USENIX

THE ADVANCED COMPUTING
SYSTEMS ASSOCIATION

Aion: Robust and Efficient Multi-Round Single-Mask Secure Aggregation Against Malicious Participants

Yizhong Liu, Zixiao Jia, Zian Jin, Xiao Chen, Song Bian,
Runhua Xu, Dawei Li, and Jianwei Liu, *Beihang University;*
Yuan Lu, *Institute of Software, Chinese Academy of Sciences*

https://www.usenix.org/conference/usenixsecurity25/presentation/liu-yizhong

This paper is included in the Proceedings of the
34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Proceedings of the
34th USENIX Security Symposium is sponsored by USENIX.

Aion: Robust and Efficient Multi-Round Single-Mask Secure Aggregation Against Malicious Participants

Yizhong Liu[†], Zixiao Jia[†], Zian Jin[†], Xiao Chen[†], Song Bian[†],
Runhua Xu[ᛤ], Dawei Li[†*], Jianwei Liu[†*], Yuan Lu[ᛋ]

[†]*School of Cyber Science and Technology, Beihang University,* [ᛤ]*School of Computer Science and Technology, Beihang University,*
[ᛋ]*Institute of Software, Chinese Academy of Sciences*

Email: {liuyizhong, jiazixiao, jinzian, chenxiao, sbian, runhua, lidawei, liujianwei}@buaa.edu.cn, luyuan@iscas.ac.cn

Abstract

Federated learning enables multiple clients to collaboratively train a model without sharing their data. Secure aggregation (SA) allows for the computation of aggregated models while protecting the private models of clients from disclosure, making it highly promising in large-scale real-world applications. Masking-based SA stands out due to its higher efficiency and accuracy. However, existing masking-based SA methods face issues such as high overhead, loss of correctness under poisoning attacks, and inability to tolerate malicious participants. In this paper, we propose Aion, a robust and efficient multi-round single-mask SA tolerating malicious participants. We introduce an aggregatable SA pattern in which each client only adds a single mask and performs only one secret sharing operation, while each aggregator only reconstructs a total secret or mask. Compared to Flamingo (S&P'23), this reduces the secret sharing times from rq to q (r for training round number and q for client number per round) and lowers n aggregators' mask reconstruction overhead from $O(n^2q)$ to $O(n)$. Furthermore, we design a lightweight evolving input validation mechanism that efficiently filters out malicious client models by dynamically updating the mask range and overall bound, thereby improving model accuracy. Besides, we present robustness enhancements that tolerate malicious clients and aggregators. These constructions support aggregator share verification and asynchronous client model utilization. Finally, experiments demonstrate that Aion outperforms Flamingo by a factor of 563.64 in speed while achieving a 97.98% reduction in message overhead with 4096 clients and 8 aggregators, effectively defending against poisoning attacks with low overhead.

1 Introduction

Federated learning [1] (FL) is a distributed learning approach where clients train models on local data without sharing it. Instead, they usually send model parameters to a server, which aggregates them to create a global model. *Secure aggregation (SA)* is a privacy-preserving method to compute the aggregated model while protecting the private models and data of clients from disclosure [2]. SA presents promising potential in large-scale real-world applications for predictive typing [3] and social user data analysis [4].

The main goals of SA are to protect the client model while maintaining high accuracy in the aggregated model, prevent malicious participants from compromising the aggregation process [5, 6], and accommodate a large-scale and dynamic group of participants [7, 8]. Currently, the methods for achieving SA include differential privacy [9], secure multi-party computation (MPC) [10], homomorphic encryption (HE) [11], and masking-based schemes [2].

Mask technique: Realizing efficient and accurate SA. Mask techniques have emerged as a promising tool to achieve high *accuracy* and *efficiency* in SA, as they effectively eliminate the privacy-preserving noise introduced by clients. Specifically, *pairwise masking* proposed in SecAgg (CCS'17) [2] and studied in the state-of-the-art schemes SecAgg+ (CCS'19) [12], Flamingo (S&P'23) [14], ACORN (Security'23) [13], combine *pairwise masks* and *individual masks* to protect local models. Each client computes $\vec{y}_i = \vec{x}_i - \sum_{j\in C, j<i} \mathrm{PRG}(h_{i,j}) + \sum_{j\in C, j>i} \mathrm{PRG}(h_{i,j}) + \mathrm{PRG}(m_i)$ where $\vec{x}_i$ is the local model and C is a client set. Each client first performs a Diffie-Hellman (DH) key exchange with other clients to generate pairwise mask seeds $h_{i,j}$, which are then used as input to compute multiple pairwise masks via a pseudo-random generator (PRG). The pairwise masks from multiple related clients cancel each other out, resulting in an accurate aggregated model. Specifically, to deal with offline clients, each client must share the pairwise masks in advance with multiple *aggregators* (specialized clients or servers) via secret sharing (SS) [17] or threshold encryption [18]. This allows for the threshold reconstruction of the offline clients' pairwise masks. Moreover, if a client sends its data $\vec{y}_i$ after its pairwise masks are reconstructed due to asynchrony [19], the client model can still be recovered. Therefore, each client must initially apply an individual mask $\mathrm{PRG}(m_i)$ generated from its

*Corresponding authors: Dawei Li and Jianwei Liu.

Table 1: Comparison of secure aggregation schemes.

Framework	Mask Technology Overhead			Input Validation		Participant Tolerance		
	Secret Sharing Times for Clients	Mask Number per Model	Comm. Complexity for Aggregators	Input Validation Support	ZK-free	Malicious Client	Malicious Aggregator	Async. Support
clear$^{\diamond}$	0	0	—	✗	—	✗	✗	✗
SecAgg [2]	$2rq^{\natural}$	q (pariwise+individual)	$O(q^3)$	✗	—	✗	✗	✗
SecAgg+ [12]	$2rq$	u (pariwise+individual)	$O(u^3)$	✗	—	✗$^{\aleph}$	✓	✗
ACORN [13]	$2rq$	u (pariwise+individual)	$O(u^3)$	✓	✗	✓	✓	✗
Flamingo [14]	$rq^{\flat}$	u (pariwise+individual)	$O(n^2q)^{\star}$	✗	—	✗	✓	✗
ELSA [15]	rq	— (Boolean SS + OT)	—	✓	✓†	✓	✓	✗
Mario [16]	—	— (Threshold HE)	$O(n^2)$	✓	✗	✓	✓	✓
Aion	q	1 (single)	$O(n)$	✓	✓	✓	✓	✓

$^{\diamond}$ "clear" indicates the normal federated learning without any privacy protection measures.
$^{\natural}$ r denotes the training round (usually 500 to 10000). q denotes the client number in each round (usually 50 to 5000). u denotes the client's neighborhood size ($u \leq q$).
$^{\aleph}$"✗" indicates a semi-honest client, who tries to infer private information but follows the protocol and avoids active attacks, like poisoning attacks.
$^{\flat}$ Flamingo reduces the times of secret sharing by introducing threshold decryption for sharing and reconstructing pairwise masks.
* n represents the number of aggregators. Please refer to Appendix D for detailed complexity analysis.
† In ELSA, the system use oblivious transfer to ensure the input validation.

own secret m_i to its data. Similarly, the individual mask must also be shared in advance to enable its reconstruction.

Existing SA schemes (please refer to Appendix C for details) primarily focus on the mask technology overhead, input validation, and participant tolerance (Table 1) to ensure security and efficiency.

Mask technology overhead. In each round, clients must first share their pairwise and individual mask seeds with multiple aggregators to facilitate reconstruction during aggregation. For *secret sharing times*, each client needs to perform $2rq$ or rq secret sharing [2, 12–14] (q for the client number per round and r for the training round number). Each secret sharing typically incurs substantial communication and computational overhead for clients. Regarding the *mask number per model*, each client needs to compute q or u masks, where u denotes the size of the subsets (termed neighborhoods) formed among q clients. For aggregators, considering *communication complexity*, they must recover the individual and pairwise masks one by one for q clients in each round. This requires each aggregator to broadcast their shares and reconstruct masks, leading to a communication cost of $O(n^2q)$.

Input validation. The aggregators are unable to know the exact model parameters due to added masks, making it difficult to prevent poisoning attacks [20–22] from *malicious clients*. Without any defenses via *input validation*, even a single incorrect gradient, such as those with a high norm, can bias the entire global model, resulting in a loss of model correctness and robustness. Recent prior work [13, 15, 16] considers the use of L_2 defenses with zero-knowledge proofs (ZK) [23] or oblivious transfer (OT) [24] to filter out malicious gradients. Additional studies [25, 26] demonstrate that this method can effectively defend against a variety of poisoning attacks.

Participant tolerance. Malicious participants include *malicious clients* and *aggregators*. Malicious clients may not only submit poisoned models but also incorrectly perform secret sharing. Meanwhile, a malicious aggregator could send incorrect reconstruction information, such as invalid client secret shares. Besides, considering *asynchronous support* [16], the utilization of late client models can mitigate the negative effects of network latency, increasing model accuracy.

1.1 Our Contributions

In this paper, we propose Aion[1], a robust and efficient multi-round single-mask SA against malicious participants. The main contributions are as follows.

Aggregatable multi-round single-mask SA pattern with high efficiency. For initialization, each client performs a one-time secret sharing. In each aggregation phase, each client computes a single mask and uploads the masked models. Then each aggregator first aggregates multiple clients' secret/mask shares locally and reconstructs the total secret/mask. Besides, we design a client set concealed sortition method to prevent adversaries from knowing the client set in advance. Notably, for each client, only a single secret sharing is conducted, reducing the overall secret sharing time from rq in Flamingo to q and supporting multiple rounds of secret usage. Moreover, each client only needs to add a single mask, compared to Flamingo's u masks, reducing the overhead for clients. Furthermore, each aggregator is required to reconstruct only one aggregated mask, reducing the communication complexity for the aggregator from Flamingo's $O(n^2q)$ to $O(n)$. The communication cost is irrelevant to the client number, making it well-suited for large-scale FL.

Lightweight evolving input validation mechanism. We design a pluggable lightweight evolving input validation module for the single-mask SA to defend against malicious clients. This mechanism filters the masked models utilizing an improved norm defense mechanism. Each client is required to add a single mask that falls within a specified range. This allows the aggregator to directly validate whether the input, after the addition of the mask, remains within the L_2 norm bound. Furthermore, as the training rounds progress, the mask

[1] Aion is the god of eternity in Greek mythology, symbolizing the sustainability of our secret for the mask.

range and L_2 norm bound evolve to accommodate the updated model. In particular, our evolving input validation is ZK-free and does not require clients to perform additional communication or computation, making it lightweight and efficient. Additionally, the evolving adjustment mechanism enhances the accuracy of identifying poisoned gradients.

Robustness enhancements tolerating malicious participants. To tolerate malicious aggregators, we design an aggregated secret share verification mechanism using cryptographic commitments to validate each aggregator's shares. Besides, we introduce a Byzantine fault tolerant (BFT) consensus to ensure consistency on the model. Moreover, we develop an asynchronous FL module allowing for the protection of late-arrival client models while fully utilizing their inputs to improve model accuracy.

Experiments and Analysis. Our experiments evaluate execution efficiency, model accuracy, and resilience to poisoning attacks. In terms of efficiency, we compare Aion with Flamingo [14], ACORN [13], SecAgg+ [12], SecAgg [2], Mario [16], and ELSA [15], and the results show Aion has significant advantages in both runtime and communication overhead. We also evaluate the evolving input validation mechanism, showing that Aion effectively mitigates poisoning attacks, leading to higher model accuracy. In experiments with multiple datasets, Aion demonstrates complete resistance to attacks even under a 50% poison ratio. Importantly, the input validation module introduces subtle overhead while maintaining comparable accuracy to the unmodified model.

1.2 High-Level Technical Overview

Reducing mask number to one and enabling multi-round secret with optimized overhead. Pairwise masks are mutually canceled among multiple clients [12, 14]. That is, in a client set C, the global gradient $\vec{x}$ of all clients can be computed by $\vec{x} = \sum_{i\in C}\left(\vec{x}_i - \sum_{j\in C, j<i}\mathrm{PRG}(h_{i,j}) + \sum_{j\in C, j>i}\mathrm{PRG}(h_{i,j})\right)$. To handle offline clients, pairwise masks need to be recovered to achieve cancellation. To prevent model leakage due to delayed inputs from offline clients, an additional individual mask is added. In each round, the individual mask for each online client and the pairwise mask for offline clients must be *recovered individually*. We observe that this is unnecessary since the goal is to recover the mask sum of all clients. Specifically, instead of recovering each client's secret or mask individually, we recover the *sum of secrets* of q clients: $M = \sum m_i$ where m_i is the secret shared by the i-th client. Locally, each aggregator computes the q secret shares sum: $[M]^j = \sum [m_i]^j$, where $[m_i]^j$ represents the secret share of m_i obtained by the j-th aggregator. The total secret M can then be reconstructed through interpolation, $M = \sum \lambda_j [M]^j$, where λ_j is the Lagrange coefficient. Meanwhile, mask computation requires homomorphic properties, and thus Homomorphic PRG (HPRG) can be employed. Each client computes $\mathrm{HPRG}(m_i)$ and the aggregator calculates $\mathrm{HPRG}(M)$ after recovering M. The *sum of mask* is obtained since $\mathrm{HPRG}(M) = \mathrm{HPRG}(\sum m_i) = \sum \mathrm{HPRG}(m_i)$ holds. In this way, only the aggregated mask $\mathrm{HPRG}(M)$ is recovered without revealing individual client masks $\mathrm{HPRG}(m_i)$, preventing model leakage even if a client goes offline. However, using the same mask $\mathrm{HPRG}(m_i)$ for each round may expose model differences across rounds. To address this, a distinct mask should be used in each round. We apply a key Homomorphic PRF (HPRF) [27], using the secret m as the key and the round r as the seed. Thus, each round's mask becomes $\mathrm{HPRF}(m,r)$, and by key homomorphism, we derive that $\sum \mathrm{HPRF}(m_i,r) = \mathrm{HPRF}(\sum m_i, r) = \mathrm{HPRF}(M,r)$. This ensures that each round's client mask is unique and private, allowing secrets to be reused across rounds. Each client only needs to perform SS once, reducing SS times from rq to q.

Furthermore, considering that the set of clients chosen across rounds may overlap, which could risk revealing client secrets, we design a *client concealed sortition* algorithm. Each client calculates a Verifiable Random Function (VRF) output based on their secret m_i to determine their participation in a given round. Consequently, the client set in each round is randomly assigned, and client sets across rounds remain independent. Clients attach their VRF output and proof when uploading inputs. This prevents an adversary from controlling honest nodes even under instant corruption.

Optionally, we propose reconstructing the mask as an alternative to reconstructing the secret. Each aggregator reveals $\mathrm{HPRF}([M]^j, r)$, allowing the reconstruction of the aggregated mask $\mathrm{HPRF}(M,r) = \mathrm{HPRF}\left(\sum \lambda_j [M]^j, r\right) = \sum \lambda_j \mathrm{HPRF}([M]^j, r)$. This avoids revealing the total secret M, permits the selection of client sets with overlap, and enables continual use of the secret for mask generation.

Designing a specialized input validation mechanism for single-mask SA. The L_2 norm, by applying a specific bound, filters out abnormal models or gradients and has been shown in numerous studies [28–31] to prevent various attacks [32–34]. ACORN [13], ELSA [15], and Mario [16] use the L_2 norm in masking-based SA to perform input validation and defend against malicious clients. We observe that the L_2 norm is directly applied to constrain the client model $\vec{x}$. However, due to the multiple added masks, ZK proofs are required to demonstrate that the masks are correctly added to the model and that $\vec{x}$ meets the L_2 bound, increasing overhead for clients and aggregators. In Aion, each client applies a single mask, making the mask range controllable. We limit each client's mask to a defined range (less than a specified percentage of the model) and enforce an overall L_2 bound on the masked input $\vec{x} + \mathrm{HPRF}(m,r)$. For honest clients, the controlled mask range and L_2 bound ensure the input stays within valid limits. For malicious clients, uploading an incorrect model or mask will exceed the L_2 bound or fail to achieve the attack.

