CERTIRANK: ALIGNING FEDERATED RANK LEARNING WITH SECURE AND CERTIFIED AGGREGATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Privacy-preserving Byzantine-robust Federated Learning (PBFL) aims to protect client updates while mitigating poisoning attacks. While homomorphic encryption (HE) has become a mainstream approach for PBFL, the combination of real-valued updates, robust aggregation, and encrypted computation often incurs substantial overhead. To address this challenge, we propose CertiRank, a rank-based PBFL that aligns federated rank learning with secure and certified aggregation. By leveraging integer-valued ranking updates and lightweight additive aggregation, CertiRank eliminates costly real-to-integer conversion and avoids complex ciphertext-domain similarity computations. To strengthen robustness against malicious clients, we introduce Certified Membership-gated Ranking Aggregation (CMGRA), which decouples subnetwork selection from fine-grained rank ordering and certifies membership changes using honest-support evidence. To securely realize CMGRA, we develop a customized secure aggregation mechanism that combines SEncode, RLWE-AHE, and privacy-preserving validity verification, enabling efficient additive aggregation and certification directly over encrypted ranking updates. Experiments demonstrate that CertiRank achieves strong robustness and competitive model utility while substantially reducing computation and communication overhead compared with existing HE-based PBFL schemes. Our code is available at https://anonymous.4open.science/r/CertiRank.

1 INTRODUCTION

Federated Learning (FL) enables multiple clients to collaboratively train a model without sharing their raw data. However, FL is not inherently secure. An honest-but-curious server may infer sensitive information from clients' model updates through inference attacks (Fan et al., 2025b; Zhang et al., 2021), while malicious clients can submit manipulated updates to degrade model utility or implant backdoors (Fang et al., 2020; Shejwalkar & Houmansadr, 2021; Fan et al., 2025a; Wang et al., 2020). To simultaneously address privacy leakage and model poisoning, Privacy-preserving Byzantine-robust Federated Learning (PBFL) has attracted significant attention. Homomorphic Encryption (HE) has emerged as a mainstream approach for PBFL because it enables aggregation-related computations to be performed directly over encrypted updates (Liu et al., 2021; Chen et al., 2024a; Ma et al., 2022; Miao et al., 2022; Lu et al., 2024; Jiang et al., 2025).

However, existing HE-based PBFL schemes often suffer from substantial efficiency concerns. First, they typically rely on Pearson correlation (Liu et al., 2021; Chen et al., 2024a), cosine similarity (Ma et al., 2022; Miao et al., 2022; Lu et al., 2024), or Euclidean distance (Jiang et al., 2025), requiring costly ciphertext-domain operations such as encrypted inner products, norm evaluations, comparisons, and divisions (Liu et al., 2021; Chen et al., 2024a; Ma et al., 2022; Miao et al., 2022; Lu et al., 2024; Jiang et al., 2025). Second, the update representation used in FL is not naturally compatible with the computation model of HE. Conventional FL exchanges real-valued model updates, whereas efficient HE schemes operate over integer or polynomial rings. Bridging this gap requires additional encoding (Liu et al., 2024; Chen et al., 2021) or quantization (Zhang et al., 2020; Dong et al., 2020), introducing a trade-off between numerical precision and cryptographic efficiency. A more detailed discussion of related work is provided in Appendix A. These challenges reflect a deeper mismatch between the learning and cryptographic layers. Existing PBFL systems often treat cryptography as a protection mechanism attached to a fixed FL workflow, forcing the cryptographic layer to ac-

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

commodate update representations and aggregation operations that were not designed with secure computation in mind.

To address this issue, we aim to align the learning representation with secure aggregation, allowing the update representation, aggregation procedure, and cryptographic functionality to form a natural match. More specifically, the learning layer should expose representations that are inherently aggregation-friendly and compatible with efficient encrypted computation, while the aggregation mechanism should preserve this simplicity and simultaneously provide the robustness required in adversarial settings. We then find that Federated Rank Learning (FRL) (Mozaffari et al., 2023) naturally achieves such an alignment between the learning layer and secure aggregation. Instead of transmitting real-valued model updates, FRL requires clients to upload integer-valued rankings that capture neural-network edge importance, while the server only performs element-wise summation to obtain a global ranking. This representation not only eliminates costly real-to-integer conversion, but also removes the need for complex ciphertext-domain similarity computations, allowing aggregation to be efficiently realized using additive homomorphic encryption. Furthermore, by restricting updates to a discrete permutation space, FRL limits arbitrary manipulation of update magnitudes and directions, thereby providing a degree of inherent robustness against poisoning attacks.

While FRL constrains the format of client updates, it does not constrain the ordering within a valid permutation. Malicious clients can still submit carefully crafted yet valid rankings to manipulate the selected subnetwork. VEM (Gong et al., 2025) demonstrates that such ranking manipulations can substantially degrade model utility. Therefore, FRL requires a lightweight certification mechanism that constrains malicious subnetwork changes while preserving additive aggregation. Motivated by these observations, we propose **Certified Membership-gated Ranking Aggregation (CMGRA)**. CMGRA strengthens Byzantine robustness while preserving the additive aggregation structure of FRL. The key idea is to decouple *which edges are selected* from *how they are ranked*. Each client submits both a complete ranking and a binary membership vector identifying its K locally selected edges. Membership counts and ranking scores are obtained through element-wise summation, retaining natural compatibility with additive homomorphic encryption. Instead of directly updating the selected subnetwork based on ranking scores, CMGRA certifies subnetwork modifications using membership evidence: a candidate edge is allowed to enter the selected set only when its support exceeds the malicious-client bound. Meanwhile, Borda scores preserve the fine-grained ordering information required by rank-based learning. As a result, CMGRA achieves stronger robustness without introducing expensive ciphertext-domain similarity computations. Furthermore, we establish bounds on the honest membership-utility gap and cumulative subnetwork changes, and prove stabilization under finite membership variation.

Building upon CMGRA, we further propose **CertiRank**, a PBFL framework that enables CMGRA over encrypted client updates through a customized cryptographic design. Rather than applying a generic HE layer to CMGRA, CertiRank co-designs the encoding, homomorphic operations, and protocol workflow. First, we introduce SEncode, a sign-aware encoding scheme that jointly encodes ranking values and membership information while supporting efficient verification and aggregation. Second, we instantiate RLWE-AHE from the BFV construction (Fan & Vercauteren, 2012) and tailor it to the minimal set of operations required by CertiRank, namely additive aggregation and prescribed monomial shifts. Third, we design a two-server protocol that combines position-obfuscated validity verification with secure aggregation, allowing clients to recover the statistics required by CMGRA. Together, these designs realize CMGRA over encrypted updates using a deliberately lightweight homomorphic computation path. In particular, RLWE-AHE implements the required coefficient shifts through monomial multiplication by x^v, avoiding costly Galois slot rotations and their associated key material. Moreover, eliminating ciphertext–ciphertext multiplication avoids multiplication-induced noise growth and can permit smaller ciphertext moduli under correctness and security constraints, thereby reducing both computation and communication overhead.

Experiments demonstrate efficiency gains at both the cryptographic and system levels. At $N = 16384$, RLWE-AHE requires approximately 17–30% of the encryption time of the tested SIMD/slot-based HE baselines, with ciphertexts approximately one-eighth their size. These gains reflect the combined encoding, operation, implementation, and parameter choices relative to the baselines' default configurations. Compared with BCPBFL (Miao et al., 2022), reported client-side and server-side processing times decrease by up to 86.6% and 97.0%, respectively. Relative to RVPFL (Lu et al., 2024), client upload and inter-server traffic decrease by up to 96.9% and 98.9%, respectively. Under the evaluated poisoning attacks, CMGRA improves accuracy over FRL by up to 70.9%.

2 CERTIFIED MEMBERSHIP-GATED RANKING AGGREGATION

Federated Rank Learning (FRL) (Mozaffari et al., 2023) builds on the Lottery Ticket Hypothesis and the supermask paradigm. The former suggests that dense networks contain sparse subnetworks that can be trained to achieve comparable accuracy (Frankle & Carbin, 2019). The latter searches for binary masks, termed supermasks, that select useful subnetworks while keeping randomly initialized weights fixed. To learn these masks, Ramanujan et al. (2020) introduced Edge-Popup (EP), which learns edge-importance scores and selects the highest-scoring edges. FRL extends this approach to the FL setting, where clients locally train edge scores and share only their rankings instead of real-valued updates. The server then performs Borda aggregation by summing the client-assigned ranks for each edge and sorting the resulting scores to obtain the global ranking for subsequent local training. For a layer with d edges indexed by $I = \{0, \dots, d-1\}$, let K be the number of selected edges, with $1 \le K < d$. In global round t, client u submits an edge-aligned rank vector $\boldsymbol{r}_u^{(t)}$: its entry $r_{u,i}^{(t)}$ is the rank assigned to edge $i \in I$. The entries form a permutation of I, with larger ranks indicating greater importance, the top-K entries specify the client's subnetwork. Appendix B summarizes the notation.

CMGRA. FRL uses Borda scores for both selection and ordering, while CMGRA decouples these two roles through membership certification. In round t, let $\mathcal{Q}^{(t)}$ denote the set of clients with well-formed submissions, with at most B Byzantine clients among them. For each client $u \in Q^{(t)}$ and edge i, define the top-K membership bit $\mu_{u,i}^{(t)} = \mathbf{1}[r_{u,i}^{(t)} \ge d-K]$, where $\mathbf{1}[\cdot]$ is the indicator function. The membership count $c_i^{(t)}$ and Borda sum $s_i^{(t)}$ are

$$c_i^{(t)} = \sum_{u \in \mathcal{Q}^{(t)}} \mu_{u,i}^{(t)}, \qquad s_i^{(t)} = \sum_{u \in \mathcal{Q}^{(t)}} r_{u,i}^{(t)}. \tag{1}$$

Membership counts govern selection, while Borda scores order edges within the selected and unselected groups. Let $A_t \subseteq I$ contain the K edges with the largest membership counts, breaking ties by fixed edge IDs, and let $S^{(t)}$ denote the final selected set. CMGRA sets $S^{(t)} = A_t$ if

$$\min_{i \in A_t} c_i^{(t)} - \max_{j \notin A_t} c_j^{(t)} > B.$$

Otherwise, it starts from the preceding selected set $S^{(t-1)}$ and performs certified exchanges between incumbents in this set and challengers outside it. Both groups are ordered by membership count, then Borda score and edge ID. The strongest challenger is paired with the weakest incumbent, followed by subsequent pairs in order. Challenger j replaces incumbent i only if $c_j^{(t)} - c_i^{(t)} > B$. These disjoint exchanges preserve $|S^{(t)}| = K$. If none is certified, the previous set is retained. Finally, CMGRA orders each group by Borda score and edge ID, assigning ranks $0, \dots, d-K-1$ to unselected edges and $d-K, \dots, d-1$ to selected edges. The resulting global ranking $\boldsymbol{r}_g^{(t)}$ guides score reordering for the next round of local training. Every certified membership change reflects an honest preference because Byzantine clients contribute at most B to any count difference. Appendix C provides the theoretical analysis of CMGRA's robustness and subnetwork dynamics.

3 PROPOSED CUSTOM ENCODING AND RLWE-BASED ENCRYPTION SCHEME

We combine sign-aware coefficient encoding with RLWE-based additively homomorphic encryption (RLWE-AHE) to support integer addition and prescribed cyclic shifts. Let $\mathcal{R}_z = \mathbb{Z}_z[x]/(x^N+1)$ and $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N+1)$ denote the plaintext and ciphertext rings, respectively. Here z and q are the plaintext and ciphertext moduli, and the polynomial degree N is a power of two.

3.1 SIGN-AWARE ENCODING

Coefficient encoding places each entry of $\boldsymbol{a} \in \mathbb{Z}_z^N$ in a polynomial coefficient, packing N integers without a slot-encoding transform. To support cyclic shifts of packed vectors, we consider multiplication by the monomial x^v, which shifts coefficient positions by v. However, reducing the result

modulo $x^N + 1$ negates coefficients that wrap around. We therefore introduce sign-aware encoding to compensate for these sign changes. For a prescribed offset $v \in \mathbb{Z}_N$, define

$$\mathsf{SEncode}(\boldsymbol{a}, v) = \sum_{i=0}^{N-v-1} a_i x^i - \sum_{i=N-v}^{N-1} a_i x^i. \tag{2}$$

The last v coefficients are pre-negated in $\mathbb{Z}_z$. Since $x^N = -1$, multiplication by x^v cancels these signs, yielding the cyclic right shift $\mathrm{rot}_v(\boldsymbol{a}) = [a_{N-v}, \ldots, a_{N-1}, a_0, \ldots, a_{N-v-1}]$.

Two decoding methods recover the original or shifted vector. The coefficient decoder $\mathsf{CDecode}$ directly reads the polynomial coefficients as a vector. The sign-aware decoder $\mathsf{SDecode}(\cdot, v)$ first applies $\mathsf{CDecode}$ and then negates the last v entries in $\mathbb{Z}_z$, canceling the sign changes introduced during encoding and recovering the original vector. When applied after the matching multiplication by x^v, $\mathsf{CDecode}$ directly returns $\mathrm{rot}_v(\boldsymbol{a})$.

For a common offset v, the encoding also preserves addition:

$$\mathsf{SEncode}(\boldsymbol{a}, v) + \mathsf{SEncode}(\boldsymbol{b}, v) = \mathsf{SEncode}(\boldsymbol{a} + \boldsymbol{b}, v).$$

Thus polynomial addition represents element-wise vector addition, while a matching monomial multiplication represents a cyclic shift. All three encoding and decoding routines require only coefficient reads and sign changes, giving $O(N)$ complexity without slot transforms. Appendix D provides more detailed definitions and descriptions of these routines.

For example, with $N = 4$ and $v = 1$, the encoding and its shifted form are

$$\mathsf{SEncode}(\boldsymbol{a}, 1) = a_0 + a_1 x + a_2 x^2 - a_3 x^3,$$
$$x\,\mathsf{SEncode}(\boldsymbol{a}, 1) = a_3 + a_0 x + a_1 x^2 + a_2 x^3 \pmod{x^4 + 1}.$$

Applying $\mathsf{SDecode}(\cdot, 1)$ to the first polynomial recovers the original vector $\boldsymbol{a} = [a_0, a_1, a_2, a_3]$, whereas applying $\mathsf{CDecode}$ to the second polynomial returns the cyclically shifted vector $[a_3, a_0, a_1, a_2]$.

3.2 RLWE-based Additively Homomorphic Encryption Scheme

We implement RLWE-AHE using the BFV public-key encryption construction (Fan & Vercauteren, 2012), specialized to coefficient encoding, addition and prescribed monomial shifts. Key generation samples a ternary secret polynomial s, a uniformly random polynomial $a \in \mathcal{R}_q$, and a small error polynomial e, then sets $b = -as + e$. The public key is $pk = (b, a)$, and the secret key is $sk = s$.

To encrypt $m \in \mathcal{R}_z$ under $pk = (b, a)$, sample a fresh ternary polynomial r and small errors e_1, e_2. With $\Delta = \lfloor q/z \rfloor$, the resulting ciphertext consists of two polynomials:

$$ct = \mathbf{Enc}(pk, m) = (c_1, c_2) = (rb + \Delta m + e_1,\ ra + e_2) \in \mathcal{R}_q^2. \tag{3}$$

Decryption recovers m by coefficient-wise rounding of $z(c_1 + c_2 s \bmod q)/q$ modulo z. Decryption and the homomorphic identities below require the noise and scaling conditions in Appendix E. All ciphertext arithmetic takes place in $\mathcal{R}_q$.

For ciphertexts $ct = (c_1, c_2)$ and $ct^* = (c_1^*, c_2^*)$ under the same key and parameters, *ciphertext–ciphertext addition* combines corresponding components:

$$\mathbf{Add}(ct, ct^*) = ct \oplus ct^* = (c_1 + c_1^*, c_2 + c_2^*).$$

If the inputs encrypt m and m^*, the result encrypts $m + m^* \in \mathcal{R}_z$ without decrypting either input. When m^* is available in plaintext, *ciphertext–plaintext addition* instead uses

$$\mathbf{PAdd}(ct, m^*) = (c_1 + \Delta m^*, c_2).$$

It produces the same plaintext sum by adding Δm^* to the first component and leaving the second unchanged, avoiding a separate encryption of m^*.

