OPSA: Efficient and Verifiable One-Pass Secure Aggregation with TEE for Federated Learning

Zhangshuang Guan, Yulin Zhao, Zhiguo Wan, and Jinsong Han

***Abstract*—Federated learning enables collaborative model training while preserving data privacy by keeping data local. To protect user privacy during model aggregation, secure aggregation (SA) protocols are widely adopted to mask models. However, existing SA protocols require at least three round trips per aggregation and lack mechanisms to verify aggregation results. Verifiable SA addresses the verification gap but incurs high communication costs. TEE-based SA minimizes round trips but faces computational bottlenecks due to TEE's limited physical memory, especially when handling larger models or numerous clients. In this work, we introduce OPSA, an efficient and verifiable one-pass SA protocol based on TEE. By handling client dropouts via server-side TEE, OPSA enables the server to aggregate masked models in a single pass, significantly reducing round trips. To mitigate TEE's limitations, OPSA offloads tasks like model aggregation and mask elimination outside TEE, with only shared keys processed within TEE. Building on this design, we propose KhPRF-OPSA (single masking) and POT-OPSA (double masking) protocols, both incorporating novel cryptographic primitives. Furthermore, OPSA integrates commitment and signature mechanisms to ensure result verifiability with only $O(1)$ additional communication overhead per client. Compared to state-of-the-art schemes, OPSA achieves a 2$\sim$10$\times$ speedup in multi-round aggregation while guaranteeing result verification.**

***Index Terms*—Federated learning, secure aggregation, verifiable aggregation, trusted execution environment.**

I. Introduction

FEDERATED learning (FL) represents a paradigm shift in machine learning, replacing traditional centralized model training methods with decentralized collaborative frameworks [1]. In this approach, a global machine learning model is trained on multiple decentralized devices with the assistance of a central server, each possessing its own local dataset. The distinguishing feature of FL is that model updates are computed locally on these devices and uploaded to the server, eliminating the necessity to transfer raw data to a central server. These devices (clients) and the server engage in interactions through the above operations for multiple rounds, ultimately achieving a convergent model. However, the original model update may also lead to the leakage of original private data [2], [3].

Corresponding authors: Zhiguo Wan and Jinsong Han.

Z. Guan and J. Han are with the College of Computer Science and Technology, Zhejiang University, Hangzhou, 310027, China (e-mail: {guanzs, hanjinsong}@zju.edu.cn).

Y. Zhao is with Hangzhou Institute for Advanced Study, University of Chinese Academy of Sciences, Hangzhou 310024, China (e-mail: zhaoyulin22@mails.ucas.ac.cn).

Z. Wan is with the Zhejiang Lab, Hangzhou, 311121, China (e-mail: wanzhiguo@zhejianglab.com).

This work was supported by the National Natural Science Foundation of China under Grant (Nos. U22B2032, 62272425, U21A20462, 62372400).

To protect the privacy of model updates, *secure aggregation* (SA) has been designed for model aggregation in FL. The core idea involves each client masking its local model in a certain manner, while the server aggregates these masked models to recover a plaintext global model. Furthermore, secret sharing schemes are adopted to ensure that the global model remains recoverable even after some clients drop out. The above idea is first proposed by Bonawitz et al. [4][1], upgraded by Bell et al. [5], and subsequently adapted into various variants [6], [7]. However, these schemes are more suitable for single-round FL, as each aggregation requires clients to regenerate and share their keys, significantly increasing communication overhead.

Fortunately, MicroFedML [8] addresses this issue by introducing a one-time setup phase to reuse secret keys, Flamingo [9] designs a multi-round SA protocol using pairwise masks while LERNA [10] achieves the same goal through a single masking mechanism. While these schemes achieve a reusable setup phase by introducing an additional group, they still require three client-server round trips per aggregation (as shown in Fig. 1), which poses challenges in scenarios with high network latency or limited bandwidth. Essentially, the special group, such as decryptors in Flamingo and a committee in LERNA, can be treated as a trusted entity. To this end, we ask the following research question: *Can hardware-based Trusted Execution Environments (TEEs) replace these special groups to enable round-trip-optimal aggregation while maintaining computational efficiency within TEE?*

Executing SA entirely within TEE optimizes communication round trips, but the limited physical memory of TEE poses significant computational bottlenecks, particularly when aggregating larger models or handling numerous clients. For example, a global model containing a million parameters takes about 3.8MB of memory, with each parameter represented in 4 bytes. However, in scenarios involving hundreds of clients, the memory requirements can easily exceed the 128MB physical memory capacity of TEE. To address this issue, SEAR [11] adopts a strategy of aggregating one layer of the model at a time. However, this approach does not alleviate the overall computation burden on TEE, and similar challenges may arise when processing layers with a large number of parameters. Of course, sparse gradient aggregation can reduce TEE memory usage, but Kato et al. [12] have introduced an attack targeting the gradient index within TEE.

Moreover, all the multi-round or TEE-based SA protocols mentioned above lack mechanisms to verify final aggregation results, making them vulnerable to manipulation by a mali-

[1]We write the SecAgg and SecAgg+ term as [4] and [5], respectively.

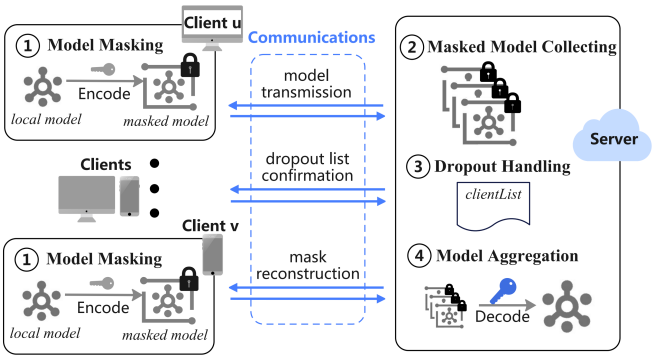


Fig. 1: Workflow of the state-of-the-art SA schemes

cious server (e.g., through inconsistent aggregation), which could lead to privacy attacks [13]–[15]. Although several verifiable SA protocols [16]–[20] have been proposed under stronger security assumptions (e.g., assuming no client-server collusion), they suffer from high communication overhead and require at least four round trips per aggregation.

In this work, we propose OPSA to effectively address these issues. First, we minimize communication round trips by handling client dropouts via server-side TEE. Then, we require TEE to process only shared keys internally, improving its computational efficiency. Finally, we present two multi-round SA protocols based on novel cryptographic primitives specifically designed for OPSA. Moreover, we design a mechanism to verify final aggregation results under a weaker security assumption. Our contributions can be summarized as follows:

- We propose OPSA, an efficient and verifiable multi-round one-pass SA framework. With the help of TEE, the server aggregates masked models in a single pass, significantly reducing communication round trips. Meanwhile, TEE is tasked only with key processing operations, keeping its memory consumption independent of model size, thereby effectively mitigating computational bottlenecks.
- Building on OPSA, we present two multi-round SA protocols: KhPRF-OPSA, a single masking protocol based on key-homomorphic pseudo-random function, and POT-OPSA, a double masking protocol built upon proxy oblivious transfer. Both protocols are designed to efficiently aggregate larger models and handle more clients within the restricted capacity of TEE.
- By incorporating commitment and signature mechanisms, we introduce OPSA-RV, a result verification method that protects against forged aggregation results. OPSA-RV achieves this under the weaker security assumption that a malicious server may collude with any semi-honest client. Compared to SOTA schemes, OPSA-RV incurs only $O(1)$ additional communication overhead per client.
- We provide a rigorous security analysis to establish the theoretical safety guarantees of OPSA. Additionally, we implement the KhPRF-OPSA and POT-OPSA protocols, along with the OPSA-RV method, and compare their performance with SOTA schemes through extensive experiments. The results demonstrate that our design excels in terms of dropout resilience and efficiency.

II. Preliminaries

Before introducing the preliminaries, we list the notations used in this work.

Notations. Unless specified otherwise, we use λ to denote the security parameter, lowercase letters (e.g., n, k, r) to denote scalars, and bold lowercase letters (e.g., $\mathbf{x}, \mathbf{y}, \mathbf{k}$) to denote vectors. The format $\{s_u\}_u$ denotes a set of values indexed by u. The variable $\mathbf{x}_u^{(r)}$ denotes the input vector $\mathbf{x}$ held by client u in round r. The sets $\mathcal{U}, \mathcal{D}, \mathcal{V}$, where $\mathcal{U} = \mathcal{D} \cup \mathcal{V}$, contain all clients, dropped clients, and surviving clients, respectively. Note that we use the terms "user" and "client" interchangeably.

A. Federated Learning

A general federated learning (FL) scenario involves n clients and a server, all participants repeat the following processes for each round r until the current global model $\boldsymbol{\theta}^{(r)}$ converges:

- Local training: each client $u \in \mathcal{U}$ receives the global model $\boldsymbol{\theta}^{(r-1)}$, which is updated in the previous round, and combines it with its private dataset to train a local model $\mathbf{x}_u^{(r)}$ to be sent to the server.
- Model aggregation: the server collects the individual model $\mathbf{x}_u^{(r)}$ from distinct clients and computes an aggregated model $\mathbf{z}^{(r)} = \sum_{u \in \mathcal{V}} \mathbf{x}_u^{(r)}$ to update the global model $\boldsymbol{\theta}^{(r)}$ to be delivered to clients for next round.

Note that these clients can drop out at any steps in each FL training round.

