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

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. This reduces the secret sharing times from rq to q (r for training round number and q for client number) 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 Aion outperforms the state-of-the-art Flamingo (S&P'23) by a factor of 214.67 in speed while achieving a 98.05% 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 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 pseudorandom 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 own secret m_i to its data. Similarly, the individual mask must also be shared in advance to enable its reconstruction.

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[◇]	0	0	—	✗	—	✗	✗	✗
SecAgg [2]	$2rq$[♮]	q (pariwise+individual)	$O(q^3)$	✗	—	✗	✗	✗
SecAgg+ [12]	$2rq$	u (pariwise+individual)	$O(u^3)$	✗	—	✗[ℵ]	✓	✗
ACORN [13]	$2rq$	u (pariwise+individual)	$O(u^3)$	✓	✗	✓	✓	✗
Flamingo [14]	rq[♭]	u (pariwise+individual)	$O(n^2q)$[★]	✗	—	✗	✓	✗
ELSA [15]	rq	— (Boolean SS + OT)	—	✓	✓[†]	✓	✓	✗
Mario [16]	—	— (Threshold HE)	$O(n^2)$	✓	✗	✓	✓	✓
Aion	$\boldsymbol{q}$	1 (single)	$O(n)$	✓	✓	✓	✓	✓

[◇] "clear" indicates the normal federated learning without any privacy protection measures.

[♮] 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 ($u \leq q$).

[ℵ] "✗" indicates a semi-honest client, who tries to infer private information but follows the protocol and avoids active attacks, like poisoning attacks.

[♭] 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 E for detailed complexity analysis.

[†] In Elsa, the system use oblivious transfer to ensure the input validation.

Existing SA schemes 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 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 (neighborhood size) masks. 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[2] 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 to q and supporting multiple rounds of secret usage. Moreover, each client only needs to add a single mask, compared to 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 $O(n^2q)$ 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 range and L_2 norm bound evolve to accommodate the updated model. In particular, our evolving input validation is ZK-free

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

[2] The aggregators can be implemented as servers (as in Mario [16]) or as selected clients (as in Flamingo [14]).

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, showing that Aion significantly improves secure aggregation efficiency. With 4096 clients and 8 aggregators, Aion outperforms Flamingo by a factor of 214.67 in speed, achieving a 98.05% reduction in message overhead. We also evaluate the evolving input validation mechanism, showing that Aion effectively mitigates poisoning attacks, leading to higher model accuracy. In experiments with the CIFAR10 and FMNIST 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]. 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, homomorphism is in need for mask computing, so Homomorphic PRG (HPRG) can be used. Each client computes $\text{HPRG}(m_i)$ and the aggregator calculates $\text{HPRG}(M)$ after recovering M. The *sum of mask* is obtained since $\text{HPRG}(M) = \text{HPRG}\left(\sum m_i\right) = \sum \text{HPRG}(m_i)$ holds. In this way, only the aggregated mask $\text{HPRG}(M)$ is recovered without revealing individual client masks $\text{HPRG}(m_i)$, preventing model leakage even if a client goes offline. However, using the same mask $\text{HPRG}(m_i)$ 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 $\text{HPRF}(m,r)$, and by key homomorphism, we derive that $\sum \text{HPRF}(m_i,r) = \text{HPRF}\left(\sum m_i, r\right) = \text{HPRF}(M,r)$. 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 $\text{HPRF}([M]^j,r)$, allowing the reconstruction of the aggregated mask $\text{HPRF}(M,r) = \text{HPRF}\left(\sum \lambda_j [M]^j, r\right) = \sum \lambda_j \text{HPRF}([M]^j,r)$. 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 [25, 26, 28–30] to prevent various attacks [31–33]. 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 $\vec{x} + \text{HPRF}(m,r)$. 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 $\text{HPRF}(M,r)$ 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 [34] 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: $f(x) = a_0 + a_1 x + \cdots + a_{t-1}x^{t-1} \mod p, \quad a_0 = m$. For each P_j, the share is computed as $[m]^j = f(x_j) \mod p$ (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, dealer sends $([m]^j, Com)$ to it, where $Com = (Com_0, Com_0, Com_1, \ldots, Com_{t-1})$.

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. Given an HPRF $F: \chi \rightarrow \gamma$, where $(\chi, \oplus)$ and $(\gamma, \otimes)$ are groups, if it satisfies: $F(m_1, r) \otimes F(m_2, r) = F(m_1 \oplus m_2, r)$ for every key m with a public input where r is a public input string, it is key-homomorphic. A key HPRF [27] based on the Learning with Errors (LWE) problem can be constructed as: $F(m,r) = \langle \mathbf{A}\cdot m + \mathbf{e}, r\rangle \pmod q$, where n, l, q are public parameters, $\mathbf{A}$ is a matrix randomly chosen from $\mathbb{Z}_q^{n\times l}$, l is a secret vector from $\mathbb{Z}_q^l$, $\mathbf{e}$ is a small error vector, and $r \in \mathbb{Z}_q^n$ is the input. 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 [35] is usually run by a leader and several normal nodes to process proposals through interactive votes. We adopt a lightweight version [36] of the HotStuff protocol [37], 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 $O(n)$.

2.4 Verifiable Random Function

Verifiable Random Function (VRF) [38, 39] 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 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 $pk = g^{sk}$.

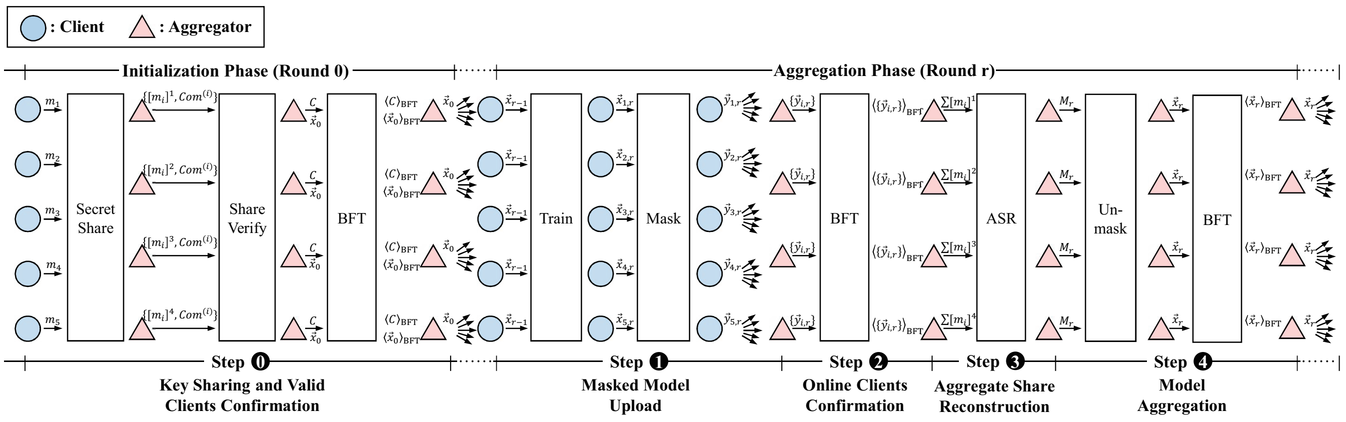
Randomness generation: $\mathsf{VRF.Prove}_{sk}(x) \rightarrow (y, \pi(y))$. Given an input x, the generator computes a randomness $y = e(g,g)^{1/x+sk}$ and its proof $\pi(y) = g^{1/(x+sk)}$.