We implement the prescribed shift by multiplying both ciphertext components by x^v: $\mathbf{Rot}(ct, v) = (c_1 x^v, c_2 x^v)$. If ct encrypts $\mathsf{SEncode}(\boldsymbol{a}, v)$, decrypting the result and applying $\mathsf{CDecode}$ yields $\mathrm{rot}_v(\boldsymbol{a})$. Compared with conventional SIMD slot rotations in general-purpose HE schemes, this prescribed coefficient-domain shift requires neither Galois keys nor key switching. Addition-only RLWE-AHE also avoids multiplication-induced noise growth and relinearization, potentially allowing a smaller ciphertext modulus under correctness and security constraints. The complete algorithms and the RLWE-based IND-CPA analysis are provided in Appendices D and F.1, respectively.

4 THE DESIGN OF CERTIRANK

This section presents CertiRank by integrating CMGRA with SEncode and RLWE-AHE. For clarity, we describe CertiRank for a single layer consisting of d edges, of which K are selected. For multilayer models, the same procedure is applied independently to each layer with its own d and K.

4.1 SYSTEM MODEL

CertiRank consists of two auxiliary servers, $\mathcal{S}_1$ and $\mathcal{S}_2$, and a fixed group of clients (Figure 1). Each client $\mathcal{C}_u$ has an identifier u, and $\mathcal{C}$ denotes the set of all client identifiers. In each global round t, at most U clients are selected to perform local training and submit encrypted rankings. Their identifiers form the participating set $\mathcal{P}^{(t)} \subseteq \mathcal{C}$, with $|\mathcal{P}^{(t)}| \leq U$. The two servers verify client submissions and securely aggregate the accepted rankings, while all clients use the aggregation results to update and maintain the global model. Both servers are semi-honest and non-colluding: they follow the protocol but may infer private information from their respective views. Malicious clients may arbitrarily manipulate submissions, collude with each other, and collude with at most one server. At most B participants are malicious in each round.

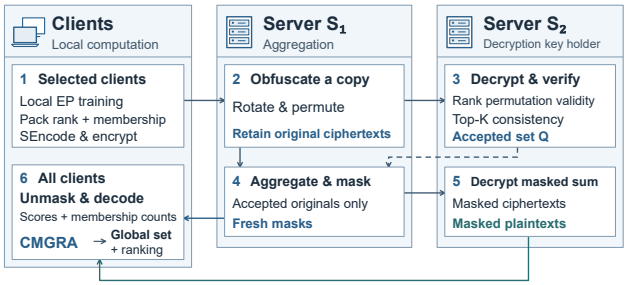


Figure 1: CertiRank system model.

4.2 OVERVIEW

CMGRA requires the aggregate Borda-score vector $\boldsymbol{s}^{(t)}$ and membership-count vector $\boldsymbol{c}^{(t)}$. Every accepted ranking must be a valid permutation, with a membership vector consistent with its top-K entries. CertiRank must therefore support private client submission, verification of ranking validity and membership consistency, and secure computation of both statistics. We meet these requirements through encrypted submission, position-obfuscated verification and masked aggregate decryption.

Each client jointly packs and encrypts its ranking and membership information. $\mathcal{S}_1$ retains the original ciphertexts for edge-aligned aggregation and sends position-obfuscated copies to $\mathcal{S}_2$ for decryption and verification. After $\mathcal{S}_2$ identifies valid clients, $\mathcal{S}_1$ aggregates their original ciphertexts and masks the sum. $\mathcal{S}_2$ decrypts the masked aggregate and forwards the result to all clients, while $\mathcal{S}_1$ separately distributes the masks. Clients recover both statistics and execute CMGRA in plaintext to obtain the selected edge set and global ranking for the next round.

4.3 CERTIRANK PROTOCOL

We next describe the main stages of CertiRank: system initialization, client-side computation and submission, server-side verification and secure aggregation, and aggregate recovery with CMGRA. The complete protocol is given in Algorithm 1 in Appendix G. Appendices H and F.3 provide the detailed workflow and security analysis, respectively.

4.3.1 SYSTEM INITIALIZATION

Choose the packing base $\lambda = 2^{\lceil \log_2(U+1) \rceil} > U$ and plaintext modulus $z > \lambda U(d-1)+U$ to prevent carries between packed fields and aggregation wrap-around. $\mathcal{S}_2$ generates (pk, sk), distributes the public parameters and retains sk. A public seed initializes fixed network weights, edge scores and $S^{(0)}$. Each layer occupies $M = \lceil d/N \rceil$ blocks. In round t, $\mathcal{S}_1$ shares independent uniform shifts $v_m^{(t)} \in \mathbb{Z}_N$ for block indices $m = 1, \ldots, M$ with all clients. Common shifts give all submissions the same encoding layout for aggregation. At most U accepted clients contribute a membership bit at each coordinate, so their sum remains below λ and cannot carry into the ranking score. The bound on z covers the largest possible joint sum, including both fields.

4.3.2 CLIENT-SIDE COMPUTATION AND SUBMISSION

Each selected client $u \in \mathcal{P}^{(t)}$ runs local EP, starting from the public scores reordered by the preceding global ranking, and obtains $\boldsymbol{r}_u^{(t)}$. It derives $\mu_{u,i}^{(t)} = \mathbf{1}[r_{u,i}^{(t)} \geq d - K]$ and packs

$$h_{u,i}^{(t)} = \lambda r_{u,i}^{(t)} + \mu_{u,i}^{(t)}. \tag{4}$$

These entries form the packed vector $\boldsymbol{h}_u^{(t)}$. Partition it into M blocks, zero-pad the final block when necessary and record its valid coordinates. Let $\boldsymbol{h}_{u,m}^{(t)}$ denote the m-th block of length N. Each block is then encoded and encrypted as $ct(\boldsymbol{h}_{u,m}^{(t)}) = \mathbf{Enc}(pk, \mathsf{SEncode}(\boldsymbol{h}_{u,m}^{(t)}, v_m^{(t)}))$ and sent to $\mathcal{S}_1$.

The first round starts from the initial scores, subsequent rounds reorder them using the preceding global ranking. Throughout local EP, the network weights remain fixed and clients update only their edge scores.

4.3.3 SERVER-SIDE VERIFICATION AND SECURE AGGREGATION

$\mathcal{S}_1$ retains the received ciphertexts for aggregation. For verification, it applies $\mathbf{Rot}(ct(\boldsymbol{h}_{u,m}^{(t)}), v_m^{(t)})$ to each block and shuffles the resulting blocks using a fresh client-specific permutation $\pi_u^{(t)}$. It then sends the transformed ciphertexts to $\mathcal{S}_2$ for decryption and validity checks.

$\mathcal{S}_2$ decrypts the ciphertexts and applies $\mathsf{CDecode}$. It checks that the padding positions indicated by $\mathcal{S}_1$ contain zeros, then excludes them from the ranking checks. From each remaining entry, it recovers $\widetilde{\mu} = \widetilde{h} \bmod \lambda$ and $\widetilde{r} = (\widetilde{h} - \widetilde{\mu})/\lambda$. It checks that the representation is well formed, the ranks form a permutation of $\{0, \ldots, d-1\}$, and each membership bit equals $\mathbf{1}[\widetilde{r} \geq d - K]$. $\mathcal{S}_2$ returns the accepted client indices $\mathcal{Q}^{(t)}$ to $\mathcal{S}_1$. Accepted rankings may still be adversarial, so CMGRA controls their effect on membership through certificates under the malicious-client bound B. If $\mathcal{Q}^{(t)}$ is empty, clients retain the previous selected set and ranking.

For a nonempty accepted set, $\mathcal{S}_1$ aggregates the retained original ciphertexts. Let $\boldsymbol{h}_m^{(t)} = \sum_{u \in \mathcal{Q}^{(t)}} \boldsymbol{h}_{u,m}^{(t)}$. Since all clients use the same encoding offset for block m, additivity gives

$$p_m^{(t)} = \sum_{u \in \mathcal{Q}^{(t)}} \mathsf{SEncode}(\boldsymbol{h}_{u,m}^{(t)}, v_m^{(t)}) = \mathsf{SEncode}(\boldsymbol{h}_m^{(t)}, v_m^{(t)}). \tag{5}$$

The corresponding ciphertext is computed directly as

$$ct(\boldsymbol{h}_m^{(t)}) = \bigoplus_{u \in \mathcal{Q}^{(t)}} ct(\boldsymbol{h}_{u,m}^{(t)}). \tag{6}$$

Before sending the aggregate for decryption, $\mathcal{S}_1$ samples a fresh uniform mask $\rho_m^{(t)} \in \mathcal{R}_z$ and computes $\widehat{ct}_m = \mathbf{PAdd}(ct(\boldsymbol{h}_m^{(t)}), \rho_m^{(t)})$. Masks are sampled independently for every block and round. After decryption, $\mathcal{S}_2$ obtains $\widehat{p}_m^{(t)} = p_m^{(t)} + \rho_m^{(t)} \bmod z$ and forwards these masked plaintexts to the clients. Meanwhile, $\mathcal{S}_1$ sends the masks to the clients.

4.3.4 AGGREGATE RECOVERY AND CMGRA

Using the masked plaintexts from $\mathcal{S}_2$ and the masks from $\mathcal{S}_1$, clients first subtract the masks modulo z. They then apply $\mathsf{SDecode}(\cdot, v_m^{(t)})$ to each recovered polynomial, remove padding, and concatenate the decoded blocks into the aggregate vector $\boldsymbol{h}^{(t)}$ in the original edge order. Each entry satisfies $h_i^{(t)} = \lambda s_i^{(t)} + c_i^{(t)}$, from which clients recover the membership count $c_i^{(t)} = h_i^{(t)} \bmod \lambda$ and the Borda score $s_i^{(t)} = (h_i^{(t)} - c_i^{(t)})/\lambda$. Since $0 \leq c_i^{(t)} \leq |\mathcal{Q}^{(t)}| \leq U < \lambda$, the two fields remain separate, ensuring exact recovery of both statistics over the accepted clients. With these statistics, clients execute CMGRA as described in Section 2 to update the selected edge set and global ranking. Deterministic tie-breaking ensures that clients obtain the same results. Recovery information is distributed to all clients, so those not participating in the current round can perform the same update and remain synchronized for subsequent rounds.

Together, these stages combine ranking validity checks with CMGRA's certified membership updates, while keeping homomorphic computation limited to additions and prescribed shifts.

5 EXPERIMENTS

We evaluate encoding, HE operations, client and server overhead, and poisoning robustness. Appendix I.1 details the implementations, datasets and training settings, while Appendix E describes the cryptographic parameter choices and correctness conditions.

5.1 PERFORMANCE OF ENCODING AND RLWE-AHE

5.1.1 PERFORMANCE EVALUATION OF SENCODE AND CDECODE

Table 1 compares our coefficient encoding with NTT-based BFV/BGV batching and FFT-based CKKS encoding. By operating directly on coefficients, our method avoids the encoding transforms used by these baselines. The results show consistently lower encoding and decoding latency across all tested polynomial degrees. At $N = 16384$, SEncode takes $50\,\mu$s, corresponding to speedups of approximately $5\times$ over NTT and $64\times$ over FFT, while CDecode takes $62\,\mu$s. For context, SEncode and NTT each carry N values per plaintext, whereas the tested FFT encoder carries $N/2$.

Table 1: Encoding and decoding time (μs) across polynomial degrees.

	$N = 4096$		$N = 8192$		$N = 16384$	
Method	Encode	Decode	Encode	Decode	Encode	Decode
SEncode/CDecode	15	18	25	30	50	62
NTT-based	58	76	114	154	255	336
FFT-based	297	343	901	1420	3186	8707

5.1.2 PERFORMANCE EVALUATION OF RLWE-AHE

Table 2 extends the comparison to HE operations and ciphertext sizes for $N \in \{4096, 8192, 16384\}$. RLWE-AHE uses a 60-bit ciphertext modulus, while the baselines use default moduli. The measured gains therefore reflect the combined effects of encoding, supported operations, implementation and parameter choices.

Table 2: Execution time (in μs) of encryption, decryption, addition and rotation operations for different schemes across several polynomial degrees.

Degree	**Scheme**	**Encryption**	**Decryption**	**Addition**	**Rotation (Min)**	**Rotation (Avg)**	**Rotation (Max)**	**Ciphertext Size**
$N = 4096$	RLWE-AHE	836	205	16	21	27	65	64.39KB
	BGV	1873	190	28	825	3419	7770	128.61KB
	BFV	1340	323	27	662	2753	6630	128.61KB
	CKKS	1650	45	26	653	2743	6432	128.61KB
$N = 8192$	RLWE-AHE	1631	403	29	40	49	120	128.61KB
	BGV	5526	690	101	4101	18429	38827	514.11KB
	BFV	3514	1092	106	3476	15691	33173	514.11KB
	CKKS	4634	161	109	3448	15613	30627	514.11KB
$N = 16384$	RLWE-AHE	3270	848	65	83	95	261	257.11KB
	BGV	19270	3102	704	24539	116561	232694	2056.11KB
	BFV	11153	4556	718	21994	105385	235465	2056.11KB
	CKKS	15092	674	719	21815	104898	228607	2056.11KB

Overall, RLWE-AHE achieves low computational overhead and compact ciphertexts under the tested configurations. At $N = 16384$, RLWE-AHE encrypts in $3270\,\mu$s, compared with 11153–$19270\,\mu$s for the tested baselines, and its ciphertexts occupy 257.11 KB rather than 2056.11 KB. Its prescribed monomial shifts avoid Galois rotation keys and average $95\,\mu$s, whereas the baselines' slot rotations take 104898–$116561\,\mu$s on average. Although CKKS has lower standalone decryption latency, RLWE-AHE achieves lower combined decryption and decoding latency across all tested degrees. Appendix I.2 further compares monomial shifts and slot rotations across different step sizes and polynomial degrees, complementing the average latencies reported in Table 2.

5.2 PERFORMANCE OF CERTIRANK

The learning experiments use MNIST/Conv2, SVHN/Conv8 and CIFAR10/ResNet18, with 1,000 clients, Dirichlet heterogeneity $\alpha = 1.0$, $U = 25$ participants per round and 500 rounds. The server evaluation additionally uses $U \in \{5, 15, 25\}$ participants. We compare CertiRank with CertiRank-BFV (CMGRA with SIMD-BFV), BCPBFL (Miao et al., 2022), RVPFL (Lu et al., 2024), and plaintext FedAvg (McMahan et al., 2017). All RLWE-based system experiments use $N = 8192$. Appendix A discusses the baselines.

5.2.1 CLIENT-SIDE OVERHEAD ANALYSIS

Table 3 reports per-client processing time and upload traffic per global round. For all schemes, processing time includes local training, encoding, encryption, decryption, decoding, and protocol processing. FedAvg provides a plaintext reference for computation and communication.

Table 3: Per-client processing time and upload traffic per global round.

	Processing time (s)			Upload traffic (MB)		
Scheme	MNIST	SVHN	CIFAR10	MNIST	SVHN	CIFAR10
CertiRank	0.37	1.48	2.70	25.25	81.14	171.44
CertiRank-BFV	0.84	3.17	6.87	100.91	324.33	685.31
BCPBFL	2.76	8.41	16.61	200.32	647.16	1368.12
RVPFL	≈ 1.37 h	≈ 4.45 h	≈ 9.45 h	793.77	2576.09	5451.34
FedAvg	0.01	0.16	0.19	6.20	20.13	42.59

CertiRank has the lowest reported client processing time among the encrypted schemes. On CIFAR10, its 2.70 s cost corresponds to 39.3% of CertiRank-BFV's time and 16.3% of BCPBFL's, while RVPFL requires about 9.45 h. CertiRank also uploads less data than the encrypted baselines: its 171.44 MB upload is 25.0%, 12.5% and 3.1% of the respective costs of CertiRank-BFV, BCPBFL and RVPFL. Across all datasets, CertiRank's upload is about $4\times$ that of plaintext FedAvg.

5.2.2 SERVER-SIDE OVERHEAD ANALYSIS

Table 4 compares server processing time and inter-server communication across datasets and participant counts, both measured per global round. For all schemes, processing time covers verification and robust aggregation, while communication includes the total volume of intermediate data exchanged between the two servers.

Table 4: Server aggregation time and total inter-server traffic per global round.