B. Trusted Execution Environment

Trusted Execution Environment (TEE), whose notable implementations include Intel Software Guard Extensions (SGX) and ARM TrustZone, is a hardware solution designed to ensure computational integrity and confidentiality by creating a secure and isolated space, often referred to as an enclave, where the data and its sensitive computations can be performed without the risk of interference or compromise. In particular, TEE allows remote users to verify the application and the hardware it is running via remote attestation.

Given the widespread use of Intel SGX in personal computers and cloud computing servers, we leverage it for secure aggregation in FL. However, Intel SGX presents the following challenges in achieving efficient SA protocols: (1) limited physical memory, Intel SGX v1 only has 128MB physical memory, exceeding its limit results in high overhead; (2) data transmission overhead, the SGX application is divided into trusted and untrusted parts, transmitting data between the two parts will incur significant overhead due to additional encryption/decryption operations; (3) side-channel attacks, which are an important factor to be considered, although they are not part of the Intel SGX threat model [21].

Similar to SEAR [11], we assume that Intel SGX is trusted and do not consider side-channel information leakage. In our approach, the enclave negotiates a shared key $k_{e,u}$ with each client $u \in \mathcal{U}$. Through remote attestation, client u can verify that the intended code is securely running within the enclave on a fully patched, Intel SGX-enabled platform. Notably, side-channel attacks, including physical attacks, do

not reveal any new or unexpected vulnerabilities in the Intel SGX architecture [11]. Recent work [22] shows that SGX attacks can often be mitigated through constant-time code by developers or microcode updates from Intel.

C. Cryptographic Primitives

In our protocols, we will employ the following cryptographic primitives for randomness generation and secure communication. A cryptographically secure pseudo-random generator (PRG) takes a uniformly random seed of some fixed length and outputs a string that is computationally indistinguishable from a random string; for example, it can be instantiated with AES-CTR in practice [4]. A cryptographically secure pseudo-random function (PRF) indexed by a key can also be regard as the PRG. A secure key-agreement (KA) protocol enables any user to input their private key and the public key of the other party, producing a shared secret key as output; for example, it can be instantiated with a Diffie–Hellman key agreement protocol followed by a key derivation function in practice [4], [5]. A standard secure signature scheme $\Pi_\sigma = (\mathsf{Sign}, \mathsf{VerSig})$ that is existentially unforgeable under chosen message attacks (EUF-CMA); for example, it can be instantiated with ECDSA followed by a key derivation function in practice. A commitment scheme that is perfectly hiding and computationally binding under the discrete logarithm assumption; for example, it can be instantiated with vector Pedersen commitment in practice [7].

III. OPSA Framework

In this section, we first give a threat model, and then present our OPSA framework after describing its high-level idea.

A. Threat Model

Let $N = |\mathcal{U}|$, there are N clients and a single untrusted server equipped with TEE, each client $u \in \mathcal{U}$ holds a local model $\mathbf{x}_u$, and all clients want to obtain the summation $\sum \mathbf{x}_u$ with the help of the server S revealing nothing about $\mathbf{x}_u$. Similar to MicroFedML [8] and Flamingo [9], we target the following failures and attacks: (1) honest clients that may dynamically drop out or respond too slowly due to factors such as unstable network conditions, power loss, etc.; and (2) arbitrary actions by an adversary who has control over the server and a bounded fraction of the clients. Note that the latter corresponds to the malicious setting.

In any given aggregation round, we assume that at most δ fraction of N clients drop out during any step, and that the adversary compromises γN clients before each aggregation round begins, where γ is a maximum fraction of corrupt clients, possibly also colluding with the server (which we call the malicious setting). To preserve the privacy of the final aggregation, we require $\delta + \gamma < 1$. For KhPRF-OPSA and POT-OPSA, we assume the server is fully malicious, but for OPSA-RV we prove security only in the case of semi-honest clients. Moreover, since the privacy of client temporary keys in our scheme relies on TEE, we assume TEE is trusted and exclude side-channel attacks and physical attacks from the scope of our work, as discussed in Section II-B. This is analogous to the assumption in SEAR [11].

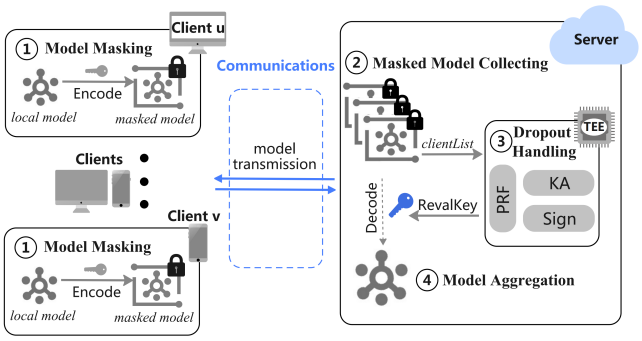


Fig. 2: Workflow of OPSA

B. Intuition of the idea

Applying masks to local models has become an effective approach for preserving privacy in FL. Each client masks its local model using a shared key, and the server subsequently utilizes revealed keys to remove these masks during model aggregation. However, this method imposes significant overhead, requiring clients to regenerate and share a new key in each round, as well as at least three client-server round trips to achieve dropout resilience. Additionally, message authentication codes (MACs) are commonly used to verify aggregation results, but they double the communication cost, adding further inefficiency.

To address these challenges, we use TEE to design multi-round aggregation. By handling client dropouts through server-side TEE, we reduce communication round trips, achieving one-pass aggregation. However, TEE-based SA protocols, which decrypt and aggregate all client models within the limited physical memory of TEE, often face significant computational bottlenecks when aggregating larger models or handling numerous clients. To mitigate this issue, we adopt a novel strategy that offloads tasks like model aggregation and mask elimination outside TEE, requiring TEE to handle only shared keys. Based on these shared keys, clients generate temporary keys to mask their local models, while TEE selectively reveals specific temporary keys to handle client dropouts. This design ensures that TEE memory consumption remains unaffected by the aggregated model size, reducing computational overhead and enhancing scalability.

During the key revealing process, the computational overhead of TEE is further impacted by mask constructions which require different numbers of temporary keys. To this end, we present two mask constructions: (1) KhPRF-OPSA, a single masking protocol based on key-homomorphic pseudo-random function (KhPRF), aggregates temporary keys of all surviving clients within TEE; (2) POT-OPSA, a double masking protocol built upon proxy oblivious transfer (POT), chooses temporary individual keys for surviving clients and temporary pairwise keys for dropped clients within TEE. Both protocols are designed to minimize the inputs and cryptographic operations performed within TEE, enabling them to aggregate larger models and handle more clients within TEE's restricted capacity.

Moreover, we integrate commitment and signature mechanisms to provide verifiable aggregation results. Specifically, each client commits to its local model, and server-side TEE

then evaluates these commitments to output an aggregated commitment along with its corresponding signature. The TEE-generated signature binds the aggregation range to a valid list of semi-honest surviving clients, while the aggregated commitment guarantees the authenticity of an aggregation result within this range. These novel methods come together to form OPSA, whose brief workflow is illustrated in Fig. 2.

C. OPSA framework

Before introducing our OPSA framework, we formally define a multi-round SA protocol as follows:

Definition 1 (Multi-round SA protocol)**.** *A multi-round SA protocol involving a client set $\mathcal{U}$, a server S, and an integer R, consists of two phases:*

- **Setup phase***: it is executed only once at the initiation of the protocol to complete initialization.*
- **Training phase***: for round $r \in [1, R]$, the SA protocol is executed once, beginning with a client generating its local model $\mathbf{x}_u^{(r)}$ and ending with the server computing an aggregated model $\sum \mathbf{x}_u^{(r)}$.*

Based on Definition 1, our OPSA is composed of three key components: one-time setup, one-pass aggregation, and result verification. Note that aggregation results are verifiable against a malicious server in the case of semi-honest clients, as discussed in our threat model.

One-Time Setup. During setup phase, all clients and the server execute the following Setup algorithm respectively to correctly generate public parameters.

- $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda, m)$: takes as inputs a security parameter λ, and a model dimension m. It outputs public parameters pp, which defines a key space $\mathcal{K}$, a model space $\mathcal{M}$, a mask space $\mathcal{Y}$, and commitment parameters $\mathsf{pp}_{\mathsf{com}}$.

Then, the server loads a program code into its TEE to create an instance, and distributes an initial global model to all clients. After verifying the correctness of the server's TEE instance through remote attestation, each client accepts the initial global model and proceeds to a FL training round.

We assume that all parties (client u, server S, and TEE instance M_e) have its own authenticated key pair (pk_*, sk_*) based on Public Key Infrastructure (PKI). Each client u can share a key with TEE instance M_e as a root seed to regenerate its temporary key as follows:

$$k_{u,e} = \mathsf{KA}(sk_u, pk_e) = \mathsf{KA}(sk_e, pk_u) = k_{e,u}. \tag{1}$$

Note that all parties initialize a round as $r = 1$, and the TEE instance sets an extra round as $r_e = 0$. The constraint $r > r_e$ means that the server can invoke the TEE instance only once during round r.

One-Pass Aggregation. By incorporating TEE and reusable shared keys, any SA protocol can be optimized into an efficient one-pass aggregation, as long as its mask construction can be decoupled into the following algorithms:

- $\mathbf{mask}_u \leftarrow \mathsf{Encode}(\mathsf{pp}, \mathsf{shKey}, r)$: takes as inputs public parameters pp, a set of shared keys shKey, and round r. It outputs a one-time mask $\mathbf{mask}_u$.
- $(\mathsf{reKey}, \mathsf{rand}) \leftarrow \mathsf{RevealKey}(\mathsf{pp}, \mathcal{V}, \mathcal{D}, r)$: takes as inputs public parameters pp, a list of surviving clients $\mathcal{V}$, a dropout list $\mathcal{D}$, and round r. It outputs a set of revealed keys reKey and a revealed random number rand.
- $\sum \mathbf{mask}_u \leftarrow \mathsf{Decode}(\mathsf{pp}, \mathsf{reKey}, r)$: takes as inputs public parameters pp, a set of revealed keys reKey, and round r. It outputs an aggregated mask $\sum \mathbf{mask}_u$.