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 $y = e(g, \pi)$. 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 [40]) and n *aggregators* (typically a few or several dozen, e.g., $n = 16$). There can be a server to issue training task and utilize the final model. 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



: Client
: Aggregator
Initialization Phase (Round 0)
Aggregation Phase (Round r)
Secret Share
Share Verify
BFT
Train
Mask
BFT
ASR
Un-mask
BFT
Step ⓿
Key Sharing and Valid Clients Confirmation
Step ❶
Masked Model Upload
Step ❷
Online Clients Confirmation
Step ❸
Aggregate Share Reconstruction
Step ❹
Model Aggregation

Figure 1: The system flow of Aion.

the masked model (or gradient, which will be referred to as the model for simplicity) to the aggregator. 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 aggregator. 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.

Typical deployment scenario. Aion is well-suited for large-scale secure aggregation applications, and the typical deployment scenarios include mobile keyboards [41], healthcare [42], and wearable devices [43].

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*, meaning that message latency between honest nodes is within Δ, though Δ is unknown. 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 a digital signature nor break an encryption scheme. 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, the adversary can control at most $q-2$ malicious clients.

3.3 System Goal

Security. The system must ensure the security property, which means the privacy information of honest clients will not be obtained by any participants (aggregators or clients). Privacy 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 ⓿). 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).

In Section 4.1 and 4.2, we detailed the initialization and aggregation phases. In Section 4.3, we design a client concealed sortition mechanism that allows clients to secretly and randomly determine their training rounds. In Section 4.4, we present an alternative aggregated mask reconstruction algorithm, which can recover the overall mask value without exposing the aggregated keys. Additionally, an input validation mechanism is introduced in Section 5. We also design pluggable robustness enhancement constructions, including the HPRF error elimination algorithm (Section 6.1) and asynchronous federated learning (Section 6.2).

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 ⓿)
    ▸ As a client $C_i \in C_0$:
 1: generate a random secret value $m_i$
 2: run Share$(m_i, f+1, n) \to (\{[m_i]^j\}_{j\in[S]}, Com^{(i)})$
 3: for $j \in [S_j]$ 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 Verify$([m_i]^j, Com^{(i)}) \to 1$ then
 7:         put client $C_i$ into set $C$; store $[m_i]^j$
 8: run BFT$(C)$ to commit the valid client set $C = \langle C \rangle_{\mathrm{BFT}}$
 9: broadcast $\vec{x}_0$ to each client in $C$
10: start round $r+1$
    ▸ As a client $C_i \in C_0$:
11: run CCS$(\vec{x}_0, K) \to (C^{(k)}, \pi(i))$    ▷Algorithm 4, concealed sortition
```

4.1 Initialization Phase

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

Step ⓿ : 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 $\vec{x}_0$ to each client in C to start the training (Lines 9-10). Each aggregator S_j stores valid clients' shares $\{[m_i]^j\}_{i\in[C]}$.

4.2 Aggregation Phase

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

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 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 $C_r^{\mathrm{on}}$
 8:     else
 9:         put $C_i$ into the offline set $C_r^{\mathrm{off}}$
10: run MGF$(C_r^{\mathrm{on}}) \to C_r^{\mathrm{valid}}$    ▷ Algorithm 6, input validation
11: set $C_r^{\mathrm{on}} = C_r^{\mathrm{valid}}$
12: run 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 ASR$(\{[M_r]^j\}, f+1) \to M_r$    ▷ 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
    can be eliminated by Algorithm 8
18: run EMA$(C_{r-1}^{\mathrm{off}}) \to \vec{x}'_{r-1}$    ▷ Algorithm 9, asynchronous FL
19: set $\vec{x}_r = \vec{x}_r + \omega \vec{x}'_{r-1}$    ▷ $\omega$ is a scaling factor
20: run 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
```

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 $\vec{y}_{i,r} = \vec{x}_{i,r} +$

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 $g^{[M_r]^j} = \prod_{i \in [C_r^{\mathrm{on}}]} \prod_{k \in [t]} (Com_k^{(i)})^{x_j^k} \mod p$ 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 $\perp$
```

$\mathsf{HPRF}(m_i, r)$ to an aggregator $S_j \in S$ (Lines 3-4).

Note that to ensure that a client's input is counted in the online client set, a client can send a request to $n-f$ aggregators to confirm that his input is received to defend against malicious aggregators. To ensure reliable delivery, a client can also choose to utilize erasure coding [44] to divide its input and send $n-f$ shares to different aggregators to reduce message complexity. Then each aggregator can recover each client's inputs.

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 $\mathsf{MGF}(C_r^{\mathrm{on}})$ (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 $\langle C_r^{\mathrm{on}} \rangle_{\mathrm{BFT}}$ (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).

In Algorithm 3, each aggregated share $[M_r]^j$ is broadcast by the corresponding aggregator S_j, $\{Com^{(i)}\}$ denotes the commitments by valid clients, and t denotes the threshold value for the reconstruction. It can be noted that Feldman commitments are homomorphic, 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. 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 an aggregator leader (also the leader of BFT). The leader collects shares, reconstructs the single mask, and transfers the result to each aggregator. If the results are invalid due to the malicious leader, view-change mechanism of BFT can be used to replace it. 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):

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

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

$$\prod_{i \in [C_r^{\mathrm{on}}]} g^{[m_i]^j} = \prod_{i \in [C_r^{\mathrm{on}}]} g^{[m_i]^j} \mod p$$

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

$$\sum_{j=1}^{t} \lambda_j \cdot [M_r]^j = \sum_{j=1}^{t} \sum_{i=1}^{|C^{\mathrm{on}}|} \lambda_j \cdot [m_i]^j = \sum_{i=1}^{|C^{\mathrm{on}}|} \sum_{j=1}^{t} \lambda_j \cdot [m_i]^j = \sum_{i=1}^{|C^{\mathrm{on}}|} m_i = M$$

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 $\mathsf{EMA}(C_{r-1}^{\mathrm{off}})$ (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 $\langle \vec{x}_r \rangle_{\mathrm{BFT}}$ (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 $\langle \vec{x}_r \rangle_{\mathrm{BFT}}$ 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:

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

4.3 Client Concealed Sortition

In Aion, if the total number of clients is sufficiently large, the number of intersecting clients between selected client sets in each round is low, thereby minimizing the risk of client secret exposure. However, when the total number of clients is small, using a random selection for each round may result in subsets with overlapping nodes across rounds. In this case, the sum of shares from differing nodes in consecutive rounds could be calculated, reducing the system's tolerance for malicious clients. This risk also arises if the overlap between consecutive rounds is excessively high. Although it is possible to prevent honest client shares from being revealed by having the aggregator store and compare each round's client set leveraging failure probability, this approach increases aggregator overhead. 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, ensuring the client sets for different rounds are independent.

Algorithm 4 Client Concealed Sortition (CCS)

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