		Aggregation time (s)			Inter-server traffic (MB)		
Dataset	Scheme	$U = 5$	$U = 15$	$U = 25$	$U = 5$	$U = 15$	$U = 25$
MNIST	CertiRank	4.89	16.04	24.42	151.50	404.03	656.50
	CertiRank-BFV	26.04	78.13	128.91	605.46	1614.56	2623.66
	BCPBFL	148.51	443.17	759.23	4006.4	12019.2	20032
	RVPFL	≈ 16.9 h	≈ 50.2 h	≈ 83.4 h	12700.32	36513.42	60326.52
SVHN	CertiRank	14.88	48.11	78.81	486.84	1298.24	2109.64
	CertiRank-BFV	77.81	259.57	441.39	1945.98	5189.28	8432.58
	BCPBFL	496.08	1504.82	2515.79	12943.2	38829.6	64716
	RVPFL	≈ 54.9 h	≈ 162.8 h	≈ 270.7 h	41217.44	118500.14	195782.84
CIFAR10	CertiRank	31.88	99.06	171.36	1028.64	2743.04	4457.44
	CertiRank-BFV	167.94	539.34	856.37	4111.86	10964.96	17818.06
	BCPBFL	990.5	3156.61	5136.21	27362.4	82087.2	136812
	RVPFL	≈ 116.2 h	≈ 344.6 h	≈ 572.9 h	87221.44	250761.64	414301.84

CertiRank achieves the lowest server processing time and inter-server traffic among the tested encrypted schemes. On CIFAR10 at $U = 25$, its processing time is 20.0% of CertiRank-BFV's and

3.3% of BCPBFL's, while its traffic is 25.0%, 3.3% and 1.1% of CertiRank-BFV, BCPBFL and RVPFL, respectively. Costs grow approximately linearly with U: each round requires verification of U client submissions and inter-server transmission of their ciphertexts plus one masked aggregate.

5.2.3 Robustness Analysis

Table 5: Robustness comparison under three datasets

Scheme	No Attack	Label Flipping			Gradient Ascent			Pixel Backdoor			VEM		
		10%	20%	30%	10%	20%	30%	10%	20%	30%	10%	20%	30%
MNIST													
FedAvg	98.18	97.47	78.39	43.37	9.80	9.80	9.80	98.23/65.42	98.23/73.15	98.22/77.31	–	–	–
BCPBFL	97.98	97.98	97.93	97.02	97.76	96.80	94.80	97.94/36.27	98.07/42.23	97.85/49.26	–	–	–
RVPFL	98.08	98.16	98.13	98.08	98.02	96.80	94.36	98.16/31.50	98.20/48.35	98.12/53.52	–	–	–
FRL	98.42	97.38	97.71	96.54	94.40	71.61	52.04	98.38/55.31	98.45/70.85	98.59/82.48	93.85	67.77	62.83
CMGRA	98.40	98.01	97.28	96.59	97.50	96.42	95.20	98.33/13.43	98.18/22.27	97.88/24.01	97.04	96.66	95.95
SVHN													
FedAvg	93.53	90.99	81.29	30.29	6.70	6.70	6.70	93.83/50.38	93.68/57.24	93.20/61.44	–	–	–
BCPBFL	93.31	93.11	92.70	92.00	93.30	92.80	91.80	93.50/10.72	93.30/16.01	93.00/18.23	–	–	–
RVPFL	92.21	91.96	91.20	90.11	92.67	92.30	90.85	93.80/8.03	93.50/20.66	93.10/24.14	–	–	–
FRL	93.04	93.16	93.10	86.89	88.88	71.31	19.59	92.81/6.42	92.66/44.51	92.68/60.59	85.50	71.28	45.02
CMGRA	92.98	93.30	93.39	92.78	92.48	89.21	86.72	93.46/2.59	93.51/12.65	93.19/18.36	89.80	88.64	86.74
CIFAR10													
FedAvg	83.36	79.16	79.34	40.49	5.64	12.97	10.00	84.10/25.28	84.17/51.36	83.86/60.36	–	–	–
BCPBFL	80.33	79.51	77.80	68.44	78.27	77.07	76.94	77.31/5.17	77.39/9.79	77.21/27.19	–	–	–
RVPFL	82.80	80.71	79.08	68.94	77.61	75.71	73.35	83.52/8.56	83.70/12.96	83.88/14.76	–	–	–
FRL	80.16	78.91	75.38	47.61	54.00	6.63	9.93	80.23/5.42	80.36/44.59	80.97/58.94	40.30	21.50	16.80
CMGRA	80.01	79.07	77.97	70.96	77.83	77.53	75.31	81.18/3.62	80.47/7.30	78.15/15.47	78.00	75.50	74.50

Note: Pixel Backdoor results are reported as ACC/ASR, with higher ACC and lower ASR indicate better performance. "–" denotes that VEM is not applicable. Results are reported after 500 global rounds.

Table 5 evaluates robustness against Label Flipping (Tolpegin et al., 2020), Gradient Ascent (Kurakin et al., 2017), Pixel Backdoor (Xie et al., 2020) and VEM (Gong et al., 2025) attacks, with detailed attack settings provided in Appendix I.3. When there is no attack, CMGRA's accuracy differs from FRL's by only 0.02, 0.06, and 0.15 percentage points on MNIST, SVHN and CIFAR10. At nominal 30% VEM, its respective gains over FRL are 33.12, 41.72, and 57.70 points. Backdoor resistance also improves, with dataset-dependent clean-accuracy trade-offs. On CIFAR10 at 30% malicious clients, CMGRA achieves 78.15% ACC/15.47% ASR versus FRL's 80.97%/58.94%. RVPFL has slightly lower ASR (14.76%). These empirical results complement our theoretical guarantees on honest membership utility and subnetwork dynamics.

6 Conclusion and Future Work

In this paper, we presented **CertiRank**, a PBFL framework that aligns FRL with secure and certified aggregation through CMGRA, sign-aware encoding, and RLWE-AHE. By replacing real-valued updates with ranking-based representations and introducing membership-certified aggregation, CertiRank bridges the gap between efficient secure computation and Byzantine-robust learning. Our theoretical analysis establishes bounds on the honest membership-utility gap and cumulative subnetwork changes, with stabilization under finite total membership variation. Experimental results demonstrate that CertiRank achieves competitive model utility and strong poisoning robustness while substantially reducing computation and communication overhead compared with existing HE-based PBFL schemes.

CertiRank currently relies on two non-colluding servers, which may limit deployment in settings where independently operated servers are unavailable. An important direction for future work is the design of a single-server variant based on zero-knowledge proofs (ZKPs), enabling verification that encrypted submissions encode valid ranking permutations and membership bits consistent with their top-K entries. However, such a design introduces significant challenges. In addition to the computational and communication costs associated with proof generation and verification, CertiRank requires verification of structured permutation constraints and rank–membership consistency, which are considerably more complex than conventional proof targets such as update norms. Developing lightweight proof systems for these properties while preserving compatibility with homomorphic aggregation remains an important challenge for practical single-server deployment.

AI USE STATEMENT

Large language models assisted with language polishing, literature retrieval and discovery, and experimental implementation, including grammar refinement and code development. The authors formulated the research problem and developed the core methodology and main technical ideas. They independently reviewed and verified all AI-assisted outputs, including language edits and experimental code, and take full responsibility for the paper's content.

ETHICS STATEMENT

This work aims to improve privacy protection and robustness in federated learning. All evaluations use publicly available benchmark datasets, without recruiting human participants or collecting new personal data. Poisoning attacks are evaluated in controlled benchmark experiments to assess defensive robustness. Clients submit encrypted ranking and membership information rather than raw data, with privacy protection subject to the stated threat model, cryptographic assumptions and verification leakage. Practical deployment should account for these limitations and the applicable data-use restrictions.

REPRODUCIBILITY STATEMENT

To support reproducibility, our implementation is available through an anonymous repository at `https://anonymous.4open.science/r/CertiRank`. All experiments use the publicly available MNIST, SVHN, and CIFAR-10 datasets. Appendix I.1 describes the experimental environment, model architectures, data partitioning, and training settings. Cryptographic parameters and attack configurations are provided in Appendices E and I.3, respectively.

REFERENCES

James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, pp. 1253–1269. ACM, 2020.

Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, et al. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pp. 1175–1191, 2017.

Chaochao Chen, Jun Zhou, Li Wang, et al. When homomorphic encryption marries secret sharing: Secure large-scale sparse logistic regression and applications in risk control. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2652–2662, 2021.

Xiangquan Chen, Chungen Xu, Bennian Dou, and Pan Zhang. PPAPAFL: A novel approach to privacy protection and anti-poisoning attacks in federated learning. In *Tools for Design, Implementation and Verification of Emerging Information Technologies*, pp. 117–138. Springer Nature Switzerland, 2024a. doi: 10.1007/978-3-031-51399-2_7.

Zekai Chen, Shengxing Yu, Farong Chen, Fuyi Wang, Ximeng Liu, and Robert H Deng. Lightweight privacy-preserving cross-cluster federated learning with heterogeneous data. *IEEE Transactions on Information Forensics and Security*, 2024b.

Ye Dong, Xiaojun Chen, Liyan Shen, and Dakui Wang. Eastfly: Efficient and secure ternary federated learning. *Computers & Security*, 94:101824, 2020.

Junfeng Fan and Frederik Vercauteren. Somewhat practical fully homomorphic encryption. *Cryptology ePrint Archive*, 2012.

Mingyuan Fan, Zhanyi Hu, Fuyi Wang, and Cen Chen. Bad-pfl: Exploiting backdoor attacks against personalized federated learning. In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net, 2025a.

Mingyuan Fan, Fuyi Wang, Cen Chen, and Jianying Zhou. Boosting gradient leakage attacks: Data reconstruction in realistic FL settings. In *34th USENIX Security Symposium, USENIX Security 2025, Seattle, WA, USA, August 13-15, 2025*, pp. 2985–3004, 2025b.

Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Gong. Local model poisoning attacks to byzantine-robust federated learning. In *29th USENIX security symposium (USENIX Security 20)*, pp. 1605–1622, 2020.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *7th International Conference on Learning Representations, ICLR, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019.

Zirui Gong, Yanjun Zhang, Leo Yu Zhang, Zhaoxi Zhang, Yong Xiang, and Shirui Pan. Not all edges are equally robust: Evaluating the robustness of ranking-based federated learning. In *2025 IEEE Symposium on Security and Privacy (SP)*, pp. 2527–2545. IEEE, 2025.

Siyang Jiang, Hao Yang, Qipeng Xie, Chuan Ma, Sen Wang, Zhe Liu, Tao Xiang, and Guoliang Xing. Towards compute-efficient byzantine-robust federated learning with fully homomorphic encryption. *Nature Machine Intelligence*, 7(10):1657–1668, 2025.

Alexey Kurakin, Ian J. Goodfellow, and Samy Bengio. Adversarial machine learning at scale. In *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings*. OpenReview.net, 2017.

Xiaoyuan Liu, Hongwei Li, Guowen Xu, Zongqi Chen, et al. Privacy-enhanced federated learning against poisoning adversaries. *IEEE Transactions on Information Forensics and Security*, 16: 4574–4588, 2021.

Ximeng Liu, Kim-Kwang Raymond Choo, Robert H. Deng, Rongxing Lu, and Jian Weng. Efficient and privacy-preserving outsourced calculation of rational numbers. *IEEE Trans. Dependable Secur. Comput.*, 15(1):27–39, 2018.

Xuanqi Liu, Zhuotao Liu, Qi Li, Ke Xu, and Mingwei Xu. Pencil: Private and extensible collaborative learning without the non-colluding assumption. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society, 2024.

Ziyao Liu, Jiale Guo, Wenzhuo Yang, Jiani Fan, Kwok-Yan Lam, and Jun Zhao. Privacy-preserving aggregation in federated learning: A survey. *IEEE Transactions on Big Data*, pp. 1–20, 2022. doi: 10.1109/TBDATA.2022.3190835.

Zhi Lu, Songfeng Lu, Xueming Tang, and Junjun Wu. Robust and verifiable privacy federated learning. *IEEE Transactions on Artificial Intelligence*, 5(4):1895–1908, 2024.

Zhuoran Ma, Jianfeng Ma, Yinbin Miao, et al. ShieldFL: Mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Transactions on Information Forensics and Security*, 17:1639–1654, 2022.

Mohamad Mansouri, Melek Önen, Wafa Ben Jaballah, and Mauro Conti. Sok: Secure aggregation based on cryptographic schemes for federated learning. *Proceedings on Privacy Enhancing Technologies*, 2023.

Brendan McMahan, Eider Moore, Daniel Ramage, et al. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pp. 1273–1282. PMLR, 2017.

Yinbin Miao, Ziteng Liu, Hongwei Li, et al. Privacy-preserving byzantine-robust federated learning via blockchain systems. *IEEE Transactions on Information Forensics and Security*, 17:2848–2861, 2022.

Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. Ppfl: Privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th annual international conference on mobile systems, applications, and services*, pp. 94–108, 2021.

Hamid Mozaffari, Virat Shejwalkar, and Amir Houmansadr. Every vote counts: Ranking-based training of federated learning to resist poisoning attacks. In *32nd USENIX Security Symposium (USENIX Security 23)*, pp. 1721–1738, 2023.

Pascal Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*, pp. 223–238. Springer, 1999.

Vivek Ramanujan, Mitchell Wortsman, Aniruddha Kembhavi, et al. What's hidden in a randomly weighted neural network? In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 11893–11902, 2020.

Phillip Rieger, Torsten Krauß, Markus Miettinen, Alexandra Dmitrienko, and Ahmad-Reza Sadeghi. Crowdguard: Federated backdoor detection in federated learning. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*. The Internet Society, 2024.

Badrul Sarwar, George Karypis, Joseph Konstan, and John Riedl. Item-based collaborative filtering recommendation algorithms. In *Proceedings of the 10th international conference on World Wide Web*, pp. 285–295, 2001.

Virat Shejwalkar and Amir Houmansadr. Manipulating the byzantine: Optimizing model poisoning attacks and defenses for federated learning. In *NDSS*, 2021.

Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *European symposium on research in computer security*, pp. 480–501. Springer, 2020.

Hongyi Wang, Kartik Sreenivasan, Shashank Rajput, et al. Attack of the tails: Yes, you really can backdoor federated learning. *Advances in Neural Information Processing Systems*, 33:16070–16084, 2020.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H. Yang, Farhad Farokhi, Shi Jin, Tony Q. S. Quek, and H. Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE Trans. Inf. Forensics Secur.*, 15:3454–3469, 2020.

Chulin Xie, Keli Huang, Pin-Yu Chen, and Bo Li. DBA: distributed backdoor attacks against federated learning. In *8th International Conference on Learning Representations, ICLR, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net, 2020.

Qipeng Xie, Siyang Jiang, Linshan Jiang, Yongzhi Huang, Zhihe Zhao, Salabat Khan, Wangchen Dai, Zhe Liu, and Kaishun Wu. Efficiency optimization techniques in privacy-preserving federated learning with homomorphic encryption: A brief survey. *IEEE Internet of Things Journal*, 11 (14):24569–24580, 2024.

Shuangqing Xu, Yifeng Zheng, and Zhongyun Hua. Camel: Communication-efficient and maliciously secure federated learning in the shuffle model of differential privacy. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pp. 243–257, 2024.

Chengliang Zhang, Suyi Li, Junzhe Xia, et al. BatchCrypt: Efficient homomorphic encryption for cross-silo federated learning. In *2020 USENIX annual technical conference (USENIX ATC 20)*, pp. 493–506, 2020.

Wanrong Zhang, Shruti Tople, and Olga Ohrimenko. Leakage of dataset properties in multi-party machine learning. In *30th USENIX security symposium (USENIX Security 21)*, pp. 2687–2704, 2021.

Yufei Zhou, Peijia Zheng, Xiaochun Cao, and Jiwu Huang. Two-tier data packing in rlwe-based homomorphic encryption for secure federated learning. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*, pp. 2844–2858, 2024.

Appendix for Submission #20878

List of Appendix

A: Related Work

B: Notation Summary

C: Theoretical Analysis of CMGRA
- C.1 Formal guarantees
- C.2 Proof of Theorem 1
- C.3 Proof of Theorem 2

D: Encoding and Encryption Details
- D.1 Sign-aware Encoding and Decoding
- D.2 RLWE-AHE algorithms

E: RLWE-AHE Parameters and Correctness Conditions
- E.1 Fresh Encryption Noise and Supported Operations
- E.2 A Conditional Bound on the Failure Probability
- E.3 Scaling, Masking, and Exact Decryption
- E.4 Plaintext Range and Interpretation of the Parameters

F: Security Analysis
- F.1 Security of RLWE-AHE
- F.2 Security of the Two-Stage Obfuscation
- F.3 Security of CertiRank

G: Complete CertiRank Algorithm

H: Detailed CertiRank Workflow
- H.1 System Initialization
- H.2 Client-side Computation and Submission
- H.3 Server-side Verification and Secure Aggregation
- H.4 Aggregate Recovery and CMGRA

I: Experimental Details and Additional Results
- I.1 Experimental Configuration and Model
- I.2 Rotation latency across step sizes
- I.3 Poisoning Attack Settings

A Related Work

Table 6: Comparison between CertiRank and existing HE-based PBFL schemes.

