\mathcal{S} = \{S_0, S_1, S_2\}$, that collaboratively provide the FL service to a dynamic set of clients $\{C_i\}$ holding local datasets. Among them, S_0

serves as the leader, while S_1 and S_2 act as auditors that independently validate the aggregate before release. Clients follow a one-shot participation model: each client performs local training on its data, submits its model update to the servers, and may then disconnect. Each client establishes an authenticated confidential channel with every server. The leader receives both ciphertext paths, whereas each auditor receives only the compact control-path ciphertexts and subsequently performs a single aggregate-level audit before release.

4.2 Threat Model and Security Guarantees

SealAgg aims to provide both input privacy and aggregate integrity for clients, and therefore primarily considers threats from the servers. We assume a non-colluding honest-majority setting. Particularly, each server in $\mathcal{S}$ may individually attempt to infer private information during the protocol execution, and at most one of the three servers may actively deviate from the protocol specification, such as dropping, substituting, or perturbing ciphertexts or recovery material. We further assume that all client-server communication is protected by trusted secure channels, such as TLS [3].

In SealAgg, individual gradients are locally encoded by clients before upload. Except for the final aggregate gradient, the update of each honest client remains hidden from the local view of any single server. Moreover, the integrity of the aggregate gradient is verifiable. Specifically, once the set of client updates included in a given round is determined, if a malicious server attempts to tamper with the corresponding aggregate result, SealAgg detects such misbehavior and aborts the protocol.

5 The design of SealAgg

5.1 Technical overview.

We aim to achieve exact privacy-preserving federated learning under strict one-shot client participation while detecting malicious-server tampering. The privacy layer must add no intentional perturbation, ensuring utility loss stems only from the learning pipeline. However, one-shot aggregation and malicious-server robustness are inherently in tension: the former dictates a single upload without post-upload client involvement, whereas the latter typically requires auxiliary per-client verification material that scales with the number of submissions n or model dimension d, creating a scalability bottleneck.

Among privacy backbones, homomorphic encryption technique best supports non-interactive, one-shot submissions without utility loss, unlike differential privacy or multi-round secure aggregation [20,5,30]. Yet, naively applying public-key HE to high-dimensional gradients is prohibitively expensive, and encrypting only partial updates degrades update confidentiality [13]. To achieve practical

[3] `https://www.rfc-editor.org/rfc/rfc8446`

full-update protection, SealAgg employs a hybrid homomorphic strategy inspired by Willow [3]. The high-dimensional payload is separated from its short decryption capability. The update is encrypted under a lightweight KAHE layer, while only the compact per-round session key is protected via threshold AHE. This shifts the expensive public-key operations from the full gradient vector to a constant-size key object, decoupling costly overhead from the model dimension.

Confidentiality alone is insufficient because homomorphic ciphertexts are inherently malleable. Prior systems mitigate this via separate verification layers (e.g., linearly homomorphic hashes, homomorphic authenticators, or MPC-style checking [20,29,11]). However, these introduce computation-heavy paths scaling with n, d, or both. To eliminate the need for a separate costly verification pipeline, SealAgg introduces an *aggregate-native auditing mechanism*. Following [22], clients embed a compact audit structure—two hidden linear projections and a context-binding heartbeat value—directly into their updates prior to encryption. Dual projections are crucial: given a correct aggregate G, a forgery $\widehat{G}$, and their difference $\Delta := \widehat{G} - G$, a single projection (e.g., $\rho^\top \Delta = 0$) cannot guarantee $\Delta = 0$. By enforcing both left and right projections ($\rho^\top \Delta = 0$ and $\Delta\sigma = 0$), SealAgg forces any forgery to evade detection in both row and column spaces, ensuring *algebraic integrity*. Complementarily, the context-binding heartbeat value binds the aggregate to the committed round and client set, allowing auditors to recompute the expected heartbeat and reject mismatches, thereby guaranteeing *context integrity*.

Since one-shot clients cannot participate in post-upload integrity checks, and a single leader cannot self-certify its output, SealAgg distributes trust across an asymmetric three-server architecture. Each client uploads a KAHE-encrypted audited update and a threshold-AHE-encrypted session key. The leader aggregates only the high-dimensional KAHE ciphertexts. The two auditors avoid this payload path, processing only the compact AHE key-path messages to independently establish the participating client set and lock the round context. Following context agreement, the servers invoke threshold key recovery. The aggregate is decrypted, and the auditors verify it against the dual projections and heartbeat derived from the locked client set, aborting upon any mismatch. This asymmetric design confines the high-dimensional aggregation bottleneck to a single server, while preventing the leader from forging results, as the output must satisfy constraints independently locked by the auditors.

5.2 Audited Encoding and Dual-Path Encryption

SealAgg initializes the control plane with system parameters and a long-term threshold AHE public key PK_{AHE} for stable key recovery. For each round r, the three-server committee publishes a commitment $\mathsf{Com}^{(r)}$ and authenticated seed shares $\{\mathsf{sd}_j^{(r)}\}$, where two valid shares reconstruct a fresh seed $\mathsf{sd}^{(r)}$. This guarantees that audit coefficients remain ephemeral and hidden until the round state is firmly committed.

As detailed in Algorithm 1, after verifying $\mathsf{Com}^{(r)}$, each client C_i encodes its local update as $\mathbf{X}_i^{(r)} \in \mathbb{F}_p^{M \times N}$. Using $\mathsf{sd}^{(r)}$, C_i derives left and right multipliers

Algorithm 1: Audited update encoding (AUE).

Input: Round r; verified round seed $\mathsf{sd}^{(r)}$; local update $\mathbf{g}_i^{(r)}$; client index i.
Output: Audited encoding $\widetilde{\mathbf{g}}_i^{(r)}$.

1 Convert the raw local update into the packed matrix:

$$\mathbf{X}_i^{(r)} \leftarrow \mathsf{Encode}(\mathbf{g}_i^{(r)}), \quad \mathbf{X}_i^{(r)} \in \mathbb{F}_p^{M\times N}$$

2 Derive the left- and right-check multipliers, where $\rho_1^{(r)} \in \mathbb{F}_p^{1\times M}$, $\rho_2^{(r)} \in \mathbb{F}_p^{N\times 1}$:

$$\rho_1^{(r)} \leftarrow \mathsf{Expand}(\mathsf{sd}^{(r)}, \texttt{"row"}\|M), \quad \rho_2^{(r)} \leftarrow \mathsf{Expand}(\mathsf{sd}^{(r)}, \texttt{"col"}\|N)$$

3 Compute the hidden left projection: $\mathbf{z}_i^{(r)} \leftarrow \rho_1^{(r)}\mathbf{X}_i^{(r)}$, where $\mathbf{z}_i^{(r)} \in \mathbb{F}_p^{1\times N}$

4 Compute the hidden right projection: $\mathbf{y}_i^{(r)} \leftarrow \mathbf{X}_i^{(r)}\rho_2^{(r)}$, where $\mathbf{y}_i^{(r)} \in \mathbb{F}_p^{M\times 1}$

5 Derive the client-specific heartbeat value: $b_i^{(r)} \leftarrow \mathsf{PRF}_{\mathsf{sd}^{(r)}}(\texttt{"b"}\|r\|i)$

6 Assemble the final audited encoding:

$$\widetilde{\mathbf{g}}_i^{(r)} \leftarrow \begin{bmatrix} [\,\mathbf{X}_i^{(r)} & \mathbf{y}_i^{(r)}\,] \\ [\,\mathbf{z}_i^{(r)} & b_i^{(r)}\,] \end{bmatrix}$$

$(\rho_1^{(r)}, \rho_2^{(r)})$ to compute the hidden projections $\mathbf{z}_i^{(r)}$ and $\mathbf{y}_i^{(r)}$. It also computes a heartbeat $b_i^{(r)} = \mathsf{PRF}_{\mathsf{sd}^{(r)}}(\texttt{"b"}\|r\|i)$. These values are packed with $\mathbf{X}_i^{(r)}$ to form the audited payload $\widetilde{\mathbf{g}}_i^{(r)}$. The dual projections enforce the aggregate's algebraic correctness, while the heartbeat binds it to the specific round and client identity.