```
    Key sharing and valid clients confirmation (Step ⓿ , Line 11)
    ▸ As a client $C_i \in C$
1:  run $\mathsf{VRF.Prove}_{m_i}(\mathsf{Hash}(\vec{x}_0)) \rightarrow (v_i, \pi(v_i))$
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 $\mathsf{VRF.Ver}_{Com_0^{(i)}}(\mathsf{Hash}(\vec{x}_0), v_i, \pi(v_i)) \rightarrow 1$ then
```

In Algorithm 4, after sending valid secret shares to aggregators, the client C_i uses the hash value of the initial model $\vec{x}_0$ (committed by aggregators running BFT consensus) sent by the aggregator 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). Assuming the number of subsets is K ($K < r$ 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 of k-th round, C_i uploads $\vec{y}_{i,k}$ 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 $\mathsf{VRF.Ver}_{pk_i}(\mathsf{Hash}(\vec{x}_0), v_i, \pi(v_i))$, where $pk_i = Com_0^{(i)} = g^{m_i}$ (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. In AMR, we leverage the additive homomorphism of HPRF. The adversary controls f malicious nodes. If there is an incorrect HPRF value among the $f+1$ HPRF values used to construct $F(x)$, it is impossible to have at least f other HPRF values on $F(x)$. Only if $F(x)$ is constructed correctly can at least $2f+1$ HPRF values be on $F(x)$. Besides, Equation 2 essentially leverages the additive homomorphism of the HPRF key. By interpreting both sides of Equation 1 as HPRF keys, it holds that:

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

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 supports both large-scale and small-scale client participation in FL, allowing the same client to participate in multiple rounds of training without the need for the CCS algorithm. Meanwhile, the communication overhead of Aion-AMR remains independent of the number of clients q.

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 (introduced in Appendix A). The L_2 norm is used for input validation by limiting the norm of overall input of each client to detect and discard anomalous updates. If the norm exceeds a set bound b, the update is discarded as potentially malicious.

In Algorithm 6, an aggregator first calculates the current round's bound b_r using the global model and mask from the previous two rounds (Lines 1-2). When each aggregator receives a masked gradient[3] $\vec{y}_r$, 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 3-6).

Algorithm 6 Masked Gradient Filtering (MGF)

Input: global gradient $\vec{x}_{r-1}$, $\vec{x}_{r-2}$, aggregated HPRF masks, bound b_{r-1} of round $r-1$
Output: valid client set C^{valid}

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

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.

Evolving constraint of individual client mask size. 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. Let the maximum value of the HPRF output elements be h_{max}. Then, the normalization coefficient for round r is given by $\alpha_r = \beta\|(\vec{x}_{r-1})\|_\infty / h_{max}$, where $\beta \in (0, 1)$. 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. MGF does not introduce additional communication overhead and only incurs minimal computational cost.

6 Robustness Enhancement Constructions

Next, we introduce the HPRF error elimination and asynchronous FL.

6.1 HPRF Error Elimination Module

We denote the dimension of the model 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 (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) $\vec{h} = \{h_k\}$ 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 the model vector.

However, the homomorphic operation of adding two HPRF values introduces an error $\vec{e}^{(2)}$, i.e., $\mathsf{HPRF}(m_1, r) + \mathsf{HPRF}(m_2, r) = \mathsf{HPRF}(m_1 + m_2, r) + \vec{e}^{(2)}$, where $\vec{e}^{(2)} = \{e_k^{(2)}\}, e_k^{(2)} \in \{-1, 0, 1\}$. Extending this to the case of adding q HPRF values, we have $\sum_{i=1}^{q} \mathsf{HPRF}(m_i, r) = \mathsf{HPRF}(\sum_{i=1}^{q} m_i, r) + \vec{e}^{(q)}$, 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 remove (DMR) algorithms, as shown in Algorithms 7 and 8. Clients run DMC when adding

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

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 Remove (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 $l_{\text{dp}}$ 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).

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 federated learning can accelerate model convergence. Therefore, we design an expired model aggregation module EMA that enables Aion to support asynchronous federated learning 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 $C_r^{\text{delay}}$          ▷ $C_r^{\text{delay}}$ 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 (earlier rounds may also be utilized by setting a round limit). 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 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).

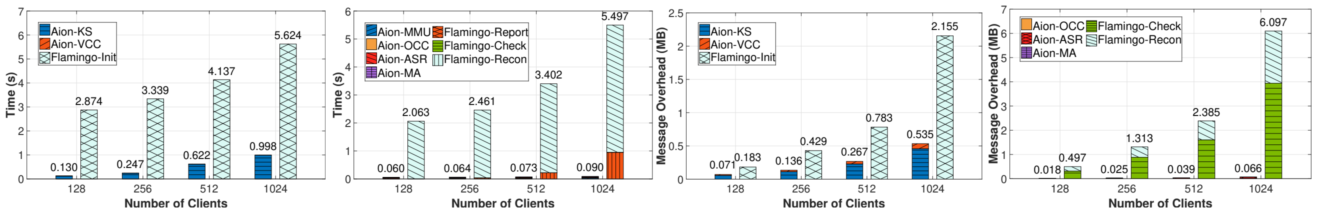
Introduction of the security threshold ξ. Privacy leakage may occur during aggregation if the aggregator colludes with $|C_r^{\text{delay}}| - 1$ malicious clients in C_r^{delay}. In this scenario, the aggregator can infer the honest client's mask by recovering the sum of the masks of the malicious clients. Thus, the algorithm sets the security threshold ξ, assuming that malicious aggregators only collude with a maximum of $\xi - 2$ clients. At this point, each aggregator only performs mask recovery when it receives masked models from more than ξ clients.

7 Implementation and Evaluation

We implement Aion using Python on a desktop equipped with RTX4090 and i9-14900KF and conduct an evaluation on time cost, message overhead, and model accuracy.

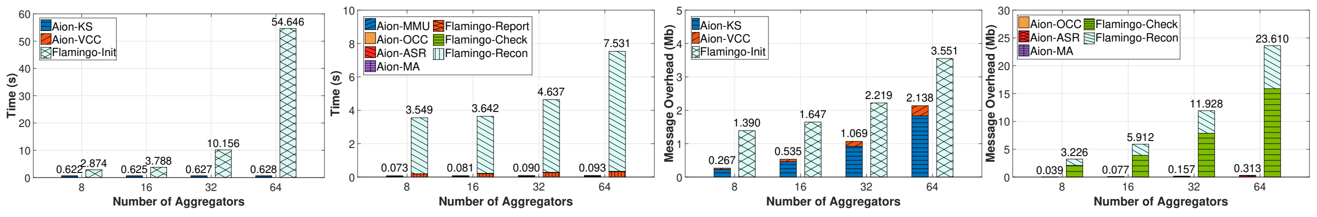
7.1 Time and Message Overhead Performance

We test the time and message overhead for both initialization and aggregation phases by adjusting the client number



(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 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 and message overhead (initialization and aggregation phase) with varying aggregator number n.