Further, since aggregators in Aion can compute the total mask $\mathrm{HPRF}(M,r)$ through reconstruction, and as training progresses, the gradients tend to decrease in magnitude, a fixed L_2 bound and mask range become less dynamic and

accurate. Hence, we introduce an *evolving input validation* mechanism where both the mask range and overall L_2 bound adapt dynamically, linked to each round's aggregated mask, thereby achieving more precise input validation.

Dealing with malicious participants and asynchronous clients. If a malicious aggregator sends incorrect aggregated shares, it can lead to an invalid total mask. So we leverage the homomorphism of verifiable secret sharing (VSS)'s commitments to verify aggregated shares. Each aggregated share can be validated using its aggregated commitment. Besides, to prevent a malicious aggregator from sending inconsistent models to different clients, we use BFT consensus for consistency.

For asynchronous clients, existing schemes [12–14] recover pairwise masks for an offline client. However, when asynchronous clients' inputs eventually arrive, individual mask recovery is not possible since their models will be revealed. In Aion, offline clients are excluded from the online client set, without recovering any single client's mask. This allows the masks of multiple asynchronous clients to be aggregated and recovered, effectively utilizing inputs from all clients.

2 Building Blocks

2.1 Verifiable Secret Sharing

We introduce a VSS scheme [35] using Feldman commitments. A dealer distributes a secret m among n nodes (denoted by P), such that at least t honest shares can reconstruct the secret m, with each node able to verify share correctness. The scheme consists of three functions.

Share: $\mathsf{Share}(m,t,n) \rightarrow (\{[m]^j\}_{j\in[P]}, Com)$. The dealer selects a random polynomial $f(x)$ of degree $t-1$ over $\mathbb{Z}_p$ with the secret m as the constant term: $f(x) = a_0 + a_1 x + \cdots + a_{t-1}x^{t-1} \mod p$, $a_0 = m$. For each P_j, the share is computed as $[m]^j = f(x_j) \mod p$, where x_j is publicly predefined and usually $x_j = j$. To enable verification, the dealer computes commitments $Com_i = g^{a_i} \mod p$ for each coefficient a_i. For each participant P_j, the dealer sends $([m]^j, Com)$ to it, where $Com = (Com_0, Com_1, \ldots, Com_{t-1})$. Note that $[m]^j$ must be sent via a secret channel, whereas Com is broadcast through a public channel (e.g., a bulletin board). If the dealer attempts to send inconsistent commitments, participants will detect it immediately via cross-verification, thereby aborting the protocol.

Verify: $\mathsf{Verify}([m]^j, Com) \rightarrow \{0,1\}$. Each P_j verifies the correctness of their share $[m]^j$ by checking $g^{[m]^j} = \prod_{k=1}^{t}(Com_k)^{x_j^k} \mod p$. If the equation holds, the share is valid and the function outputs 1. Otherwise, output 0.

Reconstruct: $\mathsf{Rec}(\{[m]^j\}_{j\in[t]}) \rightarrow m$. When t or more nodes reveal valid shares, the secret m can be reconstructed using Lagrange interpolation where $\lambda_j = \prod_{k\in[t],k\neq j} \frac{x_k}{x_k - x_j} \mod p$:

$$m = f(0) = \sum_{j\in[t]} \lambda_j \cdot [m]^j \mod p \tag{1}$$

2.2 Homomorphic Pseudorandom Functions

HPRF is a special type of pseudorandom function that exhibits homomorphic properties on keys. Let $F : \chi \rightarrow \gamma$ be an HPRF, where $(\chi, \oplus)$ and $(\gamma, \otimes)$ are groups. If the HPRF F satisfies: $F(m_1, r) \otimes F(m_2, r) = F(m_1 \oplus m_2, r)$ for keys $m_1, m_2 \in \chi$, where r is an input string, then it is called a key homomorphic HPRF. m_i is the secret chosen by each client as the key of HPRF. A key HPRF [27] based on the Learning with Errors (LWE) problem can be constructed as $F(m,r) = \lfloor \prod_{i=1}^{l} \mathbf{A}_{r_i}, m \rfloor_q \in \mathbb{Z}_q^n$, where n, l, q are public parameters, m is a secret vector from $\mathbb{Z}_q^n$, r_i is the element at index i in the input vector $r \in \mathbb{Z}_q^l$, $\mathbf{A}_{r_i}$ is the public matrix determined by r_i. The homomorphic property holds as follows: $F(m_1 + m_2, r) = F(m_1, r) + F(m_2, r) + \mathbf{E}$, where the approximation error is a small error term $\mathbf{E} \in \{-1, 0, 1\}^n$. The security of this construction depends on the hardness of the LWE problem. HPRF constructs pseudorandom outputs that exhibit homomorphic properties under key addition, albeit with a small error term. We design an HPRF error elimination module (Section 6.1) to mitigate the impact of HPRF errors on model aggregation accuracy.

2.3 Byzantine Fault Tolerance Consensus

In distributed networks, consensus is an effective way to achieve *consistency* where each node has an identical view on the committed proposals and *liveness* where proposals are sure to be processed. BFT consensus [36] is usually run by a leader and several normal nodes to process proposals through interactive votes. We adopt a lightweight version [37] of the HotStuff protocol [38], where the leader remains stable, and nodes commit to proposals via two voting rounds. In Aion, BFT commits the model or the online client set, with a communication complexity of $O(n)$. In the following, we denote the result of executing BFT consensus on a message str as $\langle str \rangle_{\mathrm{BFT}} = \big(str, \sigma(str)\big)$, where $\sigma(str)$ is the threshold signature from at least $n - f$ consensus nodes on str.

2.4 Verifiable Random Function

Verifiable Random Function (VRF) [39, 40] is a cryptographic primitive that combines randomness with verifiability. It allows one generator to produce a random value that can be verified by others using a proof and a public key.

Key generation: $\mathsf{VRF.Gen}(1^k) \rightarrow (pk, sk)$. Choose a cyclic group $\mathbb{G}$ of prime order q with a generator g, select a random private key $sk \in \mathbb{Z}_q$, and compute the public key $pk = g^{sk}$.

Randomness generation: $\mathsf{VRF.Prove}_{sk}(x) \rightarrow (y, \pi(y))$. Given an input x, the generator computes a randomness $y = e(g,g)^{1/x+sk}$ and its proof $\pi(y) = g^{1/(x+sk)}$.

Verification: $\mathsf{VRF.Ver}_{pk}(x, y, \pi) \rightarrow \{0,1\}$. To verify the output y for input x with proof π, check if $e(g^x \cdot pk, \pi) = e(g,g)$ and whether $y = e(g,\pi)$, where $e(\cdot,\cdot)$ is the bilinear pairing

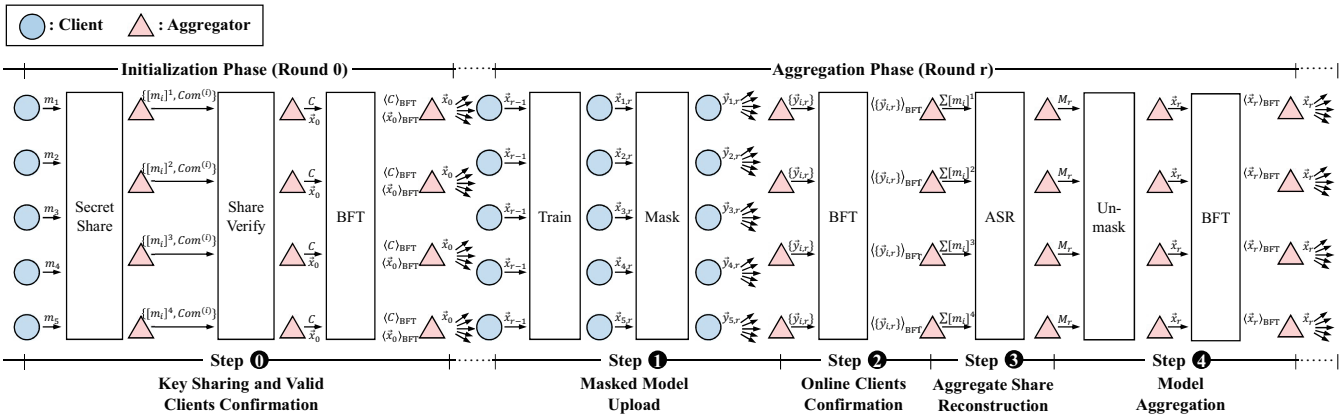


Figure 1: The system flow of Aion.

function. If both checks succeed, then y is valid and output 1. Otherwise, output 0.

3 Problem Formulation

3.1 Problem Statement

Aion consists of N *clients* (N can range from 100K to 10M in large-scale FL [41]), a single server, and n *aggregators* (typically a few or several dozen, e.g., $n = 16$). The *server* is responsible for initializing the training task and storing the final model, while the aggregators handle client coordination and model aggregation. In all masking-based SA schemes utilizing secret sharing, multiple "aggregators" are essential for performing secret share collection and reconstruction. Decryptors in Flamingo [14] and multiple clients within neighborhoods in SecAgg [2], SecAgg+ [12] and ACORN [13] are aggregators in essence. In each round, q clients participate in training ($q \leq N$, e.g., $q = 100$ or 1000). Each client has its own private data for training, and after training, the client uploads the masked model (or gradient, which will be referred to as the model for simplicity) to the aggregators. The aggregators aggregate the models in each round. The system starts with an initialization phase, followed by r rounds (about 500 to 10,000) of aggregation phases. In each round, each client C_i in the client set C trains using the previous round's global model $\vec{x}_{r-1}$ to obtain a local model $\vec{x}_{i,r}$, generates a mask, and uploads the masked model $\vec{y}_{i,r}$ to the aggregators. Each aggregator S_j reconstructs the aggregated mask and obtains the updated global model $\vec{x}_r$, which is sent back to the clients. The aggregation phase is iterated until the model converges. Practical solutions for dynamic aggregators and their real-world deployment are discussed in Appendix G.

3.2 System Model

Threat model. Aion tolerates malicious participants. *Malicious clients* may incorrectly share secrets, add erroneous masks, drop out, or engage in model poisoning to affect model accuracy or insert backdoors. *Malicious aggregators* may crash, deceive, or equivocate (sending different messages to different participants). Malicious clients and aggregators can collude to deduce the private models or data of honest clients.
Communication model. In Aion, authenticated and encrypted channels are used between clients and aggregators, as well as between aggregators. The network between aggregators is *partially synchronous*, with message latency between honest nodes bounded by an unknown Δ. The communication between clients and aggregators is *asynchronous*, meaning that messages intended for round r may arrive after round r.
Adversarial model. We consider a probabilistic polynomial time (P.P.T.) adversary $\mathcal{A}$, who cannot forge digital signatures (e.g. DSS signature [42]) nor break the encryption scheme (e.g. AES encryption [43]). For n aggregators, $\mathcal{A}$ can control at most f malicious aggregators, where $n \geq 3f+1$ in partially synchronous networks (Aion also supports $n \geq 2f+1$ in synchronous networks with a synchronous BFT). For q clients in each aggregation phase, $\mathcal{A}$ can control at most $q-2$ clients.

3.3 System Goal

Security. The system must ensure security, which means the private information of honest clients will not be obtained by any participants (aggregators or clients). Private information mainly refers to local models of honest clients.
Correctness of aggregated model. In each round r, assuming that each client C_i adds its mask correctly to its model $\vec{x}_{i,r}$, then it holds that the aggregated model $\vec{x}_r = \sum \vec{x}_{i,r}$.
Robustness against malicious participants. The system defends against attacks from malicious clients or aggregators, ensuring convergence and accuracy of the final model.

3.4 System Overview

Figure 1 introduces the main process of Aion. Aion operates in two distinct phases: the initialization phase and the aggregation phase. For initialization, each client generates a secret

value and shares it with the aggregators using VSS, and the aggregators use these shares to establish a valid client set. After reaching a consensus on the valid set and initial model via BFT, the aggregators broadcast the model to the valid clients (Step ❶). In the aggregation phase, each client performs local model training and computes a masked model using HPRF, then uploads the masked model to the aggregators (Step ❶). The aggregators collect these masked models, confirm the clients' online status via BFT consensus (Step ❷), and reconstruct the aggregated mask using aggregated share/mask reconstruction (Step ❸). The aggregators then remove the mask, compute the global model, and reach consensus via BFT. Finally, aggregators broadcast the global model to the clients for the next round of the aggregation phase (Step ❹).

4 Aggregatable Multi-Round Single-Mask SA

Aion contains two phases: the one-time initialization phase (Algorithm 1) and the repeated aggregation phase (Algorithm 2). The blue texts represent pluggable modules.

Algorithm 1 Initialization Phase (Round $r = 0$)

Input: initial model $\vec{x}_0$, initial client set C_0
Output: valid client set C, $[m_i]^j$ for $C_i \in C$ (each aggregator S_j)

◇ **Initialization phase (round $r = 0$):**
Key sharing and valid clients confirmation (Step ❶)
▸ **As a client** $C_i \in C_0$**:**
1: **generate** a random secret value m_i
2: **run** $\mathsf{Share}(m_i, f+1, n) \rightarrow (\{[m_i]^j\}_{j\in[S]}, Com^{(i)})$
3: **for** each $S_j \in S$ **do**
4: **send** $([m_i]^j, Com^{(i)})$ to corresponding S_j

▸ **As an aggregator** $S_j \in S$**:**
5: **for** each client $C_i \in C_0$ **do**
6: **if** $\mathsf{Verify}([m_i]^j, Com^{(i)}) \rightarrow 1$ **then**
7: **put** client C_i into set C; **store** $[m_i]^j$
8: **run** $\mathsf{BFT}(C) \rightarrow \langle C \rangle_{\mathrm{BFT}}$ to commit the valid client set $C = \langle C \rangle_{\mathrm{BFT}}$
9: **run** $\mathsf{BFT}(\vec{x}_0) \rightarrow \langle \vec{x}_0 \rangle_{\mathrm{BFT}} = (\vec{x}_0, \sigma(\vec{x}_0))$ and broadcast $(\vec{x}_0, \sigma(\vec{x}_0))$ to each client in C
10: **start** round $r+1$

▸ **As a client** $C_i \in C_0$**:**
11: **run** $\mathsf{CCS}(\vec{x}_0 \| \sigma(\vec{x}_0), K) \rightarrow (C^{(k)}, \pi(i))$ ▷ Algorithm 4, client concealed sortition

4.1 Initialization Phase

The initialization phase is only executed in round 0, which only contains the following Step ❶.

Step ❶ : Key sharing and valid clients confirmation

In Algorithm 1, each client C_i randomly generates a secret value m_i as the HPRF key, runs the secret sharing function $\mathsf{Share}(m_i, f, n)$, and obtains secret shares $\{[m_i]^j\}_{j\in[S]}$ with commitments $Com^{(i)}$ (Lines 1-2). Then each client sends $([m_i]^j, Com^{(i)})$ to the corresponding aggregator S_j through an authenticated and encrypted channel (Lines 3-4).

Each aggregator $S_j \in S$ first runs the share verification function on each received secret share $[m_i]^j$ from client C_i to verify the validity of the share. If the verification passes, the aggregator will consider the client corresponding to the share as a valid client and put it into the set of valid clients, and store the share (Lines 5-7). Next, the aggregator set runs the BFT consensus to achieve consistency on the valid client set $\langle C \rangle_{\mathrm{BFT}}$ (Line 8). Finally, each aggregator broadcasts the initial model $\langle \vec{x}_0 \rangle_{\mathrm{BFT}} = (\vec{x}_0, \sigma(\vec{x}_0))$ with its BFT threshold signature to each client in C to start the training (Lines 9-10). Each aggregator S_j stores valid clients' shares $\{[m_i]^j\}_{i\in[C]}$.