Specifically, in FL training round r, each client $u \in \mathcal{U}$ masks its local model as $\mathbf{y}_u^{(r)} = \mathbf{x}_u^{(r)} + \mathsf{Encode}(\mathsf{pp}, \mathsf{shKey}, r)$. Once the TEE instance outputs a set of revealed keys $\mathsf{reKey} = \mathsf{RevealKey}(\mathsf{pp}, \mathcal{V}, \mathcal{D}, r)$, the server computes the plaintext aggregated model as $\sum \mathbf{x}_u^{(r)} = \sum \mathbf{y}_u^{(r)} - \mathsf{Decode}(\mathsf{pp}, \mathsf{reKey}, r)$.

Result Verification (RV). To provide verifiable aggregation results, we divide the aggregation verification into two parts: a valid aggregation range (i.e., semi-honest surviving clients), secured through a signature corresponding to TEE integrity, and an accurate aggregation result, guaranteed by a commitment scheme. In OPSA-RV, the output of TEE, protected by its integrity, binds a valid aggregation range and proves the authenticity of an aggregation result within that range.

We require that a vector commitment scheme consists of the following algorithms:

- $\mathsf{com}_{\mathbf{v}_i} \leftarrow \mathsf{Commit}(\mathbf{v}_i, \mathsf{rand}_i)$: takes as inputs a vector $\mathbf{v}_i$ and a random number rand_i. It outputs a vector commitment $\mathsf{com}_{\mathbf{v}_i}$.
- $b \leftarrow \mathsf{Open}(\mathsf{com}_{\mathbf{v}_i'}, \mathbf{v}_i, \mathsf{rand}_i)$: takes as inputs a commitment $\mathsf{com}_{\mathbf{v}_i'}$, a vector $\mathbf{v}_i$, and a random number rand_i. It outputs a bit b, which equals 1 iff the equation $\mathsf{com}_{\mathbf{v}_i'} = \mathsf{Commit}(\mathbf{v}_i, \mathsf{rand}_i)$ holds.
- $\mathsf{com} \leftarrow \mathsf{Eval}(\{\mathsf{com}_u\}_u)$: takes as input a set of commitments $\{\mathsf{com}_u\}_u$. It outputs a final commitment com.

Each client u commits to its local model as $\mathsf{com}_u = \mathsf{Commit}(\mathbf{x}_u, \mathsf{rand}_u)$, where rand_u can be instantiated randomly via its shared key. This commitment, along with its signature, ensures that the server cannot tamper with the local model. Furthermore, TEE can aggregate these commitments into a final commitment using the Eval algorithm.

Building upon the commitment scheme, a valid aggregation range $\mathcal{V}$ can be processed in TEE, as follows:

- assert: $\forall u \in \mathcal{V}, \mathsf{VerSig}(pk_u, \sigma_u, \mathsf{com}_u) = 1$.
- compute: (dummy, rand) := $\mathsf{RevealKey}(\mathsf{pp}, \mathcal{V}, \mathcal{D}, r)$.
- compute: com := $\mathsf{Eval}(\{\mathsf{com}_u\}_{u \in \mathcal{V}})$.
- sign: $\sigma_e := \mathsf{Sign}(sk_e, \mathsf{com}||\mathsf{rand}||r)$.
- output: $(\sigma_e, \mathsf{com}, \mathsf{rand})$.

Once processed, the final result $\sum_{u \in \mathcal{V}} \mathbf{x}_u$ is uniquely bound to $\mathsf{com} = \mathsf{Commit}(\sum_{u \in \mathcal{V}} \mathbf{x}_u, \mathsf{rand}) = \mathsf{Eval}(\{\mathsf{com}_u\}_{u \in \mathcal{V}})$, ensuring its authenticity. In general, the output of TEE can be regarded as a proof $\boldsymbol{\pi}^{\mathsf{valid}(\sum \mathbf{x}_u^{(r)})} = (r, \sigma_e, \mathsf{com}, \mathsf{rand})$ satisfying the following relation:

$$\begin{cases} \mathsf{VerSig}(pk_e, \sigma_e, \mathsf{com}||\mathsf{rand}||r) = 1 \\ \mathsf{Open}(\mathsf{com}, \sum_{u \in \mathcal{V}} \mathbf{x}_u, \mathsf{rand}) = 1 \end{cases} \tag{2}$$

Finally, the server sends the plaintext aggregated model along with its proof to all clients, and each client verifies this global model before initiating a new FL training round.

Putting it all Together. A complete description without verifiable aggregation in the malicious setting is provided

Parties: N clients and a TEE-equipped server.
Let $r := 1$ be the round shared by all parties.
Let $r_e := 0$ be the round maintained by TEE.

Setup phase

All parties compute $\mathsf{pp} := \mathsf{Setup}(1^\lambda, m)$.
Server S:
- load TEE instance M_e.
- send the initial global model $\boldsymbol{\theta}^{(1)}$ to clients.

Client $u \in \mathcal{U}$:
- verify the validity of the server-side TEE M_e.
- generate a shared key $k_{u,e} := \mathsf{KA}(sk_u, pk_e)$.

Training phase // repeat steps for all $r \in [1, R]$

1. **Mask step.**

Client $u \in \mathcal{U}$: // ① *model masking*
- receive from the server $\boldsymbol{\theta}^{(r)}$.
- train a local model $\mathbf{x}_u^{(r)}$ based on $\boldsymbol{\theta}^{(r)}$.
- send the server a masked model $\mathbf{y}_u^{(r)} := \mathbf{x}_u^{(r)} + \mathsf{Encode}(\mathsf{pp}, \mathsf{shKey}, r)$.

Server S: // ② *masked model collecting*
- collect $\mathbf{y}_u^{(r)}$ from users (denote with $\mathcal{V}^{(r)} \subseteq \mathcal{U}$).
- load ($\mathcal{V}^{(r)}$, $\mathcal{D}^{(r)}$, r) into M_e.

2. **Unmask step.**

TEE instance M_e: // ③ *dropout handling*
- if $r_e \geq r$, abort; otherwise, $r_e := r$.
- assert $|\mathcal{V}^{(r)}| \geq \lceil (1-\delta)N \rceil$.
- return a set of revealed keys (reKey,dummy) := $\mathsf{RevealKey}(\mathsf{pp}, \mathcal{V}^{(r)}, \mathcal{D}^{(r)}, r)$.

Server S: // ④ *model aggregation*
- $\sum \mathbf{x}_u^{(r)} := \sum \mathbf{y}_u^{(r)} - \mathsf{Decode}(\mathsf{pp}, \mathsf{reKey}, r)$.
- update $\boldsymbol{\theta}^{(r)}$ to $\boldsymbol{\theta}^{(r+1)}$ using $\sum \mathbf{x}_u^{(r)}$.
- send $\boldsymbol{\theta}^{(r+1)}$ to clients.

Fig. 3: Detailed description of the OPSA framework without verifiable aggregation in the malicious setting.

Parties: N clients and a TEE-equipped server.
Let $r := 1$ be the round shared by all parties.
Let $r_e := 0$ be the round maintained by TEE.

Setup phase

All parties compute $\mathsf{pp} := \mathsf{Setup}(1^\lambda, m)$.
Server S:
- load TEE instance M_e.
- send the initial global model $\boldsymbol{\theta}^{(1)}$ to clients.

Client $u \in \mathcal{U}$:
- verify the validity of the server-side TEE M_e.
- generate a shared key $k_{u,e} := \mathsf{KA}(sk_u, pk_e)$.

Training phase // repeat steps for all $r \in [1, R]$

1. **Mask step.**

Client $u \in \mathcal{U}$: // ① *model masking*
- receive from the server $\boldsymbol{\theta}^{(r)}$ and a proof, and check the proof $\boldsymbol{\pi}^{\mathsf{valid}(\sum \mathbf{x}_u^{(r-1)})}$ if $r > 1$.
- train a local model $\mathbf{x}_u^{(r)}$ based on $\boldsymbol{\theta}^{(r)}$.
- send the server a message $msg_u^{(r)}$ consisting of
  - $\mathbf{y}_u^{(r)} := \mathbf{x}_u^{(r)} + \mathsf{Encode}(\mathsf{pp}, \mathsf{shKey}, r)$,
  - $\mathsf{com}_u^{(r)} := \mathsf{Commit}(\mathbf{x}_u, \mathsf{rand}_u)$,
  - $\sigma_u^{(r)} := \mathsf{Sign}(sk_u, \mathsf{com}_u^{(r)})$.

Server S: // ② *masked model collecting*
- collect $msg_u^{(r)}$ from users (denote with $\mathcal{V}^{(r)} \subseteq \mathcal{U}$).
- load ($\mathcal{V}^{(r)}$, $\mathcal{D}^{(r)}$, $\{\mathsf{com}_u^{(r)}, \sigma_u^{(r)}\}_u$, r) into M_e.