q and aggregator number n. The model dimension d is set to 10k. In Figures 2 and 3, the left and right bars represent Aion data and Flamingo data, respectively. KS stands for Key Sharing (Step ⓿), VCC denotes Valid Clients Confirmation (Step ⓿), MMU denotes Masked Model Upload (Step ❶), OCC stands for Online Clients Confirmation (Step ❷), ASR refers to Aggregated Share Reconstruction (Step ❸), and MA represents Model Aggregation (Step ❹). Due to Aion's shorter runtime, the time distribution of each phase is unclear in the bar chart. Appendix C (Figure 12) provides a detailed illustration of the time and proportion for each phase.

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, demonstrating lower execution times in both phases. With 1024 clients, Aion is 61.08 times faster than Flamingo in the aggregation phase and 5.63 times faster for initialization. This performance gap widens as the number of clients 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 is 64, Aion's time in the aggregation phase is 80.97 times faster than Flamingo's and 87.01 times faster in the initialization phase. The reason is, 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 Flamingo, a difference that increases with the client number. When q is 1024, Aion's overhead is 1.08% of Flamingo's in the aggregation phase and 19.44% in the initialization phase. 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 1.32% of Flamingo's in the aggregation phase and 60.21% in the initialization phase. As the number of aggregators increases, the message overhead for both Aion and Flamingo increases. This is expected as each added aggregator introduces extra communication. However, the increase in overhead for Aion is noticeably smaller compared to Flamingo. This is because the aggregation of client masks in Aion requires less data exchange among aggregators than in Flamingo.

7.2 Input Validation Performance

Experimental setup. The input validation experiment follows the basic setup of RoseAgg [45]. We conduct 60 rounds of poisoning attacks, with an attack probability of 50% for 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 size* (scaling factor of HPRF value) on model performance. The default values are set as follows: poison ratio 50%, boost rate 20, and mask size is constrained to be less than 10% of the global update from the last round. We compare Aion with the benchmark FedAvg [46] and the state-of-the-art scheme, Flame [47], to thoroughly demonstrate the effectiveness of the input validation module.

Adversary setup. Adversaries launch the model replacement attacks. They multiply their gradients by a scaling factor to

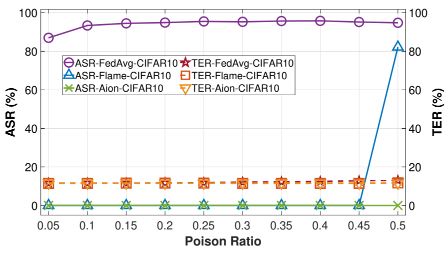


Figure 4: ASR/TER vs. Poison Ratio (CIFAR10).

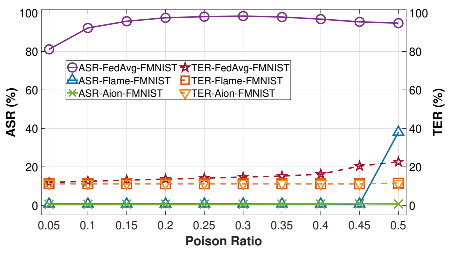


Figure 5: ASR/TER vs. Poison Ratio (FMNIST).

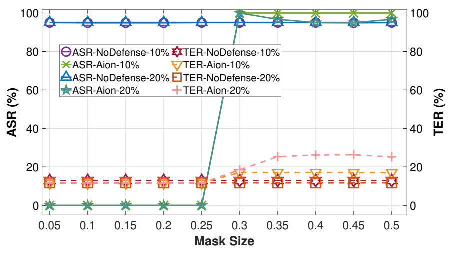


Figure 6: ASR/TER vs. Mask Size (CIFAR10).

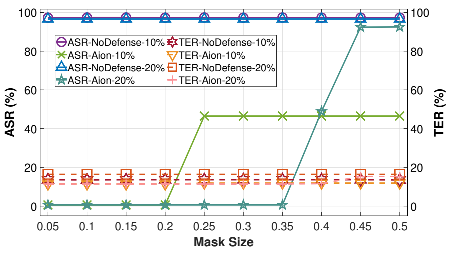


Figure 7: ASR/TER vs. Mask Size (FMNIST).

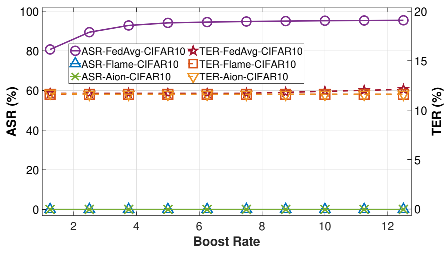


Figure 8: ASR/TER vs. Boost Rate (CIFAR10).

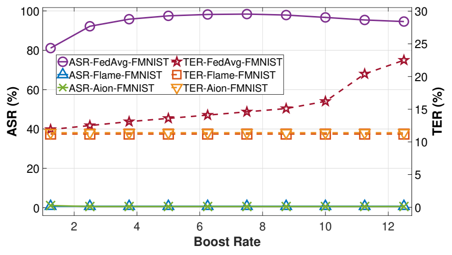


Figure 9: ASR/TER vs. Boost Rate (FMNIST).

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.

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 size added in each round must not exceed the product of mask size and the global gradient from the previous round, using the mask size 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 size is lower than 0.25. Figure 7 depicts the results for the FMNIST dataset, where Aion is effective at mask sizes lower than 0.35 with a poison ratio of 20%, but only ensures model safety at mask sizes 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 sizes 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 clients q and fixed n equal to 8. TABLE 2 presents the time and message overhead for both client and aggregator. The results indicate that the time and message costs for Aion's both client and aggregator are low.

Time cost. Aion is faster than Flamingo for both initialization

Table 2: Time and message overhead of different schemes

q	Scheme	Client: ms (KB)		Aggregator: ms (MB)	
		Initialization	**Aggregation**	**Initialization**	**Aggregation**
1024	Flamingo	67.13 (2.22)	81.09 (46.62)	132.59 (0.40)	4.19×10^3 (5.81)
	Aion-ASR	3.02 (0.44)	22.33 (39.28)	12.31 (0.07)	72.88 (0.13)
	Aion-AMR	3.01 (0.44)	22.64 (39.28)	12.30 (0.07)	89.23 (0.67)
	Aion-ASR+IV	3.06 (0.44)	22.51 (39.28)	12.36 (0.07)	74.63 (0.12)
	Aion-AMR+IV	3.08 (0.44)	23.14 (39.28)	12.30 (0.07)	91.41 (0.73)
2048	Flamingo	67.36 (4.44)	112.02 (47.44)	139.20 (0.83)	1.05×10^4 (13.17)
	Aion-ASR	3.02 (0.88)	22.31 (39.28)	14.03 (0.14)	102.36 (0.26)
	Aion-AMR	3.02 (0.88)	22.34 (39.28)	14.08 (0.14)	113.27 (1.40)
	Aion-ASR+IV	3.04 (0.88)	22.57 (39.28)	14.12 (0.14)	108.36 (0.24)
	Aion-AMR+IV	3.03 (0.88)	22.33 (39.28)	14.11 (0.14)	131.57 (1.38)
4096	Flamingo	67.81 (8.88)	127.02 (49.11)	143.58 (1.75)	3.72×10^4 (23.68)
	Aion-ASR	3.04 (1.75)	22.64 (39.28)	14.97 (0.28)	173.29 (0.51)
	Aion-AMR	3.05 (1.75)	22.42 (39.28)	15.01 (0.28)	216.29 (2.94)
	Aion-ASR+IV	3.06 (1.75)	22.37 (39.28)	15.05 (0.28)	180.57 (0.46)
	Aion-AMR+IV	3.03 (1.75)	22.74 (39.28)	15.00 (0.28)	227.93 (2.82)

and aggregation phases. When q is 4096, Aion-ASR's initialization of clients is faster than Flamingo by 20.31 times. In the aggregation phase, Aion-ASR also outperforms Flamingo and is 5.61 times faster than Flamingo.

For aggregators, Aion reduces execution times. When q is 4096, Aion-ASR is 214.67 times faster than Flamingo in the aggregation phase. Meanwhile, when q increases from 1024 to 4096, Flamingo's time increases by 887.82%, while Aion's time increases by only 237.77%. 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 22.43% 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 5.86% 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 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 46.62 KB for a client, compared to Aion-ASR's 39.28 KB. This difference is even bigger during initialization, with Aion-ASR's overhead being only 19.81% of Flamingo's.

Aion shows a significant advantage in aggregator-side message overhead, especially in initialization. With q is 1024, Aion-ASR requires only 0.13 MB for initialization, less than Flamingo's 5.81 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 Aion aggregation phase. We define the ideal functionality $\mathcal{F}$ of Π in Figure 10.

Ideal functionality $\mathcal{F}$

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