Schemes	Cryptographic technology	Byzantine mitigation	Core secure operations	Efficiency	Real-to-integer conversion
PEFL (Liu et al., 2021)	Paillier HE	Pearson correlation coefficient	Inner product, L_2 norm, mean	Low	Needed
PPAPAFL (Chen et al., 2024a)	CKKS HE	Pearson correlation coefficient	Inner product, L_2 norm, mean	Moderate	Needed
ShieldFL (Ma et al., 2022)	Paillier HE	Cosine similarity	Inner product, L_2 norm	Low	Needed
BCPBFL (Miao et al., 2022)	CKKS HE	Cosine similarity	Inner product, L_2 norm	Moderate	Needed
RVPFL (Lu et al., 2024)	Paillier HE	Adjusted cosine similarity	Inner product, L_2 norm, mean	Low	Needed
Lancelot (Jiang et al., 2025)	CKKS HE	Euclidean distance	Inner product, L_2 norm	Moderate	Needed
CertiRank	RLWE-based HE	Membership-certified ranking	Addition, rotation	High	Not Needed

Privacy-preserving FL commonly uses Differential Privacy (DP), Trusted Execution Environments (TEE), Secure Multi-party Computation (SMC), or Homomorphic Encryption (HE) (Liu et al., 2022; Mansouri et al., 2023). These approaches protect client information under different assumptions and incur different costs. DP adds noise to model updates, which can reduce accuracy as noise accumulates (Wei et al., 2020; Xu et al., 2024). TEE-based methods provide hardware-enforced isolation and depend on trust in the hardware and execution environment (Mo et al., 2021; Rieger et al., 2024). SMC supports joint computation over private inputs, often at the cost of communication and interaction (Bonawitz et al., 2017; Bell et al., 2020; Chen et al., 2024b). HE instead supports computation on encrypted inputs without requiring trusted hardware or adding DP noise to the recovered updates (Xie et al., 2024; Zhou et al., 2024). We focus on HE-based PBFL methods and the cost of combining encrypted aggregation with poisoning defenses.

Existing HE-based PBFL schemes incorporate robust aggregation rules into the encrypted computation. PEFL (Liu et al., 2021) encrypts client updates with Paillier (Paillier, 1999) and sends them to two non-colluding servers. The servers compute an element-wise median reference and assign weights using each gradient's Pearson correlation with that reference. PPAPAFL (Chen et al., 2024a) uses a similar correlation-based rule with CKKS to aggregate approximate real-valued updates. ShieldFL (Ma et al., 2022) uses two-trapdoor Paillier (Liu et al., 2018) to evaluate cosine similarity against a reference gradient and adjusts weights under non-IID data. BCPBFL (Miao et al., 2022) combines CKKS aggregation with blockchain, using similarity to a trusted gradient for weighting and blockchain to resist server-side tampering. RVPFL (Lu et al., 2024) uses Paillier and adjusted cosine similarity (Sarwar et al., 2001) to reduce the effect of gradient mean shifts. Lancelot (Jiang et al., 2025) evaluates encrypted pairwise Euclidean distances with CKKS and supports Krum and Multi-Krum aggregation.

These methods rely on encrypted similarity or distance calculations, involving operations such as inner products, L_2 norms, and means. Such operations increase computation and communication costs. Additive-only Paillier also requires interaction and blinding to evaluate them. Representing real-valued updates introduces a further consideration: Paillier typically uses fixed-point encoding, while CKKS uses approximate encoding and scale management. Both therefore require attention to numerical precision. CertiRank uses integer rankings and membership counts to express the statistics needed by CMGRA through additive aggregation. Table 6 summarizes the resulting differences in representations and secure operations.

B NOTATION SUMMARY

Table 7 summarizes the main notation. Client sets contain identifiers: $\mathcal{C}_u$ is the client indexed by $u \in \mathcal{C}$, and $\mathcal{Q}^{(t)} \subseteq \mathcal{P}^{(t)} \subseteq \mathcal{C}$. Layer-specific symbols are interpreted separately for each layer.

Table 7: Summary of the main notation.

Symbol	Meaning
Clients and ranking aggregation	
t, T	Global round and total number of rounds, where $t = 0$ denotes initialization.
$u, \mathcal{C}_u, \mathcal{D}_u$	Client identifier, corresponding client, and its local dataset.
$\mathcal{C}, \mathcal{P}^{(t)}, \mathcal{Q}^{(t)}$	Sets of all, participating, and accepted client identifiers.
U, B	Per-round upper bounds on participating and Byzantine clients, respectively.
$\mathcal{S}_1, \mathcal{S}_2$	The two auxiliary servers.
d, I, i, j	Layer size, edge-index set $I = \{0, \dots, d-1\}$, and edge indices.
K	Number of selected edges in the layer, with $1 \le K < d$.
$\theta^{\boldsymbol{w}}, \theta^{\boldsymbol{s}}, \psi$	Fixed network weights, edge scores, and number of local EP steps.
$\boldsymbol{r}_u^{(t)}, \boldsymbol{r}_g^{(t)}$	Local and global rank vectors, aligned with fixed edge IDs.
$\mu_{u,i}^{(t)}$	Indicator that edge i belongs to client u's top-K set.
$c_i^{(t)}, s_i^{(t)}$	Membership count and Borda sum over accepted submissions.
$A_t, S^{(t)}, S^{(0)}$	Membership-based candidate set, final selected set, and initial selected set.
Robustness and dynamics	
b_t, H_t, U_t	Accepted Byzantine, honest, and total client counts, where $U_t = b_t + H_t$.
$c_{i,H}^{(t)}, \Phi_t(S)$	Honest membership count for edge i and honest membership utility of set S.
$S_{H,t}^*, f_i^{(t)}$	An optimal honest top-K set and honest membership frequency.
$q_t, \mathcal{V}_T$	Number of newly selected edges and cumulative temporal variation of honest membership frequencies.
Encoding, encryption, and protocol	
N, M, m, ℓ_m	Polynomial degree, block count $M = \lceil d/N \rceil$, protocol block index, and valid block length.
$\boldsymbol{a}_m$	Indicator vector for the valid coordinates of block m.
$z, q, \mathcal{R}_z, \mathcal{R}_q$	Plaintext and ciphertext moduli and their polynomial quotient rings.
$\lambda, \boldsymbol{h}_u^{(t)}$	Packing base and packed vector, with entries $h_{u,i}^{(t)} = \lambda r_{u,i}^{(t)} + \mu_{u,i}^{(t)}$.
$v, v_m^{(t)}$	Generic cyclic-shift offset and round-specific offset for block m.
rot_v, R_v	Cyclic right shift by v positions, in encoding and security notation.
$\pi_u^{(t)}$	Permutation of client u's blocks in round t.
$\rho_m^{(t)}, \widehat{p}_m^{(t)}$	Aggregate mask and masked aggregate plaintext for block m.
$pk = (b, a), sk = s$	Public key and secret key polynomial.
r, e, e_1, e_2	Encryption randomness and key-generation/encryption error polynomials.
$ct = (c_1, c_2), C_{u,m}$	Generic ciphertext and client u's ciphertext for block m.
Δ	Plaintext scaling factor, $\lfloor q/z \rfloor$.

C THEORETICAL ANALYSIS OF CMGRA

C.1 FORMAL GUARANTEES

We analyze the complete CMGRA rule, including fallback exchanges, for $1 \le K < d$. Let $c_{i,H}^{(t)}$ be the membership count contributed by the H_t honest accepted submissions, and let $b_t \le B$ be the number of accepted Byzantine submissions, with B a nonnegative integer. These honest counts are generated along the actual training history, which may depend on previous adversarial rankings. Our deterministic guarantees allow colluding, adaptive Byzantine submissions that satisfy the prescribed validity checks, without assuming IID honest updates. Write $S_t = S^{(t)}$, $m = \min(K, d-K)$, and define

$$\Phi_t(S) = \sum_{i \in S} c_{i,H}^{(t)}, \qquad S_{H,t}^* \in \arg\max_{|S|=K} \Phi_t(S).$$

Per-round aggregation robustness. Beyond certifying individual changes, CMGRA controls the quality of its final selected set relative to the honest membership objective.

Theorem 1 (Honest membership approximation and exact recovery)**.** *For every round and every allowable Byzantine strategy,*

$$c_{j,H}^{(t)} - c_{i,H}^{(t)} \le B + b_t \quad (i \in S_t,\ j \notin S_t), \tag{7}$$

$$\begin{aligned} 0 \le \Phi_t(S_{H,t}^*) - \Phi_t(S_t) &\le |S_{H,t}^* \setminus S_t|(B + b_t) \\ &\le m(B + b_t). \end{aligned} \tag{8}$$

If the honest top-K boundary margin satisfies

$$\Delta_{H,t} := \min_{i \in S_{H,t}^*} c_{i,H}^{(t)} - \max_{j \notin S_{H,t}^*} c_{j,H}^{(t)} > B + b_t, \tag{9}$$

then CMGRA accepts a full certificate for $S_{H,t}^*$, independently of S_{t-1} and the Borda tie-breaking decisions.

This bound applies to both the full-certificate and fallback branches. Its reference is the honest membership optimum at the current history, not the output of a separate attack-free training run or an optimum of the prediction loss. A proof uses the fact that the final set admits no excluded–selected pair whose observed count difference exceeds B.

Multi-round subnetwork dynamics. The complete global ranking determines the score initialization of subsequent local EP, so changes in honest membership evidence must be included in the analysis. The following bound tracks that actual feedback through empirical membership frequencies.

Theorem 2 (Cumulative subnetwork changes)**.** *Assume $H_t \ge 1$ and define the actual honest membership frequencies, their cumulative variation, and the number of newly selected edges by*

$$f_i^{(t)} = c_{i,H}^{(t)}/H_t, \qquad q_t = |S_t \setminus S_{t-1}|,$$

$$\mathcal{V}_T = \sum_{t=2}^{T} \|\boldsymbol{f}^{(t)} - \boldsymbol{f}^{(t-1)}\|_\infty.$$

For every finite horizon T, including histories generated by adaptive Byzantine submissions,

$$\sum_{t=1}^{T} \frac{B + 1 - b_t}{H_t}\, q_t \le m(1 + \mathcal{V}_T). \tag{10}$$

If $H_t \le H_{\max}$, $B + 1 - b_t \ge \kappa > 0$, and $\mathcal{V}_\infty < \infty$, then only finitely many edges are replaced and S_t is eventually constant.

The finite-horizon bound does not require the honest membership variation to be small. Eventual stabilization requires the additional finite-variation condition and concerns the selected subnetwork, not the full ranking or training loss. The proofs below also discuss the strength of this condition and a preservation condition covering all within-group rankings.

Preliminaries. With the notation above, decompose $c_i^{(t)} = c_{i,H}^{(t)} + c_{i,B}^{(t)}$, where $0 \le c_{i,B}^{(t)} \le b_t$. Rank–membership consistency implies that each accepted submission selects exactly K edges, so $\sum_i c_{i,H}^{(t)} = H_t K$. All counts refer to the actual training history. The arguments hold for adaptive Byzantine submissions without independence or identical-distribution assumptions on honest updates. For layerwise aggregation, they apply to each layer with its own d, K.

Lemma 3 (Certificate soundness)**.** *For every certified challenger–incumbent pair (j, i) in round t,*

$$c_{j,H}^{(t)} - c_{i,H}^{(t)} \ge B + 1 - b_t \ge 1.$$

If a full top-K certificate is accepted, this inequality holds for every selected edge j and excluded edge i.

Proof. The integer-valued certificate implies $c_j^{(t)} - c_i^{(t)} \ge B + 1$, whereas $c_{j,B}^{(t)} - c_{i,B}^{(t)} \le b_t$. Subtraction proves the claim. For a full certificate, every selected–excluded count gap exceeds B. □

Lemma 4 (Boundary condition of the final set)**.** *After either branch of CMGRA, its final output satisfies*

$$\max_{j \notin S_t} c_j^{(t)} - \min_{i \in S_t} c_i^{(t)} \le B. \tag{11}$$

Proof. The full-certificate branch returns an observed top-K set, so the claim is immediate. For the fallback branch, suppress t and list incumbents as $i_1, \dots, i_K$ in nondecreasing count order and challengers as $j_1, \dots, j_{d-K}$ in nonincreasing count order. Borda and edge-ID tie-breaking leave these count inequalities unchanged. The paired gaps $c_{j_\ell} - c_{i_\ell}$ are nonincreasing in ℓ. Hence precisely a prefix of q pairs passes the strict threshold B.

After these exchanges, compare any excluded and selected edge. An unaccepted challenger cannot exceed an accepted challenger in count, while a removed incumbent cannot exceed a retained incumbent. If $q > 0$, every removed incumbent has count at most c_{i_q}, while every accepted challenger has count at least $c_{j_q} > c_{i_q} + B$. Finally, if both unaccepted challengers and retained incumbents exist, their maximum possible difference is $c_{j_{q+1}} - c_{i_{q+1}} \le B$, since the next pair failed. Empty categories require no comparison. These cases cover all excluded–selected pairs and prove the result. □

C.2 Proof of Theorem 1

Proof. For every final excluded–selected pair, Lemma 4 gives $c_j^{(t)} - c_i^{(t)} \le B$. Since $c_{i,B}^{(t)} - c_{j,B}^{(t)} \le b_t$, this yields equation 7. Pair the equally sized sets $S_{H,t}^* \setminus S_t$ and $S_t \setminus S_{H,t}^*$ and sum this inequality to obtain equation 8. Under equation 9, every observed count gap from an edge in $S_{H,t}^*$ to one outside it is at least $\Delta_{H,t} - b_t > B$. Thus $S_{H,t}^*$ is the unique observed top-K and passes the full certificate. □

Tightness and progress. The public-bound versions of the preceding guarantees replace $B + b_t$ by $2B$. This endpoint is attainable: let $d = 2$, $K = 1$, $B = b_t \ge 1$, and $H_t = 2B$. Suppose every honest client selects j, every Byzantine client selects i, and $S_{t-1} = \{i\}$. Both ballots are valid rankings. The observed correcting gap is exactly B, so neither branch changes the set, and the honest utility gap is $2B$. More generally, unanimous honest support guarantees full recovery against every allowable submission if $H_t > B + b_t$. With $U_t = H_t + b_t = |\mathcal{Q}^{(t)}|$ and $b_t = B$, this becomes $U_t > 3B$. This condition concerns worst-case progress, not the soundness of accepted changes.

C.3 Proof of Theorem 2

Proof. Use the normalized potential $F_t(S) = \sum_{i \in S} f_i^{(t)}$. For the fallback branch, sum Lemma 3 over its disjoint accepted pairs. For the full-certificate branch, pair the newly selected and newly excluded edges arbitrarily and apply the same lemma. Both branches therefore satisfy

$$F_t(S_t) - F_t(S_{t-1}) \ge \frac{B + 1 - b_t}{H_t} q_t.$$

Because $0 \le f_i^{(t)} \le 1$ and $\sum_i f_i^{(t)} = K$, every $F_t(S)$ lies in $[\max(0, 2K-d), K]$, an interval of width m. Moreover, $\sum_i (f_i^{(t)} - f_i^{(t-1)}) = 0$. Summing over S or its complement gives

$$|F_t(S) - F_{t-1}(S)| \le m \|\boldsymbol{f}^{(t)} - \boldsymbol{f}^{(t-1)}\|_\infty.$$

Telescoping the per-round improvement yields

$$\begin{aligned}\sum_{t=1}^{T} \frac{B+1-b_t}{H_t} q_t &\le F_T(S_T) - F_1(S_0) + \sum_{t=1}^{T-1} \big[F_t(S_t) - F_{t+1}(S_t)\big] \\ &\le m + m\mathcal{V}_T.\end{aligned}$$

Under the additional conditions, this bounds $\sum_t q_t$ by $H_{\max} m(1+\mathcal{V}_\infty)/\kappa$. Since q_t is a nonnegative integer, only finitely many rounds can change the selected set. □

Interpretation of stabilization. The variation in Theorem 2 concerns the actual honest responses, including effects of earlier poisoned rankings on score initialization and local EP. The finite-horizon bound does not assume that this variation is small. Finite total variation is an additional condition for eventual stabilization. Stationarity of the data distribution alone does not imply that condition for realized finite-sample updates. In fact, bounded integer H_t restricts the empirical frequencies to a finite set, so finite total variation here requires eventual exact constancy of those frequencies. Stabilization concerns the selected subnetwork, not the full ranking or convergence of the training loss. The theorem bounds the number of replacements, not a uniform communication-round deadline, since an adaptive adversary may delay admissible changes.