2. **Unmask step.**

TEE instance M_e: // ③ *dropout handling*
- if $r_e \geq r$, abort; otherwise, $r_e := r$.
- assert $|\mathcal{V}^{(r)}| \geq \lceil (1-\delta)N \rceil$.
- $\mathsf{VerSig}(pk_u, \sigma_u^{(r)}, \mathsf{com}_u^{(r)}) = 1$ for all $u \in \mathcal{V}^{(r)}$.
- return an output consisting of
  - $(\mathsf{reKey}, \mathsf{rand}) := \mathsf{RevealKey}(\mathsf{pp}, \mathcal{V}^{(r)}, \mathcal{D}^{(r)}, r)$,
  - $\mathsf{com} := \mathsf{Eval}(\{\mathsf{com}_u\}_{u \in \mathcal{V}^{(r)}})$,
  - $\sigma_e := \mathsf{Sign}(sk_e, \mathsf{com}||\mathsf{rand}||r)$.

Server S: // ④ *model aggregation*
- $\sum \mathbf{x}_u^{(r)} := \sum \mathbf{y}_u^{(r)} - \mathsf{Decode}(\mathsf{pp}, \mathsf{reKey}, r)$.
- update $\boldsymbol{\theta}^{(r)}$ to $\boldsymbol{\theta}^{(r+1)}$ using $\sum \mathbf{x}_u^{(r)}$.
- form $\boldsymbol{\pi}^{\mathsf{valid}(\sum \mathbf{x}_u^{(r)})}$ according to Equation 2.
- send $\boldsymbol{\theta}^{(r+1)}$ and $\boldsymbol{\pi}^{\mathsf{valid}(\sum \mathbf{x}_u^{(r)})}$ to clients.

Fig. 4: Detailed description of the OPSA framework with verifiable aggregation against a malicious server.

in Fig. 3, while that with verifiable aggregation against a malicious server is provided in Fig. 4. OPSA first performs the Setup algorithm to generate public parameters, the server loads the TEE instance and delivers an initial model, and each client checks the TEE instance through remote attestation and negotiates a shared key with it (**Setup phase**).

Upon receiving the global model, during the **Training phase**, all clients validate it using its corresponding proof to start a new FL training round. Each client combines the approved global model with its private data to train a local model, which is then encoded as a masked model. To achieve result verifiability, each client commits to its local model and signs the corresponding commitment. A message, including the masked model, commitment, and signature, is sent to the server (① *Model Masking*). Subsequently, the server collects masked models from distinct clients, which are appended into a list of surviving clients. Based on this, the server easily determines a dropout list. Both the surviving and dropout lists are then loaded into the TEE instance (② *Masked Model Collecting*). Following that, the TEE instance verifies that it has been invoked only once per round to prevent multiple executions. Utilizing the loaded lists, the TEE instance publishes revealed keys, an aggregated commitment, and its signature to the server (③ *Dropout Handling*). Upon obtaining the output from the TEE instance, the server utilizes the revealed keys

to unmask an aggregated model and takes the commitment and the signature as a proof, proving the authenticity of this model (④ *Model Aggregation*). Finally, a new global model is updated and sent to all clients for the next round.

IV. Construction

Based on the OPSA framework, we instantiate two types of SA protocols: one is a single masking scheme based on the key-homomorphic pseudo-random function (KhPRF), and the other is a double masking scheme based on proxy oblivious transfer (POT). Moreover, the one-time setup and the result verification (RV) are independent of SA protocols, allowing us to share them for both constructions mentioned above.

A. Construction Based on KhPRF

We construct a single masking scheme with KhPRF [23] based on the learning with rounding (LWR) assumption [24].

Definition 2 (LWR [24])**.** *Given a security parameter λ, let $\ell = \ell(\lambda)$, $q = q(\lambda)$, $p = p(\lambda)$, and moduli $q \geq p \geq 2$ be integers, for a vector $\mathbf{s} \in \mathbb{Z}_q^\ell$, the LWR$_{\ell,q,p}$ assumption states that for any $\mathbf{a} \xleftarrow{\$} \mathbb{Z}_q^\ell$, $b \xleftarrow{\$} \mathbb{Z}_q$, the following indistinguishability holds: $(\mathbf{a}, \lfloor \langle \mathbf{a}, \mathbf{s} \rangle \rceil_p) \approx_c (\mathbf{a}, \lfloor b \rceil_p)$, where $\lfloor \cdot \rceil_p : \mathbb{Z}_q \to \mathbb{Z}_p$ is the rounding function defined as $\lfloor x \rceil_p = \lfloor x \cdot (p/q) \rfloor \mod p$.*

Definition 3 (KhPRF by LWR [23])**.** *Let $H_1 : \mathcal{X} \to \mathbb{Z}_q^\ell$ be a hash function modeled as a random oracle, define the function $F_{\mathrm{LWR}} : \mathbb{Z}_q^\ell \times \mathcal{X} \to \mathbb{Z}_p$ as $F_{\mathrm{LWR}}(\mathbf{k}, x) \leftarrow \lfloor \langle \mathbf{k}, H_1(x) \rangle \rceil_p$ under the LWR$_{\ell,q,p}$ assumption. For every $\mathbf{k}_1, \mathbf{k}_2 \in \mathbb{Z}_q^\ell$, F_{LWR} is 1-almost key homomorphic in the sense that $F_{\mathrm{LWR}}(\mathbf{k_1}, x) + F_{\mathrm{LWR}}(\mathbf{k_2}, x) = \mathrm{F}(\mathbf{k_1} + \mathbf{k_2}, x) + e$ where $e \in \{0, 1\}$.*

Note that KhPRF can also be instantiated based on the Ring-LWR assumption [24]. In this work, we instantiate it as $\mathrm{KF}(\mathbf{k}, \mathbf{x}) = (\lfloor \langle \mathbf{k}, H_1(x_1) \rangle \rceil_p, \cdots, \lfloor \langle \mathbf{k}, H_1(x_m) \rangle \rceil_p)$.

Construction 1 (KhPRF-OPSA)**.** *Let $F_1 : \mathbb{Z}_q \times \mathbb{Z}_q^* \to \mathbb{Z}_q^\ell$ be a secure PRF, and $F_0 : \mathbb{Z}_q \times \mathbb{Z}_q^* \to \mathbb{Z}_q$ be a secure PRF. Based on the LWR$_{\ell,q,p}$ assumption, let $\mathrm{KF}(\cdot, \cdot)$ be 1-almost KhPRF, the key space be $\mathcal{K} = \mathbb{Z}_q^\ell$, the model space be $\mathcal{M} = \mathbb{Z}_p^m$, and the mask space be $\mathcal{Y} = \mathbb{Z}_p^m$.*

- $\mathbf{mask}_u \leftarrow \mathsf{Encode}(\mathsf{pp}, \mathsf{shKey}, r)$*: each client u parses the shared key $k_{u,e}$ from* shKey*, computes a temporary key $\mathbf{k}_u = F_1(k_{u,e}, r)$, and outputs $\mathbf{mask}_u = \mathrm{KF}(\mathbf{k}_u, \mathbf{b})$ where $\mathbf{b} \in \mathbb{Z}_q^m$ is parsed from public parameters* pp*.*
- $(\mathsf{reKey}, \mathsf{rand}) \leftarrow \mathsf{RevealKey}(\mathsf{pp}, \mathcal{V}, \mathcal{D}, r)$*: TEE instance M_e loads the surviving list $\mathcal{V}$, computes the temporary key $\mathbf{k}_u = F_1(k_{e,u}, r)$ and $\mathsf{rand}_u = F_0(k_{e,u}, 0||r)$ for all $u \in \mathcal{V}$, and sums up temporary keys $\mathbf{k}_\mathcal{V} = \sum_{u \in \mathcal{V}} \mathbf{k}_u$ and random numbers $\mathsf{rand} = \sum_{u \in \mathcal{V}} \mathsf{rand}_u$. It outputs $\mathsf{reKey} = \{\mathbf{k}_\mathcal{V}\}$ and* rand*.*
- $\mathbf{mask}_\mathcal{V} \leftarrow \mathsf{Decode}(\mathsf{pp}, \mathsf{reKey}, r)$*: the server parses the revealed key $\mathbf{k}_\mathcal{V}$ from* reKey*, and outputs a mask $\mathbf{mask}_\mathcal{V} = \mathrm{KF}(\mathbf{k}_\mathcal{V}, \mathbf{b})$ where $\mathbf{b} \in \mathbb{Z}_q^m$ is parsed from public parameters* pp*.*

Since 1-almost KhPRF causes an error $\mathbf{e} \in \{0, 1\}^m$, the summation of masks is given by $\sum_{u \in \mathcal{V}} \mathbf{mask}_u = \mathbf{mask}_\mathcal{V} + \mathbf{e}_\mathcal{V}$ where $|\mathbf{e}_{\mathcal{V},i}|_{i \in [1,m]} < |\mathcal{V}|$. To remove the mask correctly, we can set a model scaling factor Δ sufficiently large to ensure that errors are removed by the rounding operation. The process of the above construction is the same as shown in Fig. 3 and Fig. 4, except for the following modifications:

- In the masking step, each client u masks its local model $\mathbf{y}_u = \Delta \cdot \mathbf{x}_u + \mathsf{Encode}(\mathsf{pp}, \mathsf{shKey}, r)$.
- In the unmasking step, the server obtains the result $\sum_{u \in \mathcal{V}} \mathbf{x}_u = \frac{1}{\Delta} \left(\sum_{u \in \mathcal{V}} \mathbf{y}_u - \mathsf{Decode}(\mathsf{pp}, \mathsf{reKey}, r) \right)$.

B. Construction Based on Proxy-OT

Here, we extend the notion of non-interactive proxy oblivious transfer (POT_2^1) defined by Choi et al. [25] to *k-out-of-n proxy oblivious transfer* (POT_n^k), and then utilize it to construct a double masking scheme.