To avoid the dimension-dependent public-key overhead, the client separates payload encryption from key recovery. C_i samples a fresh symmetric KAHE session key k_i to encrypt the high-dimensional payload $\widetilde{\mathbf{g}}_i^{(r)}$, and protects only k_i under the threshold AHE: $c_{\mathsf{K},i}^{(r)} \leftarrow \mathsf{KAHE.Enc}(\widetilde{\mathbf{g}}_i^{(r)}, k_i)$, $\quad c_{\mathsf{A},i}^{(r)} \leftarrow \mathsf{AHE.Enc}(PK_{\mathsf{AHE}}, k_i)$. For one-shot submission, C_i sends $(c_{\mathsf{K},i}^{(r)}, c_{\mathsf{A},i}^{(r)})$ to the leader S_0, which aggregates the payload path and forwards the compact control path $c_{\mathsf{A},i}^{(r)}$ to auditors S_1 and S_2. The client then safely disconnects for the remainder of the round.

5.3 Integrity Auditing and Model Update

After collecting client submissions, the leader S_0 determines its accepted client set $\mathcal{C}_0^{(r)}$ and homomorphically aggregates the paths into $C_{\mathsf{K}}^{(r)}$ and $C_{\mathsf{A}}^{(r)}$. In parallel, each auditor S_j forms its local view $\mathcal{C}_j^{(r)}$ from the received control-path ciphertexts. Before decryption shares are released, the leader must commit to a consistent round state. Thus, it computes the digest $h_{\mathsf{A}}^{(r)} := \mathsf{H}(C_{\mathsf{A}}^{(r)})$ and broadcasts the candidate round claim $T^{(r)} := \big(\mathcal{C}_0^{(r)}, C_{\mathsf{K}}^{(r)}, C_{\mathsf{A}}^{(r)}, h_{\mathsf{A}}^{(r)}\big)$ to the auditors.

The auditors use the control plane to verify this claim. Each auditor checks participant set consistency $\mathcal{C}_j^{(r)} = \mathcal{C}_0^{(r)}$, recomputes the control-path aggregate and digest, and enforces the participation threshold $n_r := |\mathcal{C}_0^{(r)}| \geq \tau$. This threshold prevents a malicious leader from trivializing the anonymity set to compromise privacy or bias the round. An auditor releases its partial decryption share $\mathsf{pd}_j^{(r)}$ and seed share $\mathsf{sd}_j^{(r)}$ only if the claim matches its local view as in Algorithm 2.

Algorithm 2: Key recovery and decryption (KRD).

Input: Role $j \in \{0,1,2\}$; round state $T^{(r)}$; for auditors S_j with $j \in \{1,2\}$, local accepted set $\mathcal{C}_j^{(r)}$ and locally received ciphertexts $\{c_{\mathsf{A},i}^{(r)}\}_{i\in\mathcal{C}_j^{(r)}}$ from clients; AHE secret share sk_j^{AHE}; authenticated seed share $\mathsf{sd}_j^{(r)}$.

Output: Decrypted aggregate state $\widetilde{\mathbf{G}}^{(r)}$, or $\perp$.

if $j \in \{1,2\}$ **then**
  if $\mathcal{C}_j^{(r)} \neq \mathcal{C}_0^{(r)}$ **then**
    return $\perp$
  Recompute $\widehat{C}_{\mathsf{A},j}^{(r)} \leftarrow \mathsf{AHE.EvalAdd}(\{c_{\mathsf{A},i}^{(r)}\}_{i\in\mathcal{C}_j^{(r)}})$, $\widehat{h}_{\mathsf{A},j}^{(r)} \leftarrow \mathsf{H}(\widehat{C}_{\mathsf{A},j}^{(r)})$, $n_r \leftarrow |\mathcal{C}_0^{(r)}|$
  if $\widehat{h}_{\mathsf{A},j}^{(r)} \neq h_{\mathsf{A}}^{(r)}$ ***or*** $n_r < \tau$ **then**
    return $\perp$
Compute a partial decryption share: $\mathsf{pd}_j^{(r)} \leftarrow \mathsf{AHE.PartDec}(sk_j^{\mathsf{AHE}}, C_{\mathsf{A}}^{(r)})$
Disseminate $(\mathsf{sd}_j^{(r)}, \mathsf{pd}_j^{(r)})$ to the server committee
if $j \in \{1,2\}$ **then**
  Reconstruct and store $\mathsf{sd}^{(r)} \leftarrow \mathsf{VSS.Recover}(\{\mathsf{sd}_k^{(r)}\}_{k=0}^{2})$ for AVR
Reconstruct the aggregate key: $K_{\mathrm{agg}}^{(r)} \leftarrow \mathsf{AHE.Recover}(C_{\mathsf{A}}^{(r)}, \{\mathsf{pd}_k^{(r)}\}_{k=0}^{2})$
if $K_{\mathrm{agg}}^{(r)} = \perp$ **then**
  return $\perp$
Decrypt: $\widetilde{\mathbf{G}}^{(r)} \leftarrow \mathsf{KAHE.Dec}(C_{\mathsf{K}}^{(r)}, K_{\mathrm{agg}}^{(r)})$
if $\widetilde{\mathbf{G}}^{(r)}$ *cannot be parsed into the expected audited layout* **then**
  return $\perp$
return $\widetilde{\mathbf{G}}^{(r)}$

In our instantiation, two valid shares can reconstruct the aggregated decryption key $K_{\mathrm{agg}}^{(r)}$ and round seed $\mathsf{sd}^{(r)}$. Crucially, honest servers only release shares after validating the candidate claim.

However, recovering the correct aggregated key only authenticates the control path. Because homomorphic ciphertexts are malleable, a malicious leader could tamper with the payload ciphertext $C_{\mathsf{K}}^{(r)}$ while forging a consistent $C_{\mathsf{A}}^{(r)}$. To prevent this, SealAgg performs a second, payload-level audit after decryption, once the seed $\mathsf{sd}^{(r)}$ is reconstructed. Each auditor uses this seed to re-derive $\rho_1^{(r)}$ and $\rho_2^{(r)}$, recomputes the expected heartbeat sum for the agreed client set $\mathcal{C}^{(r)} := \mathcal{C}_0^{(r)}$, and parses the decrypted aggregate $\widetilde{\mathbf{G}}^{(r)}$. As outlined in Algorithm 3, the audit passes only if both projection identities and the heartbeat relation hold. Projections prevent algebraic inconsistencies, while the heartbeat binds an otherwise valid aggregate to the correct round and participant set.