Proposition 5 (Preservation under within-group rank feedback)**.** *Fix a K-set S^* and let $\mathcal{R}(S^*)$ contain all full rankings whose top-K set is S^*. Suppose that, at every subsequent round, every aggregate of accepted honest local responses obtainable from a ranking in $\mathcal{R}(S^*)$ and any admissible associated client set and training states satisfies*

$$\min_{i \in S^*} c_{i,H} \ge \max_{j \notin S^*} c_{j,H}.$$

Once S^* is selected, it remains selected under arbitrary allowable adaptive Byzantine rankings.

Proof. For every outsider j and incumbent i, the premise gives $c_j - c_i \le b_t \le B$, so no fallback exchange passes. A full certificate selecting another set would require an inserted–removed pair with $c_j - c_i > B$, which is also impossible. Induction over rounds proves preservation. The quantifier over all within-group rankings explicitly includes the feedback through reinitialization of local scores. □

Guarantee boundaries. The preservation premise is a condition on honest learning responses, not an automatic property of membership gating. Similarly, $S^*_{H,t}$ uses the honest counts at the actual attacked history and need not match the output of a separate attack-free training run. Borda scores determine within-group ordering and resolve count ties in fallback pairing, so they may affect the identities of exchanged edges, but cannot bypass the membership certificate. The results establish aggregation robustness and conditional subnetwork stability, without asserting unconditional optimization convergence or immunity of prediction accuracy to all multi-round poisoning strategies.

D Encoding and Encryption Details

Let $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N+1)$ and $\mathcal{R}_z = \mathbb{Z}_z[x]/(x^N+1)$ denote the ciphertext and plaintext polynomial rings, respectively, where N is a power of two.

This appendix expands the encoding and encryption operations used in the main text. SEncode packs an integer vector into polynomial coefficients and preserves vector addition when the encoding offset is shared. Its sign adjustments also allow multiplication by the matching monomial to produce a cyclic shift of the vector entries. We first define these identities and their decoding rules, then describe the BFV-based RLWE-AHE operations that preserve them under encryption. Vectors are written in bold, such as $\boldsymbol{a}$, with entries a_i. Polynomials are written as $p(x)$, or simply p when unambiguous.

D.1 SIGN-AWARE ENCODING AND DECODING

The sign-aware encoding $\mathsf{SEncode}(\boldsymbol{a}, v)$ maps a vector into $\mathcal{R}_z$ using an offset v. Given an N-dimensional integer vector $\boldsymbol{a} = [a_0, a_1, \cdots, a_{N-1}]$ and a random integer $v \in [0, N-1]$, the encoding is defined as:

$$\mathsf{SEncode}(\boldsymbol{a}, v) = p_{\boldsymbol{a}}(x) = \sum_{i=0}^{N-v-1} a_i x^i - \sum_{i=N-v}^{N-1} a_i x^i.$$

The last v coefficients are negated in $\mathcal{R}_z$, while no coefficients change sign when $v = 0$. These sign changes compensate for the wrap-around signs introduced by the matching monomial multiplication.

The encoding preserves addition for a common offset. For two vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ encoded with the same v,

$$\mathsf{SEncode}(\boldsymbol{a}, v) + \mathsf{SEncode}(\boldsymbol{b}, v) = \mathsf{SEncode}(\boldsymbol{a} + \boldsymbol{b}, v).$$

This property ensures that addition in the encoded (or encrypted) domain directly corresponds to plaintext vector addition. The same sign pattern supports a cyclic shift by the prescribed offset. Let $p_{\boldsymbol{a}}(x) = \mathsf{SEncode}(\boldsymbol{a}, v)$ denote the polynomial encoding of the vector $\boldsymbol{a} = [a_0, a_1, \cdots, a_{N-1}]$. Multiplication by x^v modulo $x^N + 1$ produces the cyclic right shift of $\boldsymbol{a}$ by v positions. Formally, we have

$$x^v \cdot p_{\boldsymbol{a}}(x) \pmod{x^N + 1} = \sum_{i=0}^{v-1} a_{N-v+i} x^i + \sum_{i=v}^{N-1} a_{i-v} x^i.$$

Since $x^N \equiv -1 \pmod{x^N + 1}$, the coefficients that wrap from higher degrees to lower degrees would ordinarily incur a sign flip during such a rotation. However, in $\mathsf{SEncode}$, the last v coefficients of $p_{\boldsymbol{a}}(x)$ are pre-negated, so that these sign flips are exactly neutralized. The resulting coefficients therefore contain the original entries in cyclically shifted order. This identity requires the monomial exponent to match the encoding offset.

The decoding rule depends on whether the polynomial has undergone this matching shift.

- Sign-aware Decoding ($\mathsf{SDecode}$): Given a polynomial $p(x) = \sum_{i=0}^{N-1} b_i x^i \in \mathcal{R}_z$ and an offset v, the sign-aware decoding is defined as

  $$\mathsf{SDecode}(p(x), v) = [b_0, \cdots, b_{N-v-1}, -b_{N-v}, \cdots, -b_{N-1}].$$

  All negations are taken modulo z. This reverses the encoding signs, giving $\mathsf{SDecode}(\mathsf{SEncode}(\boldsymbol{a}, v), v) = \boldsymbol{a}$.

- Coefficient Decoding ($\mathsf{CDecode}$): Given a polynomial $p(x) = \sum_{i=0}^{N-1} b_i x^i \in \mathcal{R}_z$, $\mathsf{CDecode}$ directly maps its coefficients to a vector as

  $$\mathsf{CDecode}(p(x)) = [b_0, b_1, \cdots, b_{N-1}].$$

  Unlike $\mathsf{SDecode}$, this method does not perform sign restoration and is mainly used by the verification server $\mathcal{S}_2$ to recover the coefficients of the rotated plaintext for validity verification.

D.2 RLWE-AHE ALGORITHMS

We adopt an HE scheme following the BFV construction (Fan & Vercauteren, 2012), while retaining only the operations required by our protocol, namely ciphertext addition, ciphertext-plaintext addition, and lightweight ciphertext rotation. We refer to this streamlined scheme as RLWE-AHE, whose security relies on the small-secret decisional RLWE assumptions specified in Appendix F.1.

Let χ_{key} denote the distribution used to sample the secret polynomial, with each coefficient independently and uniformly drawn from $\{-1, 0, 1\}$, and let χ_{err} denote a discrete Gaussian distribution with a small standard deviation for sampling encryption errors. The RLWE-AHE scheme consists of the following algorithms and operations:

- **ParamGen**(γ, z): Given a security parameter γ and plaintext modulus z, this algorithm generates the public parameters $\texttt{Param}$, including the ring dimension N, ciphertext modulus q, and the parameters of the error distribution χ_{err}.
- **KeyGen**$(\texttt{Param})$: Given $\texttt{Param}$, sample a secret polynomial $s \leftarrow \chi_{\text{key}}$, a uniformly random polynomial $a \xleftarrow{\$} \mathcal{R}_q$, and an error polynomial $e \leftarrow \chi_{\text{err}}$. Compute $b = -as + e$, and output the secret key $sk = s$ and public key $pk = (b, a) \in \mathcal{R}_q^2$.
- **Enc**(pk, m): Given the public key $pk = (b, a)$ and a plaintext $m \in \mathcal{R}_z$, let $\Delta = \lfloor \frac{q}{z} \rfloor$ denote the plaintext scaling factor. Sample $r \leftarrow \chi_{\text{key}}$ and $e_1, e_2 \leftarrow \chi_{\text{err}}$, and compute the ciphertext
$$ct = (c_1, c_2) = (r \cdot b + \Delta \cdot m + e_1,\ r \cdot a + e_2) \in \mathcal{R}_q^2.$$
- **Dec**(sk, ct): Given the secret key $sk = s$ and a ciphertext $ct = (c_1, c_2)$, recover the plaintext as
$$m' = \left\lfloor \frac{z \cdot (c_1 + c_2 \cdot s \mod q)}{q} \right\rceil \in \mathcal{R}_z.$$
The decryption succeeds, i.e., $m' = m$, provided that the resulting phase satisfies the noise and scaling conditions in Appendix E.
- **Rot**(ct, v): Given a ciphertext $ct = (c_1, c_2)$ and an offset v, the rotation operation is defined as multiplication by the monomial x^v, i.e.,
$$\textbf{Rot}(ct, v) = (c_1 \cdot x^v, c_2 \cdot x^v) \in \mathcal{R}_q^2.$$
For $p_{\boldsymbol{a}}(x) = \mathsf{SEncode}(\boldsymbol{a}, v)$, where $\boldsymbol{a} = [a_0, \ldots, a_{N-1}]$, and its corresponding ciphertext $ct = \textbf{Enc}(pk, p_{\boldsymbol{a}}(x))$, suppose the resulting phase satisfies the noise and scaling conditions in Appendix E. Then the rotation operation satisfies $\textbf{Dec}(sk, \textbf{Rot}(ct, v)) = x^v p_{\boldsymbol{a}}(x) = \widetilde{m}(x) \in \mathcal{R}_z$. By the encoding identity, $\mathsf{CDecode}(\widetilde{m}(x)) = \text{rot}_v(\boldsymbol{a})$.
- **Add**(ct, ct^*): Given two ciphertexts $ct = (c_1, c_2)$ and $ct^* = (c_1^*, c_2^*)$ encrypting plaintexts $m, m^* \in \mathcal{R}_z$, respectively, their homomorphic addition is defined as
$$\textbf{Add}(ct, ct^*) = ct \oplus ct^* = (c_1 + c_1^*, c_2 + c_2^*) \in \mathcal{R}_q^2.$$
Accordingly, this scheme supports additive homomorphism, i.e., $\textbf{Dec}(sk, \textbf{Add}(ct, ct^*)) = m + m^* \in \mathcal{R}_z$, provided that the resulting phase satisfies the noise and scaling conditions in Appendix E.
- **PAdd**(ct, m^*): Given a ciphertext $ct = (c_1, c_2)$ encrypting $m \in \mathcal{R}_z$ and a plaintext $m^* \in \mathcal{R}_z$, their ciphertext-plaintext addition is defined as
$$\textbf{PAdd}(ct, m^*) = (c_1 + \Delta m^*,\ c_2) \in \mathcal{R}_q^2.$$
Accordingly, $\textbf{Dec}(sk, \textbf{PAdd}(ct, m^*)) = m + m^* \in \mathcal{R}_z$, provided that the resulting phase satisfies the noise and scaling conditions in Appendix E.

The conditional IND-CPA security of RLWE-AHE under these assumptions is established in Appendix F.1.

E RLWE-AHE Parameters and Correctness Conditions

This appendix states the correctness conditions for the RLWE-AHE operations in Section 3.2 and distinguishes them from the experimental parameter choices. The reported experiments use a 60-bit ciphertext modulus, $z = 2^{35}$, and polynomial degrees $N \in \{4096, 8192, 16384\}$. The error distribution has reported standard deviation $\sigma = 8$, and each coefficient of r and s is sampled uniformly from $\{-1, 0, 1\}$. The evaluated aggregation workload includes up to $U = 256$ clients. The authors report no observed decryption errors with the experimental 60-bit configuration. This empirical observation does not specify a cryptographic failure probability or replace the conditions below.

E.1 Fresh Encryption Noise and Supported Operations

All polynomial products below are taken modulo $x^N + 1$. Substituting $b = -as + e$ into the encryption equation gives

$$c_1 + c_2 s = \Delta \bar{m} + \eta \pmod{q}, \qquad \eta = re + e_1 + e_2 s, \qquad \Delta = \lfloor q/z \rfloor, \tag{12}$$

where $\bar{m}$ denotes the integer coefficient lift used to encode $m \in \mathcal{R}_z$. The two polynomial products contribute to the noise and cannot be replaced by a single draw from the error distribution. For independent, centered error coefficients with variance σ^2, each fresh noise coefficient satisfies

$$\mathrm{Var}(\eta_i) = \sigma^2 \left(\frac{4N}{3} + 1 \right). \tag{13}$$

Thus, the standard deviation of an error sample is distinct from the standard deviation of a ciphertext's decryption noise.

Multiplication by a monomial permutes the noise coefficients and may change their signs. Consequently,

$$\|x^v \eta \bmod (x^N + 1)\|_\infty = \|\eta\|_\infty. \tag{14}$$

Ciphertext addition adds the corresponding noise polynomials. The verification branch applies a monomial rotation, whereas aggregation uses the retained ciphertexts before verification obfuscation. For at most U honestly generated ciphertexts, each with noise norm at most B, the aggregate noise norm is at most UB. Plaintext addition introduces no fresh encryption randomness, but its effect on scaling must also be included, as shown below.

E.2 A Conditional Bound on the Failure Probability

To obtain a tail bound, assume that each independent, centered error coefficient E satisfies

$$\mathbb{E}[\exp(tE)] \le \exp(\tau^2 t^2/2) \quad \text{for every } t \in \mathbb{R}. \tag{15}$$

Here τ is a verified sub-Gaussian parameter for the actual sampler. A reported standard deviation alone does not establish Eq. equation 15 with $\tau = \sigma$. The sampler's distribution convention and any truncation must therefore be specified when instantiating this bound.

Conditioned on r and s, each coefficient in Eq. equation 12 is a sum of independent error coefficients with weights in $\{-1, 0, 1\}$. Its squared sub-Gaussian parameter is at most

$$\tau^2 \big(\|r\|_2^2 + \|s\|_2^2 + 1 \big) \le \tau^2 (2N + 1). \tag{16}$$

Chernoff's inequality and a union bound over all coefficients of L fresh ciphertexts give

$$\Pr\left[\max_{\ell \le L} \|\eta_\ell\|_\infty > B \right] \le 2NL \exp\left(-\frac{B^2}{2\tau^2(2N+1)} \right). \tag{17}$$

This union bound does not require independent ciphertext noise polynomials. It also permits reuse of the public-key error polynomial e. For a target probability $\varepsilon \in (0, 1)$, define

$$B_\varepsilon = \tau \sqrt{2(2N+1) \log\left(\frac{2NL}{\varepsilon} \right)}. \tag{18}$$

With probability at least $1 - \varepsilon$, every such ciphertext has noise norm at most B_ε. Every aggregate of at most U of these ciphertexts, including any monomial rotations, then has noise norm at most UB_ε. The count L must cover all packed ciphertexts, layers, and rounds included in the claimed guarantee. Runs with different parameter sets may be assigned separate failure budgets whose sum bounds the overall failure probability.

E.3 Scaling, Masking, and Exact Decryption

The floor in $\Delta = \lfloor q/z \rfloor$ contributes an additional term to decryption correctness. Let $r_q = q - z\Delta$, so that $0 \le r_q < z$. For any supported computation, let $F \in \mathbb{Z}[x]/(x^N + 1)$ be the unreduced sum of the message lifts after the prescribed signed coefficient permutations. When a plaintext mask is

added, its implemented integer lift is also included in F. The resulting decryption phase has the form

$$c_1 + c_2 s = \Delta F + \eta_{\mathrm{tot}} \pmod{q}. \tag{19}$$

After multiplication by z/q, coefficientwise rounding, and reduction modulo z, decryption returns

$$F + \left\lfloor \frac{z\eta_{\mathrm{tot}} - r_q F}{q} \right\rceil \pmod{z}. \tag{20}$$

Hence, a sufficient condition for recovering $F \bmod z$ is

$$\left\| \eta_{\mathrm{tot}} - \frac{r_q}{z} F \right\|_\infty < \frac{q}{2z}. \tag{21}$$

In particular, if $\|F\|_\infty \le F_{\max}$ throughout the computation, the conditional noise bound above gives the sufficient condition

$$U B_\varepsilon + \frac{r_q}{z} F_{\max} < \frac{q}{2z}. \tag{22}$$

The bound $F_{\max}$ depends on the actual message lifts and must include additions before reduction modulo z. For example, canonical lifts in $\{0, \dots, z-1\}$ give $F_{\max} \le (U+1)(z-1)$ for an aggregate followed by one plaintext mask. This bound remains valid under the signed coefficient permutations induced by monomial multiplication. A tighter bound may use the supported input range and the implemented signed lifts.