We introduce the syntax of our POT_n^k protocol in Appendix A. The POT_n^k protocol involves three parties: a sender, a chooser, and a proxy. Let $F_2 : \mathbb{Z}_q \times \mathbb{Z}_q^* \to \mathbb{Z}_q^n$ be a secure PRF, and $k_{s,c} = \mathrm{KA}(pk_C, sk_S)$ be the shared key between the sender and the chooser, we can instantiate POT_n^k from the key agreement protocol as follows:

- $A \leftarrow \mathsf{Send}(r, sk_S, pk_C, \{m_i\}_{i \in [n]})$: the sender computes $(z_1, z_2, \cdots, z_n) = F_2(k_{s,c}, r)$ where $|z_i| = \lambda$, generates a permuted order $\pi := \mathrm{Perm}[n]$, and sets $A = \{ct_i\}_{i \in [n]}$ where $ct_{\pi(i)} = z_i \oplus m_i$.
- $B \leftarrow \mathsf{Choose}(r, sk_C, pk_S, \{b_j\}_{j \in [k]})$: the chooser computes $(z_1, z_2, \cdots, z_n) = F_2(k_{c,s}, r)$ where $|z_i| = \lambda$, generates a permuted order $\pi = \mathrm{Perm}[n]$, and reveals only the part associated with the choice $B = \{aux_j\}_{j \in [k]}$ where $aux_j = (\pi(b_j), z_{b_j})$.
- $M \leftarrow \mathsf{Proxy}(r, pk_S, pk_C, A, B)$: the proxy parses $A = (ct_1, ct_2, \cdots, ct_n)$ and $B = \{(b'_j, z_{b_j})\}_{j \in [k]}$, and computes $M = \{m_{b'_j}\}_{j \in [k]}$ where $m_{b'_j} = ct_{b'_j} \oplus z_{b_j}$.

In our POT_n^k protocol designed for one-pass aggregation, we designate TEE as the chooser (rather than the proxy) to alleviate its computational burden, with the clients as the senders and the server as the proxy. Given that OPSA only requires safeguarding the privacy of clients, our focus is solely on ensuring *sender privacy* in POT. This implies that the proxy can know choice of the chooser, thereby enabling TEE to make choices based on the client list provided by the server. Moreover, the permuted operation in the above instantiation can also be removed. Note that the chooser (i.e., TEE) is required to choose only once per round.

Construction 2 (POT-OPSA)**.** *Let $F_0 : \mathbb{Z}_q \times \mathbb{Z}_q^* \to \mathbb{Z}_q$ be a secure PRF, $G : \mathbb{Z}_q \to \mathbb{Z}_q^m$ be a secure PRG. Based on the instantiation POT_n^k, let the key space be $\mathcal{K} = \mathbb{Z}_q$, the model space be $\mathcal{M} = \mathbb{Z}_q^m$, and the mask space be $\mathcal{Y} = \mathbb{Z}_q^m$.*

- $\mathbf{mask}_u \leftarrow \mathsf{Encode}(\mathsf{pp}, \mathsf{shKey}, r)$*: each client u parses its shared keys $\{k_{u,v}\}_{v \in \mathcal{U}}$ from* shKey*, samples a temporary individual key $\widetilde{k}_u \xleftarrow{\$} \{0, 1\}^\lambda$, and computes a temporary pairwise key $\widetilde{k}_{u,v} = F_0(k_{u,v}, r)$ for $v \in \mathcal{U}$. It outputs a one-time mask summation $\mathbf{mask}_u = G(\widetilde{k}_u) + \sum_{v \in \mathcal{U}, u < v} G(\widetilde{k}_{u,v}) - \sum_{v \in \mathcal{U}, u > v} G(\widetilde{k}_{u,v})$.*
- $(\mathsf{reKey}, \mathsf{rand}) \leftarrow \mathsf{RevealKey}(\mathsf{pp}, \mathcal{V}, \mathcal{D}, r)$*: TEE instance M_e loads the surviving list $\mathcal{V}$ and the dropout list $\mathcal{D}$,*

constructs a choice set $\{b_v\}_{v\in\mathcal{D}}$*, and computes an auxiliary set* $B_u = \mathrm{POT}_n^k.\mathsf{Choose}(r, sk_e, pk_u, \{0\}\cup\{b_v\}_{v\in\mathcal{D}})$ *and a random number* $\mathsf{rand}_u = F_0(k_{e,u}, 0||r)$ *for* $u \in \mathcal{V}$*. It outputs* $\widetilde{\mathsf{reKey}} = \bigcup_{u\in\mathcal{V}} B_u$ *and* $\mathsf{rand} = \sum_{u\in\mathcal{V}} \mathsf{rand}_u$*.*

- $\mathbf{mask}_{\mathcal{U}} \leftarrow \mathsf{Decode}(\mathsf{pp}, \widetilde{\mathsf{reKey}}, r)$*: the server parses the revealed keys* $\{\widetilde{k}_u, \widetilde{k}_{u,v}\}_{u\in\mathcal{V},v\in\mathcal{D}}$ *from* $\widetilde{\mathsf{reKey}}$*, and outputs the summation of all masks* $\mathbf{mask}_{\mathcal{U}} = \sum_{u\in\mathcal{V},v\in\mathcal{D}} \Big(G(\widetilde{k}_u) + \sum_{u<v} G(\widetilde{k}_{u,v}) - \sum_{u>v} G(\widetilde{k}_{u,v})\Big)$.

In fact, POT_n^k is instantiated as $\mathrm{POT}_{|\mathcal{U}|}^{|\mathcal{D}|+1}$. For all $u \in \mathcal{V}$, the TEE instance chooses $\{0\}$ and $\{b_v\}_{v\in\mathcal{D}}$ to obtain the auxiliary messages corresponding to $\widetilde{k}_u$ and $\{\widetilde{k}_{u,v}\}_{v\in\mathcal{D}}$, respectively. The process of this construction is the same as shown in Fig. 3 and Fig. 4, except for the following new additions and modifications:

- In the masking step, each client u sends the server
$$A_u = \mathrm{POT}_n^k.\mathsf{Send}(r, sk_u, pk_e, \{\widetilde{k}_u\}\cup\{\widetilde{k}_{u,v}\}_{v\in\mathcal{U}}),$$
and the server stores A_u received from distinct clients.
- In the unmasking step, the server parses $\bigcup_{u\in\mathcal{V}} B_u$ from $\widetilde{\mathsf{reKey}}$, runs $\mathrm{POT}_n^k.\mathsf{Proxy}(r, pk_u, pk_e, A_u, B_u)$ to recover $\widetilde{\mathsf{reKey}}_u = \{\widetilde{k}_u\}\cup\{\widetilde{k}_{u,v}\}_{v\in\mathcal{D}}$, and obtains the aggregated model $\sum_{u\in\mathcal{V}} \mathbf{x}_u = \sum_{u\in\mathcal{V}} \mathbf{y}_u - \mathsf{Decode}(\mathsf{pp}, \widetilde{\mathsf{reKey}}, r)$ where $\widetilde{\mathsf{reKey}} = \bigcup_{u\in\mathcal{V}} \widetilde{\mathsf{reKey}}_u$.

To reduce the number of pairwise masks, we can employ the sub-graph technique [5], [9] to generate fewer pairwise masks, directly reducing the values of n and k to improve the performance of our POT_n^k for one-pass aggregation.

C. Construction of Result Verification

Let $\mathsf{pp}_{\mathsf{com}} = (\mathbf{g}, h)$ be commitment public parameters where $\mathbf{g} = (g_1, \cdots, g_m) \xleftarrow{\$} \mathbb{G}^m$ and $h \xleftarrow{\$} \mathbb{G}$, and $F_0 : \mathbb{Z}_q \times \mathbb{Z}_q^* \to \mathbb{Z}_q$ be a secure PRF. Each client u can generate a random number $\mathsf{rand}_u = F_0(k_{u,e}, 0||r)$ where r is the round and $k_{u,e}$ is the shared key between client u and TEE instance M_e.

Commitments. We first extend the Pedersen commitment scheme [26] on vectors as follows:

- $\mathsf{com}_u \leftarrow \mathsf{Commit}(\mathbf{x}_u, \mathsf{rand}_u)$: outputs a model commitment $\mathsf{com}_u = \mathbf{g}^{\mathbf{x}_u} h^{\mathsf{rand}_u}$.
- $b \leftarrow \mathsf{Open}(\mathsf{com}_{u'}, \mathbf{x}_u, \mathsf{rand}_u)$: outputs $b = 1$ iff the equation $\mathsf{com}_{u'} = \mathsf{Commit}(\mathbf{x}_u, \mathsf{rand}_u)$ holds.
- $\mathsf{com} \leftarrow \mathsf{Eval}(\{\mathsf{com}_u\}_u)$: outputs an aggregated commitment $\mathsf{com} = \prod_u \mathsf{com}_u$.

TEE Verification. Upon loading $(\mathcal{V}, \mathcal{D}, \{\mathsf{com}_u, \sigma_u\}_u, r)$, TEE instance M_e performs the following checks to ensure that the aggregation range $\mathcal{V}$ is valid:

- assert $\mathsf{VerSig}(pk_u, \sigma_u, \mathsf{com}_u) = 1$ for all $u \in \mathcal{V}$.
- run RevealKey to obtain $\mathsf{rand} := \sum_{u\in\mathcal{V}} \mathsf{rand}_u$.
- run Eval to obtain $\mathsf{com} := \prod_{u\in\mathcal{V}} \mathsf{com}_u$.
- generate $\sigma_e := \mathsf{Sign}(sk_e, \mathsf{com}||\mathsf{rand}||r)$.
- output $(\sigma_e, \mathsf{com}, \mathsf{rand})$.