Algorithm 2 Aggregation Phase (Round r, $r = 1, 2, \ldots$)

Input: global model $\vec{x}_{r-1}$ (client)
Output: global model $\vec{x}_r$ (aggregator)

Masked model upload (Step ❶)
▸ **As a client** $C_i \in C$**:**
1: **upon** receiving a valid committed model $\langle \vec{x}_{r-1} \rangle_{\mathrm{BFT}}$ from an aggregator
2: **perform** local training on $\vec{x}_{r-1}$ to obtain $\vec{x}_{i,r}$
3: **calculate** $\vec{y}_{i,r} = \vec{x}_{i,r} + \mathsf{HPRF}(m_i, r)$ ▷ The error of HPRF on the client side can be eliminated by Algorithm 7
4: **send** $\vec{y}_{i,r}$ to an aggregator $S_j \in S$

Online clients confirmation (Step ❷)
▸ **As an aggregator** $S_j \in S$**:**
5: **for** each client $C_i \in C$ **do**
6: **if** $\vec{y}_{i,r}$ sent by C_i is received within the $timer$ **then**
7: **put** C_i into the online set C_r^{on}
8: **else**
9: **put** C_i into the offline set C_r^{off}
10: **run** $\mathsf{MGF}(C_r^{\mathrm{on}}) \rightarrow C_r^{\mathrm{valid}}$ ▷ Algorithm 6, input validation
11: **set** $C_r^{\mathrm{on}} = C_r^{\mathrm{valid}}$
12: **run** $\mathsf{BFT}(C_r^{\mathrm{on}})$ to commit the online client set $\langle C_r^{\mathrm{on}} \rangle_{\mathrm{BFT}}$

Aggregated share reconstruction (Step ❸)
▸ **As an aggregator** $S_j \in S$**:**
13: **calculate** $[M_r]^j = \sum_{i\in[C_r^{\mathrm{on}}]} [m_i]^j$
14: **broadcast** $[M_r]^j$ among S
15: **upon** receiving $f+1$ valid $[M_r]^{j'}$ from $S_{j'}$
16: **run** $\mathsf{ASR}(\{[M_r]^j\}, f+1) \rightarrow M_r$ ▷ Algorithm 3, ASR, which can also be replaced with Algorithm 5, AMR

Model aggregation (Step ❹)
▸ **As an aggregator** $S_j \in S$**:**
17: **calculate** $\vec{x}_r = \sum_{i\in[C_r^{\mathrm{on}}]} \vec{y}_{i,r} - \mathsf{HPRF}(M_r, r)$ ▷ The error of HPRF on the aggregator side can be eliminated by Algorithm 8
18: **run** $\mathsf{EMA}(C_{r-1}^{\mathrm{off}}) \rightarrow \vec{x}'_{r-1}$ ▷ Algorithm 9, asynchronous FL
19: **set** $\vec{x}_r = \vec{x}_r + \omega \vec{x}'_{r-1}$ ▷ ω is a scaling factor
20: **run** $\mathsf{BFT}(\vec{x}_r)$ to commit the model $\langle \vec{x}_r \rangle_{\mathrm{BFT}}$
21: **if** $|\vec{x}_r - \vec{x}_{r-1}| \geq \epsilon$ **then** ▷ check whether $\vec{x}_r$ is convergent
22: **send** the committed model $\langle \vec{x}_r \rangle_{\mathrm{BFT}}$ to each client in round $r+1$
23: **set** a timer $timer$ and **start** round $r+1$
24: **else**
25: **set** $\vec{x}^c = \vec{x}_r$ ▷ $\vec{x}^c$ represents the convergent model
26: **return**

4.2 Aggregation Phase

The aggregation phases contain the following 4 steps for each round r, which are shown in Algorithm 2.

Step ❶ : Masked model upload

For each client $C_i \in C$, on receiving a committed model $\langle \vec{x}_{r-1} \rangle$ from an aggregator, it verifies the commitment proof of BFT (usually an aggregated signature of aggregators) and

starts the aggregation phase of round r. C_i obtains the global model $\vec{x}_{r-1}$, trains the local model $\vec{x}_{i,r}$ for round r (Lines 1-2). Using the HPRF key m_i and r as HPRF seed, C_i generates a mask $\mathsf{HPRF}(m_i, r)$ and sends the masked model $\vec{y}_{i,r} = \vec{x}_{i,r} + \mathsf{HPRF}(m_i, r)$ to an aggregator $S_j \in S$ (Lines 3-4).

Step ❷ : Online clients confirmation

Each aggregator $S_j \in S$ determines the online client set C_r^{on} by checking if it has received $\vec{y}_{i,r}$ from each client $C_i \in C$. If $\vec{y}_{i,r}$ is received within the $timer$, C_i is considered online and added to the set C_r^{on}. Offline clients are added to the set C_r^{off} for asynchronous FL (Lines 5-9). The aggregator then runs the input validation function $\mathsf{MGF}(C_r^{\text{on}})$ (Algorithm 6) to filter out malicious inputs and updates the online client set to C_r^{valid} (Lines 10-11). Subsequently, S_j runs BFT consensus on C_r^{on} to commit the online set $\langle C_r^{\text{on}} \rangle_{\text{BFT}}$ (Line 12).

Step ❸ : Aggregated share reconstruction

S_j computes the sum of the shares of the online clients $[M_r]^j$ and broadcasts it among aggregators (Lines 13-14). Upon receiving at least $f+1$ shares, S_j will run the aggregated share reconstruction function $\mathsf{ASR}(\{[M_r]^j\}, f+1)$ of Algorithm 3 to obtain the aggregated key M_r (Lines 15-16).

Algorithm 3 Aggregated Share Reconstruction (ASR)

Input: aggregated shares $\{[M_r]^j\}$, commitments $\{Com^{(i)}\}$, threshold t
Output: aggregated secret M_r

1: **set** $VAS = \emptyset$ ▷ VAS denotes the valid aggregated share set
2: **for** each received $[M_r]^j$ **do**
3: **if** $g^{[M_r]^j} = \prod_{i\in[C_r^{\text{on}}]} \prod_{k\in[t]} (Com_k^{(i)})^{x_j^k} \mod p$ **then**
4: **put** $[M_r]^j$ into VAS
5: **if** $|VAS| \geq t$ **then**
6: **return** $M_r = \mathsf{Rec}(\{[M_r]^j\}_{[M_r]^j \in VAS}, t)$
7: **else return** $\perp$

In Algorithm 3, each aggregated share $[M_r]^j$ is broadcast by the corresponding aggregator S_j, t denotes the threshold value for the reconstruction, $Com^{(i)} = \{Com_k^{(i)}\}_{k\in[t]}$ denotes the public commitments of client C_i, and $\{Com^{(i)}\}$ denotes the commitments of all the online clients. It can be noted that Feldman commitments are homomorphic (i.e. $g^{[m_1]^j + [m_2]^j} = \prod_{k\in[t]} (Com_k^{(1)} \cdot Com_k^{(2)})^{x_j^k} \mod p$ for each $j \in [S]$), so to verify the validity of share addition, the aggregator simply needs to multiply the corresponding commitments. Therefore, the algorithm verifies the correctness of the aggregation share $[M_r]^j$ using the aggregated commitments from valid clients. It should be noted that VSS protocols utilizing a commitment with homomorphic properties, such as Pedersen commitment [44] or KZG commitment [45], can be applied to the scheme. The Feldman commitment is adopted here for clarity in describing the protocol. If the verification is successful, $[M_r]^j$ is added to the valid set VAS (Lines 1-4). When the size of VAS reaches t, the Rec function of VSS can be invoked to recover the aggregated secret M_r by using the aggregation shares $\{[M_r]^j\}$ (Lines 5-7).

Utilizing collect-aggregate-transfer to reduce complexity. Instead of broadcasting, each aggregator can send its share to the aggregator leader (also the leader of BFT). The leader collects shares, reconstructs the mask, and transfers the result to each aggregator. The result includes the reconstructed secret and its corresponding t shares. A malicious leader may either refuse to send the result to honest aggregators or submit tampered shares. For the former, honest aggregators will initiate the view-change of BFT to replace the malicious leader upon timeout; for the latter, they verify the t shares against their commitments. If the validation fails, the view-change will also be initiated. Therefore, the communication complexity of aggregators is $O(n)$, remaining the same as BFT.

Correctness of aggregated share reconstruction. We first prove the correctness of the aggregated verification process (Algorithm 3, Line 3):

$$g^{[M_r]^j} = g^{\sum_{i\in[C_r^{\text{on}}]} [m_i]^j} = \prod_{i\in[C_r^{\text{on}}]} g^{[m_i]^j} = \prod_{i\in[C_r^{\text{on}}]} g^{[m_i]^j} = \prod_{i\in[C_r^{\text{on}}]} g^{\sum_{k\in[t]} a_k^{(i)} x_j^k} = \prod_{i\in[C_r^{\text{on}}]} \prod_{k\in[t]} (Com_k^{(i)})^{x_j^k} \mod p$$

Next, we prove the correctness of the aggregated share reconstruction process $\mathsf{Rec}(\{[M_r]^j\}_{[M_r]^j\in VAS}, t)$ in Line 6:

$$\sum_{j=1}^{t} \lambda_j \cdot [M_r]^j = \sum_{j=1}^{t} \sum_{i=1}^{|C_r^{\text{on}}|} \lambda_j \cdot [m_i]^j = \sum_{i=1}^{|C_r^{\text{on}}|} \sum_{j=1}^{t} \lambda_j \cdot [m_i]^j = \sum_{i=1}^{|C_r^{\text{on}}|} m_i = M_r$$

Step ❹ : Model aggregation

In Algorithm 2, after reconstructing the aggregated share M_r, each aggregator computes the overall mask using $\mathsf{HPRF}(M_r, r)$. Then S_j removes the mask to obtain the accurate aggregated model $\vec{x}_r$, where $\vec{x}_r = \vec{y}_r - \mathsf{HPRF}(M_r, r)$ (Line 17). For asynchronous federated learning, the aggregator runs $\mathsf{EMA}(C_{r-1}^{\text{off}})$ (Algorithm 9) to get $\vec{x}'_{r-1}$ and updates $\vec{x}_r$ by adding a scaled version of $\vec{x}'_{r-1}$: $\vec{x}_r = \vec{x}_r + \omega \vec{x}'_{r-1}$, where $\omega \in (0, 1)$ is a scaling factor (Lines 18-19).

Finally, aggregators run the BFT consensus on $\vec{x}_r$ to get the committed $\langle \vec{x}_r \rangle_{\text{BFT}}$ (Line 20). Each aggregator determines whether model $\vec{x}_r$ has converged. If $\vec{x}_r$ converges, the aggregator terminates training and outputs $\vec{x}^c = \vec{x}_r$; otherwise, it sends $\langle \vec{x}_r \rangle_{\text{BFT}}$ to the client of round $r+1$. Besides, all aggregators should set a timer, and the clients that successfully upload their masked models before the timer expires will be put into the online client set of the next round (Lines 21-26).

Correctness of model aggregation. The sum of all clients' masks is equal to the overall mask. Each client's mask is $\mathsf{HPRF}(m_i, r)$, while the overall mask is $\mathsf{HPRF}(M_r, r)$. By the key homomorphism, it holds that:

$$\sum_{i=1}^{|C^{\text{on}}|} \mathsf{HPRF}(m_i, r) = \mathsf{HPRF}\left(\sum_{i=1}^{|C^{\text{on}}|} m_i, r \right) = \mathsf{HPRF}(M_r, r)$$

4.3 Client Concealed Sortition

Trivially selecting a subset of clients from the entire client set in each round may result in overlaps between client subsets

across different rounds. In such cases, the adversary may have the opportunity to infer the secret of an honest client. For example, in round i, the online client set is $\{c_1, ..., c_l, c_{l+1}\}$ with the aggregated HPRF key $M_i = \sum_{k=1}^{l+1} m_k$, while in round j, the set is $\{c_1, ..., c_l\}$, producing $M_j = \sum_{k=1}^{l} m_k$. An adversary could derive the HPRF key m_{l+1} from $M_i - M_j$, exposing C_{l+1}'s mask value and model. A more detailed analysis is presented in Appendix E. To address this, we propose the client concealed sortition (CCS) in Algorithm 4, where each client secretly and randomly determines its assignment to a specific round while ensuring the client subsets for different rounds are independent.

Algorithm 4 Client Concealed Sortition (CCS)

Private Input: m_i as VRF secret key
Public Input: initial model $\vec{x}_0$ and its threshold signature $\sigma(\vec{x}_0)$; number of client subsets K; $Com_0^{(i)}$ as VRF public key
Private Output: subset $C^{(k)}$ of the client set C to which C_i belongs
Public Output: proof of the random value $\pi(v_i)$

Key sharing and valid clients confirmation (Step ⓪, Line 11)
▶ As a client $C_i \in C$
1: **run** $\mathsf{VRF.Prove}_{m_i}\left(\mathsf{Hash}(\vec{x}_0 \| \sigma(\vec{x}_0))\right) \rightarrow (v_i, \pi(v_i))$
2: **calculate** $k = v_i \mod K$
3: **select** $C^{(k)}$ as the subset to which it belongs

Masked model upload (Step ❶, Line 4)
▶ As a client $C_i \in C$
4: **send** $\vec{y}_{i,r}, v_i, \pi(v_i)$ to an aggregator $S_j \in S$

Online clients confirmation (Step ❷, Line 6)
▶ As an aggregator $S_j \in S$:
5: **if** $\vec{y}_{i,r}$ sent by C_i is received within the $timer$ and $\mathsf{VRF.Ver}_{Com_0^{(i)}}\left(\mathsf{Hash}(\vec{x}_0 \| \sigma(\vec{x}_0)), v_i, \pi(v_i)\right) \rightarrow 1$ **then**

In Algorithm 4, after sending valid secret shares to the aggregators, the client C_i uses the hash value of the initial model $\vec{x}_0$ with its threshold signature $\sigma(\vec{x}_0)$ generated by aggregators through BFT as input to the VRF. Using the secret m_i as private key sk_i, C_i computes the VRF result v_i and the corresponding proof $\pi(v_i)$ (Line 1). The malicious aggregator or the server might attempt to leak $\vec{x}_0$ early to let malicious clients adjust m_i before secret sharing to influence the output of VRF. However, the BFT signature is published only after clients share secrets. Even if $\vec{x}_0$ may be leaked prematurely, clients cannot know the threshold signature $\sigma(\vec{x}_0)$ as it contains at least one honest client's signature. Assuming the number of subsets is K (K is less than the total training rounds to let each client contribute), C_i computes $k = v_i \mod K$ (typically, $v_i \gg K$ in large-scale FL). This allows C_i to determine which subset $C^{(k)}$ and round it belongs to (Lines 2-3). In the Step ❶, C_i uploads $\vec{y}_{i,r}$ along with $(v_i, \pi(v_i))$ (Line 4). In Step ❷, the aggregator verifies whether C_i should be put into C_r^{on} by additionally invoking $\mathsf{VRF.Ver}_{Com_0^{(i)}}\left(\mathsf{Hash}(\vec{x}_0 \| \sigma(\vec{x}_0)), v_i, \pi(v_i)\right)$ (Line 5).

Extra advantages of client concealed sortition. CCS has several extra advantages. First, since the VRF result is kept secret until revealed, it prevents an instant corruption adversary from controlling honest clients. Second, since the client shares its secret before the aggregators issue the committed initial model, the client cannot predict the VRF input in advance, preventing a malicious aggregator or clients from unfairly partitioning the client subsets. Third, the client knows in advance which round it will participate in, allowing it to remain offline until needed, thus reducing unnecessary overhead.