The aggregate mask is sampled uniformly from $\mathcal{R}_z$, so its contribution cannot be bounded using only the range of unmasked client values. Consequently, the ciphertext modulus bit length alone does not establish Eq. equation 22, since the exact value of $q \bmod z$ and the lifting convention also matter. If an implementation instead embeds coefficients by rounding qm/z, its rounding errors must replace the floor-scaling term in the analysis. These embedding conventions should not be interchanged without updating the encryption and plaintext-addition definitions.

E.4 Plaintext Range and Interpretation of the Parameters

Correct recovery of an element of $\mathcal{R}_z$ is distinct from recovery of the intended integer aggregate. For the joint representation $h_{u,i} = \lambda r_{u,i} + \mu_{u,i}$, the protocol uses $\lambda > U$ and requires

$$z > \lambda U(d-1) + U, \qquad \lambda = 2^{\lceil \log_2(U+1) \rceil}. \tag{23}$$

This condition prevents integer wrap-around after mask removal and sign-aware decoding. At $U = 256$, $\lambda = 512$ and $z = 2^{35}$ satisfy Eq. equation 23 for layers with $d \le 262144$. Larger layers require a separate range check and parameter choice. The primitive benchmark's sampled values lie in $[2^{25}, 2^{26})$, whose sums over 256 clients are below 2^{34}. That benchmark range does not replace the layer-specific check for the joint representation.

For a 60-bit q and $z = 2^{35}$, the nominal threshold satisfies $2^{23} \le q/(2z) < 2^{24}$. Comparison with this threshold must use the complete expression in Eq. equation 22, rather than an individual error sample's standard deviation. The reported absence of decryption errors supports retaining the experimental configuration for the evaluated runs. An explicit failure-probability guarantee additionally requires the exact modulus, the sampler's verified tail parameter, the implemented message lifts, and the full ciphertext count L. The available experimental report does not supply all these quantities or a measured noise-budget distribution. Accordingly, the 60-bit configuration is reported as an evaluated setting, rather than as an unconditional correctness guarantee.

The reported Lattice Estimator evaluation places the 60-bit configuration at at least 128-bit classical security for $N = 4096$. The original parameter assessment extends this security claim to the larger dimensions at the same ciphertext modulus. This security estimate and the correctness conditions address different questions. Reproduction of the estimate requires the exact modulus, secret and error distributions, estimator revision, attack models, and cost assumptions. The dimension and modulus bit length alone do not provide that record.

F Security Analysis

This section analyzes the security components of CertiRank under the threat model in Section 4. We establish the conditional IND-CPA security of RLWE-AHE, characterize the plaintext leakage of two-stage obfuscation, and state the resulting component guarantees under server–client collusion.

F.1 SECURITY OF RLWE-AHE

RLWE-AHE uses the BFV-style public-key encryption construction described in Section 3.2. The following argument states the small-secret RLWE assumption needed for its IND-CPA security. Decryption correctness additionally requires the noise and scaling conditions in Appendix E.

F.1.1 IND-CPA SECURITY DEFINITION

Definition 6 (IND-CPA Security)**.** The challenger generates public parameters $\texttt{Param}$ and a key pair (pk, sk), and gives $(\texttt{Param}, pk)$ to an adversary $\mathcal{A}$. After receiving the public key, $\mathcal{A}$ chooses two messages $m_0, m_1 \in \mathcal{R}_z$. The challenger samples $\beta \xleftarrow{\$} \{0,1\}$ and returns $ct^* = \mathbf{Enc}(pk, m_\beta)$. The adversary then outputs a bit β'. The scheme is IND-CPA secure if every probabilistic polynomial-time adversary has negligible advantage

$$\mathrm{Adv}_{\mathcal{A}}^{\text{IND-CPA}}(\gamma) = \left| \Pr[\beta' = \beta] - \frac{1}{2} \right|.$$

Because encryption is public, the adversary can generate additional encryptions locally, so no encryption oracle is required in this game.

F.1.2 RLWE ASSUMPTION

Let $\mathcal{R}_q = \mathbb{Z}_q[x]/(x^N + 1)$, with the parameters and sampling distributions used by RLWE-AHE. The coefficients of a secret $w \leftarrow \chi_{\text{key}}$ are independently and uniformly sampled from $\{-1, 0, 1\}$. For $k \in \{1, 2\}$, define the following distributions over k pairs of ring elements:

$$\mathsf{L}_k = \big((a_j, a_j w + e_j)\big)_{j=1}^{k}, \qquad \mathsf{U}_k = \big((a_j, u_j)\big)_{j=1}^{k}.$$

Here all $a_j, u_j \xleftarrow{\$} \mathcal{R}_q$ are mutually independent, and the errors $e_j \leftarrow \chi_{\text{err}}$ are independently sampled. The secret and errors are independent of the uniform ring elements, and the same secret w is used throughout L_k. For a probabilistic polynomial-time distinguisher $\mathcal{D}$, write

$$\mathrm{Adv}_{\mathcal{D}}^{k\text{-RLWE}}(\gamma) = \left| \Pr[\mathcal{D}(\mathsf{L}_k) = 1] - \Pr[\mathcal{D}(\mathsf{U}_k) = 1] \right|.$$

We assume these advantages are negligible for the parameter family and secret and error distributions used by the scheme. The two-sample form explicitly covers the correlated ciphertext components. It is included in the assumption rather than inferred from a single isolated sample. The one-sample form also follows from this two-sample assumption by discarding the second pair.

F.1.3 SECURITY THEOREM

Theorem 7. *Under the small-secret RLWE assumption above, RLWE-AHE is IND-CPA secure. For every probabilistic polynomial-time adversary $\mathcal{A}$, there exist distinguishers $\mathcal{D}_1$ and $\mathcal{D}_2$, with polynomial simulation overhead, such that*

$$\mathrm{Adv}_{\mathcal{A}}^{\text{IND-CPA}}(\gamma) \le \mathrm{Adv}_{\mathcal{D}_1}^{1\text{-RLWE}}(\gamma) + \mathrm{Adv}_{\mathcal{D}_2}^{2\text{-RLWE}}(\gamma).$$

Proof. Let G_0 be the real IND-CPA game, and let p_i denote the adversary's success probability in game G_i. Set $\Delta = \lfloor q/z \rfloor$ and let $\bar{m}$ denote the fixed public integer coefficient lift used by the encryption algorithm. All ciphertext expressions below are reduced modulo q and $x^N + 1$.

Game G_1: replacing the public-key component. Replace $b = -as + e$ by an independent uniform $\widetilde{b} \xleftarrow{\$} \mathcal{R}_q$ while keeping a uniform. The challenger still encrypts using the resulting public key $(\widetilde{b}, a)$. Because χ_{key} is symmetric, $-s$ has the same distribution as s. Thus $(a, -as + e)$ has distribution L_1. Given a one-sample challenge (a, y), $\mathcal{D}_1$ supplies $pk = (y, a)$ to $\mathcal{A}$ and generates the challenge encryption using fresh r, e_1, e_2. It samples β itself and outputs 1 exactly when $\mathcal{A}$ returns $\beta' = \beta$. This simulation produces G_0 for a real RLWE sample and G_1 for a uniform sample, giving

$$|p_0 - p_1| \le \mathrm{Adv}_{\mathcal{D}_1}^{1\text{-RLWE}}(\gamma).$$

Game G_2: jointly replacing both ciphertext components. In G_1, the challenge ciphertext is

$$ct^* = (r\widetilde{b} + e_1 + \Delta\bar{m}_\beta,\ ra + e_2).$$

Replace the pair $(r\widetilde{b}+e_1, ra+e_2)$ jointly by independent uniform elements (u_1, u_2). To implement this step, $\mathcal{D}_2$ receives a two-sample challenge $((a_1, y_1), (a_2, y_2))$ and supplies $pk = (a_1, a_2)$ to $\mathcal{A}$. After receiving m_0, m_1, it samples β and returns

$$ct^* = (y_1 + \Delta\bar{m}_\beta,\ y_2).$$

It again outputs 1 exactly when $\beta' = \beta$. For a real challenge, set $\widetilde{b} = a_1$, $a = a_2$, and interpret the shared RLWE secret as the encryption randomness r. The resulting view is exactly G_1, including the correlation between ciphertext components. For a uniform challenge, the view is exactly G_2. Consequently,

$$|p_1 - p_2| \le \mathrm{Adv}^{\text{2-RLWE}}_{\mathcal{D}_2}(\gamma).$$

In G_2, the ciphertext is $(u_1 + \Delta\bar{m}_\beta, u_2)$. Conditioned on the public key and the chosen messages, it consists of independent uniform ring elements and is independent of β. Therefore $p_2 = 1/2$, and the triangle inequality yields

$$\begin{aligned}\mathrm{Adv}^{\text{IND-CPA}}_{\mathcal{A}}(\gamma) &= |p_0 - 1/2| \\ &\le |p_0 - p_1| + |p_1 - p_2| \\ &\le \mathrm{Adv}^{\text{1-RLWE}}_{\mathcal{D}_1}(\gamma) + \mathrm{Adv}^{\text{2-RLWE}}_{\mathcal{D}_2}(\gamma).\end{aligned}$$

Both terms are negligible under the stated assumption, which proves the claim. □

F.2 Security of the Two-Stage Obfuscation

The two-stage obfuscation preserves the ranking values and membership bits needed for verification while changing their coordinate positions. We first analyze one honest client's submission in a single round and layer, omitting the client and round indices. For clarity, assume initially that $d = MN$, so all M sub-vectors contain N valid entries.

Let R_v denote a cyclic right shift by v positions. The offsets $v_1, \ldots, v_M$ are sampled independently and uniformly from $\mathbb{Z}_N$ and are shared across clients within the round. For each client, $\mathcal{S}_1$ independently samples a uniform permutation π and retains it. With $\pi(k)$ denoting the original identity of the kth received sub-vector, the verification plaintext is

$$V_{\boldsymbol{v},\pi}(\boldsymbol{h}) = \big[R_{v_{\pi(1)}}\boldsymbol{h}_{\pi(1)}, \ldots, R_{v_{\pi(M)}}\boldsymbol{h}_{\pi(M)}\big].$$

Thus, each offset remains associated with its original sub-vector, even though the shifted sub-vectors are presented in a different order.

Theorem 8 (Position randomization and verification leakage)**.** *For a valid submission with $d = MN$, the two-stage obfuscation has the following properties.*

- ***Marginal position randomization.*** *When the offsets and permutation are undisclosed, each fixed original position is mapped uniformly to the d output positions. Under a uniform prior over original positions, the posterior remains uniform when conditioned only on its observed position. This statement does not condition on the remaining verification plaintext or auxiliary information.*
- ***Structural leakage.*** *The verification plaintext reveals which ranks belong to the same sub-vector, their relative cyclic positions, and their associated membership bits.*
- ***Ambiguity with disclosed offsets.*** *Given $\mathbf{v}$, the verification plaintext alone admits exactly $M!$ distinct valid edge-aligned ranking vectors. Each received sub-vector has M compatible original identities, and each rank has M compatible original coordinates. Each candidate identity specifies the corresponding inverse shift.*

Proof. Represent an original position by (m, j), where $m \in \{1, \ldots, M\}$ and $j \in \mathbb{Z}_N$. Its obfuscated position is

$$\tau(m, j) = \big(\pi^{-1}(m), j + v_m \bmod N\big).$$

Independence and uniform sampling imply that, for any output position (k, j'),

$$\Pr[\tau(m, j) = (k, j')] = \Pr[\pi^{-1}(m) = k] \Pr[v_m = j' - j \bmod N] = \frac{1}{M}\frac{1}{N} = \frac{1}{d}.$$

The stated posterior follows from Bayes' rule under the uniform prior and the specified conditioning.

A sub-vector permutation preserves the grouping of its entries. Within each sub-vector, a common cyclic shift also preserves every position difference modulo N. Because the transformation changes positions without changing entry values, the ranks and their membership bits remain visible in the verification plaintext.

Now suppose that $\boldsymbol{v}$ is disclosed, and write the received sub-vectors as $\boldsymbol{y}_1, \ldots, \boldsymbol{y}_M$. Assigning $\boldsymbol{y}_k$ to original identity m gives the candidate

$$\boldsymbol{h}_m = R_{-v_m} \boldsymbol{y}_k.$$

Every bijection between received sub-vectors and original identities produces a valid candidate ranking. It preserves the rank permutation and rank–membership consistency, since it only reorders the joint entries. The $M!$ assignments yield distinct vectors because every rank occurs exactly once. For a rank observed at position j' in a received sub-vector, the compatible original coordinates are

$$\big\{(m, j' - v_m \bmod N) : m = 1, \ldots, M\big\}.$$

Hence, disclosing the offsets supplies an inverse shift for each candidate identity without supplying the identity correspondence. Repeated offsets may make some inverse shifts identical, but do not eliminate the distinct candidate identities. □

Padding and layerwise interpretation. For a general layer of dimension d, only its final sub-vector may contain padding. The disclosed validity information can identify and align this partial sub-vector, exposing its valid entries in their original coordinates. Let $F = \lfloor d/N \rfloor$ denote the number of full sub-vectors. Thus, $F = M$ without padding and $F = M - 1$ when a partial sub-vector is present. Even after fixing the partial sub-vector, the verification plaintext, disclosed offsets, and padding metadata alone admit $F!$ compatible assignments of the full sub-vectors. The preceding proof applies to these assignments: each bijection specifies the inverse shifts and preserves ranking validity. The remaining identities are therefore non-unique when $F \geq 2$, whereas no permutation ambiguity remains for $F \leq 1$. This characterization applies separately to each layer.

These counts describe compatibility with the stated verification information, not equally likely candidates under arbitrary priors or a computational security level. When $F \geq 2$, these counts identify the role of the client-specific permutation: learning the offsets does not, by itself, determine the correspondence between received and original full sub-vectors.

F.3 Security of CertiRank

We now relate the obfuscation analysis to CertiRank's encrypted uploads and masked aggregate recovery under the threat model in Section 4. These mechanisms serve complementary purposes: encryption protects uploads from parties without the secret key, obfuscation changes the positions exposed during verification, and masking randomizes the aggregate plaintext returned by decryption. Aggregate statistics and global outputs are intended client outputs and are therefore available to a server that colludes with a client.

We first establish the plaintext property of the aggregate masks.

Lemma 9 (Uniform aggregate masking)**.** *Let p be an arbitrary $\mathcal{R}_z$-valued plaintext and let ρ be sampled uniformly from $\mathcal{R}_z$, independently of p. Then $\widehat{p} = p + \rho \bmod z$ is uniform over $\mathcal{R}_z$ and statistically independent of p.*

Proof. For every fixed p and $\widehat{p}$, exactly one mask satisfies $\rho = \widehat{p} - p \bmod z$. Since $\mathcal{R}_z$ contains z^N elements,

$$\Pr[\widehat{p} = \widehat{p}_0 \mid p = p_0] = \Pr[\rho = \widehat{p}_0 - p_0 \bmod z] = z^{-N}.$$

This conditional distribution is uniform and does not depend on p_0, establishing the claim. □

CertiRank uses fresh independent masks for each sub-vector and round. Under the decryption conditions in Appendix E, the decrypted masked aggregate is precisely the plaintext covered by Lemma 9.

Theorem 10 (Privacy properties of CertiRank)**.** *Assume correct decryption and that the RLWE assumption holds for the parameters of RLWE-AHE. The auxiliary servers $\mathcal{S}_1$ and $\mathcal{S}_2$ are semi-honest and non-colluding. Malicious clients may collude with each other and with at most one auxiliary server. Then CertiRank provides three component guarantees: IND-CPA protection of encrypted uploads against parties without sk, verification-plaintext obfuscation with the leakage characterized in Theorem 8 and the padding analysis above, and uniform masking of aggregate plaintexts as established in Lemma 9. In particular, when $F \geq 2$, the remaining full sub-vectors have non-unique compatible original identities in the verification plaintext, even when the shared offsets are disclosed. This positional claim considers only that plaintext, the offsets, and the disclosed padding information.*

Proof. We examine the three adversarial cases and identify the applicable component guarantees.

No server–client collusion. Server $\mathcal{S}_1$ receives encrypted client submissions but does not possess sk. Theorem 7 establishes the IND-CPA security of this upload encryption. Server $\mathcal{S}_2$ decrypts the position-obfuscated verification copies, whose values and positional structure are characterized in Theorem 8 and the padding analysis. During aggregate recovery, it decrypts independently masked sums. Lemma 9 establishes that each resulting masked plaintext is independent of the corresponding aggregate plaintext.