- $\mathcal{F}$ 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 $\mathcal{F}$ outputs $\vec{x}_r = \sum_{i \in C^{\text{on}}/C_{\mathcal{A}}}$; otherwise, sends abort to all honest clients.

Figure 10: Ideal functionality of Π.

The inputs of $\mathcal{F}$ are the masked local model $\vec{y}_{i,r}$ ($\vec{y}_{i,r} = \vec{x}_{i,r} + \mathsf{HPRF}(m_i, r)$) of valid clients in set C^{valid}, the number of malicious aggregators f, and malicious client f'. The output of $\mathcal{F}$ is the global model $\vec{x}_{i,r}$. $\mathcal{F}$ determines the output based on the values of f and f'. In any round r, if f and f' satisfy the security assumption, i.e., $n \geq 3f+1$ and $q \geq f'+2$, then $\mathcal{F}$ outputs the final model aggregation result $\vec{x}_r$. Otherwise, $\mathcal{F}$ outputs abort. Please see Appendix B for detailed proof.

Theorem 1 (Security of Π)**.** *Assume a secure HPRF algorithm with key homomorphism where adversary $\mathcal{A}$ 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 the ideal functionality $\mathcal{F}$. 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' ($q \geq f'+2$), Π satisfies security if the ideal-world joint view $View^{\mathcal{F}}_{\mathsf{Sim}}$ is indistinguishable from the real-world joint view $View^{\Pi}_{\mathcal{A}}$, i.e., $View^{\mathcal{F}}_{\mathsf{Sim}} \approx View^{\Pi}_{\mathcal{A}}$.*

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. In the future, we would like to (1) investigate additional input validation methods to prevent a broader range of poisoning attacks; (2) explore efficient message transmission methods between clients and multiple aggregators, leveraging error correction codes.

Ethics Considerations

This section details the ethical considerations that guided the development of Aion, our robust and efficient multi-round secure aggregation protocol for federated learning. We are deeply committed to upholding the highest standards of ethical conduct, ensuring that our research promotes security, privacy, and societal benefit while proactively addressing potential risks and adhering to established ethical principles.

Potential for Unintended Negative Applications. Our research on Aion provides a mechanism for secure and privacy-preserving federated learning. We recognize that the core technology we develop could potentially be misused for unethical purposes. While our primary aim is to establish a robust privacy-preserving framework for large-scale collaborative model training, this technology could, in principle, be adapted to conceal malicious activities or compromise data integrity. However, we strongly believe that the positive benefits offered by Aion—specifically, enabling secure and privacy-preserving large-scale federated learning applications that respect data ownership and privacy—vastly outweigh the potential for misuse. We emphasize our commitment to the responsible development and deployment of our protocol, encouraging its application in scenarios that are aligned with societal benefit. We pledge to engage with the research and policy community to understand and mitigate the risks.

Legal Compliance, Data Privacy, and Security. Our research adheres strictly to all relevant legal frameworks and principles of data privacy. We prioritize respect for law and public interest by ensuring that all data practices, including the selection and use of datasets, are fully compliant with applicable laws and regulations. Our experiments employed only publicly available, open datasets, and we meticulously reviewed their associated licenses to ensure full compliance. We also avoided any experiments on live systems that may cause unexpected vulnerability.

Team Well-being and Responsible Research Practices. The well-being of our research team was a paramount consideration. We actively avoided exposing our team to any form of physical harm, psychologically distressing data, or unsafe research conditions. We fostered an open, supportive, and collaborative team environment, encouraging open communication, and addressing any concerns promptly. We implemented equitable task distribution, ensured reasonable work hours, and actively promoted a healthy work-life balance, recognizing that well-being is essential for sustained research success and creativity. We actively promoted diversity and inclusivity to foster an enriching working environment that is accessible to all team members.

Addressing Potential Harm and Unintended Consequences. We acknowledge that even well-intentioned research may have unforeseen consequences. As such, we commit to continuous monitoring of the practical impact of the Aion protocol when deployed in real-world settings and will immediately address any negative outcomes identified by promptly refining the protocol or providing actionable mitigation recommendations. Guided by the principle of Justice, we take full responsibility for the implications of our work. This includes continuous engagement with relevant stakeholders to identify potential negative impacts and to find effective solutions to these problems.

Transparent and Responsible Vulnerability Disclosure. We are committed to a culture of responsible and transparent vulnerability disclosure. Should any vulnerabilities be discovered within the Aion protocol or its implementation, we will promptly notify relevant stakeholders through established channels. This notification will include a comprehensive description of the vulnerability, steps to reproduce, and potential mitigation strategies. We will provide a reasonable time frame for affected parties to address identified vulnerabilities before making any public disclosure.

Open Science

We are committed to upholding the principles of open science to ensure transparency and facilitate the reproducibility of our research. All research code and necessary documentation have been anonymously published at: https://anonymous.4open.science/r/Aion-8D05, to facilitate other researchers in replicating the experiments. This level of transparency ensures the integrity of our findings and allows for external validation.

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 *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 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*, pages 1175–1191, 2017.