Once the audit passes, each auditor deterministically decodes the averaged update, applies the model transition, and hashes the resulting next-model view. The new model is released only if the two auditor hashes match. This final agreement step prevents a malicious leader from turning one audited aggregate into

Algorithm 3: Audited verification and release (AVR).

Input: Auditor role $j \in \{1,2\}$; current global model $\mathbf{w}^{(r)}$; learning rate η_r; accepted client set $\mathcal{C}^{(r)} := \mathcal{C}_0^{(r)}$; decrypted aggregate state $\widetilde{\mathbf{G}}^{(r)}$; round seed $\mathsf{sd}^{(r)}$ reconstructed from the server shares.

Output: Updated global model $\mathbf{w}^{(r+1)}$, or $\perp$.

Derive the hidden left-check multiplier: $\rho_1^{(r)} \leftarrow \mathsf{Expand}(\mathsf{sd}^{(r)}, \texttt{"row"} \| M)$

Derive the hidden right-check multiplier: $\rho_2^{(r)} \leftarrow \mathsf{Expand}(\mathsf{sd}^{(r)}, \texttt{"col"} \| N)$

Parse the decrypted aggregate state as

$$\widetilde{\mathbf{G}}^{(r)} = \begin{bmatrix} [\,\mathbf{G}^{(r)} & \mathbf{y}_{\mathrm{agg}}^{(r)}\,] \\ [\,\mathbf{z}_{\mathrm{agg}}^{(r)} & b_{\mathrm{agg}}^{(r)}\,] \end{bmatrix}.$$

Recompute the expected heartbeat sum: $b_{\mathrm{exp}}^{(r)} \leftarrow \sum_{i \in \mathcal{C}^{(r)}} \mathsf{PRF}_{\mathsf{sd}^{(r)}}(\texttt{"b"} \| r \| i)$

if $b_{\mathrm{agg}}^{(r)} \neq b_{\mathrm{exp}}^{(r)}$ ***or*** $\mathbf{z}_{\mathrm{agg}}^{(r)} \neq \rho_1^{(r)} \cdot \mathbf{G}^{(r)}$ ***or*** $\mathbf{y}_{\mathrm{agg}}^{(r)} \neq \mathbf{G}^{(r)} \cdot \rho_2^{(r)}$ **then**

  return $\perp$

Compute the averaged model update in model space: $\bar{\mathbf{g}}^{(r)} \leftarrow \mathsf{Decode}(\mathbf{G}^{(r)})/|\mathcal{C}^{(r)}|$

Update the global model: $\mathbf{w}^{(r+1)} \leftarrow \mathit{\Pi}_{\mathcal{G}}\big(\mathbf{w}^{(r)} - \eta_r \bar{\mathbf{g}}^{(r)}\big)$

Hash the local next-model view: $h_j^{(r)} \leftarrow \mathsf{H}(\mathbf{w}^{(r+1)})$

Exchange model hashes with the peer auditor

if $h_j^{(r)} \neq h_{3-j}^{(r)}$ **then**

  Abort because the two auditor views disagree

  return $\perp$

Broadcast the agreed next global model $\mathbf{w}^{(r+1)}$ to clients

divergent client-visible states and makes the state transition jointly authorized rather than leader dictated.

6 Security Analysis

6.1 Correctness

Theorem 1 (Round correctness). *Fix a round r. Let $\mathcal{C}^{(r)}$ be the accepted client set in the surviving claim $T^{(r)}$, and write $n_r := |\mathcal{C}^{(r)}| \geq \tau$. Assume that every client in $\mathcal{C}^{(r)}$ honestly executes AUE and the submission step, the leader honestly forms $C_{\mathsf{K}}^{(r)}$, $C_{\mathsf{A}}^{(r)}$, and $T^{(r)}$, both auditors receive exactly the* AHE*-side ciphertexts indexed by $\mathcal{C}^{(r)}$ and verify $h_{\mathsf{A}}^{(r)}$, $(2,3)$-*VSS *recovery returns the unique seed $\mathsf{sd}^{(r)}$, $(2,3)$-threshold-*AHE *recovery returns $K_{\mathrm{agg}}^{(r)} = \sum_{i \in \mathcal{C}^{(r)}} k_i$,* KAHE *decryption of $C_{\mathsf{K}}^{(r)}$ under $K_{\mathrm{agg}}^{(r)}$ is correct, and the no-wraparound / exact-*Decode *condition holds. Define $\mathbf{X}_i^{(r)} := \mathsf{Encode}(\mathbf{g}_i^{(r)})$, $\widetilde{\mathbf{X}}_i^{(r)} := \mathsf{AUE}(\mathsf{sd}^{(r)}, r, i, \mathbf{X}_i^{(r)})$, and $\mathbf{G}^{(r)} := \sum_{i \in \mathcal{C}^{(r)}} \mathbf{X}_i^{(r)}$. Then KRD does not abort, AVR accepts at both honest auditors, and both output $\mathbf{w}_1^{(r+1)} = \mathbf{w}_2^{(r+1)} = \mathit{\Pi}_{\mathcal{G}}\big(\mathbf{w}^{(r)} - \eta_r \mathsf{Decode}(\mathbf{G}^{(r)})/n_r\big)$. Consequently, $\mathsf{H}(\mathbf{w}_1^{(r+1)}) = \mathsf{H}(\mathbf{w}_2^{(r+1)})$, and the release step succeeds.*

Proof. By correctness of $(2,3)$-VSS, threshold-AHE, and KAHE, both honest auditors reconstruct the same seed $\mathsf{sd}^{(r)}$, the same aggregate key $K_{\mathrm{agg}}^{(r)}$, and the same audited aggregate $\widetilde{\mathbf{G}}^{(r)} := \sum_{i\in\mathcal{C}^{(r)}} \widetilde{\mathbf{X}}_i^{(r)}$. Since all honest parties use the same verified seed, they derive the same hidden projections and heartbeat values, so the two projection checks and the heartbeat check in AVR all pass. The update rule is then deterministic given $\mathbf{G}^{(r)}$, n_r, $\mathbf{w}^{(r)}$, and η_r, hence both auditors compute the same $\mathbf{w}^{(r+1)}$ and therefore the same hash.

6.2 Input Privacy

We assume a single passively corrupted server. To unify the analysis across different server roles, we upper-bound any individual server's view using a leakage:

$$\mathsf{Leak}_{\max}^{(r)} := \big(\mathsf{ra}^{(r)}, \{(c_{\mathsf{K},i}^{(r)}, c_{\mathsf{A},i}^{(r)})\}_{i\in\mathcal{C}^{(r)}}, T^{(r)}, \{(\mathsf{sd}_k^{(r)}, \mathsf{pd}_k^{(r)})\}_{k=0}^{2}, \widetilde{\mathbf{G}}^{(r)}, \mathbf{w}^{(r+1)}\big).$$

Since any single-server view $\mathsf{View}_{S_j}^{(r)}$ can be efficiently computed via a PPT projection of $\mathsf{Leak}_{\max}^{(r)}$ and independent local state st_{S_j}, proving the privacy of $\mathsf{Leak}_{\max}^{(r)}$ is sufficient to guarantee the privacy of any specific server's view.