4.4 Aggregated Mask Reconstruction

We design aggregated mask reconstruction (AMR) (Algorithm 5) as an alternative to ASR (Algorithm 3). Unlike ASR, which reconstructs aggregated secret shares, AMR directly reconstructs the aggregated mask (HPRF) values without exposing the aggregated shares.

Algorithm 5 Aggregated Mask Reconstruction (AMR)

Input: secret share $[m_i]^j$ for $C_i \in C_r^{\text{on}}$, masked global model $\vec{y}_r$
Output: global model $\vec{x}_r$

▶ As an aggregator $S_j \in S$:
1: **calculate** $[M_r]^j = \sum_{C_i \in C_r^{\text{on}}} [m_i]^j$
2: **broadcast** $\mathsf{HPRF}([M_r]^j, r)$ within S
3: **set** $HV = \emptyset$ ▷ HV denotes the HPRF value set
4: **put** received HPRF value into HV
5: **for** $0 \le z \le f$ **do**
6: **wait** till $|HV| \ge 2f + 1 + z$
7: **selects** different $f+1$ values repeatly
8: **calculate** $\mathsf{HPRF}(M_r, r) = \lambda_j \cdot \sum_{j \in [f+1]} \mathsf{HPRF}([M_r]^j, r)$
9: **let** $F(x)$ be the corresponding preconstructed polynomial function
10: **if** at least $2f+1$ HPRF values in HV match $F(x)$ **then**
11: **return** $\mathsf{HPRF}(M_r, r)$

In Algorithm 5, each aggregator S_j first computes the sum of the shares of the online clients C_r^{on}, denoted as $[M_r]^j$ (Line 1). Then, the aggregator directly broadcasts $\mathsf{HPRF}([M_r]^j, r)$ in S instead of $[M_r]^j$ (Line 2). Next, S_j collects the received HPRF values into a set HV (Lines 3-4). When there are at least $2f+1$ HPRF values in HV, it repeatedly selects different $f+1$ values and calculates:

$$\mathsf{HPRF}(M_r, r) = \sum_{j \in [f+1]} \lambda_j \cdot \mathsf{HPRF}([M_r]^j, r) \tag{2}$$

Let $F(x)$ represent the polynomial function reconstructed by these $f+1$ HPRF values. If at least $2f+1$ of the HPRF values in HV lie on $F(x)$, the reconstructed $\mathsf{HPRF}(M_r, r)$ is considered correct and is output by the algorithm (Lines 5-11). Otherwise, S_j will continue to wait for new HPRF values until the verification passes.

Correctness of AMR. Equation 2 essentially leverages the additive homomorphism of the HPRF key. By interpreting both sides of Equation 1 as HPRF keys, it holds that:

$$\mathsf{HPRF}(M_r, r) = \mathsf{HPRF}\Big(\sum_{j \in [f+1]} \lambda_j \cdot [M_r]^j, r\Big) = \sum_{j \in [f+1]} \lambda_j \cdot \mathsf{HPRF}([M_r]^j, r)$$

Comparison of Aion-ASR and Aion-AMR. It can be observed that the communication and computation overhead of

Aion-AMR are higher than those of Aion-ASR. However, the advantage of Aion-AMR is that it ensures the entire aggregation process does not leak any aggregated keys. Besides, AMR allows the same client to participate in multiple rounds of training without the need for the CCS algorithm.

5 Lightweight Evolving Input Validation

We propose a lightweight evolving input validation mechanism named masked gradient filtering (MGF) (Algorithm 6) for Aion based on norm defense [32]. The L_2 norm is used for input validation by limiting the norm of each client's input to detect and discard anomalous updates. If the norm exceeds a set bound b, the update is discarded as potentially malicious.

Algorithm 6 Masked Gradient Filtering (MGF)

Input: global gradients $\vec{x}_{r-1}$, $\vec{x}_{r-2}$, mask ratio β, maximum HPRF element $h_{\max}$, aggregated HPRF masks $\mathsf{HPRF}(M_{r-1}, r-1)$, $\mathsf{HPRF}(M_{r-2}, r-2)$, mask scaling factors α_{r-1}, α_{r-2}, bound b_{r-1}, online client set C_r^{on}
Output: valid client set C^{valid}, global gradient $\vec{x}_r$

Key sharing and valid clients confirmation (Step ❶, Line 3)
▸ As a client $C_i \in C$:
1: **calculate** $\alpha_r = \beta \|(\vec{x}_{r-1})\|_\infty / h_{\max}$
2: **calculate** $\vec{y}_{i,r} = \vec{x}_{i,r} + \alpha_r \cdot \mathsf{HPRF}(m_i, r)$

Online clients confirmation (Step ❷, Line 10)
▸ As an aggregator $S_j \in S$:
3: **calculate** $\mu_r = \frac{(\|\vec{x}_{r-1}\|_2 + \alpha_{r-1}\|\mathsf{HPRF}(M_{r-1}, r-1)\|_\infty)}{(\|\vec{x}_{r-2}\|_2 + \alpha_{r-2}\|\mathsf{HPRF}(M_{r-2}, r-2)\|_\infty)}$
4: **set** $b_r = \mu_r b_{r-1}$
5: **set** $C^{\text{valid}} = \emptyset$
6: **for** $i = 1$ to $|C_r^{\text{on}}|$ **do**
7: **if** $\|\vec{y}_{i,r}\|_2 \le b_r$ **then**
8: **put** C_i into set C^{valid}

Model aggregation (Step ❹, Line 17)
▸ As an aggregator $S_j \in S$:
9: **calculate** $\alpha_r = \beta \|(\vec{x}_{r-1})\|_\infty / h_{\max}$
10: **calculate** $\vec{x}_r = \sum_{i \in [C_r^{\text{valid}}]} \vec{y}_{i,r} - \alpha_r \cdot \mathsf{HPRF}(M_r, r)$

In Algorithm 6, before computing the masked gradient[2] $\vec{y}_{i,r}$, each client first calculates the scaling factor $\alpha_r = \beta\|(\vec{x}_{r-1})\|_\infty / h_{\max}$, where $\beta \in (0,1)$ is termed the mask ratio, and $h_{\max}$ is the theoretical maximum value of the HPRF output elements. The client then computes $\vec{y}_{i,r} = \vec{x}_{i,r} + \alpha_r \cdot \mathsf{HPRF}(m_i, r)$ (Lines 1-2). For an aggregator, it first calculates the current round's bound b_r using the global model and mask from the previous two rounds (Lines 3-4). When each aggregator receives a masked gradient $\vec{y}_r$, it computes and compares whether the norm exceeds a bound b_r of round r. If it does, the gradient is filtered out and not included in the aggregation; otherwise, it is retained in the valid set C^{valid} (Lines 5-8). Each aggregator, when calculating $\vec{x}_r$, also multiplies the output of HPRF by α_r (Lines 9-10).

Evolving bound value. The bound b_r has a significant impact on the final training accuracy and poisoning success rate. If the bound is too low, it will filter too many clients, reducing the final training accuracy. Conversely, if the bound is too high, the poisoning success rate will increase, also lowering the final accuracy. To address this, we design an evolving bound adjustment strategy. Specifically, we adjust the bound after each training round. b_r is determined by the previous round's bound b_{r-1} and a reduction factor μ_r. We can effectively determine μ_r in each round since Aion can compute the sum of the mask values during the aggregation process in each round. As the training rounds r increase, the global gradient $\vec{x}_r$ decreases, hence typically $0 < \mu_r < 1$. Finally, we set:

$$\mu_r = \frac{(\|\vec{x}_{r-1}\|_2 + \alpha_{r-1}\|\mathsf{HPRF}(M_{r-1}, r-1)\|_\infty)}{(\|\vec{x}_{r-2}\|_2 + \alpha_{r-2}\|\mathsf{HPRF}(M_{r-2}, r-2)\|_\infty)}$$

Since we are processing gradients with masks, the norm filtering is influenced by the mask values. Given that we are aware of the total (or average) mask value, and that the mask values are controllable within a certain range, we incorporate the mask values obtained from the HPRF calculation into the formula to adjust the dynamic constraint. We chose the L_∞ norm of HPRF instead of the L_2 norm because each bit of the HPRF result is random. If the L_2 norm were used, it might filter out normal gradients with small masks added due to a few large elements, leading to instability in μ_r. Using the L_∞ norm effectively controls the range and amplitude of μ_r, allowing b_r to decrease steadily. Importantly, we set the bound b_r to be consistent with configurations that do not apply masking. When the mask magnitude is small, poisoned gradients typically still exceed the threshold and are thus filtered out effectively. In contrast, although large masks may occasionally cause a small number of benign gradients to be discarded, this has minimal impact on convergence (honest gradients are generally smaller than poisoning gradients, and their masked norms usually remain below b_r). Moreover, since poisoning attacks usually rely on significantly boosted gradients to be effective, these malicious updates are more likely to be filtered. We provide an experimental evaluation of Aion's robustness against poisoning attacks in comparison with ACORN in Section 7.2.

Evolving constraint of individual client mask ratio. Note that during the aggregation phase, the aggregator invokes MGF (Algorithm 6) for input validation, so the size of the individual mask (HPRF value) cannot be arbitrarily large; it must be smaller than the size of the model. Concretely, we let the aggregator calculate a normalization coefficient for the clients. The normalization coefficient for round r is given by $\alpha_r = \beta\|(\vec{x}_{r-1})\|_\infty / h_{\max}$. In this way, the client C_i can calculate the masked local gradient by $\vec{y}_{i,r} = \vec{x}_{i,r} + \alpha_r \cdot \mathsf{HPRF}(m_i, r)$. In our experiments (Section 7.2), we set $\beta = 0.2$, and the results indicate that this setting achieves good defenses against poisoning attacks as well as effective model convergence. Additionally, we evaluate Aion's resistance to gradient inversion attacks (GIA) in Appendix F to verify whether the scaling masks can protect the privacy of the clients. MGF introduces almost no additional communication or computational cost.

[2] L_2 norm defense typically applies to the gradients. The gradient represents the model's parameter change, and the two are easily converted.

6 Robustness Enhancement Constructions

6.1 HPRF Error Elimination Module

We denote the dimension of $\vec{x}_{i,r}$ as d, and each element has l_{dp} digits in decimal places. Denote the output of $\mathsf{HPRF}(m_i, r)$ as a pseudorandom vector[3] $\vec{h} = \{h_k\}$ of size d, where h_k is an integer (finite field element) and can vary in size. To mask $\vec{x}_{i,r}$ effectively with $\vec{h}$, we can scale $\{h_k\}$ to have the same number of decimal places as the elements in $\vec{x}_{i,r}$.

However, the homomorphic operation of adding q HPRF values introduces an error $\vec{e}^{(q)}$, i.e., $\sum_{i=1}^{q} \mathsf{HPRF}(m_i, r) = \mathsf{HPRF}(\sum_{i=1}^{q} m_i, r) + \vec{e}^{(q)}$, where $\vec{e}^{(q)} = \{e_k^{(q)}\}$, $e_k^{(q)} \in \{-(q-1), \ldots, q-1\}$. Therefore, we introduce the dynamic mask coverage (DMC) algorithms and corresponding dynamic mask removal (DMR) algorithms, as shown in Algorithms 7 and 8. Clients run DMC when adding masks, increasing the length of elements in the HPRF value based on the number of clients q, ensuring that errors only appear in the additional HPRF digits. As for aggregators, they run DMR to remove the masks, truncating the extra digits after computing the overall mask to obtain the accurate aggregated model.

Algorithm 7 Dynamic Mask Coverage (DMC)

Input: local model $\vec{x}_{i,r}$, HPRF key m_i
Output: masked local model $\vec{y}_{i,r}$
Parameter: client number q, decimal places l_{dp} for elements $\vec{x}_{i,r}$

▶ **As a client** $C_i \in C$
1: **calculate** $l_{\text{ex}} = \lceil \log_{10}(2q) \rceil$ ▷ $\lceil . \rceil$ denotes ceiling function
2: **execute** $\mathsf{HPRF}(m_i, r) \rightarrow \vec{h} = \{h_k\}$
3: **for** each $h_k \in \vec{h}$ **do**
4: **calculate** $h_k = h_k / 10^{l_{\text{dp}}+l_{\text{ex}}}$
5: **set** $\vec{y}_{i,r} = \vec{x}_{i,r} + \vec{h}$

Algorithm 8 Dynamic Mask Removal (DMR)

Input: masked global model $\vec{y}_r$, aggregated HPRF key M_r
Output: global model $\vec{x}_r$
Parameter: client number q, decimal places l_{dp} for elements of $\vec{x}_r$

▶ **As an aggregator** $S_j \in S$**:**
1: **calculate** $l_{\text{ex}} = \lceil \log_{10}(2q) \rceil$ ▷ $\lceil . \rceil$ denotes ceiling function
2: **execute** $\mathsf{HPRF}(M_r, r) \rightarrow \vec{h}^* = \{h_k^*\}$
3: **for** each $h_k^* \in \vec{h}^*$ **do**
4: **calculate** $h_k^* = h_k^* / 10^{l_{\text{dp}}+l_{\text{ex}}}$
5: **set** $\vec{x}_r = \vec{y}_r - \vec{h}^*$
6: **set** $\vec{x}_r = \mathsf{Round}(\vec{x}_r, l_{\text{dp}})$ ▷round elements in $\vec{x}_r$ to l_{dp} decimal places

In Algorithm 7, a client C_i first computes $l_{\text{ex}} = \lceil \log_{10}(2q) \rceil$, where l_{ex} represents the extra digits that should be added (Line 1). Next, C_i computes $\mathsf{HPRF}(m_i, r)$ to generate a pseudorandom vector $\vec{h} = \{h_k\}$ of size d, where each h_k is a random integer with sufficient digits (Line 2). Next, C_i scales each h_k from an integer to a decimal with $(l_{\text{dp}} + l_{\text{ex}})$ decimal places (Lines 3-4). Finally, C_i computes $\vec{y}_{i,r} = \vec{x}_{i,r} + \vec{h}$ (Line 5).

[3]The HPRF output is generally in matrix form, but it can be easily converted to a vector in a row-major or column-major manner.

In Algorithm 8, each aggregator S_j obtains the random vector $\vec{h}^*$ using the similar operations as in Algorithm 7, then computes the aggregated model $\vec{x}_r = \vec{y}_r - \vec{h}^*$ (Lines 1-5). Finally, S_j performs a rounding operation to retain l_{dp} decimal places for the elements of $\vec{x}_r$, obtaining the accurate aggregated model (Line 6).

6.2 Asynchronous Federated Learning

In large-scale FL, requiring all devices to complete training and upload updates synchronously in each round results in the training speed being limited by the slowest device. Therefore, each round is limited by a timer, and some offline or high-latency clients may fail to upload their inputs within the current round. However, high-latency clients' inputs may arrive after the round has ended. Leveraging the inputs from high-latency clients for asynchronous FL can accelerate model convergence. Therefore, we design an expired model aggregation (EMA) module that enables Aion to support asynchronous FL while protecting the client model, as shown in Algorithm 9.