[3] Florian Hartmann. Predicting text selections with federated learning, 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 Trans. Netw.*, 32(5):3654–3668, 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*, pages 739–753. IEEE, 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 Transactions on Medical Imaging*, 42(7):2044–2056, 2023.

[7] Pushpita Chatterjee, Debashis Das, and Danda B. Rawat. Federated learning empowered recommendation model for financial consumer services. *IEEE Trans. Consumer Electron.*, 70(1):2508–2516, 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 Comput. Surv.*, 55(3):60:1–60:37, 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*, 15:3454–3469, 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*, pages 56–62. IEEE, 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*, pages 493–506, 2020.

[12] James Henry Bell, Kallista A Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly) logarithmic overhead. In *ACM CCS 2020*, pages 1253–1269, 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 23*, pages 4805–4822, 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*, pages 477–496. IEEE, 2023.

[15] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. Elsa: Secure aggregation for federated learning with malicious actors. In *2023 IEEE SP*, pages 1961–1979. IEEE, 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. *Commun. ACM*, 22(11):612–613, 1979.

[18] Cécile Delerablée and David Pointcheval. Dynamic threshold public-key encryption. In *CRYPTO 2008*, volume 5157, pages 317–334. Springer, 2008.

[19] Chenhao Xu, Youyang Qu, Yong Xiang, and Longxiang Gao. Asynchronous federated learning on heterogeneous devices: A survey. *Comput. Sci. Rev.*, 50:100595, 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*, 2021.

[22] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. Data poisoning attacks against federated learning systems. In *ESORICS 2020*, pages 480–501. Springer, 2020.

[23] Uriel Fiege, Amos Fiat, and Adi Shamir. Zero knowledge proofs of identity. In *ACM TOC 1987*, pages 210–217, 1987.

[24] Yuval Ishai, Joe Kilian, Kobbi Nissim, and Erez Petrank. Extending oblivious transfers efficiently. In *CRYPTO 2003*, volume 2729, pages 145–161. Springer, 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*, pages 1354–1371. IEEE, 2022.

[26] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. Eiffel: Ensuring integrity for federated learning. In Heng Yin, Angelos Stavrou, Cas Cremers, and Elaine Shi, editors, *ACM CCS 2022*, pages 2535–2549. ACM, 2022.

[27] Abhishek Banerjee and Chris Peikert. New and improved key-homomorphic pseudorandom functions. In *CRYPTO 2014*, pages 353–370. Springer, 2014.

[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] Lukas Burkhalter, Hidde Lycklama, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. Rofl: Attestable robustness for secure federated learning, 2021. https://arxiv.org/abs/2107.03311.

[30] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. CRFL: certifiably robust federated learning against backdoor attacks. In *ICML 2021*, volume 139, pages 11372–11382. PMLR, 2021.

[31] Gilad Baruch, Moran Baruch, and Yoav Goldberg. A little is enough: Circumventing defenses for distributed learning. In *NeurIPS 2019*, pages 8632–8642, 2019.

[32] Minghong Fang, Xiaoyu Cao, Jinyuan Jia, and Neil Zhenqiang Gong. Local model poisoning attacks to byzantine-robust federated learning. In *USENIX Security 2020*, pages 1605–1622, 2020.

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

[34] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, USA, 27-29 October 1987*, pages 427–437. IEEE Computer Society, 1987.

[35] Miguel Castro, Barbara Liskov, et al. Practical byzantine fault tolerance. In *OSDI 1999*, volume 99, pages 173–186, 1999.

[36] Jianyu Niu, Fangyu Gai, Mohammad M. Jalalzai, and Chen Feng. On the performance of pipelined hotstuff. In *40th IEEE Conference on Computer Communications, INFOCOM 2021, Vancouver, BC, Canada, May 10-13, 2021*, pages 1–10. IEEE, 2021.

[37] Maofan Yin, Dahlia Malkhi, Michael K Reiter, Guy Golan Gueta, and Ittai Abraham. Hotstuff: Bft consensus with linearity and responsiveness. In *PODC 2019*, pages 347–356, 2019.

[38] Silvio Micali, Michael Rabin, and Salil Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[39] Yevgeniy Dodis and Aleksandr Yampolskiy. A verifiable random function with short proofs and keys. In *International Workshop on Public Key Cryptography*, pages 416–431. Springer, 2005.

[40] 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*, 14(1–2):1–210, 2021.

[41] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329*, 2019.

[42] Jie Xu, Benjamin S Glicksberg, Chang Su, Peter Walker, Jiang Bian, and Fei Wang. Federated learning for healthcare informatics. *Journal of healthcare informatics research*, 5:1–19, 2021.

[43] KS Arikumar, Sahaya Beni Prathiba, Mamoun Alazab, Thippa Reddy Gadekallu, Sharnil Pandya, Javed Masood Khan, and Rajalakshmi Shenbaga Moorthy. Fl-pmi: federated learning-based person movement identification through wearable devices in smart healthcare systems. *Sensors*, 22(4):1377, 2022.

[44] Michael G Luby, Michael Mitzenmacher, Mohammad Amin Shokrollahi, and Daniel A Spielman. Efficient erasure correcting codes. *IEEE Transactions on Information Theory*, 47(2):569–584, 2001.

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

[46] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[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*, pages 1415–1432, 2022.

[48] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963*, 2017.

[49] Galen Andrew, Om Thakkar, Brendan McMahan, and Swaroop Ramaswamy. Differentially private learning with adaptive clipping. *NeurIPS 2021*, 34:17455–17466, 2021.

[50] Kang Wei, Jun Li, Chuan Ma, Ming Ding, Wen Chen, Jun Wu, Meixia Tao, and H Vincent Poor. Personalized federated learning with differential privacy and convergence guarantee. *IEEE Transactions on Information Forensics and Security*, 2023.

[51] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677*, 2017.

[52] 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 20*, pages 493–506, 2020.

[53] Kun Zhao, Wei Xi, Zhi Wang, Jizhong Zhao, Ruimeng Wang, and Zhiping Jiang. Smss: Secure member selection strategy in federated learning. *IEEE Intelligent Systems*, 35(4):37–49, 2020.

[54] Vaikkunth Mugunthan, Antigoni Polychroniadou, David Byrd, and Tucker Hybinette Balch. Smpai: Secure multi-party computation for federated learning. In *NeurIPS 2019*, volume 21. MIT Press Cambridge, MA, USA, 2019.