Clients collude with $\mathcal{S}_1$. The coalition knows its own inputs and the offsets, permutations, and masks generated or distributed by $\mathcal{S}_1$. It also obtains the intended client outputs, but still does not possess sk. The upload-encryption guarantee of Theorem 7 therefore remains the applicable cryptographic property.

Clients collude with $\mathcal{S}_2$. The coalition possesses sk and learns the shared offsets $\boldsymbol{v}$ from colluding clients. It can therefore compute an inverse shift for every candidate original sub-vector identity. The client-specific permutation is generated and retained by $\mathcal{S}_1$, rather than distributed to clients. For a single honest client's layer, Theorem 8 and the padding analysis give $F!$ valid candidate assignments from the verification plaintext, disclosed offsets, and padding information alone. When $F \geq 2$, these data do not uniquely determine which original identity belongs to each received full sub-vector. The identified partial sub-vector is excluded from this ambiguity. As in the preceding case, the coalition also receives the aggregate statistics through its colluding clients. Their availability is consistent with the intended output of the protocol. □

G COMPLETE CERTIRANK ALGORITHM

Algorithm 1 gives the protocol for one ranked layer, while multi-layer execution uses each layer's d, K and the common round schedule. Algorithm 2 specifies both certification branches and the complete global ranking. All sorting is lexicographic, with smaller edge identifiers preferred when substantive criteria tie. The empty-submission convention retains the preceding state.

Algorithm 1 Complete CertiRank protocol

Require: Client index set $\mathcal{C}$, participant cap U, bound B, layer size d, $1 \le K < d$, security parameter γ, rounds T, local steps ψ

Ensure: Selected sets $S^{(t)}$ and edge-aligned global rankings $\boldsymbol{r}_g^{(t)}$ at all clients

Public setup: fix edge IDs $I = \{0, \dots, d-1\}$ and a common random seed.
Clients: initialize fixed weights $\theta^{\boldsymbol{w}}$ and scores $\theta^{\boldsymbol{s}}$ from the seed; derive $\boldsymbol{r}_g^{(0)}$ and its top-K set $S^{(0)}$.
$\mathcal{S}_2$: set $\lambda = 2^{\lceil \log_2(U+1) \rceil}$; choose $z > \lambda U(d-1) + U$.
$\mathcal{S}_2$: generate $\texttt{Param} = \textbf{ParamGen}(\gamma, z)$ and $(pk, sk) = \textbf{KeyGen}(\texttt{Param})$; distribute $(\texttt{Param}, pk)$ and retain sk.
Set $M = \lceil d/N \rceil$, $\ell_m = \min(N, d - (m-1)N)$, and $a_{m,j} = \mathbf{1}[j < \ell_m]$ for $0 \le j < N$.
for $t = 1, \dots, T$ **do**
    Select the participating clients $\mathcal{P}^{(t)} \subseteq \mathcal{C}$, with $|\mathcal{P}^{(t)}| \le U$.
    $\mathcal{S}_1$: sample independent uniform $v_m^{(t)} \leftarrow \mathbb{Z}_N$ for $m = 1, \dots, M$; send this common shift vector to all clients.
    for each participating client $u \in \mathcal{P}^{(t)}$ **do**
        $\mathcal{C}_u$: use initial scores if $t = 1$; otherwise reorder initial scores using $\boldsymbol{r}_g^{(t-1)}$; run ψ local EP steps on $\mathcal{D}_u$ with fixed weights.
        $\mathcal{C}_u$: obtain the edge ranking $\boldsymbol{r}_u^{(t)}$; set $\mu_{u,i}^{(t)} = \mathbf{1}[r_{u,i}^{(t)} \ge d - K]$ and $h_{u,i}^{(t)} = \lambda r_{u,i}^{(t)} + \mu_{u,i}^{(t)}$.
        $\mathcal{C}_u$: partition $\boldsymbol{h}_u^{(t)}$ into M sub-vectors, zero-pad to length N, and send $C_{u,m} = \textbf{Enc}(pk, \mathsf{SEncode}(\boldsymbol{h}_{u,m}^{(t)}, v_m^{(t)}))$ to $\mathcal{S}_1$.
    end for
    $\mathcal{S}_1$: retain all original $C_{u,m}$ for aggregation; initialize $\mathcal{Q}^{(t)} = \varnothing$.
    for each received submission $u \in \mathcal{P}^{(t)}$ **do**
        $\mathcal{S}_1$: sample an independent uniform permutation $\pi_u^{(t)}$ of $\{1, \dots, M\}$.
        $\mathcal{S}_1 \to \mathcal{S}_2$: send, for each k, $\widetilde{C}_{u,k} = \textbf{Rot}(C_{u,m}, v_m^{(t)})$ and $\widetilde{\boldsymbol{a}}_{u,k} = R_{v_m^{(t)}} \boldsymbol{a}_m$, where $m = \pi_u^{(t)}(k)$.
        $\mathcal{S}_2$: compute $\boldsymbol{y}_{u,k} = \mathsf{CDecode}(\textbf{Dec}(sk, \widetilde{C}_{u,k}))$; reject and skip malformed submissions or those with nonzero padding.
        $\mathcal{S}_2$: concatenate coordinates with $\widetilde{a}_{u,k,j} = 1$; extract $\widetilde{\mu} = h \bmod \lambda$ and $\widetilde{r} = (h - \widetilde{\mu})/\lambda$ at each coordinate.
        $\mathcal{S}_2$: add u to $\mathcal{Q}^{(t)}$ iff these d ranks form a permutation of I, every $\widetilde{\mu} \in \{0, 1\}$, and $\widetilde{\mu} = \mathbf{1}[\widetilde{r} \ge d - K]$ everywhere.
    end for
    $\mathcal{S}_2 \to \mathcal{S}_1$: return $\mathcal{Q}^{(t)}$.
    if $\mathcal{Q}^{(t)} = \varnothing$ **then**
        $\mathcal{S}_1$: notify all clients; retain $(S^{(t)}, \boldsymbol{r}_g^{(t)}) = (S^{(t-1)}, \boldsymbol{r}_g^{(t-1)})$ and **continue**.
    end if
    for $m = 1, \dots, M$ **do**
        $\mathcal{S}_1$: aggregate the original copies, $C_m = \bigoplus_{u \in \mathcal{Q}^{(t)}} C_{u,m}$; sample $\rho_m^{(t)} \leftarrow \mathcal{R}_z$ uniformly and independently.
        $\mathcal{S}_1 \to \mathcal{S}_2$: send $\widehat{C}_m = \textbf{PAdd}(C_m, \rho_m^{(t)})$; $\mathcal{S}_1 \to \mathcal{C}$: send $\rho_m^{(t)}$.
        $\mathcal{S}_2 \to \mathcal{C}$: send $\widehat{p}_m = \textbf{Dec}(sk, \widehat{C}_m)$.
        Each client: recover $\boldsymbol{h}_m^{(t)} = \mathsf{SDecode}((\widehat{p}_m - \rho_m^{(t)}) \bmod z, v_m^{(t)}) \bmod z$; keep its first ℓ_m coordinates.
    end for
    Each client: concatenate the recovered sub-vectors; set $c_i^{(t)} = h_i^{(t)} \bmod \lambda$ and $s_i^{(t)} = (h_i^{(t)} - c_i^{(t)})/\lambda$.
    Each client: compute $(S^{(t)}, \boldsymbol{r}_g^{(t)}) = \mathrm{CMGRA}(\boldsymbol{c}^{(t)}, \boldsymbol{s}^{(t)}, S^{(t-1)}, K, B, I)$ using Algorithm 2.
end for

Algorithm 2 CMGRA: certified selection and complete rank reconstruction

Require: Membership counts $\boldsymbol{c}$, Borda scores $\boldsymbol{s}$, preceding set S_{prev}, K, B, and edge IDs I
Ensure: A set S with $|S| = K$ and a permutation $\boldsymbol{r}_g$ of $\{0, \dots, d-1\}$

function CMGRA($\boldsymbol{c}, \boldsymbol{s}, S_{\text{prev}}, K, B, I$)
  Let L be I sorted by $(-c_i, i)$ in ascending lexicographic order.
  $T_K \leftarrow \{L_1, \dots, L_K\}$; $c_{\text{in}} \leftarrow \min_{i \in T_K} c_i$; $c_{\text{out}} \leftarrow \max_{j \notin T_K} c_j$.
  if $c_{\text{in}} - c_{\text{out}} > B$ **then**
    $S \leftarrow T_K$. ▷ Full membership certificate
  else
    $S \leftarrow S_{\text{prev}}$.
    List incumbents $(i_1, \dots, i_K)$ from S_{prev} in ascending order of $(c_i, s_i, -i)$.
    List challengers $(j_1, \dots, j_{d-K})$ from $I \setminus S_{\text{prev}}$ in ascending order of $(-c_j, -s_j, j)$.
    for $a = 1, \dots, \min(K, d-K)$ **do**
      if $c_{j_a} - c_{i_a} > B$ **then**
        $S \leftarrow (S \setminus \{i_a\}) \cup \{j_a\}$.
      end if
    end for
  end if
  Sort $I \setminus S$ into $(b_1, \dots, b_{d-K})$ by $(s_i, -i)$ in ascending order.
  Sort S into $(a_1, \dots, a_K)$ by $(s_i, -i)$ in ascending order.
  for $q = 1, \dots, d-K$ **do**
    $r_{g,b_q} \leftarrow q - 1$.
  end for
  for $q = 1, \dots, K$ **do**
    $r_{g,a_q} \leftarrow d - K + q - 1$.
  end for
  return $(S, \boldsymbol{r}_g)$.
end function

H Detailed CertiRank Workflow

This appendix describes each protocol stage in detail, following Algorithm 1. The system architecture is illustrated in Figure 1.

H.1 System Initialization

Given the maximum number of participating clients U in each round, server $\mathcal{S}_2$ sets the packing base as $\lambda = 2^{\lceil \log_2(U+1) \rceil}$, i.e., the smallest power of two strictly larger than U. It further chooses the plaintext modulus z sufficiently large to avoid modular wrap-around during aggregation, satisfying $z > \lambda U(d-1)+U$. Given the security parameter γ and plaintext modulus z, $\mathcal{S}_2$ generates the public parameters $\texttt{Param} = \textbf{ParamGen}(\gamma, z)$ and an RLWE-AHE key pair $(pk, sk) = \textbf{KeyGen}(\texttt{Param})$. It distributes $\texttt{Param}$ and pk to $\mathcal{S}_1$ and all clients while retaining sk locally.

Following FRL, all clients use a common public random seed to initialize the fixed network weights $\theta^{\boldsymbol{w}}$ and initial edge scores $\theta^{\boldsymbol{s}}$, as well as to determine the initial selected set $S^{(0)}$ required by CMGRA.

To support subsequent encrypted aggregation, at the beginning of each global round t, $\mathcal{S}_1$ generates a common shift vector $\boldsymbol{v}^{(t)} = [v_1^{(t)}, \dots, v_M^{(t)}]$, where $M = \lceil d/N \rceil$ denotes the number of sub-vectors obtained by partitioning a d-dimensional vector into segments of at most N elements, and each $v_m^{(t)}$ is independently sampled as $v_m^{(t)} \xleftarrow{\$} \mathbb{Z}_N$. The shift vector is shared with all clients to maintain a consistent encoding layout and enable subsequent aggregate recovery.

H.2 Client-side Computation and Submission

At the beginning of round t, each participating client $\mathcal{C}_u$, indexed by $u \in \mathcal{P}^{(t)}$, performs local EP training. For $t = 1$, the client starts from the initial score vector $\theta^{\boldsymbol{s}}$. For $t > 1$, it uses the global ranking $\boldsymbol{r}_g^{(t-1)}$ obtained in the preceding round to reorder the initial scores before local training. After ψ local EP iterations on its private dataset $\mathcal{D}_u$, client $\mathcal{C}_u$ obtains its local ranking vector $\boldsymbol{r}_u^{(t)}$.

Based on $\boldsymbol{r}_u^{(t)}$, the client derives the corresponding membership vector $\boldsymbol{\mu}_u^{(t)}$ by assigning 1 to the K highest-ranked edges and 0 to all others. It then combines the ranking and membership information into a joint integer vector $\boldsymbol{h}_u^{(t)} = [h_{u,0}^{(t)}, \ldots, h_{u,d-1}^{(t)}]$, where $h_{u,i}^{(t)} = \lambda r_{u,i}^{(t)} + \mu_{u,i}^{(t)}$, $i = 0, \ldots, d-1$. Since $\lambda > U$, the aggregated ranking and membership information can later be uniquely separated from their joint representation.

The joint vector $\boldsymbol{h}_u^{(t)}$ is then partitioned into M sub-vectors $\boldsymbol{h}_{u,1}^{(t)}, \ldots, \boldsymbol{h}_{u,M}^{(t)}$, such that $\boldsymbol{h}_u^{(t)} = \boldsymbol{h}_{u,1}^{(t)} \| \cdots \| \boldsymbol{h}_{u,M}^{(t)}$, where $\|$ denotes vector concatenation. If necessary, the final sub-vector is zero-padded for encoding, while its valid length is retained so that the padded coordinates are excluded from subsequent verification and reconstruction. For each sub-vector $\boldsymbol{h}_{u,m}^{(t)}, m = 1, \ldots, M$, client $\mathcal{C}_u$ applies $\mathsf{SEncode}(\boldsymbol{h}_{u,m}^{(t)}, v_m^{(t)})$ to produce the encoded polynomial $p_{\boldsymbol{h}_{u,m}}^{(t)}(x)$. The common round-specific shift $v_m^{(t)}$ preserves a consistent positional structure across participating clients for subsequent homomorphic aggregation. Client $\mathcal{C}_u$ then encrypts each encoded polynomial under the public key pk using RLWE-AHE, yielding $ct(\boldsymbol{h}_{u,m}^{(t)})$. Finally, it sends the ciphertext set $\{ct(\boldsymbol{h}_{u,m}^{(t)})\}_{m=1}^{M}$ to $\mathcal{S}_1$.

H.3 Server-side Verification and Secure Aggregation

Server $\mathcal{S}_1$ retains the submitted ciphertexts for aggregation. To prepare them for verification, it applies two transformations:

- **Intra-group rotation.** For each ciphertext $ct(\boldsymbol{h}_{u,m}^{(t)})$, $\mathcal{S}_1$ applies $\mathbf{Rot}(ct(\boldsymbol{h}_{u,m}^{(t)}), v_m^{(t)})$, obtaining $ct(\widetilde{\boldsymbol{h}}_{u,m}^{(t)})$, where $\widetilde{\boldsymbol{h}}_{u,m}^{(t)}$ denotes the cyclically rotated version of $\boldsymbol{h}_{u,m}^{(t)}$. Together with the corresponding $\mathsf{SEncode}$ operation, this changes the positions of the elements within each sub-vector while preserving their values.
- **Inter-group shuffling.** For each participant $u \in \mathcal{P}^{(t)}$, $\mathcal{S}_1$ independently samples a random permutation $\pi_u^{(t)}$ over $\{1, \ldots, M\}$ and reorders the rotated ciphertexts accordingly. The resulting obfuscated ciphertext sequence is

$$\left\{ct(\widetilde{\boldsymbol{h}}^{(t)}_{u,\pi_u^{(t)}(1)}), \ldots, ct(\widetilde{\boldsymbol{h}}^{(t)}_{u,\pi_u^{(t)}(M)})\right\}.$$

  Since $\pi_u^{(t)}$ is independently generated for each client, different client submissions are presented to $\mathcal{S}_2$ under different sub-vector orders.

Server $\mathcal{S}_1$ then forwards the obfuscated ciphertexts to $\mathcal{S}_2$ for validity verification. It also derives validity masks from the public block lengths and applies the same cyclic shifts and block permutation. These masks allow $\mathcal{S}_2$ to check that padding remains zero and exclude it from the rank and membership checks. For each client, $\mathcal{S}_2$ decrypts the received ciphertexts and applies $\mathsf{CDecode}$ to obtain the corresponding sub-vectors. It then concatenates these sub-vectors in their received order to reconstruct the position-obfuscated joint vector

$$\widetilde{\boldsymbol{h}}_u^{(t)} = \widetilde{\boldsymbol{h}}^{(t)}_{u,\pi_u^{(t)}(1)} \| \cdots \| \widetilde{\boldsymbol{h}}^{(t)}_{u,\pi_u^{(t)}(M)}.$$

For each valid (non-padding) coordinate, $\mathcal{S}_2$ extracts the membership bit as $\widetilde{\mu}_{u,i}^{(t)} = \widetilde{h}_{u,i}^{(t)} \bmod \lambda$ and the rank value as $\widetilde{r}_{u,i}^{(t)} = (\widetilde{h}_{u,i}^{(t)} - \widetilde{\mu}_{u,i}^{(t)})/\lambda$. Because the transformations preserve entry values, $\mathcal{S}_2$ can check the ranking and membership conditions in the received order. A submission is accepted only if $\widetilde{\boldsymbol{r}}_u^{(t)}$ forms a permutation of $\{0, \ldots, d-1\}$ and its membership vector is consistent with the top-K entries, i.e., $\widetilde{\mu}_{u,i}^{(t)} = 1$ if $\widetilde{r}_{u,i}^{(t)} \geq d - K$, and $\widetilde{\mu}_{u,i}^{(t)} = 0$ otherwise.