Definition 1 (Simulation-based input privacy). *Fix a round r and the accepted client set $\mathcal{C}^{(r)}$. Let $\mathsf{Fix}^{(r)}$ denote all round-local variables fixed independently of the honest updates $\{\mathbf{X}_i^{(r)}\}_{i\in\mathcal{C}^{(r)}}$, such as public metadata and the corrupted server's internal state. Define the aggregate key $K_{\mathsf{agg}}^{(r)} := \sum_{i\in\mathcal{C}^{(r)}} k_i$ and the audited aggregate $\widetilde{\mathbf{G}}^{(r)} := \sum_{i\in\mathcal{C}^{(r)}} \widetilde{\mathbf{g}}_i^{(r)}$. The ideal leakage is*

$$\mathsf{IdLeak}^{(r)} := \big(\mathsf{Fix}^{(r)}, K_{\mathsf{agg}}^{(r)}, \widetilde{\mathbf{G}}^{(r)}, \mathbf{w}^{(r+1)}\big).$$

The protocol Π achieves simulation-based passive input privacy *against a single passive corrupted server if there exists a PPT simulator* Sim *such that for any PPT distinguisher $\mathcal{A}$:*

$$\Big|\Pr[\mathcal{A}(\mathsf{Leak}_{\max}^{(r)}) = 1] - \Pr[\mathcal{A}(\mathsf{Sim}(\mathsf{IdLeak}^{(r)})) = 1]\Big| \le \mathsf{negl}(\lambda).$$

Theorem 2 (Input privacy against one server). *Assume the* AHE *scheme is IND-CPA secure with decryption simulatability, and the* KAHE *scheme provides leakage-resilient semantic security under aggregate-key leakage [3]. Then Π achieves simulation-based passive input privacy against any PPT adversary passively corrupting a single server.*

Proof (Proof Sketch). The proof proceeds via a sequence of hybrid games. First, exploiting the decryption simulatability of threshold-AHE, we replace honest partial decryptions with simulated equivalents that reveal nothing beyond the aggregate key. Next, invoking the IND-CPA security of AHE, we substitute all honest control-path ciphertexts with dummy encryptions. Finally, the leakage-resilient semantic security of KAHE ensures the high-dimensional payload ciphertexts can be securely simulated from the audited aggregate. The formal proof is deferred to Appendix B.1.

6.3 Auditable Aggregate Integrity

We upper-bound the wrong-release probability by decoupling *context forgery* from *algebraic forgery*. Let $\mathsf{Adv}^{\mathsf{set}}_{\mathsf{PRF},\mathcal{B}}(\lambda)$ denote the multi-point PRF advantage of distinguishing $\{\mathsf{PRF}_{\mathsf{sd}^{(r)}}(\texttt{"b"}\|r\|i)\}_{i\in\mathcal{C}^{(r)}}$ from independent uniform scalars in $\mathbb{F}_p$. Similarly, let $\mathsf{Adv}^{\mathsf{2prg}}_{\mathsf{Expand},\mathcal{B}}(\lambda)$ denote the advantage of distinguishing the expanded pair $(\mathsf{Expand}(\mathsf{sd},\texttt{"row"}\|M),\mathsf{Expand}(\mathsf{sd},\texttt{"col"}\|N))$ from an independent, uniformly random pair in $\mathbb{F}_p^{1\times M}\times\mathbb{F}_p^{N\times 1}$.

Theorem 3 (Auditable aggregate integrity). *Fix a round r. Assume that at most one server is corrupted, honest auditors execute KRD and AVR, seed shares are revealed only after a fixed claim $T^{(r)}$ passes the control-path checks,* H *is collision resistant, and* $(2,3)$-VSS *recovery together with* $(2,3)$*-threshold-*AHE *recovery is robust for the accepted claim. Let $\mathcal{C}^{(r)}$ be the unique accepted client set determined by the common accepted control claim; if no such common claim exists, the event below is false. Define $\mathbf{G}^{(r)}_{\mathsf{ref}}:=\sum_{i\in\mathcal{C}^{(r)}}\mathbf{X}^{(r)}_i$, $\widetilde{\mathbf{G}}^{(r)}_{\mathsf{ref}}:=\sum_{i\in\mathcal{C}^{(r)}}\widetilde{\mathbf{X}}^{(r)}_i$, and let $\mathbf{w}^{(r+1)}_{\mathsf{ref}}$ be the output of AVR on $\widetilde{\mathbf{G}}^{(r)}_{\mathsf{ref}}$ under round inputs. Then for every PPT adversary $\mathcal{A}$ there exist PPT adversaries $\mathcal{B}_{\mathsf{H}}$, $\mathcal{B}_{\mathsf{PRF}}$, and $\mathcal{B}_{\mathsf{Expand}}$ such that*

$$\begin{aligned}&\Pr\big[\exists\ \textit{honest auditor outputting non-}\perp,\ \mathbf{w}^{(r+1)}_{\mathsf{rel}}\neq\mathbf{w}^{(r+1)}_{\mathsf{ref}}\big]\\&\quad\le\mathsf{Adv}^{\mathsf{cr}}_{\mathsf{H},\mathcal{B}_{\mathsf{H}}}(\lambda)+\mathsf{Adv}^{\mathsf{set}}_{\mathsf{PRF},\mathcal{B}_{\mathsf{PRF}}}(\lambda)+\mathsf{Adv}^{\mathsf{2prg}}_{\mathsf{Expand},\mathcal{B}_{\mathsf{Expand}}}(\lambda)+p^{-1}+p^{-2}.\end{aligned}$$

Moreover, conditioned on a nonzero payload deviation $\Delta_G:=\mathbf{G}^{(r)}_{\mathsf{rel}}-\mathbf{G}^{(r)}_{\mathsf{ref}}$ of rank $r_\Delta\ge 1$, the algebraic term refines to p^{-2r_Δ}. Thus, if $\mathsf{Adv}^{\mathsf{cr}}_{\mathsf{H}}(\lambda)$, $\mathsf{Adv}^{\mathsf{set}}_{\mathsf{PRF}}(\lambda)$, and $\mathsf{Adv}^{\mathsf{2prg}}_{\mathsf{Expand}}(\lambda)$ are $\mathsf{negl}(\lambda)$ and $p\ge 2^\lambda$, the wrong-release probability is $\mathsf{negl}(\lambda)$.

Proof (Proof Sketch). We upper-bound the adversary's success probability by decoupling context from algebraic forgery. Assuming a collision-resistant hash, the accepted control claim is uniquely bound. Bypassing context verification requires breaking PRF security, bounded by $\mathsf{Adv}^{\mathsf{set}}_{\mathsf{PRF}}(\lambda)$. Furthermore, introducing payload deviations while satisfying dual hidden projections requires guessing the expanded multipliers. This algebraic forgery is bounded by $\mathsf{Adv}^{\mathsf{2prg}}_{\mathsf{Expand}}(\lambda)$ and the field size $\mathcal{O}(p^{-2})$. The formal proof is deferred to Appendix B.2.