[55] 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*, pages 69–76. IEEE, 2023.

[56] 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*, pages 1172–1182. IEEE, 2021.

[57] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. Rofl: Robustness of secure federated learning. In *IEEE SP 2023*, pages 453–476. IEEE, 2023.

[58] 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.

A Norm Defenses in Federated Learning

Norms are mathematical functions that provide a measure of the "size" of a vector or matrix. In FL, norms are used to constrain the magnitude of updates, preventing malicious gradients. A norm is a function $||\cdot|| : V \to \mathbb{R}$ that maps from a vector space V to the non-negative real numbers. Commonly used norms include:

- L_1 norm: $||\vec{v}||_1 = \sum_{i=1}^{n} |v_i|$
- L_2 norm (Euclidean norm): $||\vec{v}||_2 = \sqrt{\sum_{i=1}^{n} v_i^2}$
- L_∞ norm (maximum norm): $||\vec{v}||_\infty = \max_{1\le i\le n} |v_i|$

Large gradients can destabilize the optimization process. Bounding the gradient magnitude (e.g., the L_2 norm) with a bound prevents overly large updates to filter malicious gradients, thereby enhancing the robustness and accuracy of training in FL [28–31].

B 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 $\mathcal{A}$ as a subroutine.

Hybrid 1. Initially, the adversary $\mathcal{A}$ 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 the adversary potentially controlling up to f malicious aggregators. The view in this hybrid is:

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

Hybrid 2. In this hybrid world, Sim behaves as the honest clients in the actual protocol execution. Sim knows the honest clients' keys $\{m_i\}_{i\in C\setminus C_{\mathcal{A}}}$ and inputs $\{\vec{x}_{i,r}\}_{i\in C\setminus C_{\mathcal{A}}}$, generating the same aggregated result $\vec{x}_r$ as in the real execution. The view of $\mathcal{A}$ in this hybrid is still:

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

Since Sim perfectly replicates the honest clients' execution, there is:

$$\text{View}_{\text{Sim}}^{\text{Hyb2}} = \text{View}_{\mathcal{A}}^{\Pi}$$

Hybrid 3. In this hybrid, Sim replaces the honest clients' HPRF keys $\{m_i\}$ with zero values $\{m'_i = 0\}$ 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 zero keys, so the view becomes:

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

Hybrid 4. In this hybrid, Sim uses virtual inputs $\{\vec{x}'_{i,r}\}$ to replace the actual client inputs $\{\vec{x}_{i,r}\}$, 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 $\vec{x}_r$ remains consistent. The view is now:

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

Hybrid 5. 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:

$$\text{View}_{\text{Sim}}^{\text{Hyb5}} = \left\{\{\vec{x}'_{i,r}\}_{i\in[C_{\mathcal{A}}]}, \{m_i\}_{i\in[C_{\mathcal{A}}]}, \{\vec{y}''\}_{i\in[C^{\text{on}}/C_{\mathcal{A}}]}, \vec{x}'_r\right\}$$

Hybrid 6. In this hybrid step, Sim assumes that the adversary 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 can be described as:

$$\text{View}_{\text{Sim}}^{\text{Hyb6}} = \begin{cases} \perp, & \text{if } \{C_{\mathcal{A}}\} \text{ is inconsistent} \\ \text{View}_{\text{Sim}}^{\text{Hyb5}}, & \text{otherwise} \end{cases}$$

Hybrid 7. 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 masked local models $\{\vec{y}_i\}_{i\in[C^{\text{on}}]}$ from the online clients C^{on}, the malicious aggregator number f and malicious client number f', and the malicious client set C^{on}. The output from $\mathcal{F}$ is then used to simulate the view. In this hybrid, the view can be expressed as:

$$\text{View}_{\text{Sim}}^{\text{Hyb7}} = \left\{\mathcal{F}(\{\vec{y}_i\}_{i\in[C^{\text{valid}}]}, f, \eta), \{m''_i\}_{i\in[C_{\mathcal{A}}]}, \text{Comm}'_{\mathcal{A}}, C^{\text{on}}, \vec{x}'_r\right\}$$

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 the ideal functionality $\mathcal{F}$ is revealed to the adversary. By the security properties of HPRF, BFT, and VSS, the view in Hybrid 7 is indistinguishable from that in Hybrid 6. Hence, $\text{View}_{\mathcal{A}}^{\Pi} \approx \text{View}_{\text{Sim}}^{\text{Hyb7}} = \text{View}_{\text{Sim}}^{\mathcal{F}}$, concluding the proof.

Through these seven 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 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. □

C Additional Experiment Result

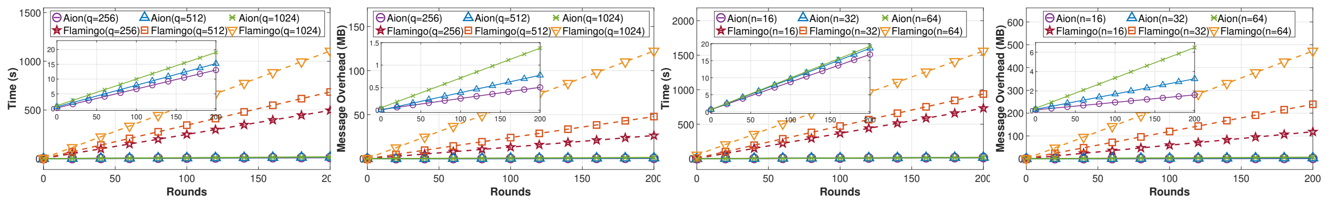
Time cost and message overhead with successive rounds. To evaluate the cumulative impact of various parameters on performance, we compare Aion and Flamingo over 200 training rounds. The time cost and message overhead are shown in Figure 11, where the inset small figures highlight Aion's performance.

Figures 11a and 11b show the effects of varying the client number. For example, with client number q fixed at 1024 and aggregator number n fixed at 8, Aion achieves an execution time of 19.01 seconds with a message overhead of 13.76 MB. In contrast, Flamingo requires 1105.12 seconds and 1222.11 MB over 200 rounds, resulting in execution times and message overheads that are 58.16-fold and 88.79-fold those of Aion, respectively.

Similarly, Figures 11c and 11d give the impact of different aggregator numbers. Here, when client number q is 512 and aggregator number n fixed at 64, Aion completes its operations in 19.23 seconds with a message overhead of 64.78 MB, whereas Flamingo requires 1560.87 seconds and 4725.62 MB which are 81.16-fold and 72.94-fold that of Aion respectively. These experiments demonstrate Aion's efficiency and scalability advantages for large-scale federated learning applications requiring a large number of training rounds.

Time 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 1024 clients and 8 aggregators, 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 0.99 seconds, accounting for approximately 98.80% of the total time, while valid clients confirmation requires only 0.012 seconds (1.20% of the phase's duration). In terms of message overhead, key sharing also incurs the highest cost, with a message size of 458.75 KB, representing 85.80% of the total communication load in this phase, whereas valid clients confirmation accounts for 75.92 KB (14.20% of the communication cost). In the aggregation phase, the time 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 37.78% of the total time. This is followed by aggregate share reconstruction at 0.022 seconds (24.44%), model aggregation at 0.018 seconds (20.0%), and online clients confirmation at 0.016 seconds (17.78%). For message overhead, aggregate share reconstruction contributes