Algorithm 9 Expired Model Aggregation (EMA)

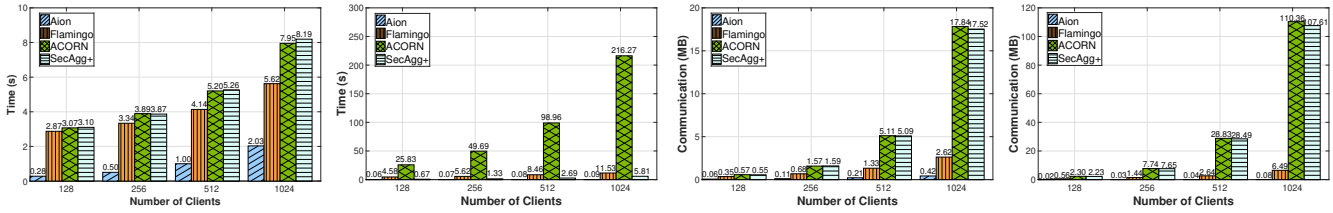
Input: offline client set of round $r-1$ C_{r-1}^{off}
Output: reused expired global model $\vec{x}'_{r-1}$
Parameter: security threshold for the number of clients ξ

▶ **As an aggregator** $S_j \in S$**:**
1: **for** each $C_i \in C_{r-1}^{\text{off}}$ **do**
2: **if** $\vec{y}_{i,r-1}$ sent by C_i is received in round r **then**
3: **put** $\vec{y}_{i,r-1}$ into set C_r^{delay} ▷ C_r^{delay} is initialized to $\emptyset$
4: **if** $|C_r^{\text{delay}}| \geq \xi$ **then**
5: **calculate** $[M'_{r-1}]^j = \sum_{i \in [C_r^{\text{delay}}]} [m_i]^j$
6: **broadcast** $[M'_{r-1}]^j$ among S
7: **upon** receiving $f+1$ valid $[M'_{r-1}]^{j'}$ from $S_{j'}$
8: **run** $\mathsf{ASR}(\{[M'_{r-1}]^j\}, f+1) \rightarrow M'_{r-1}$ ▷ Algorithm 3 or 5
9: **calculate** $\vec{x}'_{r-1} = \sum_{i \in [C_r^{\text{delay}}]} \vec{y}_{i,r-1} - \mathsf{HPRF}(M'_{r-1}, r-1)$
10: **else set** $\vec{x}'_{r-1} = \vec{0}$ ▷ model aggregation is not performed

Algorithm 9 can be executed in Step ❹ of the aggregation phase. We consider the most common scenario, where masked models sent by offline clients from round $r-1$ arrive in round r. The aggregator puts them in the set C_r^{delay} (Lines 1-3). If the size of C_r^{delay} exceeds the security parameter ξ ($\xi \geq 2$) before the end of round r, the aggregators will follow the same process as in Step ❸ to recover the aggregated key M'_{r-1} for all clients in C_r^{delay}, thus obtaining the aggregated model $\vec{x}'_{r-1}$ of these clients (Lines 4-9). If the size of C_r^{delay} is less than ξ, the aggregator deems the aggregation insecure and does not execute the aggregation process (Line 10).

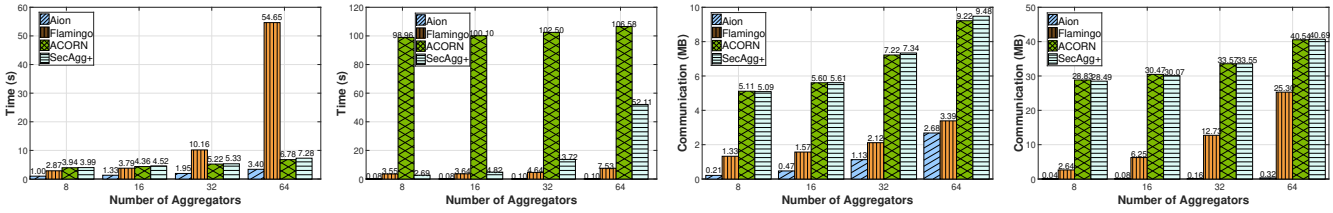
7 Implementation and Evaluation

We implement Aion using Python on a workstation with RTX4090 and i9-14900KF and conduct an evaluation on time



(a) Init. Time v.s. q (b) Agg. Time v.s. q (c) Init. Overhead v.s. q (d) Agg. Overhead v.s. q

Figure 2: Time cost and message overhead (initialization and aggregation phase) with varying client number q.



(a) Init. Time v.s. n (b) Agg. Time v.s. n (c) Init. Overhead v.s. n (d) Agg. Overhead v.s. n

Figure 3: Time cost and message overhead (initialization and aggregation phase) with varying aggregator number n.

cost, message overhead, and model accuracy. We also implement the Flamingo [14], ACORN [13], and SecAgg+ [12], as well as SecAgg [2], Mario [16], and ELSA [15], for comprehensive experimental comparison. For all masking-based schemes, we use the single-server, multi-aggregator architecture for fair comparison. Due to space constraints, we present additional experiment results in Appendix B.

7.1 Time and Message Overhead Performance

We test the time cost and message overhead for both initialization and aggregation phases by adjusting the client number q in each round and aggregator number n. The model dimension d is set to 10K and the modulus p in secret sharing is set to 2048 bits unless otherwise specified.

Time cost. Figures 2a and 2b show the execution time for the initialization and aggregation phases with varying q, fixed n at 8. Aion consistently outperforms Flamingo, ACORN, and SecAgg+. For $q = 1024$, Aion is 128.11× faster than Flamingo, 2324.80× faster than ACORN, and 62.41× faster than SecAgg+ for aggregation and 2.77×, 3.91×, 4.03× faster for initialization, respectively. ACORN incurs longer aggregation time mainly due to the use of ZK. This performance gap widens as the client number increases. Aion's single mask and one-time SS minimize computational overhead.

Figures 3a and 3b show execution times with varying n for q fixed at 512. Aion again shows a faster time. When $n = 64$, Aion achieves a speedup of 75.30× over Flamingo, 1,065.80× over ACORN, and 521.10× over SecAgg+ in the aggregation phase. In the initialization phase, Aion is 16.07×, 1.99×, and 2.14× faster than Flamingo, ACORN, and SecAgg+, respectively. The distributed key generation of Flamingo brings higher time cost. In Aion, aggregators use a collect-aggregate-transfer mechanism to reconstruct a total mask.

Message overhead. Figures 2c and 2d present the additional message overhead (excluding model data) for the initialization and aggregation phases across varying q with n fixed at 8. Aion consistently shows lower overhead than compared schemes, a difference that increases with the client number. When q is 1024, Aion's overhead is 1.23% of Flamingo's , 0.07% of ACORN's and SecAgg+'s in the aggregation phase and 16.03%, 2.38%, 2.34% in the initialization phase respectively. ACORN and SecAgg+ perform secret sharing and reconstruction across a greater number of client neighborhoods. Aion's advantage comes from sharing fewer secrets and reconstructing only a single aggregated mask.

Figures 3c and 3d present the message overhead for different n. With n fixed at 64, Aion's overhead is 1.26% of Flamingo's, 0.08% of ACORN's and SecAgg+'s in the aggregation phase, and 79.06%, 29.07%, 28.27% in the initialization phase, respectively. As the number of aggregators increases, the message overhead increases. This is expected, as each added aggregator introduces extra communication. However, the increase in overhead for Aion is noticeably smaller. This is because the aggregation of client masks in Aion requires less data exchange among aggregators.

7.2 Input Validation Performance

Experimental setup. The input validation experiment follows the basic setup of RoseAgg [46]. We conduct 60 rounds of poisoning attacks, with an attack probability of 50% for

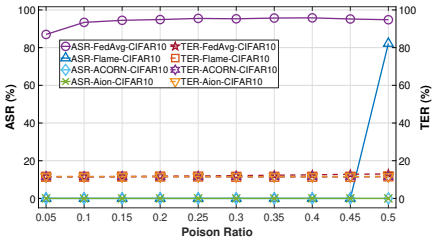


Figure 4: ASR/TER vs. Poison Ratio (CIFAR10).

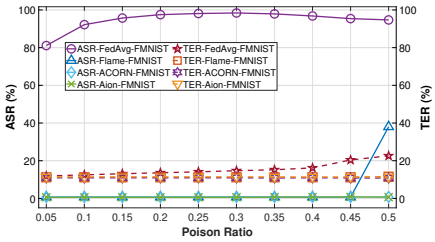


Figure 5: ASR/TER vs. Poison Ratio (FMNIST).

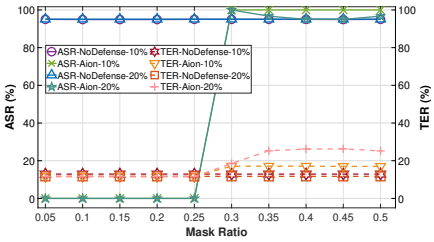


Figure 6: ASR/TER vs. Mask Ratio (CIFAR10).

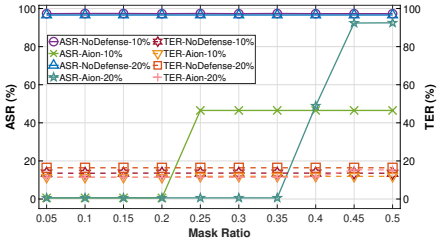


Figure 7: ASR/TER vs. Mask Ratio (FMNIST).

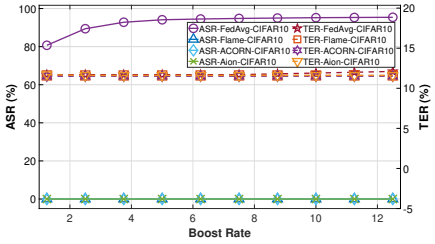


Figure 8: ASR/TER vs. Boost Rate (CIFAR10).

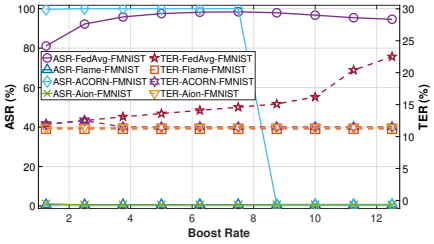


Figure 9: ASR/TER vs. Boost Rate (FMNIST).

each round. We design three experiments to investigate the effects of *poison ratio* (proportion of clients that attack), *boost rate* (amplification magnitude of malicious gradient), and *mask ratio* (scaling factor of HPRF value) on model performance. The default values are set as follows: poison ratio 50%, boost rate 20, and mask ratio is constrained to be less than 10% of the global update from the last round. We compare Aion with the benchmark FedAvg [1] and the state-of-the-art scheme, Flame [47], to thoroughly demonstrate the effectiveness of the input validation module.

Adversary setup. Adversaries multiply their gradients by a scaling factor to amplify the influence on the global model.

Evaluation metrics. The performance of the defense methods is evaluated by two metrics: Attack Success Rate (ASR) and Test Error Rate (TER). ASR represents the proportion of adversarial attacks that successfully induce the model to produce the intended output, while TER reflects the impact of poisoning attacks on the model's primary task performance.

Effectiveness of input validation. We test the performance of models under different poison ratios. As shown in Figure 4 and 5, the ASR of Aion remains at 0, indicating that even with up to 50% malicious clients, model security is not threatened. The clustering-based method, Flame, can resist poisoning attacks when malicious clients are fewer than honest ones. However, Flame cannot accurately distinguish between benign and malicious clusters if the numbers of them are comparable, allowing some malicious gradients to pass through and affect the global model. For comparison, we also evaluate ACORN's defense against poisoning attacks, with results presented in Figures 4, 5, 8, and 9. ACORN exhibits robustness on the CIFAR10 dataset, but its effectiveness deteriorates on the FMNIST dataset when the boost rate drops below 7.5. Specifically, at low boost rates, poisoned gradients in ACORN fail to exceed the clipping threshold, thereby bypassing the defense mechanism. In contrast, Aion's masking mechanism amplifies the magnitude of gradients. As a result, even at lower boost rates, poisoned gradients remain above the L_2 clipping threshold and are effectively filtered out.

Impact of privacy protection on input validation. We explore the impact of incorporating masks of varying sizes when poison ratios are 10% or 20%. Since both masks and gradients are multi-dimensional vectors with positive and negative values, their combination may reduce the absolute values of certain dimensions, potentially affecting the filtering process by decreasing the L_2 norm. To mitigate this situation, we specify that the mask ratio added in each round must not exceed the product of the mask ratio and the global gradient from the previous round, using the mask ratio hyperparameter as a means to regulate privacy protection. We include Aion without the input validation module as a control group, termed NoDefense. As shown in Figure 6, with the CIFAR10 dataset, Aion's input validation module fully resists attacks when the mask ratio is lower than 0.25. Figure 7 depicts the results for the FMNIST dataset, where Aion is effective at mask ratios lower than 0.35 with a poison ratio of 20%, but only ensures model safety at mask ratios lower than 0.2 with a poison ratio of 10%. These findings suggest that the inherent randomness of masks may introduce instability to the input validation module, highlighting the importance of carefully selecting mask ratios to ensure model security.

Tolerable maximum boost rate. We explore the influence of the boost rate on input validation when the poison ratio is 10%. Figures 8 and 9 demonstrate that Aion successfully filters all poisoned gradients when the boost rate ranges from

1.25 to 12.5. When the boost rate exceeds 10, the excessively large gradients begin to disrupt the FedAvg training process, resulting in a slight increase in the TER. However, these excessively large gradients are effectively mitigated through L_2 norm filtering, indicating that the use of a large scaling factor does not adversely affect the functionality of Aion.

7.3 Aion-ASR (AMR) with Input Validation

We compare Aion-ASR (AMR) with or without input validation (IV) across various q and $n = 8$. Table 2 presents the time cost and message overhead for both the client and aggregator.

Table 2: Time cost and message overhead of Aion.

q	Scheme	Client: ms (KB)		Aggregator: ms (MB)	
		Initialization	Aggregation	Initialization	Aggregation
1024	Flamingo	67.12 (2.21)	96.28 (7.78)	132.03 (0.40)	5159.00 (6.15)
	Aion-ASR	7.02 (0.95)	22.01 (0.11)	52.01 (0.07)	75.03 (0.16)
	Aion-AMR	7.04 (0.93)	23.03 (0.11)	51.97 (0.07)	85.00 (0.62)
	Aion-ASR+IV	6.99 (0.94)	22.04 (0.11)	52.03 (0.07)	78.05 (0.12)
	Aion-AMR+IV	7.01 (0.95)	22.00 (0.11)	52.00 (0.07)	84.41 (0.58)
2048	Flamingo	67.31 (2.23)	120.55 (7.78)	134.98 (0.78)	14439.50 (13.03)
	Aion-ASR	7.03 (0.95)	22.05 (0.11)	54.02 (0.14)	75.01 (0.32)
	Aion-AMR	6.98 (0.93)	23.00 (0.11)	54.05 (0.14)	85.03 (0.93)
	Aion-ASR+IV	7.05 (0.94)	22.30 (0.11)	54.01 (0.14)	78.00 (0.24)
	Aion-AMR+IV	7.00 (0.95)	22.00 (0.11)	54.04 (0.14)	84.31 (0.78)
4096	Flamingo	67.79 (2.20)	137.51 (7.78)	136.01 (1.53)	42359.10 (32.14)
	Aion-ASR	7.01 (0.93)	22.00 (0.11)	55.03 (0.26)	75.02 (0.65)
	Aion-AMR	6.97 (0.94)	23.02 (0.11)	55.03 (0.26)	85.00 (1.26)
	Aion-ASR+IV	7.04 (0.94)	22.03 (0.11)	55.02 (0.26)	78.04 (0.48)
	Aion-AMR+IV	7.00 (0.95)	23.01 (0.11)	55.01 (0.26)	84.42 (1.09)

Time cost. Aion is faster than Flamingo for both initialization and aggregation phases. When q is 4096, Aion-ASR's initialization of clients is faster than Flamingo by 9.67×. In the aggregation phase, Aion-ASR also outperforms Flamingo and is 6.25× faster than Flamingo.

For aggregators, Aion reduces execution times. When q is 4096, Aion-ASR is 563.64× faster than Flamingo in the aggregation phase. Meanwhile, when q increases from 1024 to 4096, Flamingo's time increases by 721.07%, while Aion's time remains nearly unchanged. Furthermore, input validation and selecting between ASR and AMR introduce a minor impact on runtime efficiency. For instance, when q is 1024, Aion-AMR shows a 13.29% increase in time compared to Aion-ASR, due to the greater computational complexity involved in reconstructing the masked HPRF values. Additionally, input validation introduces a small overhead in both cases. When q is 2048, the results in Aion-ASR+IV only have a 3.99% increase in runtime compared to Aion-ASR. These overheads are acceptable given the significant security improvements offered.