In practice, these TEE operations can be encapsulated within another TEE instance, which only requires data and code integrity to be protected (transparent enclave model [27]) and can be deployed on any other party equipped with TEE.

Proofs. Based on the above instantiations, the server can form a proof $\boldsymbol{\pi}^{\mathrm{valid}(\sum \mathbf{x}_u)} = (\mathsf{rand}, \sigma_e)$ and send it along with an aggregated model $\sum_{u\in\mathcal{V}} \mathbf{x}_u^{(r)}$ to all clients. Each client first computes a commitment $\mathsf{com}' = \mathbf{g}^{\sum_{u\in\mathcal{V}} \mathbf{x}_u^{(r)}} h^{\mathsf{rand}}$ and then verifies the signature using $\mathsf{VerSig}(pk_e, \sigma_e, \mathsf{com}'||\mathsf{rand}||r) = 1$ to confirm the authenticity of $\sum_{u\in\mathcal{V}} \mathbf{x}_u^{(r)}$.

V. Security Analysis and Discussion

Here, we first analyze the security of our framework given in Fig. 3 and Fig. 4, and then discuss its privacy attacks and robustness.

A. Security of OPSA

Following Definition 4.1 and Theorem 4.9 in SecAgg+ [5], we define a SA protocol as being α-secure if a client can be guaranteed that its private input is aggregated at most once with at least $\alpha \cdot |\mathcal{H}|$ other inputs from honest clients, where $\mathcal{H}$ is a list of all honest clients. Note that the following ideal functionality can be queried once for round $r \in [1, R]$.

Definition 4 (α-Summation Ideal Functionality)**.** *Let* q, m *be integers,* $\alpha \in [0, 1]$*,* $\mathcal{H} \subseteq \mathcal{U}$*, and* $\mathcal{X}_{\mathcal{H}} = \{\mathbf{x}_u\}_{u\in\mathcal{H}}$*, where* $\mathbf{x}_u \in \mathbb{Z}_q^m$*. For any subset* $\mathcal{L} \subseteq \mathcal{H}$*, the* α*-summation ideal functionality over* $\mathcal{X}_{\mathcal{H}}$ *for* $\mathcal{L}$ *is defined as follows:*

$$F_{\alpha,\mathcal{X}_{\mathcal{H}}}(\mathcal{L}) = \begin{cases} \sum_{u\in\mathcal{L}} \mathbf{x}_u & \text{if } |\mathcal{L}| \geq \alpha \cdot |\mathcal{H}|, \\ \perp & \text{otherwise.} \end{cases}$$

Then, the security of our protocol is given by the following theorems, and their full proofs, based on a standard hybrid argument, are presented in Appendix B.

Theorem 1 (Dropout resilience)**.** *Given secure cryptographic primitives and TEE, as described in Section II, a dropout rate* δ*, and a fraction* α *defined in Definition 4, the OPSA framework (Fig. 3 and Fig. 4), instantiated with Construction 1 or 2 with parameter* $(1-\delta) \geq \alpha$*, satisfies dropout resilience: when all parities follow the protocol, for round* $r \in [1, R]$*, all input vectors* $\mathcal{X}^{(r)}$*, and all sets of dropout clients* $\mathcal{D} \subseteq \mathcal{U}$ *with* $|\mathcal{D}| \leq \lfloor \delta N \rfloor$*, the server* S *outputs* $\sum_{u\in\mathcal{U}\setminus\mathcal{D}} \mathbf{x}_u^{(r)}$ *at the end of round* r*, except with negligible probability.*

Theorem 2 (Security)**.** *Given secure cryptographic primitives and TEE, as described in Section II, a corruption rate* γ*, and a fraction* α *defined in Definition 4, the OPSA framework (Fig. 3), instantiated with Construction 1 or 2 with parameter* $(1-\gamma) \geq \alpha$*, guarantees security: when there exists a PPT simulator* SIM *such that for any round* $r \in [1, R]$*, all input vectors* $\mathcal{X}^{(r)}$*, and all sets of corrupted parities* $\mathcal{C} \subseteq \mathcal{U}\cup\{S\}$ *of size* $|\mathcal{C}\setminus\{S\}| \leq \lfloor \gamma N \rfloor$*, the output of* SIM *is computationally indistinguishable from the view of a malicious adversary* $\mathcal{A}$*, i.e.,* $\mathsf{REAL}_{\mathcal{C},r}^{\mathcal{U},S,R}(\mathcal{A}, \{\mathbf{x}_u\}_{u\in\mathcal{U}\setminus\mathcal{C}}) \approx_c \mathsf{SIM}_{\mathcal{C},r}^{\mathcal{U},S,R,F_{\alpha,\{\mathbf{x}_u^{(r)}\}_{u\in\mathcal{U}\setminus\mathcal{C}}}}(\mathcal{A})$.

The next theorem states that OPSA achieves result authenticity against a malicious server. Let $\mathbf{z}_{\mathcal{H}} = \sum_{u\in\mathcal{U}\setminus\mathcal{C}} \mathbf{x}_u$ be the sum of honest survivors, $\mathbf{e}$ be a random vector, and $\mathcal{Z} = \{\mathbf{z}_r\}_{r\in R}$ be aggregation results to be sent to clients. We introduce $\mathsf{verify}(\mathcal{Z})$ to verify aggregation results of the server.

Theorem 3 (Result authenticity)**.** *Given secure cryptographic primitives and TEE, as described in Section II, a dropout*

rate δ, and a fraction α defined in Definition 4, we say that the OPSA framework (Fig. 4), with parameter $(1-\delta) \geq \alpha$, achieves security (Theorem 2) and result authenticity in the malicious setting for any round $r \in [1, R]$, all $\mathcal{X}_{\mathcal{U}}$, and $\mathcal{C}$ with $|\mathcal{C}| \leq \lfloor \gamma N \rfloor$, if a PPT adversary $\mathcal{A}$ can get clients to accept its forged output with a negligible probability,

$$\Pr\left[\mathsf{verify}(\mathbf{z}_{\mathcal{A}}) = 1 \;\middle|\; \begin{array}{l} \mathbf{z}_{\mathcal{A}} = \mathbf{z}_{\mathcal{H}} + \sum_{u \in \mathcal{C}} \mathbf{x}_u + \mathbf{e} \\ \mathsf{verify}(\mathbf{z}_{\mathcal{H}}) = 1,\ \mathbf{z}_{\mathcal{H}} \neq \mathbf{z}_{\mathcal{A}} \end{array}\right] \leq \mathsf{negl}(\lambda).$$

B. Privacy Attacks

Here, we discuss privacy attacks on SA protocols and analyze how to resist them within the OPSA framework. Fowl et al. [13] make changes to the shared model architecture on a malicious server, enabling it to directly access precise copies of user data from gradient updates. Pasquini et al. [14] manipulate model parameters on a malicious server to execute two privacy attacks targeting SA protocols. Zhao et al. [15] demonstrate how the privacy of SA protocols in FL can be compromised by a malicious server sending customized models to clients. It is evident that such attacks are initiated by a malicious server modifying the model, and result verification of the OPSA framework can prevent these attacks.

Furthermore, the work [28] proposes a privacy attack that leverages the aggregated models and participation information across multiple training rounds to reconstruct individual models, leading to privacy leakage. To counteract this attack, the OPSA framework requires hiding client participation information, either by enhancing the structured strategy for client selection [28] or mandating clients to form minimum batches [29] for uploading models. Another study [30] introduces $t-1$ sybil devices to assist a malicious server in reconstructing individual models, and the OPSA framework can prevent it by setting a low dropout rate $\delta \in [0, 1 - \frac{t-1}{N}]$.

In our threat model, we consider two scenarios: (1) up to δN honest clients may drop out during any step, and (2) up to γN clients may be corrupted before each aggregation. Our scheme is designed to handle any dropout rate $\delta \in [0, 1]$, as the enclave securely stores shared keys for all clients, enabling the server to remove these masks from any subset of clients. However, a malicious server can simulate client dropouts by arbitrarily disrupting communication during any round, excluding up to $(\delta + \gamma)N$ clients from the final aggregation. Since clients cannot distinguish real from simulated dropouts, we require corruption fraction $\gamma \in [0, 1 - \delta]$ and utilize TEE to check $|\mathcal{V}| \geq \lceil (1-\delta)N \rceil$, where $\mathcal{V}$ is a surviving client list. This condition prevents the malicious server, controlling γN corrupted clients, from inferring any client's private input. To guarantee that an aggregation result includes contributions from at least $c \geq 3$ honest clients, this condition is expressed as $1 - \delta - \gamma \geq \frac{c}{N}$, with $\gamma \in [0, 1 - \delta]$. For instance, if the dropout rate is set to $\delta < 0.4$, our scheme provides privacy against at most $\gamma < 0.6$ fraction of clients being corrupted.