(a) Total Time v.s. q (b) Total Overhead v.s. q (c) Total Time v.s. n (d) Total Overhead v.s. n

Figure 11: Time and message overhead (initialization and aggregation phase) with varying round r.

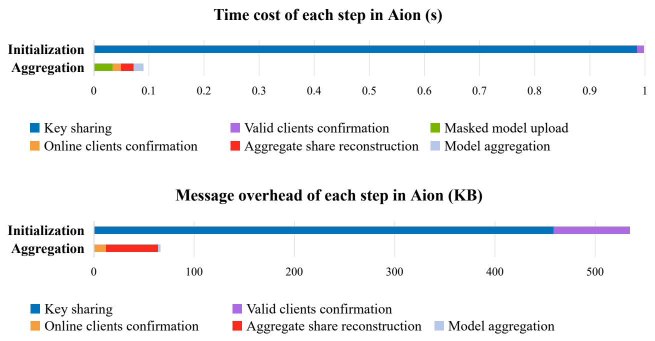


Figure 12: Time cost and message overhead of each step in Aion.

the most with 51.49 KB, accounting for 77.85% of the aggregation phase's communication cost, while online clients confirmation adds 12.31 KB (18.61%), and model aggregation has the smallest message size at 2.34 KB (3.53%).

D Related Work

Masking-based secure aggregation. Pairwise masking is an efficient SA technique for protecting data privacy. 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 to mask their input vectors. When all masked inputs are aggregated, the pairwise masks cancel out, leaving the sum of the actual inputs.

Bell et al. [12] introduced an additional layer of individual masks generated from random seeds. Clients mask their input using pairwise masks and an individual mask shared with neighbors. If a client drops out, the server requests shares of the individual mask from neighbors to reconstruct and remove it from the sum, preventing late arrival issues. Based on this, ACORN [13] optimizes the mask generation using ring learning-with-errors (RLWE) and implements input validation with L_2 norm and ZK.

Flamingo [14] treats pairwise masks as long-term secrets and uses a PRF to generate 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 privacy-preserving FL methods. DP in FL can be applied through Gaussian noise addition during global aggregation [48], adaptive quantile clipping [49], and privacy budget allocation based on Rényi DP composition [50], ensuring robust privacy protection for user data by sacrificing model accuracy. Hardy et al. [51] applied HE in vertical FL and proposed a privacy-preserving logistic regression algorithm. Zhang et al. [52] 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, 53–55], but it often results in high computational and communication overhead.

Anti-poisoning attacks schemes using the L_2 norm. In FL, adversaries either train with contaminated data or alter gradients to disrupt normal updates. Model poisoning, which amplifies adversarial influence, is typically more effective than other attacks. Suresh et al. [28] show that backdoor attacks depend on the presence of many adversaries; in scenarios with fewer adversaries, imposing a norm bound on model updates can limit their effect and defend against poisoning. As training progresses, the average magnitude of benign updates decreases, making a fixed norm bound less effective in later stages. To address this, Guo et al. [56] proposed a dynamic norm bound defense that limits malicious updates without impacting benign aggregation throughout training. A loose norm may allow harmful updates to persist after clipping, while a tight norm can over-clip benign updates, slowing convergence. RoFL [57] uses a dynamic norm, optimizing the bound through hyperparameter tuning. However, it relies on secure aggregation with ZK proofs, adding computational and communication overhead. After clipping with the L_2 norm, some methods add Gaussian noise to counter backdoors. Naseri et al. [58] analyzed two differential privacy approaches, LDP and CDP, showing that combining clipping with differential privacy effectively defends against backdoor attacks without significantly harming model performance.

E Complexity Analysis

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

Proof. In the initialization phase, the communication between aggregators involves one round of BFT to commit the valid client set. Aion uses a lightweight Hotstuff [36] as the BFT consensus with $O(n)$ communication complexity through the collection, aggregation, and forwarding by the aggregator leader. Multiple signatures from different aggregators are aggregated into a single threshold signature by the leader and transferred to others.

In the aggregation phase, besides running the BFT protocol, aggregators also execute the ASR or AMR algorithms to reconstruct a single total mask instead of reconstructing q secrets. So the complexity is irrelevant to q (comparing with $O(n^2q)$ in Flamingo). Further, in both ASR and AMR, a collect-aggregate-transfer mechanism can be employed. The aggregator leader is responsible for collecting shares from $f+1$ aggregators and then using the reconstruction algorithm to recover the secret or mask. Subsequently, the aggregator leader transfers the recovered value to all other aggregators. In this way, the communication complexity is reduced to $O(n)$. If a malicious aggregator leader fails to execute the above operations, the view-change mechanism of BFT is triggered, where an honest aggregator will be selected to act as the leader. The asynchronous FL component also involves the invocation of ASR or AMR algorithms. Additionally, input validation only increases the validation overhead on each aggregator without affecting the communication complexity. $\square$

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

Proof. In the initialization phase, the communication complexity between clients and aggregators is $O(nq)$, as each client shares a single secret to aggregators. The introduction of the client concealed sortition algorithm does not increase the communication complexity.

In the aggregation phase, the communication complexity between clients and aggregators is $O(nq)$, as each client uploads masked gradients to each aggregator. If the client sends the input to only one aggregator, a malicious aggregator may fail to take the input into account. To ensure reliable input upload, the client can send the input to $f+1$ aggregators, ensuring that at least one honest aggregator receives the input. Furthermore, to reduce the client's overhead while ensuring reliable upload, erasure coding can be applied, splitting the client's input into n parts. Note there is no communication between clients in Aion, and no DH key exchange is needed. $\square$