Message overhead. The message overhead focuses on the additional costs incurred by the respective secure aggregation except for the model-related communication (i.e., uploading local models and broadcasting the aggregated global model). This allows for a clearer comparison of the overhead specifically attributed to the secure aggregation. Flamingo has a higher message overhead than Aion, particularly in the aggregation phase. When q is 1024, Flamingo requires 7.78 KB for a client, compared to Aion-ASR's 0.11 KB. This difference is even bigger during initialization, with Aion-ASR's overhead being only 1.41% of Flamingo's.

Aion shows a significant advantage in aggregator-side message overhead, especially in initialization. With q is 1024, Aion-ASR requires only 0.07 MB for initialization, less than Flamingo's 0.40 MB. For aggregation, Aion-ASR also has a lower overhead. Aion-ASR+IV results in a slightly reduced message overhead compared to Aion-ASR since IV filters out some client inputs, while the slightly higher communication cost of Aion-AMR reflects the increased data exchange to reconstruct the HPRF values.

8 Security Analysis

We provide a formal definition of the ideal functionality of Aion. Let Π denote the protocol of the Aion aggregation phase. We define the ideal functionality $\mathcal{F}$ of Π in Figure 10.

Ideal functionality $\mathcal{F}$

Parties: clients $\{C_1, ..., C_q\} \in C$ and aggregators $\{S_1, , ..., S_n\} \in S$

- $\mathcal{F}$ receives from $\mathcal{A}$ a set of malicious clients $C_{\mathcal{A}} \subset [q]$ and malicious aggregators $S_{\mathcal{A}} \subset [n]$, where $|C_{\mathcal{A}}| = f' \leq q-2$ and $|S_{\mathcal{A}}| = f, n \geq 3f+1$:
  1. $\mathcal{F}$ receives online client $C^{\text{on}} \subset [q]$ and input $\vec{x}_{i,r}$ of client $C_i \in C^{\text{on}}/C_{\mathcal{A}}$.
  2. $\mathcal{F}$ asks $\mathcal{A}$ for a set: If $\mathcal{A}$ replies with a set $C_{\mathcal{A}}$, then $\mathcal{F}$ calculates $\vec{x}_r = \sum_{i \in [C^{\text{on}}/C_{\mathcal{A}}]} \vec{x}_{i,r}$; otherwise, sends abort to all honest clients.
  3. Depending on whether the server is corrupted by $\mathcal{A}$. If the server is corrupted by $\mathcal{A}$, then $\mathcal{F}$ outputs $\vec{x}_r$ to to all the parties corrupted by $\mathcal{A}$; otherwise, $\mathcal{F}$ asks $\mathcal{A}$ for a shift $\vec{a}_r$ and outputs $\vec{x}_r + \vec{a}_r$.

Figure 10: Ideal functionality $\mathcal{F}$ of Aion.

$\mathcal{F}$ defines the adversary $\mathcal{A}$'s corruption capability, i.e., $\mathcal{A}$ can control f aggregators and at most $q-2$ clients in each round. The privacy property of $\mathcal{F}$ ensures that $\mathcal{A}$ just knows the sum of all honest clients' local gradients $\vec{x}_r = \sum_{i \in [C^{\text{on}}/C_{\mathcal{A}}]} \vec{x}_{i,r}$; it cannot infer the local gradient $\vec{x}_{i,r}$. The correctness property of $\mathcal{F}$ guarantees $\vec{x}_r$ is indeed computed from the clients' inputs. In particular, Theorem 1 proves that $\mathcal{F}$ is securely realized by Aion. For page limit, we defer the detailed proof of the theorem in Appendix A.

Theorem 1 (Security of Π)**.** *Assume a secure HPRF with key homomorphism where $\mathcal{A}$ cannot recover m from* $\mathsf{HPRF}(m, r)$*, a BFT protocol satisfying consistency and liveness, and a VSS protocol ensuring correctness and security. A P.P.T. simulator* Sim *queries $\mathcal{F}$. For any $\mathcal{A}$ controlling an aggregator set $S_{\mathcal{A}}$ of size f ($n \geq 3f+1$) and a client set $C_{\mathcal{A}}$ of size f' ($f' \leq q-2$), Π satisfies security if the ideal-world joint view $\text{View}^{\mathcal{F}}_{\text{Sim}}$ is indistinguishable from the real-world joint view $\text{View}^{\Pi}_{\mathcal{A}}$.*

9 Conclusion

In this work, we propose Aion, a multi-round single-mask SA scheme with evolving input validation against malicious clients and aggregators. The computation and communication for both clients and aggregators are reduced, realizing robustness and efficiency.

Acknowledgment

This paper is supported by the National Key R&D Program of China through project 2022YFB2701600, the Natural Science Foundation of China (62202027, U21B2021, U22B2008, U21A20467, 61932014, 61932011), the Young Elite Scientists Sponsorship Program by the China Association for Science and Technology (2022QNRC001), the Beijing Natural Science Foundation (M23016), and the Fundamental Research Funds for the Central Universities.

Ethics Considerations

We develop Aion, an SA protocol for FL, guided by ethical principles prioritizing security, privacy, societal benefit, and proactive risk mitigation. While acknowledging the potential for misuse to conceal malicious activities, we assert that Aion's benefits in enabling privacy-preserving, large-scale collaborative learning respecting data ownership vastly outweigh this risk; we commit to responsible development and deployment. Our research adheres to all legal frameworks and data privacy principles, employing only compliant, public datasets and avoiding unsafe experiments. We prioritize team well-being by preventing harm, fostering a supportive environment, and promoting balance and diversity. We commit to continuously monitor for real-world harm, address negative outcomes, and take responsibility for implications. We disclose any discovered vulnerabilities responsibly to stakeholders with details and mitigations.

Open Science

We are committed to upholding the principles of open science to ensure the transparency of our research. All research codes and necessary documentation have been published at: https://doi.org/10.5281/zenodo.15605466.

References

[1] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, 2017.

[2] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *ACM CCS*, 2017.

[3] Florian Hartmann. Predicting text selections with federated learning. *Google AI Blog*, 2021.

[4] Yuntao Wang, Zhou Su, Yanghe Pan, Tom H. Luan, Ruidong Li, and Shui Yu. Social-aware clustered federated learning with customized privacy preservation. *IEEE/ACM TON*, 2024.

[5] Milad Nasr, Reza Shokri, and Amir Houmansadr. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *IEEE SP*, 2019.

[6] Ali Hatamizadeh, Hongxu Yin, Pavlo Molchanov, Andriy Myronenko, Wenqi Li, Prerna Dogra, Andrew Feng, Mona G Flores, Jan Kautz, Daguang Xu, et al. Do gradient inversion attacks make federated learning unsafe? *IEEE TMI*, 2023.

[7] Pushpita Chatterjee, Debashis Das, and Danda B. Rawat. Federated learning empowered recommendation model for financial consumer services. *IEEE TCE*, 2024.

[8] Dinh C. Nguyen, Quoc-Viet Pham, Pubudu N. Pathirana, Ming Ding, Aruna Seneviratne, Zihuai Lin, Octavia A. Dobre, and Won-Joo Hwang. Federated learning for smart healthcare: A survey. *ACM CSUR*, 2023.

[9] Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard H Yang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. Federated learning with differential privacy: Algorithms and performance analysis. *IEEE TIFS*, 2020.

[10] Hossein Fereidooni, Samuel Marchal, Markus Miettinen, Azalia Mirhoseini, Helen Möllering, Thien Duc Nguyen, Phillip Rieger, Ahmad-Reza Sadeghi, Thomas Schneider, Hossein Yalame, et al. Safelearn: Secure aggregation for private federated learning. In *SPW*, 2021.

[11] Chengliang Zhang, Suyi Li, Junzhe Xia, Wei Wang, Feng Yan, and Yang Liu. Batchcrypt: Efficient homomorphic encryption for cross-silo federated learning. In *USENIX ATC*, 2020.

[12] James Bell, Kallista Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *ACM CCS*, 2020.

[13] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. Acorn: input validation for secure aggregation. In *USENIX Security*, 2023.

[14] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *IEEE SP*, 2023.

[15] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. Elsa: Secure aggregation for federated learning with malicious actors. In *IEEE SP*, 2023.

[16] Truong Son Nguyen, Tancrède Lepoint, and Ni Trieu. Mario: Multi-round multiple-aggregator secure aggregation with robustness against malicious actors. *Cryptology ePrint Archive*, 2024.

[17] Adi Shamir. How to share a secret. *CACM*, 1979.

[18] Cécile Delerablée and David Pointcheval. Dynamic threshold public-key encryption. In *CRYPTO*, 2008.

[19] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. Asynchronous federated learning on heterogeneous devices: A survey. *Comput. Sci. Rev.*, 2023.

[20] Battista Biggio, Blaine Nelson, and Pavel Laskov. Poisoning attacks against support vector machines. *arXiv preprint arXiv:1206.6389*, 2012.

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

[22] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *ESORICS*, 2020.

[23] Uriel Fiege, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In *ACM TOC 1987*, 1987.

[24] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO*, 2003.

[25] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning. In *IEEE SP*, 2022.

[26] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. Eiffel: Ensuring integrity for federated learning. In *ACM CCS*, 2022.

[27] Dan Boneh, Kevin Lewi, Hart Montgomery, and Ananth Raghunathan. Key homomorphic prfs and their applications. In *CRYPTO*, 2013.

[28] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. Can you really backdoor federated learning?, 2019. http://arxiv.org/abs/1911.07963.

[29] Yifan Guo, Qianlong Wang, Tianxi Ji, Xufei Wang, and Pan Li. Resisting distributed backdoor attacks in federated learning: A dynamic norm clipping approach. In *IEEE Big Data*, 2021.

[30] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. Rofl: Robustness of secure federated learning. In *IEEE SP*, 2023.

[31] Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. Local and central differential privacy for robustness and privacy in federated learning. *arXiv preprint arXiv:2009.03561*, 2020.

[32] Gilad Baruch, Moran Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. *NeurIPS*, 2019.

[33] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security*, 2020.

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

[35] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, 1987.

[36] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI*, 1999.

[37] Jianyu Niu, Fangyu Gai, Mohammad M. Jalalzai, and Chen Feng. On the performance of pipelined hotstuff. In *INFOCOM*, 2021.

[38] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *PODC*, 2019.

[39] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *FOCS*, 1999.

[40] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *PKC*, 2005.

[41] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 2021.

[42] Rosario Gennaro, Stanisław Jarecki, Hugo Krawczyk, and Tal Rabin. Robust threshold dss signatures. In *EUROCRYPT*, 1996.

[43] Joan Daemen and Vincent Rijmen. Aes proposal: Rijndael. 1999.

[44] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, 1991.

[45] Aniket Kate, Gregory M Zaverucha, and Ian Goldberg. Constant-size commitments to polynomials and their applications. In *ASIACRYPT*, 2010.

[46] He Yang, Wei Xi, Yuhao Shen, Canhui Wu, and Jizhong Zhao. Roseagg: Robust defense against targeted collusion attacks in federated learning. *IEEE TIFS*, 2024.

[47] Thien Duc Nguyen, Phillip Rieger, Roberta De Viti, Huili Chen, Björn B Brandenburg, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, et al. Flame: Taming backdoors in federated learning. In *USENIX Security*, 2022.

[48] Dario Pasquini, Danilo Francati, and Giuseppe Ateniese. Eluding secure aggregation in federated learning via model inconsistency. In *ACM CCS*, 2022.

[49] Vaikkunth Mugunthan, Antigoni Polychroniadou, David Byrd, and Tucker Hybinette Balch. Smpai: Secure multi-party computation for federated learning. *NeurIPS*, 2019.

[50] Till Gehlhar, Felix Marx, Thomas Schneider, Ajith Suresh, Tobias Wehrle, and Hossein Yalame. Safefl: Mpc-friendly framework for private and robust federated learning. In *IEEE SPW*, 2023.

[51] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. Inverting gradients-how easy is it to break privacy in federated learning? *NeurIPS*, 2020.

[52] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. *NeurIPS*, 2019.

[53] Yangsibo Huang, Samyak Gupta, Zhao Song, Kai Li, and Sanjeev Arora. Evaluating gradient inversion attacks and defenses in federated learning. *NeurIPS*, 2021.

[54] Anda Cheng, Peisong Wang, Xi Sheryl Zhang, and Jian Cheng. Differentially private federated learning with local regularization and sparsification. In *CVPR*, 2022.

[55] Bingyong Guo, Zhenliang Lu, Qiang Tang, Jing Xu, and Zhenfeng Zhang. Dumbo: Faster asynchronous bft protocols. In *ACM CCS*, 2020.

A Security Analysis

Proof. To prove Theorem 1, we construct a series of hybrid executions, gradually transitioning from the real world to the ideal world. We introduce the simulator Sim in the ideal world that runs the adversary $\mathcal{A}$ as a subroutine.

Hybrid 1. Initially, $\mathcal{A}$ observes the actual execution of the Aion protocol, including the inputs from the adversary-controlled clients $\{\vec{x}_{i,r}\}_{i\in[C_\mathcal{A}]}$, the HPRF keys $\{m_i\}_{i\in[C_\mathcal{A}]}$, and the final aggregated result $\vec{x}_r = \sum_{i\in[C^{\text{on}}]}\vec{x}_{i,r}$. The BFT consensus ensures that C^{on} is correctly agreed upon by $n-f$ honest aggregators, despite $\mathcal{A}$ potentially controlling up to f malicious aggregators. The view in this hybrid is:

$$\text{View}^{\Pi}_{\mathcal{A}} = \text{View}^{\text{Hyb1}}_{\mathcal{A}} = \left\{\{\vec{x}_{i,r}\}_{i\in[C_\mathcal{A}]}, \{m_i\}_{i\in[C_\mathcal{A}]}, \{\vec{y}_{i,r}\}_{i\in[C^{\text{on}}/C_\mathcal{A}]}, \vec{x}_r\right\}$$

Hybrid 2. In this hybrid, Sim uses each virtual input $\vec{x}'_{i,r}$ generated by each client in $C^{\text{on}}/C_\mathcal{A}$ to replace the actual client input $\vec{x}_{i,r}$, generating a virtual aggregated result $\vec{x}'_r = \sum_{i\in[C^{\text{on}}/C_\mathcal{A}]}\vec{x}'_{i,r} + \sum_{i\in[C_\mathcal{A}]}\vec{x}_{i,r}$, ensuring that the final aggregation $\vec{x}'_r = \vec{x}_r$. The view is now:

$$\text{View}^{\text{Hyb2}}_{\text{Sim}} = \left\{\{\vec{x}_{i,r}\}_{i\in[C_\mathcal{A}]}, \{m_i\}_{i\in[C_\mathcal{A}]}, \{\vec{y}_{i,r}\}_{i\in[C^{\text{on}}/C_\mathcal{A}]}, \vec{x}'_r\right\}$$

Hybrid 3. In this hybrid, Sim replaces the honest clients' HPRF keys $\{m_i\}$ with random values $\{m'_i\}$ and recalculates all intermediate results, ensuring that the final aggregated result $\vec{x}_r$ remains unchanged. Due to the pseudorandomness of the HPRF, the adversary cannot distinguish between the real keys and the random keys, so the view becomes:

$$\text{View}^{\text{Hyb3}}_{\text{Sim}} = \left\{\{\vec{x}_{i,r}\}_{i\in[C_\mathcal{A}]}, \{m_i\}_{i\in[C_\mathcal{A}]}, \{\vec{y}'_{i,r}\}_{i\in[C^{\text{on}}/C_\mathcal{A}]}, \vec{x}'_r\right\}$$