7 Experimental Evaluation

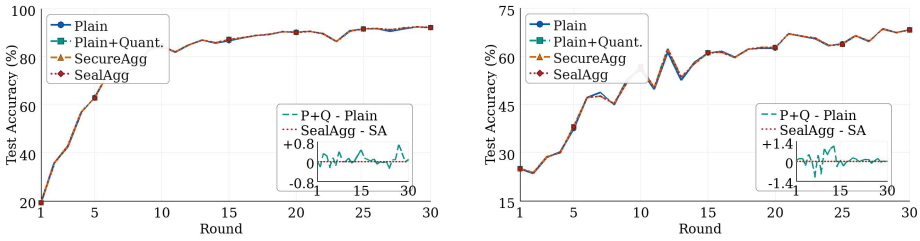
We implement a prototype of SealAgg in C++17 using OpenSSL for SHA-256 and HMAC-SHA256. The implementation supports both single-node simulation and full end-to-end execution in the three-server setting. For the underlying cryptographic components, we build the RLWE-based KAHE and threshold-AHE layers on top of SHELL [4], with ring degrees $N_1=N_2=4096$. We use the lattice estimator [1] to assess the hardness of the RLWE parameter sets used in KAHE and AHE, and choose parameters that provide at least 128-bit computational security. All experiments are conducted on a workstation equipped with dual

[4] `https://github.com/google/shell-encryption`

Intel Xeon Gold 6133 CPUs (40 physical cores / 80 hardware threads, 2.50 GHz), one NVIDIA A800 80GB PCIe GPU, 1.0 TiB RAM, and Ubuntu 24.04.1 LTS. Unless otherwise stated, end-to-end protocol experiments are run on the CPU with multi-threading enabled, using up to 32 threads.

7.1 Utility

We first evaluate the utility of SealAgg under shard-based non-IID federated training. We compare four schemes: Plain (plaintext training without quantization or cryptographic protection), Plain+Quantization (quantized training without secure aggregation), SecureAgg-only (quantized secure aggregation without integrity protection), and Our Scheme (SealAgg). Experiments are conducted on MNIST with the same small CNN as in [27] ($d = 9594$) and on Fashion-MNIST with the same 2NN model with two hidden layers of width 200 as in [27] ($d = 199{,}210$). Figure 2 shows that SealAgg closely matches SecureAgg-only on both datasets, indicating that integrity protection has essentially no visible impact on convergence. The small inset further shows that the larger utility gap comes from quantization, not from the integrity layer.



(a) MNIST (b) FMNIST

Fig. 2. Utility performance of four training pipelines.

Table 2. Ablation of integrity-layer computational overhead.

Method	Dataset	Per-Client (ms)		Server-Side (ms)			
		Offline	Online	Aggreg.	Audit	S_0 Total	$S_{1,2}$ Total
SecureAgg-only	MNIST	25.7	0.37	2.81	–	146.2	145.5
	FMNIST	30.7	6.68	21.43	–	182.8	154.7
SealAgg	MNIST	25.8	0.42	2.98	0.16	148.2	150.5
	FMNIST	31.9	7.31	22.20	0.60	202.4	194.0

Efficiency. We further conduct microbenchmarks to isolate the extra cost of integrity enforcement. Table 2 shows that the added overhead is small. Threshold-AHE encryption of session keys is precomputable and counted as client offline cost. For the server side, aggregation and audit report the core cryptographic overheads, while S_0 and $S_{\{1,2\}}$ report end-to-end wall-time. Relative to the SecureAgg-only version, aggregation adds only 0.17–0.77 ms and audit adds 0.16–0.60 ms, so S_0's round time increases by at most 10.7%.

Table 3. Overhead comparison with Camel and BatchCrypt.

Scheme	Dataset	Per-Client (ms)		Server-side (ms)	
		Offline	Online	$n = 10^3$	$n = 10^5$
Camel	MNIST	–	2.52	**122**	4,242
	FMNIST	–	13.02	**125**	4,304
BatchCrypt	MNIST	–	1,459	583	29,262
	FMNIST	–	30,249	5,454	381,699
SealAgg	MNIST	25.8	**0.42**	161	**1,210**
	FMNIST	31.9	**7.31**	189	**2,023**

Table 4. Microbenchmarks of concrete computation and communication costs across model dimensions d and client scales n.

(a) Client computation and communication overhead

Model (d)	Inter-Server Comm. (MB)			C→S Comm. (MB)		Client Comp. (ms)	
	S_0 TX	$S_{1,2}$ RX	$S_{1,2}$ TX	$\to S_0$	$\to S_{1,2}$	Offline	Online
10^3	0.079	0.070	0.061	0.070	0.060	29.37	0.146
10^5	1.690	0.875	0.061	0.875	0.060	30.42	4.288
10^7	160.2	80.11	0.061	80.12	0.060	64.86	413.3

(b) Server-side workload per round

Model (d)	Clients (n)	Micro-Benchmark (ms)		Total Round Runtime (s)	
		Aggregation	Audit	S_0 Total	$S_{1,2}$ Total
10^3	10^2	0.475	0.048	0.102	0.088
	10^3	0.865	0.047	0.158	0.158
	10^4	3.481	0.047	0.328	0.328
	10^5	16.74	0.048	1.307	1.307
10^5	10^2	1.957	0.302	0.176	0.086
	10^3	4.639	0.485	0.170	0.170
	10^4	19.69	0.493	0.417	0.367
	10^5	151.9	0.604	1.709	1.533
10^7	10^2	20.57	22.09	0.744	0.072
	10^3	141.7	22.22	1.475	1.417
	10^4	1250	22.28	5.364	2.093
	10^5	10791	22.33	53.37	4.937

Efficiency Comparison. Table 3 compares SealAgg with BatchCrypt [30] and Camel [27]. Compared with the HE-oriented BatchCrypt, SealAgg reduces both client online cost and overall server runtime. Compared with Camel, a three-server DP-based framework that also targets privacy and integrity, SealAgg remains competitive: its fixed cost is higher at small scale, but the server-side runtime gap narrows and then reverses as the client population grows. At larger scales, SealAgg achieves lower server-side runtime while keeping client online cost below 10 ms.

7.2 Scalability

We evaluate the scalability of SealAgg via microbenchmarks using simulated client gradients. As Figure 3 and Table 4 show, per-client cost is nearly invariant to n, leader runtime scales with aggregation workload, and audit overhead

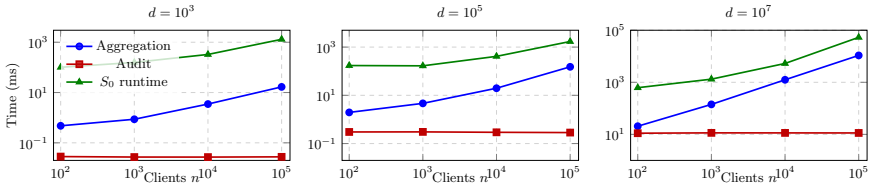


Fig. 3. Server-side scaling of SealAgg with the client population.

remains flat for a fixed d. Consequently, SealAgg scales efficiently with client population while ensuring lightweight verification.

8 Conclusion

We presented SealAgg, a verifiable secure aggregation framework for federated learning in a non-colluding three-server setting. It provides update confidentiality against any single server and aggregate integrity against one actively malicious server, while supporting one-shot participation that lets clients disconnect immediately after upload. We achieve this via an asymmetric three-server architecture and aggregate-native auditing. By embedding lightweight audit material into updates prior to encryption, our design shifts integrity verification entirely to the server, eliminating separate costly verification pipelines and heavy cryptographic proofs. Formal analysis and experiments confirm that SealAgg scales efficiently with both client count and model dimension, while the embedded auditing incurs only a marginal fraction of the per-round computational cost.