C. Extension with Robustness

OPSA relies on the integrity and confidentiality of TEE to process the clients' temporary secret keys. At the core of TEE, memory isolation ensures that the code, data, and runtime states of enclaves are inaccessible to untrusted entities, typically implemented through static partitioning mechanisms, such as the fixed-size Processor Reserved Memory in Intel SGX and the secure world memory in ARM TrustZone. Additionally, enclave management uses hardware instructions to control the life cycle of enclaves, including their creation, execution, and teardown. Specifically, enclaves are mapped to an encrypted physical memory area called Enclave Page Cache (EPC), and their life cycle is closely tied to enclave resource management, especially EPC page allocation. Due to memory usage of model aggregation exceeding EPC limits, previous TEE-based SA schemes often encounter EPC paging, leading to performance bottlenecks. In contrast, OPSA decouples TEE computations from aggregated model size and minimizes input and cryptographic operations within TEE, enabling efficient aggregation of larger models without exceeding EPC limits, indirectly reducing the attack surface. For scenarios where EPC limits might still be exceeded (e.g., handling numerous clients), OPSA can integrate strategies like dynamic memory management [31], [32] and enclave management monitors [33], [34] to enhance efficiency and robustness.

Although OPSA follows the assumption of SEAR [11] (i.e., TEE is fully trusted), its practical deployment must consider potential risks like compromised enclaves, physical attacks, and side-channel attacks. Since OPSA uses TEE solely for handling temporary keys, it remains compatible with various TEE frameworks, allowing the integration of existing approaches to mitigate these threats. For instance, tools (e.g., SgxMonitor [35]) can detect malicious enclave behavior, and mechanisms like HyperEnclave [36] can defend against compromised enclaves by preventing arbitrary memory accesses and control flows. Furthermore, hardware memory encryption techniques, such as AMD SME and Intel MKTME, could be used to protect enclaves from physical attacks [36]. To thwart side-channel attacks, OPSA can employ side-channel resistant cryptographic libraries [37] to safeguard secret keys, frameworks like eOPF [33] to implement defenses at a resource level (e.g., page tables, caches), and tools like OBELIX [38] to provide software-level protections through oblivious code execution and data accesses. These countermeasures, inherently tied to specific TEE implementations, enable OPSA to achieve robustness and comprehensive protection.

While the OPSA framework can prevent the server from tampering with the aggregation results when all clients are semi-honest, challenges arise when clients exhibit malicious behavior. In cases like model poisoning attacks, which involve intentional submission of errors or malicious models, the OPSA framework cannot guarantee the validity of the final aggregated model. To enhance client robustness, the OPSA framework can further integrate existing strategies to counteract these attacks. One strategy is to permit the server to aggregate models on a small scale, followed by scoring the aggregation results using cosine similarity [39] to identify malicious clients. Another approach is to prove that the model input satisfies the L_∞ specification through zero-knowledge proofs [7]. However, when both the server and client may act maliciously simultaneously, it is essential to combine

TABLE I: Comparison of communication (Comm.) and computation (Comp.) overhead in a secure aggregation round.

Setting	**Flamingo**	**POT-OPSA**	**LERNA**	**khPRF-OPSA**
Client Comm.	Regular client: $O(m+L+A)$ Decryptor: $O(L+\delta AN+(1-\delta)N)$	$O(m+A)$	Regular client: $O(m)$ Committee: $O(m+M)$	$O(m)$
Client Comp.	Regular client: $O(mA+L)$ Decryptor: $O(\delta AN+(1-\delta)N+\epsilon N^2)$	$O(mA)$	Regular client: $O(m\log\ell)$ Committee: $O(m\log\ell+\delta N\ell)$	$O(m\log\ell)$
Server Comm.	$O((m+L+A)N)$	$O((m+A)N)$	$O(m(N+M))$	$O(mN)$
Server Comp.	$O(t^2+\delta mAN+(1-\delta)mN)$	$O(\delta mAN+(1-\delta)mN+\epsilon N^2)$ TEE: $O((1-\delta)AN)$	$O((1-\delta)mN+t^*m)$	$O((1-\delta)mN+m\log\ell)$ TEE: $(1-\delta)N\ell$
C/S Comm.	3	1	3	1

Remark N: the number of clients, m: the model vector size, δ: the dropout rate. C/S Comm. denotes the number of client-server round-trip communications.
Flamingo A: the number of neighbors of a client, L: the number of decryptors, ϵ: graph generation parameter, t: threshold of Shamir secret sharing.
LERNA M: the number of committee members, ℓ: the key size of KhPRF, t^*: threshold of flat secret sharing.

result verification with a robust defense strategy to ensure the authenticity of the final aggregation result.

VI. Evaluation

In this section, we compare KhPRF-OPSA and POT-OPSA with state-of-the-art schemes, conducting theoretical analysis and performance analysis, respectively.

A. Theoretical Analysis

The number of client-server communications has a critical impact on the overall execution time of SA protocols. In real-world FL training scenarios across wide area networks, the round-trip time (RTT) delay can reach tens of milliseconds, resulting in notable differences in the arrival time of messages from different clients. Consequently, the server spends considerable waiting time in each communication round.

Table I provides a comprehensive comparison of OPSA's communication and computation costs with those of Flamingo and LERNA. Compared with double masking schemes, the computation cost of single masking schemes increases more obviously with the model dimension. This is because single mask computations involve the multiplication of polynomials of the model dimensions by the Ring-LWR key polynomials, requiring a computational complexity of $O(m\log\ell)$ with NTT acceleration. In contrast, double mask calculations can be regarded as a PRG instantiated by AES encryption, involving an $O(m)$ computational complexity. When considering server computation cost, double masking schemes are more expensive than single masking schemes, especially in scenarios with a large number of dropped clients, which require recovering their pairwise masks. Moreover, single masking schemes support dynamic client access. Overall, among the two proposed constructions, KhPRF-OPSA is more suitable for scenarios with low-dimensional models and more dropped clients, while POT-OPSA is better suited for situations with high-dimensional models and fewer dropped clients.

B. Experimental Setup

Implementation. The OPSA framework is implemented in Python, and the TEE code is developed in C++. In our implementation, we leverage the Intel SGX SDK to construct trusted applications inside the enclave, instantiate cryptographic primitives as detailed in subsection II-C, and employ the NIST P-256 curve. All trainable parameters of each client's model can be considered as a high-dimensional input vector, with the model dimension corresponding to the size of the input vector. For experiments evaluating the performance of SA protocols, we extract fixed-size segments from the real model parameters to serve as client input vectors. Meanwhile, the entire model parameters are used as client input vectors to evaluate model performance and accuracy. For the input vector, each element is set to 64 bits, requiring the encoding of floating-point model weights into 64 bits. To achieve a 128-bit security level, we set the key size to 128 bits for AES-CTR, configure the RLWR dimension as 2^{11}, and set the modulus q to $n\Delta\cdot 2^{286}$, consistent with the LERNA setting. The hardness of this configuration can be validated by the evaluator [40].

Experimental environment. We integrate all of our code into the ABIDES simulation framework [41] and run simulation on a machine equipped with a 2.90GHz Intel Core i7-10700 CPU and 32GB memory, running 64-bit Ubuntu 20.04.2 LTS. Moreover, this machine supports Intel SGX, and its protected physical memory region, known as Processor Reserved Memory (PRM), is limited to 128MB. Our baseline is set as the current state-of-the-art solutions, Flamingo and LERNA. Since the committee members or decryptors are selected from the clients, we set them to the same dropout rate. To ensure an equivalent level of security, Flamingo is set with 60 decryptors for a 1% dropout rate and 170 decryptors for a 5% dropout rate, while LERNA's committee size is consistently set to 2^{14}. Unless otherwise specified, the default dropout rate of clients in our experiments is set as $\delta = 5\%$.

Training setting. We conduct extensive experiments using common datasets, MNIST and CIFAR-10. MNIST comprises 60K training images and 10K testing images of handwritten digits across 10 classes. CIFAR-10 consists of 60K total samples, each being a 32×32 pixel RGB image, divided into 50K training samples and 10K testing samples. The number of SA rounds for the global model is set to 50 for MNIST and 200 for CIFAR-10. In every SA round, each client undergoes 5 iterations of local mini-batch training, with batch size 32 and learning rate 0.01. To simulate realistic FL training, we distribute the training data among clients in a non-IID

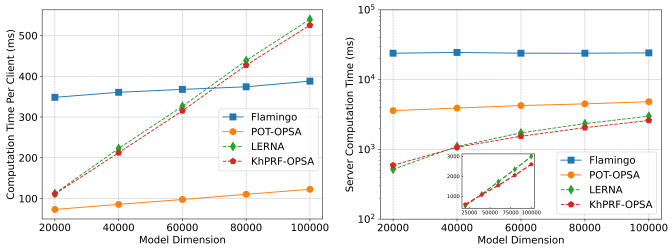
setting, using a Dirichlet distribution with hyperparameter 0.9 as described in [42].

To perform our experiments on the aforementioned datasets, we employ two models based on the classic CNN structure. For MNIST, the model includes two 5×5 convolution layers with ReLU activation, max pooling, and dropout. It also includes a ReLU-activated dense layer and a softmax-activated output layer, totaling $21,840$ trainable parameters. For CIFAR-10, the model contains $551,722$ parameters, consisting of three pairs of ReLU-activated convolution layers with batch normalization after each layer, and max pooling and dropout after each pair. Finally, the model also includes a ReLU-activated dense layer and a softmax-activated output layer.

C. Experimental Performance

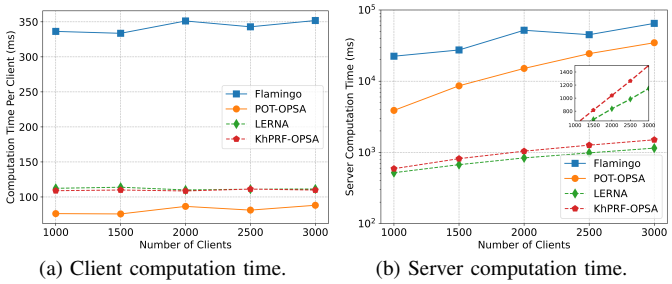
Here, we provide a benchmark to compare our two types of SA protocols with Flamingo and LERNA. For detailed comparison, we divide the entire protocol into five stages: "setup" (one-time setup phase), "mask" (masked model generation and upload), "check" (consistency check), "unmask" (aggregated model recovery), and "verify" (result verification).