Hybrid 4. Sim replaces the real masks $\text{HPRF}(m_i, r)$ with randomly generated virtual masks $\vec{h}$, while ensuring that the final aggregated result $\vec{x}'_r$ remains unchanged:

$$\text{View}^{\text{Hyb4}}_{\text{Sim}} = \left\{\{\vec{x}_{i,r}\}_{i\in[C_\mathcal{A}]}, \{m_i\}_{i\in[C_\mathcal{A}]}, \{\vec{y}''_{i,r}\}_{i\in[C^{\text{on}}/C_\mathcal{A}]}, \vec{x}'_r\right\}$$

Hybrid 5. In this hybrid step, Sim assumes that $\mathcal{A}$ may attempt to cheat by submitting multiple sets $\{C_\mathcal{A}\}$. If inconsistencies are detected during this process by BFT consensus, the protocol execution is aborted. Thus, the view becomes:

$$\text{View}^{\text{Hyb5}}_{\text{Sim}} = \begin{cases} \perp, & \text{if } \{C_\mathcal{A}\} \text{ is inconsistent} \\ \text{View}^{\text{Hyb5}}_{\text{Sim}}, & \text{otherwise} \end{cases}$$

Hybrid 6. Finally, Sim relies entirely on the virtual data obtained from the ideal functionality $\mathcal{F}$ to generate all necessary intermediate results and the final output, without utilizing the real input data or masks. Specifically, the ideal functionality $\mathcal{F}$ receives as input the online set of clients C^{on}, the local

models $\{\vec{x}_{i,r}\}_{i\in[C^{\text{on}}]}$ from the online clients C^{on}, the malicious aggregator number f and malicious client number f'. The output $\vec{x}''_r$ from $\mathcal{F}$ is then used to simulate the view. In this hybrid, the view can be expressed as:

$$\text{View}_{\text{Sim}}^{\text{Hyb6}} = \left\{\{\vec{x}_{i,r}\}_{i\in[C_{\mathcal{A}}]}, \{m_i\}_{i\in[C_{\mathcal{A}}]}, \{\vec{y}''_{i,r}\}_{i\in[C^{\text{on}}/C_{\mathcal{A}}]}, \vec{x}''_r\right\}$$

In this final hybrid, the adversary's view is indistinguishable from the previous hybrid steps, ensuring that no additional information beyond what is allowed by $\mathcal{F}$ is revealed to $\mathcal{A}$. By the security properties of HPRF, BFT, and VSS, the view in Hybrid 6 is indistinguishable from that in Hybrid 6. Hence, $\text{View}_{\mathcal{A}}^{\Pi} \approx \text{View}_{\text{Sim}}^{\text{Hyb6}} = \text{View}_{\text{Sim}}^{\mathcal{F}}$, concluding the proof.

Through these hybrid steps, we have demonstrated that the execution of the Aion protocol under the control of no more than f malicious aggregators and f' malicious clients each round is indistinguishable from the ideal execution of the functionality $\mathcal{F}$. Consequently, this implies that the adversary cannot gain any additional information beyond the final aggregated result, thus ensuring the privacy and security of the client data throughout the entire protocol execution. □

In the following, we discuss how Aion resists collusion attacks and gradient isolation attacks.

Defenses against collusion attacks. Aion ensures security against collusion between aggregators or the server and up to $q-2$ clients through the following aspects. (1) When the aggregated secret M is recovered and the adversary compromises $q-2$ clients, the remaining honest keys m_j and m_k satisfy $m_j + m_k \equiv M - \sum_{i=1}^{q-2} m_i = c \mod p$, with a solution space $\mathcal{S} = \{m_j, m_k \in \mathbb{Z}_p \mid m_j \equiv c - m_k\}$. All possible solutions for m_j and m_k are uniformly distributed in $\mathbb{Z}_p$, making it impossible for the adversary to determine their exact values. (2) Among the aggregators, at most f can be malicious. Since reconstruction requires at least $f+1$ valid shares, the secret of any honest client cannot be recovered.

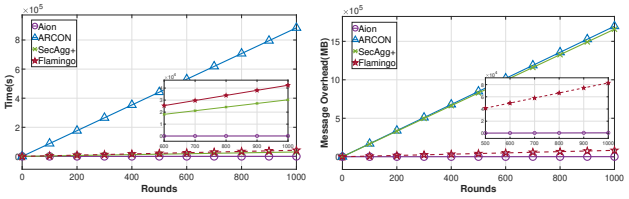
Defenses against gradient isolation attacks. Pasquini et al. [48] propose a method to bypass secure aggregation and obtain client training data, which we term *gradient isolation attacks*. Its core strategy is that the server sends distinct models to different clients to infer the gradient information. Specifically, in [48], the server sends a normal model to a target client while distributing modified models to other clients, termed as non-target clients. The modified models force non-target clients to output zero gradients , so the aggregated result corresponds solely to the target client's model (or gradient). However, Aion can effectively mitigate such attacks. In Aion, BFT consensus by multi-aggregators is enforced on the global model in each round (Algorithm 2, Line 20) to ensure the consistency of the model. Clients can detect model tampering by verifying the BFT result (at least $n-f$ aggregators' signatures on the model), thereby resisting gradient isolation attacks.

B Additional Experiment Results

Performance comparison on different datasets including end-to-end overhead. We evaluate the performance of masking-based schemes (including Aion, Flamingo [14], ACORN [13], SecAgg+ [12], and SecAgg [2]) using diverse real-world datasets with $q = 256$, $n = 8$, as shown in Table 3. In addition to reporting the time and communication costs of the initialization and aggregation phases, we also measured the end-to-end performance per round, which includes one round of local training and one execution of the SA protocol. Note that communication costs exclude model parameters since their inclusion would obscure differences between schemes.

Results demonstrate Aion's significant advantage in end-to-end performance, particularly as model size increases. On EMNIST-Byclass with 6700k params, Aion completes each round in 74.05 seconds (including 63.57s for training), outperforming Flamingo by 33.80%, ACORN by 57.03%, SecAgg+ by 26.19%, and SecAgg by 77.51%. As model size grows, Aion's initialization overhead remains stable since it primarily involves VSS protocols independent of model dimensions. Aion also excels in aggregation: EMNIST-Byclass tests show its computation time on the client side is 0.66s, which is 2.69× faster than Flamingo's, 4.97× than ACORN's, 4.34× than SecAgg+'s, and 152.38× than SecAgg's. Similarly, Aion's aggregator computation is 2.22s, which outperforms others by factors of 17.34 (Flamingo), 48.81 (ACORN), 11.46 (SecAgg+), and 69.70 (SecAgg).

Time cost and message overhead with successive rounds. To evaluate the cumulative impact of various parameters on performance, we set $q = 4096$, $n = 8$, and test Aion over 1000 training rounds. The time cost and message overhead are shown in Figure 11. It shows that Aion achieves an execution time of 188.05 seconds with a message overhead of 928.59 MB. In contrast, Flamingo requires 42367.02 seconds and 82876.89 MB, resulting in execution times and message overheads that are 225.30× and 89.25× those of Aion, respectively.



(a) Total Time v.s. r (b) Total Overhead v.s. r

Figure 11: Time cost and message overhead with varying r.

Time cost and message overhead ratios across different steps. As shown in Figure 12, we plot the time cost and message overhead ratio for each step with $q = 1024$ and $n = 8$, in order to better show the performance of each step.

For the initialization phase, the majority of the time is consumed by the key sharing, which takes 2.019 seconds, accounting for approximately 99.41% of the total time, while valid clients confirmation requires only 0.012 seconds (0.59% of the phase's duration). In terms of message overhead, key sharing also incurs the highest cost, with a message size of

Table 3: Performance comparison on different datasets and schemes including end-to-end overhead.

Dataset (Model)	Scheme	Client: ms (KB)		Aggregator: ms (KB)		End-to-end (per round)	
		Initialization	Aggregation	Initialization	Aggregation	Runtime (s)	Comm. (MB)
FMNIST (LeNet5) 61k params 44 rounds training time per round 7.90 s	Aion	7.00 (0.94)	30.01 (0.11)	52.00 (102.88)	34.01 (36.20)	15.56	0.60
	Flamingo	67.10 (2.22)	102.91 (7.78)	132.00 (17.40)	5160.00 (1545.71)	21.15	5.66
	ACORN	202.42 (0.95)	606.77 (2.72)	356.71 (1386.83)	50500.00 (7279.44)	67.60	9.34
	SecAgg+	235.06 (0.95)	585.36 (2.62)	404.09 (1385.63)	915.21 (7197.68)	18.00	9.27
	SecAgg	358.08 (8.38)	69690.39 (31.11)	398.09 (5460.75)	46107.58 (14175.39)	133.73	29.05
SHAKESPEARE (LSTM) 818k params 23 rounds training time per round 257.34 s	Aion	7.00 (0.94)	85.02 (0.11)	52.00 (102.88)	331.08 (36.67)	265.36	0.60
	Flamingo	67.10 (2.22)	329.32 (7.78)	132.00 (17.40)	11437.18 (1548.87)	277.10	5.42
	ACORN	208.31 (0.94)	841.87 (2.71)	356.93 (1378.92)	51696.18 (7248.34)	318.48	9.42
	SecAgg+	241.06 (0.95)	790.48 (2.59)	393.09 (1388.13)	3476.86 (7110.33)	270.21	9.18
	SecAgg	328.40 (8.21)	66634.85 (30.69)	376.33 (5333.45)	51321.03 (13953.64)	385.33	28.56
CIFAR10 (ResNet18) 2797k params 173 rounds training time per round 30.52 s	Aion	7.00 (0.94)	279.07 (0.11)	52.00 (102.88)	1096.08 (36.73)	39.50	0.60
	Flamingo	67.10 (2.22)	865.13 (7.78)	132.00 (69.59)	25144.81 (1544.19)	64.53	5.27
	ACORN	213.74 (0.95)	1754.7 (2.73)	388.92 (1395.18)	77819.42 (7308.76)	118.70	9.38
	SecAgg+	228.05 (0.96)	1378.87 (2.68)	386.58 (1417.41)	8748.41 (7344.74)	49.26	9.46
	SecAgg	349.08 (8.18)	74023.84 (30.78)	383.09 (5308.09)	73895.28 (13967.84)	188.48	28.56
EMNIST-Byclass (ResNet9) 6700k params 34 rounds training time per round 63.57 s	Aion	7.00 (0.94)	662.02 (0.11)	52.00 (102.88)	2220.71 (36.53)	74.05	0.60
	Flamingo	67.10 (2.22)	1780.20 (7.78)	132.00 (69.59)	38514.17 (1548.87)	111.85	5.51
	ACORN	312.55 (0.97)	3288.92 (2.73)	542.55 (1386.51)	108390.19 (7278.48)	172.34	9.38
	SecAgg+	345.08 (0.95)	2875.85 (2.64)	543.87 (1408.36)	25446.12 (7308.57)	100.49	9.41
	SecAgg	347.08 (8.34)	100881.33 (31.30)	423.10 (5435.23)	154791.48 (14226.09)	329.28	29.11

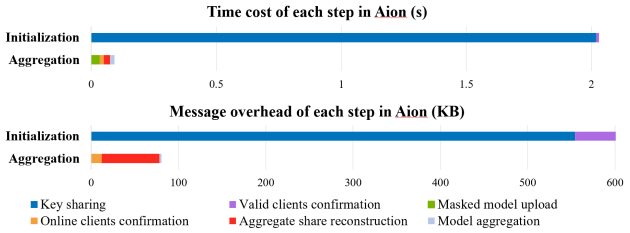
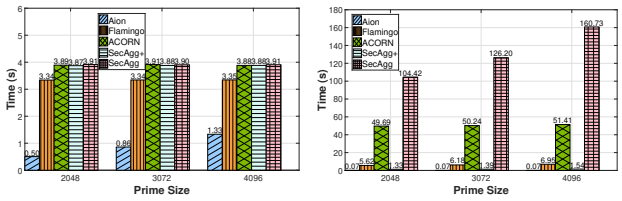


Figure 12: Time cost and message overhead of each step.

554.13 KB, representing 88.20% of the total communication load in this phase, whereas valid clients confirmation accounts for 74.14 KB (11.80% of the communication cost). In the aggregation phase, the time cost and message overheads are more evenly distributed across multiple steps. Masked model upload has the highest time consumption in this phase at 0.034 seconds, comprising 36.56% of the total time. This is followed by aggregate share reconstruction at 0.025 seconds (26.88%), model aggregation at 0.018 seconds (19.35%), and online clients confirmation at 0.016 seconds (17.20%). For message overhead, aggregate share reconstruction contributes the most with 66.11 KB, accounting for 82.21% of the aggregation phase's communication cost, while online clients confirmation adds 12.02 KB (14.95%, and model aggregation has the smallest message size at 2.28 KB (2.84%).

Impact of security parameter p on performance. We evaluate the performance impact of secret sharing modulus p (set to 2048 bits, 3072 bits, and 4096 bits respectively) on Aion, SecAgg, SecAgg+, ACORN, and Flamingo under consistent parameters ($q = 256$, $n = 8$). Figure 13 shows that Aion maintains strong performance as p increases. When $p = 4096$ bits, Flamingo, ACORN, SecAgg+, and SecAgg require 2.52, 2.92, 2.92, and 2.94 times longer initialization time than Aion, while their aggregation times are 80.29, 734.43, 22.00, and 2296.14 times longer for each round, respectively.

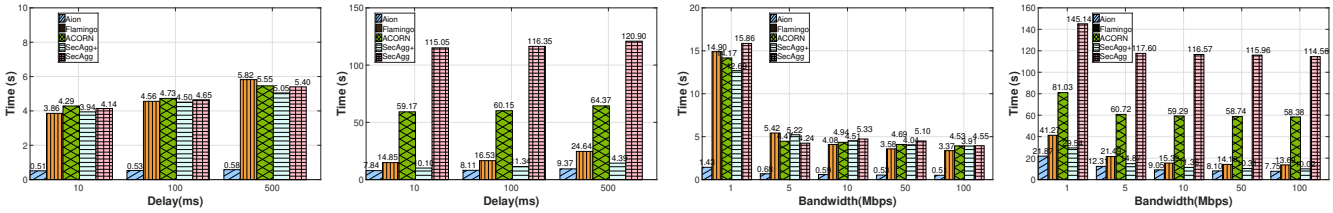


(a) Init. Time vs. p (b) Agg Time vs. p

Figure 13: Time with varying security parameter p.

In initialization, Aion's dominant cost is only each client's secret sharing, which is sensitive to p. Flamingo's comes from distributed key generation, while others mainly involve pairwise DH key exchanges. During aggregation, Aion only needs to perform secret reconstruction once, so the overhead remains almost unchanged as p increases. ACORN spends the most time on ZK proofs, and both Flamingo and SecAgg+ incur mask reconstruction costs. SecAgg+ is faster as it only reconstructs secrets within each aggregator's neighborhood. SecAgg shows the highest overhead as it requires all clients to recover secrets from all others.

Performance under varying network conditions. We set $q = 256$ and $n = 8$ and test Aion's time overhead of communication combined with computation under varying network bandwidth and latency conditions, as shown in Figure 14. The results demonstrate Aion's consistently strong performance under extremely low bandwidth constraints (poor network conditions). At 1 Mbps bandwidth, the initialization time of Aion is only 1.43 seconds, while Flamingo, ACORN, SecAgg+, and SecAgg require 10.42, 9.91, 8.87, and 11.09 times longer, respectively. For aggregation, Aion completes in 21.87 seconds, whereas the other schemes take 1.89 (Flamingo), 3.71 (ACORN), 1.35 (SecAgg+), and 6.64



(a) Init. Time v.s. latency (b) Agg. Time v.s. latency (c) Init. Time v.s. bandwidth (d) Agg. Time v.s. bandwidth

Figure 14: Time with varying network conditions.