References

1. Albrecht, M.R., Player, R., Scott, S.: On the concrete hardness of learning with errors. J. Math. Cryptol. **9**(3), 169–203 (2015)
2. Behnia, R., Riasi, A., Ebrahimi, M., Chow, S.S.M., Padmanabhan, B., Hoang, T.: Efficient secure aggregation for privacy-preserving federated machine learning. In: ACSAC. pp. 778–793 (2024)
3. Bell-Clark, J., Gascón, A., Li, B., Raykova, M., Schoppmann, P.: Willow: Secure aggregation with one-shot clients. In: CRYPTO. pp. 285–318 (2025)
4. Bendlin, R., Damgård, I.B.: Threshold decryption and zero-knowledge proofs for lattice-based cryptosystems. In: TCC. pp. 201–218 (2010)
5. Bonawitz, K.A., Ivanov, V., Kreuter, B., Marcedone, A., McMahan, H.B., Patel, S., Ramage, D., Segal, A., Seth, K.: Practical secure aggregation for privacy-preserving machine learning. In: ACM CCS. pp. 1175–1191 (2017)
6. Boudgoust, K., Scholl, P.: Simple threshold (fully homomorphic) encryption from LWE with polynomial modulus. In: ASIACRYPT. pp. 371–404 (2023)
7. Buyukates, B., So, J., Mahdavifar, H., Avestimehr, S.: LightVeriFL: A lightweight and verifiable secure aggregation for federated learning. IEEE J. Sel. Areas Inf. Theory **5**, 285–301 (2024)
8. Catalano, D., Fiore, D., Tucker, I.: Additive-homomorphic functional commitments and applications to homomorphic signatures. In: ASIACRYPT. pp. 159–188 (2022)
9. Dasu, V.A., Sarkar, S., Mandal, K.: PROV-FL: Privacy-preserving round optimal verifiable federated learning. In: AISec. pp. 33–44 (2022)

10. Groth, J.: Efficient zero-knowledge arguments from two-tiered homomorphic commitments. In: ASIACRYPT. pp. 431–448 (2011)
11. Guo, X., Liu, Z., Li, J., Gao, J., Hou, B., Dong, C., Baker, T.: VeriFL: Communication-efficient and fast verifiable aggregation for federated learning. IEEE Trans. Inf. Forensics Secur. **16**, 1736–1751 (2021)
12. Hahn, C., Kim, H., Kim, M., Hur, J.: VerSA: Verifiable secure aggregation for cross-device federated learning. IEEE Trans. Dependable Secur. Comput. **20**(1), 36–52 (2023)
13. Hu, C., Li, B.: MaskCrypt: Federated learning with selective homomorphic encryption. IEEE Trans. Dependable Secur. Comput. **22**(1), 221–233 (2025)
14. Jiang, Y., Zarezadeh, M., Dai, T., Köpsell, S.: AlphaFL: Secure aggregation with malicious^2 security for federated learning against dishonest majority. Proc. Priv. Enhancing Technol. **2025**(4), 348–368 (2025)
15. Karthikeyan, H., Polychroniadou, A.: OPA: One-shot private aggregation with single client interaction and its applications to federated learning. In: CRYPTO. pp. 319–353 (2025)
16. Krohn, M.N., Freedman, M.J., Mazières, D.: On-the-fly verification of rateless erasure codes for efficient content distribution. In: IEEE S&P. pp. 226–240 (2004)
17. Lai, P., Zhang, M., Tang, Y., Yue, Y., Di, F.: VPAFL: Verifiable privacy-preserving aggregation for federated learning based on single server. Comput. Mater. Contin. **84**(2), 2935–2957 (2025)
18. Li, H., Lin, H., Polychroniadou, A., Tessaro, S.: LERNA: Secure single-server aggregation via key-homomorphic masking. In: ASIACRYPT. pp. 302–334 (2023)
19. Ma, Y., Woods, J., Angel, S., Polychroniadou, A., Rabin, T.: Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In: IEEE S&P. pp. 477–496 (2023)
20. Mansouri, M., Önen, M., Ben Jaballah, W., Conti, M.: SoK: Secure aggregation based on cryptographic schemes for federated learning. Proc. Priv. Enhancing Technol. **2023**(1), 140–157 (2023)
21. Ren, Y., Li, Y., Feng, G., Zhang, X.: VPFLI: Verifiable privacy-preserving federated learning with irregular users based on single server. IEEE Trans. Serv. Comput. **18**(2), 1124–1136 (2025)
22. Santriaji, M.H., Xue, J., Zhang, Y., Lou, Q., Solihin, Y.: DataSeal: Ensuring the verifiability of private computation on encrypted data. In: IEEE S&P. pp. 2378–2394 (2025)
23. Song, Z., Huang, Z., Cai, Y.: A verifiable privacy-preserving federated learning scheme based on homomorphic proxy re-encryption. Inf. Sci. **730**, 122875 (2026)
24. Wang, X., Chen, T., Dai, H.N., Long, P., Yang, H., Xiong, Z., Susilo, W.: A privacy-enhanced method for privacy-preserving and verifiable federated learning. IEEE Internet Things J. **12**(14), 26768–26781 (2025)
25. Wang, Z., Dong, N., Sun, J., Knottenbelt, W.J., Guo, Y.: zkFL: Zero-knowledge proof-based gradient aggregation for federated learning. IEEE Trans. Big Data **11**(2), 447–460 (2025)
26. Xu, G., Li, H., Liu, S., Yang, K., Lin, X.: VerifyNet: Secure and verifiable federated learning. IEEE Trans. Inf. Forensics Secur. **15**, 911–926 (2020)
27. Xu, S., Zheng, Y., Hua, Z.: Camel: Communication-efficient and maliciously secure federated learning in the shuffle model of differential privacy. In: ACM CCS. pp. 243–257 (2024)
28. Yang, M., Wang, S., Wang, C., Liu, H., Deng, Y., Zhao, Y., Feng, J.: VCSA: Verifiable and collusion-resistant secure aggregation for federated learning using symmetric homomorphic encryption. J. Syst. Archit. **156**, 103279 (2024)

29. Yang, X., Ma, M., Tang, X.: An efficient privacy-preserving and verifiable scheme for federated learning. Future Gener. Comput. Syst. **160**, 238–250 (2024)
30. Zhang, C., Li, S., Xia, J., Wang, W., Yan, F., Liu, Y.: BatchCrypt: Efficient homomorphic encryption for Cross-Silo federated learning. In: USENIX ATC. pp. 493–506 (2020)
31. Zhou, P., Xie, Y., Peng, C., Kang, Y., He, D., Mu, T.: RLVP-FL: Robust and lightweight verifiable privacy-preserving federated learning scheme. Comput. Netw. **273**, 111747 (2025)

A Comparison with State-of-the-Art HE-based Schemes

Table 5 compares the dominant online operations across representative HE-based schemes. In this comparison, SealAgg confines lattice operations to $O(1)$ and uses finite-field arithmetic for the $O(d)$ workload, rather than placing the $O(d)$ term on exponentiation or pairing operations.