Let $\mathcal{Q}^{(t)} \subseteq \mathcal{P}^{(t)}$ denote the indices of clients whose submissions pass verification. If no client qualifies, all clients retain the preceding selected set and global ranking for this round. Server $\mathcal{S}_2$ returns $\mathcal{Q}^{(t)}$ to $\mathcal{S}_1$, which then homomorphically aggregates the original, non-obfuscated ciphertexts of these clients. For each sub-vector index $m = 1, \ldots, M$, $\mathcal{S}_1$ computes $ct(\boldsymbol{h}_m^{(t)}) = \bigoplus_{u \in \mathcal{Q}^{(t)}} ct(\boldsymbol{h}_{u,m}^{(t)})$. The

corresponding aggregated sub-vectors satisfy $\boldsymbol{h}_m^{(t)} = \sum_{u \in \mathcal{Q}^{(t)}} \boldsymbol{h}_{u,m}^{(t)}$. Concatenating them yields the aggregate joint vector

$$\boldsymbol{h}^{(t)} = \boldsymbol{h}_1^{(t)} \| \cdots \| \boldsymbol{h}_M^{(t)} = \sum_{u \in \mathcal{Q}^{(t)}} \boldsymbol{h}_u^{(t)} = \lambda \boldsymbol{s}^{(t)} + \boldsymbol{c}^{(t)},$$

where $\boldsymbol{s}^{(t)}$ and $\boldsymbol{c}^{(t)}$ are the aggregated Borda-score and membership-count vectors required by CMGRA, respectively.

Before decryption, $\mathcal{S}_1$ masks the aggregate plaintexts using fresh independent randomness for every block and round. For each ciphertext $ct(\boldsymbol{h}_m^{(t)})$, it samples $\rho_m^{(t)} \xleftarrow{\$} \mathcal{R}_z$ and constructs the masked ciphertext as $\widehat{ct} = \mathbf{PAdd}\big(ct(\boldsymbol{h}_m^{(t)}), \rho_m^{(t)}\big)$. Server $\mathcal{S}_2$ then decrypts the masked ciphertext to obtain

$$\widehat{p}_{\boldsymbol{h}_m}^{(t)} = \mathbf{Dec}(sk, \widehat{ct}) = p_{\boldsymbol{h}_m}^{(t)} + \rho_m^{(t)} \bmod z.$$

It sends $\widehat{p}_{\boldsymbol{h}_m}^{(t)}$ to all clients, while $\mathcal{S}_1$ separately distributes the corresponding mask $\rho_m^{(t)}$ to them.

H.4 Aggregate Recovery and CMGRA

Upon receiving the masked plaintexts and the corresponding masks, each client $\mathcal{C}_u$ with $u \in \mathcal{C}$ recovers the aggregate information as follows. For each sub-vector index $m = 1, \ldots, M$, the client removes $\rho_m^{(t)}$ from $\widehat{p}_{\boldsymbol{h}_m}^{(t)}$ to obtain the encoded aggregate polynomial

$$p_{\boldsymbol{h}_m}^{(t)} = \widehat{p}_{\boldsymbol{h}_m}^{(t)} - \rho_m^{(t)} \bmod z.$$

It then applies SDecode with the corresponding shift $v_m^{(t)}$ to recover $\boldsymbol{h}_m^{(t)}$. Concatenating the recovered sub-vectors yields $\boldsymbol{h}^{(t)}$.

Since each coordinate of $\boldsymbol{h}^{(t)}$ satisfies $h_i^{(t)} = \lambda s_i^{(t)} + c_i^{(t)}$ and $0 \le c_i^{(t)} \le U < \lambda$, the membership count and Borda score can be losslessly separated as

$$c_i^{(t)} = h_i^{(t)} \bmod \lambda, \qquad s_i^{(t)} = \frac{h_i^{(t)} - c_i^{(t)}}{\lambda}.$$

Each client thus recovers the aggregated membership-count vector $\boldsymbol{c}^{(t)}$ and Borda-score vector $\boldsymbol{s}^{(t)}$.

Using $\boldsymbol{c}^{(t)}$ and $\boldsymbol{s}^{(t)}$, each client $\mathcal{C}_u$ with $u \in \mathcal{C}$ executes CMGRA in plaintext according to Section 2, obtaining the selected edge set $S^{(t)}$ and global ranking vector $\boldsymbol{r}_g^{(t)}$. Since all honest clients recover the same aggregate statistics and CMGRA employs deterministic tie-breaking, they obtain the same result, which is maintained by all clients and used to guide local training in the next round. The position-obfuscation leakage and component security guarantees are analyzed in Appendix F.

I Experimental Details and Additional Results

This appendix collects the experimental configuration, supplementary HE results, and attack settings.

I.1 Experimental Configuration and Model

This subsection describes the released configuration for reproducing the learning experiments and the model dimensions used to calculate cryptographic costs. The configuration contains 168 runs, including five methods without attacks on each of three datasets, five methods under Label Flipping, Gradient Ascent, and Pixel Backdoor at three nominal malicious-client fractions, and FRL/CMGRA under VEM at the same three fractions.

Data and optimization. Each training set is partitioned among 1,000 clients using the source-compatible Dirichlet partition with $\alpha = 1.0$ and partition seed 0. Each round samples 25 clients without replacement, and each run contains 500 rounds. The released manifest uses run seed 0 and shares the same partition across methods. Table 8 summarizes the dataset-specific settings. Within these configurations, rank methods optimize Edge-Popup scores over fixed weights, whereas gradient methods optimize the corresponding ordinary weight tensors. All methods use SGD, cross-entropy loss, training batch size 8, weight decay 10^{-4}, and round-wise learning-rate decay 0.999. FRL and CMGRA use initial learning rate 0.4, momentum 0.99, and cosine annealing over each local training phase. FedAvg, BCPBFL, and RVPFL use momentum 0.9 without local cosine annealing, with initial learning rates of 0.001 on MNIST and 0.01 on SVHN/CIFAR10. The BCPBFL configuration uses a 200-example root set and server learning rate 0.1.

Table 8: Released dataset-specific reproduction settings. The keep ratio applies to rank-based models.

Setting	MNIST	SVHN	CIFAR10
Model	Conv2	Conv8	ResNet18
Local epochs	1	5	5
Keep ratio	0.2	0.5	0.5
Evaluation batch size	128	512	512
Rank fixed-weight initialization	Signed constant	Kaiming uniform	Signed constant
CMGRA initialization seed	2026	0	2026
Ranked tensors	4	11	21
Ranked coordinates	1,682,496	5,275,840	11,164,352
Blocks at $N = 8192$	207	646	1,366

MNIST preprocessing consists of tensor conversion and normalization. For SVHN and CIFAR10, training augmentation uses random 32×32 crops with padding 4, while CIFAR10 additionally uses random horizontal flips. Evaluation applies only tensor conversion and normalization. The channel means and standard deviations are

$$\begin{aligned} \text{MNIST}: &\quad (0.1307)/(0.3081), \\ \text{SVHN}: &\quad (0.4377, 0.4438, 0.4728)/(0.1980, 0.2010, 0.1970), \\ \text{CIFAR10}: &\quad (0.4914, 0.4822, 0.4465)/(0.2470, 0.2435, 0.2616). \end{aligned}$$

SVHN uses the standard training and test splits, without its extra split.

Architectures and counted coordinates. Conv2 uses two 3×3 convolutions with output widths 64 and 128, each followed by ReLU and 2×2 max pooling. The resulting features pass through $6272 \to 256 \to 10$ classification layers implemented as 1×1 convolutions. Conv8 instead uses four pairs of 3×3 convolutions with widths $(64, 64)$, $(128, 128)$, $(256, 256)$, and $(512, 512)$, with pooling after each pair, followed by a $2048 \to 256 \to 256 \to 10$ head. Its spatial convolutions use BatchNorm. The CIFAR ResNet18 uses a 3×3 stem without the ImageNet max-pooling stage, four stages with two basic residual blocks each and widths 64, 128, 256, and 512, global average pooling, and a $512 \to 10$ head. All convolutions omit biases, and BatchNorm has neither affine parameters nor tracked running statistics. Accordingly, the counts below cover communicated weight or score coordinates. Each rank model also stores a fixed weight for each score, but this storage does not increase the reported ranked dimension.

Table 9: Layerwise dimensions of Conv2 and Conv8. Each head layer is implemented as a 1×1 convolution.

Model	Layer	Coordinates d_ℓ	$\lceil d_\ell/8192 \rceil$
Conv2	Conv1	576	1
	Conv2	73,728	9
	Head1	1,605,632	196
	Head2	2,560	1
Conv8	Conv1	1,728	1
	Conv2	36,864	5
	Conv3	73,728	9
	Conv4	147,456	18
	Conv5	294,912	36
	Conv6	589,824	72
	Conv7	1,179,648	144
	Conv8	2,359,296	288
	Head1	524,288	64
	Head2	65,536	8
	Head3	2,560	1

Table 10: Layerwise dimensions of the CIFAR ResNet18. Stage/block indices refer to the four residual stages and their two basic blocks.

Layer	Coordinates d_ℓ	$\lceil d_\ell/8192 \rceil$
Stem	1,728	1
Stage1/Block1/Conv1	36,864	5
Stage1/Block1/Conv2	36,864	5
Stage1/Block2/Conv1	36,864	5
Stage1/Block2/Conv2	36,864	5
Stage2/Block1/Conv1	73,728	9
Stage2/Block1/Conv2	147,456	18
Stage2/Block1/Shortcut	8,192	1
Stage2/Block2/Conv1	147,456	18
Stage2/Block2/Conv2	147,456	18
Stage3/Block1/Conv1	294,912	36
Stage3/Block1/Conv2	589,824	72
Stage3/Block1/Shortcut	32,768	4
Stage3/Block2/Conv1	589,824	72
Stage3/Block2/Conv2	589,824	72
Stage4/Block1/Conv1	1,179,648	144
Stage4/Block1/Conv2	2,359,296	288
Stage4/Block1/Shortcut	131,072	16
Stage4/Block2/Conv1	2,359,296	288
Stage4/Block2/Conv2	2,359,296	288
Head	5,120	1

Under layerwise packing, the total ciphertext-block count is $\sum_\ell \lceil d_\ell/N \rceil$, including padding in each layer's last block. At $N = 8192$, all three models contain layers with a single block: the first and last layers of Conv2 and Conv8, and the ResNet18 stem, first stage-2 shortcut, and classification head. Any privacy statement for layerwise processing must therefore cover $M = 1$ explicitly.

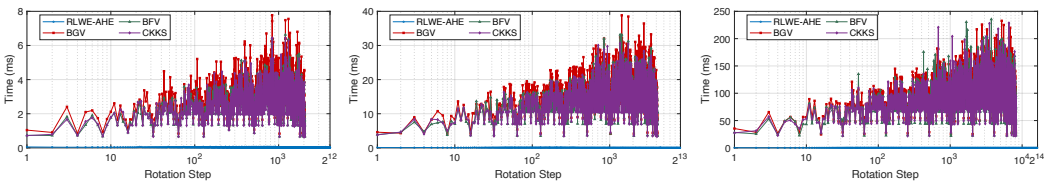
Nominal malicious fractions and actual counts. The released configuration identifies attacks by nominal fractions $f \in \{0.1, 0.2, 0.3\}$. For Label Flipping, Gradient Ascent, and Pixel Backdoor, round t uses $m_t = \lfloor 25tf \rfloor - \lfloor 25(t-1)f \rfloor$ malicious uploads, so 10% and 30% alternate between 2/3 and 7/8 clients, respectively, while 20% uses five clients per round. The VEM adapter preserves the source implementation's floor rule $m = \lfloor 25f \rfloor$: its three nominal settings therefore use 2, 5, and 7 malicious uploads per round, corresponding to effective fractions 8%, 20%, and 28%. The simulator supplies CMGRA with the realized malicious-client count as its adversarial budget. Consequently, the reported configuration does not test budget misspecification.

Implementation and benchmark platform. The reported benchmarks use an NVIDIA RTX 4090 GPU, an Intel Xeon Platinum 8481C CPU, and 90 GB of RAM, running 64-bit Ubuntu 22.04.1 LTS. The cryptographic implementation uses C++ and Microsoft SEAL, with the coefficient encoding and decoding routines integrated into the library. SEAL-Python connects this implementation to the Python experiment code. The learning simulator uses PyTorch and simulated client–server interactions in a LAN setting. The primitive benchmark results are averaged over 1,000 runs. These details specify the platform and implementation used for the reported experiments, complementing the model dimensions and reproduction configuration in Appendix I.1.

Comparison scope and communication accounting. The standard SIMD-BFV comparator uses SEAL's native batching encoder, Galois-key-based slot rotations, and default coefficient moduli. The Paillier baseline uses a 2048-bit modulus. The reported RLWE-AHE configuration combines sign-aware coefficient encoding, monomial-based coefficient shifts, and a 60-bit ciphertext modulus. The comparison therefore evaluates the combined choices of representation, supported operations, and parameters. The reported speed and size differences apply to these configurations. Isolating a gain from the encryption primitive would require an additional BFV comparison with matched encoding, operations, and parameters suited to the workload. Inter-server communication is calculated from the ciphertext collections transmitted by the protocol: U verification collections and one masked aggregate collection give $(U+1)B_{\mathrm{ct}}$, where B_{ct} is the serialized volume of one complete packed client submission. The small qualified-client identifier list is omitted from this accounting.

I.2 Rotation latency across step sizes

Figure 2 compares the latency of RLWE-AHE's prescribed monomial shifts and the baselines' slot rotations across different step sizes and polynomial degrees.



(a) $N = 4096$ (b) $N = 8192$ (c) $N = 16384$

Figure 2: Runtime evaluation of ciphertext rotation under varying step sizes.

I.3 Poisoning Attack Settings

We evaluate CertiRank against four representative poisoning attacks: Label Flipping, Gradient Ascent, Pixel Backdoor, and VEM. The descriptions below specify their objectives and how malicious clients construct their submissions.

Label Flipping. Label Flipping (Tolpegin et al., 2020) corrupts the correspondence between training samples and their labels. For a classification task with f classes, each malicious client replaces the original label $l \in \{0, \ldots, f-1\}$ with $l' = f - l - 1$ during local training. Training on these labels biases the learned class mapping and, consequently, the uploaded update or ranking. Honest clients retain the original labels throughout training. We assess the resulting effect using test accuracy after 500 global rounds.

Gradient Ascent. The Gradient Ascent attack (Kurakin et al., 2017) reverses the direction of local updates. In the released implementation for gradient-based schemes, malicious clients negate their normally trained local model deltas before aggregation. For rank-based schemes, they instead train Edge-Popup scores using negative cross-entropy and convert the resulting scores to valid rankings. Appendix I.1 details the implementation and its ranking conventions.

Pixel Backdoor. Pixel Backdoor (Xie et al., 2020) evaluates resistance to targeted model poisoning. A fixed F-shaped trigger is applied to the same nine originally non-target examples, which are relabeled as class 2 and added to every malicious client's training data. Each client trains on this

poisoned dataset and submits its update or ranking through the corresponding aggregation scheme. The attack aims to preserve predictions on clean inputs while causing triggered samples to be classified as the target class. We therefore report clean test accuracy (ACC) alongside attack success rate (ASR). ASR is the fraction of triggered test samples with original labels other than class 2 that are classified as class 2. Robustness requires preserving high ACC while limiting ASR.

VEM. VEM (Gong et al., 2025) targets edges near the top-K selection boundary in rank-based learning. Each malicious client first trains normally to obtain a benign ranking, then identifies vulnerable edges whose ranks are most likely to affect the selected subnetwork. The attacker optimizes their ordering using a differentiable Gumbel–Sinkhorn relaxation and projects the result to a valid permutation with the Hungarian algorithm. Inserting these manipulated ranks into the original ranking produces a malicious submission that preserves the permutation constraint.