(SecAgg) times longer.

Under high latency conditions of 500 ms, Aion maintains its efficiency with 0.58 seconds initialization time, outperforming Flamingo (10.03×), ACORN (9.41×), SecAgg+ (8.71×), and SecAgg (9.31×). Its aggregation time of 9.37 seconds remains significantly faster than competitors, which require 2.63× (Flamingo), 6.87× (ACORN), 1.54× (SecAgg+), and 12.90× (SecAgg) longer. These results confirm that Aion is better suited than other masking-based schemes for network environments with latency and bandwidth constraints.

Overhead of Mask Creation and Reconstruction. We set $q = 4096$, $n = 8$, $r = 1000$ and test the runtime of mask creation and reconstruction for masking-based schemes, as shown in Table 4. For a fair comparison, we assume all protocols use VSS based on Feldman commitment. The results show that Aion does have significant advantages in both mask creation and reconstruction. The time cost and communication overhead account for the total across all training rounds. The overhead of SecAgg is relatively high because secret sharing and reconstruction are performed among all clients, rather than within local neighborhoods as in other schemes.

Table 4: Mask creation and reconstruction performance.

Scheme	Mask Creation		Reconstruction	
	Runtime (s)	Comm. (GB)	Runtime (s)	Comm. (GB)
Aion	22.00	0.01	34.01	0.47
Flamingo	155.77	48.15	38103.16	26.26
ACORN	569.70	541.50	9044.61	1102.45
SecAgg+	568.50	554.24	10752.41	1077.41
SecAgg	1.71×10^8	1.66×10^4	7.36×10^7	2.15×10^4

Performance of different types of SA schemes. We evaluate Aion's end-to-end performance compared to different types of SA schemes, including OT-based ELSA [15] and ThHE-based Mario [16]. The experiment is conducted with $q = 256$ clients, where Aion and Mario use 1 server with $n = 8$ aggregators, while ELSA uses 2 servers. Performance is measured on the FMNIST dataset. As shown in Table 5, ELSA demonstrates 6.15× longer runtime and 16.93× higher communication overhead compared to Aion. Mario requires 5.71× longer runtime and 135.00× higher communication overhead than Aion. This indicates that Aion's implementation of the masking-based SA scheme using VSS is lightweight.

Table 5: Performance of different types of SA schemes.

Scheme	Client: s (KB)	Aggregator: s (MB)	End-to-End	
			Runtime (s)	Comm. (MB)
Aion	0.03 (0.11)	0.03 (0.04)	15.56	0.60
ELSA	0.50 (3.60)	82.68 (9.26)	95.66	10.16
Mario	45.94 (268.53)	26.57 (13.82)	88.87	81.00

Performance of input validation on large datasets. We further evaluate Aion's robustness against poisoning attacks on large-scale datasets, as shown in Figure 15. Specifically, we adopt the EMNIST-Byclass dataset, which contains approximately 700,000 training samples, and the ResNet9 model with 6.598 million trainable parameters. Experimental results demonstrate that Aion consistently achieves an ASR below 2.6% for poisoning ratios under 50%, indicating that Aion remains resilient to poisoning attacks, even in scenarios involving substantially larger datasets and models.

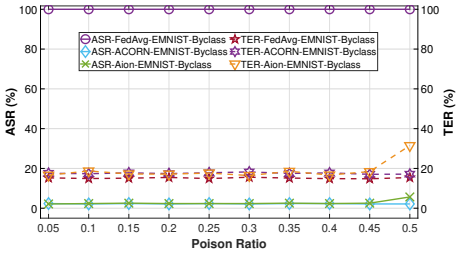


Figure 15: ASR/TER vs. Poison Ratio (EMNIST-Byclass).

C Related Work

Masking-based SA schemes. Bonawitz et al. [2] first introduced a pairwise masking technique involving shared random keys between pairs of clients using the DH key exchange. Each client derives pairwise masks from these shared keys and an individual mask to its input vector. For each client, other clients will recover its individual mask via secret reconstruction if online, or the pairwise masks if offline. Bell et al. [12] partitioned the client set into multiple neighborhoods, where both secret sharing and reconstruction are performed in the neighborhood. Based on this, ACORN [13] implements input validation with L_2 norm and ZK. Flamingo [14] treats pairwise masks as long-term secrets and uses a PRF to gener-

ate fresh seeds each round, ensuring mask uniqueness without repeated DH key exchanges. Clients encrypt per-round seeds for pairwise masks through threshold encryption. However, each client still has to perform secret sharing in each round to allow the reconstruction of individual masks. The decryptors act as aggregators and reconstruct q secrets of online clients.

Other types of SA schemes. Zhang et al. [11] proposed an accelerated HE-based SA scheme, reducing communication overhead for cross-organization scenarios. Mario [16] uses ThHE to implement an efficient SA scheme that supports input validation and dynamic servers. However, HE is computationally intensive and may slow down the training process. MPC can also achieve accurate aggregation of models in the SA process [15, 49, 50], but it often results in high computational and communication overhead.

Anti-poisoning attacks schemes using the L_2 norm. In FL, poisoning attacks amplify adversarial impact. Suresh et al. [28] showed that for fewer adversaries, imposing an L_2 norm bound on updates can mitigate poisoning. However, as benign updates shrink during training, a fixed norm becomes less effective. Guo et al. [29] proposed a dynamic norm to balance defense and convergence, avoiding over-clipping benign updates. RoFL [30] optimizes the norm dynamically but incurs overhead from secure aggregation with ZK proofs. Naseri et al. [31] further combined L_2 clipping with Gaussian noise or differential privacy to neutralize backdoors without significantly degrading model performance.

D Complexity Analysis

Theorem 2 (Communication complexity between aggregators)**.** *In Aion, the communication complexity between n aggregators is $O(n)$.*

Proof. In Aion, there are three components that require communication between aggregators, including the BFT protocol, ASR or AMR algorithms for mask reconstruction, and the view-change mechanism. We discuss them categorically.

BFT consensus. Let C_{BFT} denote the communication complexity of BFT. Aion adopts HotStuff [37] with linear message complexity. Each aggregator sends or receives a constant number of messages (quorum certificates, proposals) of size $O(1)$. The leader aggregates $2f+1$ signatures into a single threshold signature, reducing message size from $O(n)$ to $O(1)$. Therefore, for n aggregators, $C_{\text{BFT}} = O(n)+O(n) = O(n)$.

Mask reconstruction (ASR/AMR). Define C_{ASR} and C_{AMR} as the complexity of aggregated share (mask) reconstruction, which can be decomposed into two steps: share collection and result broadcast. (1) Share collection: The leader collects shares from $f+1$ aggregators. Each share is $O(1)$, total $O(f+1) = O(n)$ (since $f < n/3$); (2) Result broadcast: The leader sends the reconstructed M_r or HPRF(M_r) to n aggregators, requiring n messages. Therefore, for both algorithms, $C_{\text{ASR}} = C_{\text{AMR}} = O(n)+O(n) = O(n)$.

View-change. If the leader fails, the view-change mechanism incurs, where all honest aggregators broadcast view-change information to the new leader. The new leader then generates a proposal based on the highest view and broadcasts it. Thus, $C_{\text{view-change}} = O(n)+O(n) = O(n)$.

Overall, the communication complexity between aggregators in Aion is $C_{\text{aggs}} = C_{\text{BFT}} + C_{\text{ASR}} + C_{\text{view-change}} = O(n)$. □

Theorem 3 (Communication complexity between clients and aggregators)**.** *In Aion, the communication complexity between q clients per training round and n aggregators is $O(nq)$.*

Proof. We analyze the communication between q clients and n aggregators in both initialization and aggregation phases.

Initialization phase. Each of q clients C_i generates a single secret m_i and distributes shares via VSS to all n aggregators, yielding $O(nq)$ complexity. For n aggregators, they broadcast the BFT-committed initial model to the clients, with a complexity of $O(nq)$. Therefore, for the initialization phase, $C_{\text{initial}} = O(nq)+O(nq) = O(nq)$.

Aggregation phase. Each client C_i uploads a masked model to the aggregators. Then, aggregators broadcast the BFT-committed global model to clients. For the aggregation phase, $C_{\text{aggregate}} = O(nq)+O(nq) = O(nq)$.

Overall, the communication complexity between q clients and n aggregators is $C_{\text{clts-aggs}} = C_{\text{initial}} + C_{\text{aggregate}} = O(nq)$. □

E Privacy Leakage Probability without CCS

In Section 4.3, we indicate that when considering two rounds of client sets, if the CCS algorithm is not invoked so that one online client set is a subset of the other with a size difference of one (referred to as event A), this could lead to client privacy leakage. Let the total number of clients be N. In each round, q clients are randomly sampled to participate in training ($q \le N$). Each sampled client goes offline with probability $\eta \in [0,1]$.

Let the online sets of two rounds be C_1 and C_2. The probability of C_1 and C_2 having k identical clients is $P_{\text{inter}}(k) = \binom{q}{k}\binom{N-q}{q-k}/\binom{N}{q}$. The occurrence of event A requires exactly one client to have different online statuses in C_1 and C_2. The probability of a single client changing from offline to online is $\eta(1-\eta)$, and from online to offline is $(1-\eta)\eta$, so the total probability of a state change is $2\eta(1-\eta)$. Meanwhile, for each client in the intersection set, the probability of maintaining unchanged statuses across both rounds is $(1-\eta)^2+\eta^2$.

We decompose event A into $A_1 \cup A_2$ where:

- A_1: All $2(q-k)$ non-overlapping clients are offline; while among k overlapping clients, 1 client shows different statuses between rounds, and $k-1$ clients maintain identical statuses in both rounds. $P(A_1)$ is calculated as:

$$P(A_1) = \sum_{k=1}^{q} P_{\text{inter}}(k) \cdot \eta^{2(q-k)} \cdot \binom{k}{1} \cdot 2\eta(1-\eta) \cdot [(1-\eta)^2+\eta^2]^{k-1}$$

- A_2: Among $2(q-k)$ non-overlapping clients, 1 client is online, and $2(q-k)-1$ clients are offline; while k overlapping clients maintain identical statuses. It holds:

$$P(A_2) = \sum_{k=1}^{q} P_{\text{inter}}(k) \cdot \binom{2(q-k)}{1} \cdot (1-\eta) \cdot \eta^{2(q-k)-1} \cdot [(1-\eta)^2+\eta^2]^{k}$$

As A_1 and A_2 are mutually exclusive, we have $P(A) = P(A_1 \cup A_2) = P(A_1) + P(A_2)$. Assuming $\eta = 10\%$, $q = 5$, $N = 100$, it yields $P(A) < 10^{-8}$. However, when $q = 5$, $N = 10$, $P(A) = 1.09 \times 10^{-2}$, indicating a 1.09% probability of client privacy leakage, which is unacceptable. Therefore, it remains necessary to employ the CCS algorithm to efficiently sort clients into non-overlapping subsets.

F Evaluation of Privacy Protection

In FL, since client data remains local and is never directly accessed by the server, adversaries seeking to extract private information rely on gradient inversion attack (GIA) [51]. In GIA, an attacker initializes a random input and iteratively updates it so that its gradients approximate those uploaded by the client. When successful, this process can yield reconstructed inputs that closely resemble the client's original data. Existing FL studies [52, 53] typically evaluate the privacy-preserving capability of their schemes by analyzing their resistance to GIA. To defend against such attacks and protect privacy, noises (masks) can be applied to the gradients of the clients, introducing bias into the gradients obtained by the attacker and thereby reducing reconstruction accuracy. The magnitude of noise directly affects gradient distortion, which in turn influences the attack's reconstruction performance. In our experiments (Section F.2), we demonstrate that Aion's scaling masks can defend against GIA, proving that Aion's noise level realizes privacy protection. In the following, we first provide a theoretical analysis of the guarantees offered by the masking mechanism. We then conduct empirical evaluations to assess its robustness against GIA , thereby validating its capability to safeguard client data privacy.

F.1 Theoretical Analysis

Cheng et al. [54] theoretically prove that adding proper Gaussian noise to gradients can protect the privacy of the clients. Meanwhile, Zhu et al. [52] demonstrate that adding $\mathcal{N}(0, 0.01)$ Gaussian noise with mean $\mu(\mathcal{N}) = 0$ and variance $\sigma^2(\mathcal{N}) = 0.01$ to gradients successfully resists GIA. In Aion, the masks generated by clients follow a uniform distribution $\mathcal{U}(-a, a)$, where a depends on the make ratio β and the magnitude of the previous global gradient. Following the same configuration as in [52], when setting β to 10%, the range of a is 0.254 to 0.686 across training rounds. For the variance, $\sigma^2(\mathcal{U}) = \frac{a^2}{3} \in (0.022, 0.157) > \sigma^2(N)$. For the differential entropy, the entropy of $\mathcal{N}$ is $H(\mathcal{N}) = \frac{1}{2}\ln(2\pi e\sigma^2(\mathcal{N}))$, and the entropy of $\mathcal{U}$ is $H(\mathcal{U}) = \frac{1}{2}\ln(12\sigma^2(\mathcal{U}))$. Substituting $\sigma^2(\mathcal{N})$ and $\sigma^2(\mathcal{U})$, we obtain $H(\mathcal{N}) = ln(0.171)$ and $H(\mathcal{U}) \in (ln(0.264), ln(1.884)) > H(\mathcal{N})$. In information theory, higher entropy indicates greater randomness. The masks added to the gradients introduce reconstruction errors during the inversion process, impeding the attacker's ability to recover the original input. Notably, the masks generated in Aion exhibit higher entropy and greater randomness, making it more difficult for adversaries to perform accurate gradient matching. As a result, Aion demonstrates strong resilience against GIA and effectively enhances privacy protection.

F.2 Experimental Verification

We conduct experiments in resisting GIA to further demonstrate Aion's privacy protection capability. The experiments are carried out under a strong attack setting, where the attacker is assumed to have access to both private labels and batch normalization statistics [53]. To preserve model utility, the mask ratio β is set to 10%. Figure 16a shows the original data. Figure 16b and Figure 16c show the reconstructed images from masked gradients at round 20 and 48 respectively. The results indicate that the masking strategy of Aion effectively mitigates GIA, and reconstructed images become increasingly blurred as training progresses.

(a) Original (b) Aion round 20 (c) Aion round 48

Figure 16: Images reconstructed from local updates.

G Discussion

Supporting for dynamic aggregators. In practical deployments, aggregators may dynamically join or leave. This involves securely transferring shares from old aggregators to new ones while ensuring each client's total secret remains identical and confidential. For each client secret m, an aggregator distributes random values r and r' (where $r = r'$) to the old and new aggregator committees S and S', respectively, using distinct polynomials for secret sharing. S and S' can differ in size. As a result, each aggregator $S_j \in S$ holds secret shares $[m]^j$ and $[r]^j$, and each aggregator $S_j{}' \in S'$ holds $[m']^j$ and $[r']^j$. Each S_j broadcasts $[m]^j + [r]^j$, allowing reconstruction of $m + r$, which is then sent to S'. Then each $S_j{}'$ can compute the new share $[m']^j = m + r - [r']^j$.

Real-world deployment. In Aion, aggregators can be deployed on edge nodes or dedicated client devices. A critical challenge involves that in the real world, multi-aggregator deployments may suffer from high latency or unstable networks. Aion can address this by integrating partially synchronous and asynchronous consensus. Under normal network conditions, multiple aggregators leverage the partially synchronous BFT protocol (e.g., HotStuff [38]) to achieve efficient model aggregation . In cases of severe network fluctuations or partitions, Aion can switch to an asynchronous consensus (e.g., Dumbo [55]), ensuring eventual consistency even if some aggregators remain offline for extended periods.