Table 5. Complexity of HE-based verifiable secure aggregation schemes.

Scheme	Mechanisms	Client Online Cost	Server total Cost
VCSA [28]	SHE VMSS	$\boldsymbol{\mathcal{O}(nt)T_{mG}} + \mathcal{O}(n)T_{bp} + \mathcal{O}(n)T_{eZ_{N^2}}$	$\boldsymbol{\mathcal{O}(n^2)T_{mG}} + \mathcal{O}(n^2)T_{mZ_q^*}$
Wang [24]	MKHE VHH	$\boldsymbol{\mathcal{O}(dn)T_{eZ_{N^2}}} + \mathcal{O}(dn)T_{mZ_{N^2}}$	$\boldsymbol{\mathcal{O}(dn^2)T_{mZ_{N^2}}} + \mathcal{O}(d)T_{eZ_{N^2}}$
VPFLI [21]	FTSA-Paillier HE LHH + Sig	$\boldsymbol{\mathcal{O}(d)T_{eZ_{N^2}}} + \mathcal{O}(d)T_{mZ_{N^2}} + \mathcal{O}(d)T_{mG} + \mathcal{O}(n)T_{aG}$	$\boldsymbol{\mathcal{O}(nd)T_{mZ_{N^2}}} + \mathcal{O}(d)T_{eZ_{N^2}}$
VPAFL [17]	Paillier + RDHED LHH + Sig	$\boldsymbol{\mathcal{O}(d)T_{eZ_{N^2}}} + \mathcal{O}(d)T_{mZ_{N^2}} + \mathcal{O}(n)T_{mG}$	$\boldsymbol{\mathcal{O}(nd)T_{mZ_{N^2}}}$
Yang [29]	LWE-HE LHH + Sig	$\boldsymbol{\mathcal{O}(d^*)T_{mG}} + \mathcal{O}(d^*)T^{\mathsf{M}}_{\mathsf{Lat.Add}} + \mathcal{O}(1)T^{\mathsf{M}}_{\mathsf{Lat.Mul}}$	$\boldsymbol{\mathcal{O}(n)T^{\mathsf{M}}_{\mathsf{Lat.Add}}} + \mathcal{O}(n)T_{aG} + \mathcal{O}(1)T_{mG} + \mathcal{O}(1)T_{mZ_q^*}$
Song [23]	H-PRE LHH + Sig	$\boldsymbol{\mathcal{O}(d)T_{eG_T}} + \mathcal{O}(d)T_{mG}$	$\boldsymbol{\mathcal{O}(nd)T_{mG_T}} + \mathcal{O}(n)T_{bp}$
SealAgg	AHE+KAHE Hidden Proj.	$\boldsymbol{\mathcal{O}(1)T^{\mathsf{R}}_{\mathsf{Lat.Mul}}} + \mathcal{O}(1)T^{\mathsf{R}}_{\mathsf{Lat.Add}} + \mathcal{O}(d)T_{m\mathbb{F}_p} + \mathcal{O}(d)T_{a\mathbb{F}_p}$	$\boldsymbol{\mathcal{O}(1)T^{\mathsf{R}}_{\mathsf{Lat.Mul}}} + \mathcal{O}(n)T^{\mathsf{R}}_{\mathsf{Lat.Add}} + \mathcal{O}(d)T_{m\mathbb{F}_p} + \mathcal{O}(d+n)T_{a\mathbb{F}_p}$

Note. d is the model dimension, n is the number of participating clients per round, t is the number of local training steps, and d^* is the reduced verification dimension. T_{bp} denotes bilinear pairing; $T_{eZ_{N^2}}$ and $T_{mZ_{N^2}}$ denote exponentiation and multiplication in $\mathbb{Z}^*_{N^2}$; $T_{mZ_q^*}$ denotes multiplication in $\mathbb{Z}_q^*$; T_{mG} and T_{aG} denote elliptic-curve scalar multiplication and point addition; T_{eG_T} and T_{mG_T} denote operations in $\mathbb{G}_T$; $T^{\mathsf{R}}_{\mathsf{Lat.Mul}}$ and $T^{\mathsf{R}}_{\mathsf{Lat.Add}}$ denote ring-form lattice multiplication and addition; $T^{\mathsf{M}}_{\mathsf{Lat.Mul}}$ and $T^{\mathsf{M}}_{\mathsf{Lat.Add}}$ denote matrix/vector-form lattice multiplication and addition; $T_{m\mathbb{F}_p}$ and $T_{a\mathbb{F}_p}$ denote multiplication and addition in $\mathbb{F}_p$. Mechanism abbreviations: SHE = symmetric homomorphic encryption; VMSS = Verifiable Multi-Secret Sharing; LWE-HE = learning-with-errors-based homomorphic encryption; MKHE = multi-key homomorphic encryption; VHH = Vector Homomorphic Hashing; H-PRE = homomorphic proxy re-encryption; FTSA = Fault-Tolerant Secure Aggregation; RDHED = Reversible Data Hiding in Encrypted Domains.

B Formal Security Proofs

B.1 Proof of Theorem 2 (Input Privacy)

Proof. We prove the claim via a sequence of hybrid games. Let H_0 denote the real protocol leakage $\mathsf{Leak}^{(r)}_{\max}$.

In H_1, we replace all honest AHE partial decryptions in the leakage with simulated ones generated via $\mathsf{Sim}^{\mathsf{dec}}_{\mathsf{AHE}}(1^\lambda, C^{(r)}_{\mathsf{A}}, K^{(r)}_{\mathrm{agg}})$. The corrupted server's own

shares remain unchanged. Due to the decryption simulatability of the AHE scheme, valid honest partial decryptions reveal nothing beyond the recovered plaintext $K_{\mathrm{agg}}^{(r)}$. Therefore, $H_0 \approx_c H_1$.

In H_2, we replace the control-path ciphertext family $\{c_{\mathsf{A},i}^{(r)}\}_{i\in\mathcal{C}^{(r)}}$ with encryptions of dummy keys $\{\widehat{k}_i\}_{i\in\mathcal{C}^{(r)}}$ that satisfy $\sum_{i\in\mathcal{C}^{(r)}} \widehat{k}_i = K_{\mathrm{agg}}^{(r)}$. We then consistently recompute the aggregate control-path ciphertext, the control-path binding components of $T^{(r)}$, and the simulated honest partial decryptions using this dummy family and the preserved aggregate key $K_{\mathrm{agg}}^{(r)}$. Since the distinguisher now only observes simulated honest decryption shares, this transition reduces to the IND-CPA security of the AHE scheme. Thus, $H_1 \approx_c H_2$.

In H_3, we replace the KAHE-side ciphertext family $\{c_{\mathsf{K},i}^{(r)}\}_{i\in\mathcal{C}^{(r)}}$ with a simulated family produced by the simulator from [3], using the public KAHE parameters and the audited aggregate payload $\widetilde{\mathbf{G}}^{(r)}$. All payload-derived ciphertext-side objects are then deterministically recomputed from this simulated family. The plaintext-side outputs, $\widetilde{\mathbf{G}}^{(r)}$ and $\mathbf{w}^{(r+1)}$, remain unchanged because the recovery and release procedures are entirely deterministic given $\widetilde{\mathbf{G}}^{(r)}$ and the fixed data in $\mathsf{Fix}^{(r)}$. By the leakage-resilient semantic security of KAHE, we have $H_2 \approx_c H_3$.

Finally, let H_4 represent the output of our simulator Sim. Given the ideal leakage $\mathsf{IdLeak}^{(r)}$, Sim constructs its view as follows. First, it restores all protected-input-independent components using $\mathsf{Fix}^{(r)}$. Next, it samples dummy keys summing to $K_{\mathrm{agg}}^{(r)}$ to form the control-path ciphertexts and aggregates, and generates the simulated AHE partial decryptions via $\mathsf{Sim}_{\mathsf{AHE}}^{\mathsf{dec}}(1^\lambda, \widehat{C}_{\mathsf{A}}^{(r)}, K_{\mathrm{agg}}^{(r)})$. It then recomputes the payload-derived ciphertext components using the KAHE simulator of [3] on input $\widetilde{\mathbf{G}}^{(r)}$, and appends the plaintext outputs $\widetilde{\mathbf{G}}^{(r)}$ and $\mathbf{w}^{(r+1)}$. By construction, the distribution of H_4 is identical to H_3 ($H_3 \equiv H_4$). Therefore, $H_0 \approx_c H_1 \approx_c H_2 \approx_c H_3 \equiv H_4$, completing the proof.

B.2 Proof of Theorem 3 (Aggregate Integrity)

Proof. If the corrupted server is an auditor, then the leader is honest and fixes the unique correct claim and payload aggregate, so every honest auditor either outputs $\mathbf{w}_{\mathsf{ref}}^{(r+1)}$ or aborts. It therefore suffices to consider a corrupted leader.

Let G_0 be the real experiment. In G_1, abort on any collision in the hash-based digest binding that fixes the accepted control claim or the released object. Then $|\Pr[G_0 = 1] - \Pr[G_1 = 1]| \le \mathsf{Adv}_{\mathsf{H},\mathcal{B}_{\mathsf{H}}}^{\mathsf{cr}}(\lambda)$. Conditioned on G_1, the accepted control claim is unique; by robustness of $(2,3)$-VSS and threshold-AHE, all honest auditors reconstruct the same $\mathsf{sd}^{(r)}$ and $K_{\mathrm{agg}}^{(r)}$, or abort. Thus every non-aborting execution has a well-defined released audited plaintext $\widetilde{\mathbf{G}}_{\mathsf{rel}}^{(r)}$.

In G_2, replace the hidden heartbeat values by independent uniform field elements both in the honest audited payloads and in the auditors' recomputed heartbeat sum. Since the leader must fix the accepted claim, the released ciphertext, and $\mathcal{C}^{(r)}$ before seed opening, $\mathcal{B}_{\mathsf{PRF}}$ can embed its challenge at exactly those positions; hence $|\Pr[G_1 = 1] - \Pr[G_2 = 1]| \le \mathsf{Adv}_{\mathsf{PRF},\mathcal{B}_{\mathsf{PRF}}}^{\mathsf{set}}(\lambda)$. Conditioned on the pre-opening transcript, if the released heartbeat entry is incorrect, then the

adversary-fixed value must equal the auditors' recomputed value $b_{\mathsf{exp}}^{(r)}$, which is uniform in $\mathbb{F}_p$; this occurs with probability at most $1/p$.

Now assume the heartbeat check passes but $\Delta_G := \mathbf{G}_{\mathsf{rel}}^{(r)} - \mathbf{G}_{\mathsf{ref}}^{(r)} \neq 0$. In G_3, replace $(\rho_1^{(r)}, \rho_2^{(r)})$ by an independent uniform pair $(\widehat{\rho}_1, \widehat{\rho}_2)$. Then $|\Pr[G_2 = 1] - \Pr[G_3 = 1]| \leq \mathsf{Adv}^{\mathsf{2prg}}_{\mathsf{Expand},\mathcal{B}_{\mathsf{Expand}}}(\lambda)$. Because the leader already fixed the released ciphertext before seed opening, the induced Δ_G is fixed independently of $(\widehat{\rho}_1, \widehat{\rho}_2)$. Writing $r_\Delta := \mathrm{rank}(\Delta_G) \geq 1$, acceptance additionally requires $\widehat{\rho}_1 \Delta_G = 0$ and $\Delta_G \widehat{\rho}_2 = 0$. Each holds with probability p^{-r_Δ}, since the left and right kernels of a rank-r_Δ matrix have codimension r_Δ; independence of $\widehat{\rho}_1$ and $\widehat{\rho}_2$ gives joint probability $p^{-2r_\Delta} \leq p^{-2}$. Summing the bad events over the three hops yields the claim. If $p \geq 2^\lambda$ and all three assumption advantages are negligible, then $p^{-1} + p^{-2} \leq 2^{-\lambda} + 2^{-2\lambda}$, so the wrong-release probability is negligible.

C Detection of Malicious Server Behavior

Table 6 evaluates SealAgg under malicious server behavior. Cryptographic and structural checks directly abort control- and payload-path manipulations.

Table 6. Malicious server behaviors and first detection points.

Attack Family	Object	Representative Behavior	First Detection Point
I. Key-Recovery-Path Inconsistencies			
Key-aggregate manipulation	$c_{\mathsf{A},i}, C_{\mathsf{A}}$	Omit, duplicate, substitute, or replay key-side ciphertexts.	`AHE_DIGEST_MISMATCH`
Split-view round context	$\mathcal{C}^{(r)}, \mathsf{ctx}^{(r)}$	Deliver inconsistent accepted-set or context data to S_1 and S_2.	`CTX_MISMATCH`
Participation shrinking	$\mathcal{C}^{(r)}$	Reduce one auditor's accepted set so that $n_r < \tau$.	`PARTICIPATION_THRESHOLD_FAIL`
II. Encrypted-Update Aggregate Inconsistencies			
Encrypted-update aggregation	$c_{\mathsf{K},i}, C_{\mathsf{K}}$	Omit, duplicate, substitute, or replay encrypted updates.	`RESERVED_SLOT_FAIL`
Reserved heartbeat-slot tampering	C_{K}	Perturb the reserved heartbeat-slot coefficient after aggregation.	`RESERVED_SLOT_FAIL`
Encrypted-update perturbation	C_{K}	Perturb an encrypted-update coefficient after aggregation.	`CHECK_ROW_FAIL`
Malformed encrypted-update aggregate	C_{K}	Truncate the aggregated KAHE ciphertext or chunk stream.	`DECRYPT_PARSE_FAIL`
Key–update misbinding	$C_{\mathsf{A}}, C_{\mathsf{K}}$	Combine inconsistent key-side and encrypted-update aggregates.	`RESERVED_SLOT_FAIL`
III. Recovery-Share Faults and Tolerance			
Share-context tampering	$(pd_j, \mathsf{ctx}^{(r)})$	Attach an incorrect round context to one partial share.	`CTX_MISMATCH`
Partial-share corruption	pd_j	Alter one partial decryption share.	`RESERVED_SLOT_FAIL`
Partial-share dropping	pd_j	Drop one partial decryption share under $(2, 3)$ recovery.	`TOLERATED_BY_THRESHOLD`
IV. Release-Path Inconsistencies			
Release-path tampering	Hash exchange	Tamper with the final cross-auditor hash check.	`AUDITOR_HASH_MISMATCH`