Computation costs. To fairly evaluate the computation time of clients and server, we only choose the regular clients of Flamingo and LERNA for comparison with our scheme. Note that the server computation time of our scheme includes the execution time within TEE. For computation time measurements, we report an average over 10 experiment runs.



(a) Client computation time. (b) Server computation time.

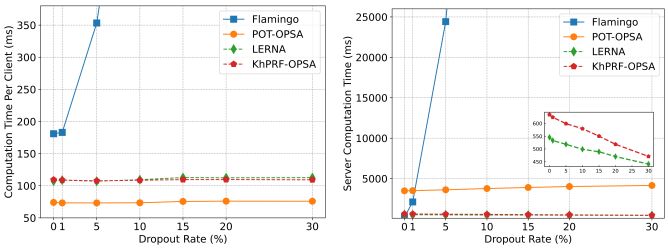
Fig. 5: Computation time vs. model dimension, with fixed number of clients $N = 1$K and dropout rate $\delta = 5\%$.



(a) Client computation time. (b) Server computation time.

Fig. 6: Computation time vs. number of clients, with fixed model dimension $m = 20$K and dropout rate $\delta = 5\%$.

Fig. 5 and Fig. 6 show the computation time for different model dimensions and different number of clients respectively. For double masking schemes, the client computation time of POT-OPSA is notably shorter than that of Flamingo. This advantage arises from the fact that Proxy-OT in our scheme utilizes simple XOR calculations instead of complex encryption operations. Moreover, at the server side, the process of mask recovery in POT-OPSA is significantly lighter compared to Flamingo, highlighting the efficiency of our approach. For single masking schemes, the server computation time of KhPRF-OPSA is slightly higher than that of LERNA due to the additional execution time required by the RevealKey operation within TEE, which is calculated by the committee in LERNA. However, when the model dimension is larger, our server computation time will be more advantageous, because TEE aggregates all low-dimensional secret keys, while LERNA aggregates all high-dimensional masks received from the committee. Overall, the computation time of KhPRF-OPSA is comparable to that of LERNA.



(a) Client computation time. (b) Server computation time.

Fig. 7: Computation time vs. dropout rate, with fixed number of clients $N = 1$K and model dimension $m = 20$K.

As shown in Fig. 7, Flamingo's computation time increases significantly as the dropout rate rises. This can be attributed to the varying number of decryptors required by Flamingo to maintain a same level of security across different dropout rates. In contrast, the client-side computation time for other schemes remains relatively stable, since the number of LERNA's committee is fixed, and our scheme only requires to interact with TEE. At the server side, double masking schemes experience a slight increase in computation time with the dropout rate due to their involvement in recovering pairwise masks of dropped clients. In contrast, the computation time of single masking schemes decreases as the dropout rate increases, as the server doesn't need to recover masks for dropped clients. While maintaining the same security level, our scheme can tolerate any dropout rate $\delta \in [0, 1]$, whereas Flamingo and LERNA are limited to smaller ranges of $\delta \in [0, \frac{1}{3})$ and $\delta \in [0, \frac{1}{2})$, respectively.

We compare our scheme with Flamingo and LERNA across various stages of a SA round, and their computation times are presented in Table II. The key advantage of our scheme lies in its significant reduction of the overall computation time. This efficiency is achieved by transforming the complex recovery operation carried out by Flamingo's decryptors and LERNA's committee members into a concise and lightweight TEE responsible for performing the recovery of secret seeds. Moreover, our scheme provides verifiable aggregation results, and the detailed computation times are provided in Table III.

TABLE II: Computation time for different stages per round of secure aggregation with $N = 1$K, $m = 20$K, and $\delta = 5\%$.

Scheme	Type	Stages for secure aggregation (ms)				
		setup	mask	check	unmask	verify
Flamingo	Server	40.37	335.75	7.94	24397.08	–
	Client	–	355.34	–	–	–
	Decryptor	7143.84	–	4.44	833.11	–
POT-OPSA	Server	1192.60	–	–	3914.28	–
	TEE	178.64	–	–	13.28	122.60
	Client	0.89	79.23	–	–	64.89
LERNA	Server	208.96	–	–	518.39	–
	Client	4245.30	107.33	–	–	–
	Committee	392.69	–	3.98	107.11	–
KhPRF-OPSA	Server	–	–	–	573.02	–
	TEE	178.73	–	–	104.16	123.94
	Client	0.37	111.97	–	–	65.44

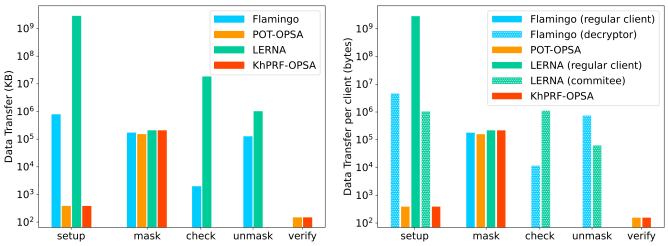
Compared with VerSA [18] and the work proposed by Eltaras et al. [20] (which we call ESLAM), our OPSA-RV has fixed TEE computation time and communication cost.

TABLE III: Computation time and communication (comm.) cost of result verification with $N = 1$K and $\delta = 5\%$.

Model dimension	Scheme	Client (ms)		Server/TEE (ms)	Comm. cost per client (B)
		commit	verify		
20K	VerSA	768.07	6.22	23879.57	160000
	ESLAM	214.49	4.60	21.19	191506
	OPSA-RV	32.91	32.26	125.35	152
40K	VerSA	1526.02	10.18	43921.27	320000
	ESLAM	222.43	10.41	45.71	351506
	OPSA-RV	65.25	64.33	124.89	152
60K	VerSA	2223.32	14.14	64412.10	480000
	ESLAM	224.26	14.18	65.39	511506
	OPSA-RV	98.21	97.18	123.81	152
80K	VerSA	2998.85	18.49	86986.30	640000
	ESLAM	229.63	17.92	86.03	671506
	OPSA-RV	130.02	129.32	121.79	152
100K	VerSA	3769.42	22.45	108853.58	800000
	ESLAM	238.34	23.59	103.32	831506
	OPSA-RV	163.53	163.11	121.52	152

Communication cost. As shown in Fig. 8, we evaluate the communication cost across different stages of a single SA round. During the "setup" phase, our scheme requires only TEE and clients to execute the key agreement protocol and remote attestation, eliminating the need for secret sharing among clients. In the "mask" stage, all schemes must upload masked models, with the communication cost of single masking schemes slightly higher than that of double masking schemes due to the larger value of key-homomorphic PRF. Compared with Flamingo and LERNA, our scheme saves communication costs in the "check" and "unmask" stages with the help of TEE. Furthermore, during the "verify" stage, our communication cost is fixed for the size of a commitment and a signature, significantly lower than that of a global model, which is also provided in Table III.

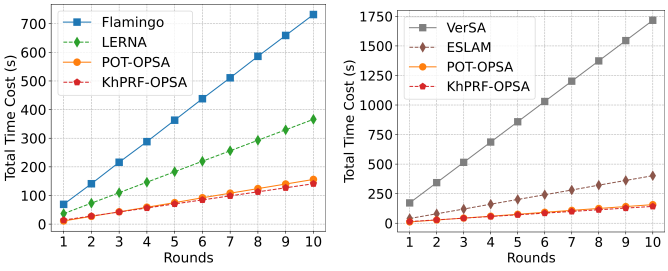
Total time cost. To demonstrate the practicality of our scheme, we evaluate the overall execution time required for the client and server to complete a 10-round SA across all schemes, including the server's wait time, the time for all messages to be delivered and received, and various computation times, but excluding the one-time setup time. As



(a) Server communication cost. (b) Client communication cost.

Fig. 8: Communication cost for different stages in a single round over 1K clients, with $m = 20$K and $\delta = 5\%$.

shown in Fig. 9, our scheme has significant advantages in both verifiable and non-verifiable SA. For SA without verification, our scheme is nearly 5× faster than Flamingo and almost 2× faster than LERNA. When verification is involved, our scheme is nearly 10× faster than VerSA [18] and almost 3× faster than ESLAM [20]. This improvement is expected to be even more significant in scenarios with high network latency, thanks to the one client-server round-trip communication provided by the OPSA framework.



(a) SA without verification. (b) SA with verification.

Fig. 9: End-to-end completion time of 10 rounds of secure aggregation over 1K clients with $m = 20$K and $\delta = 5\%$.

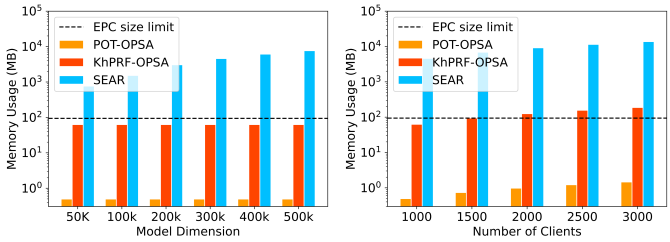
As shown in Table IV, our scheme demonstrates advantages in server-side runtime compared with SEAR [11]. Obviously, the runtime per client is independent of the number of clients, and the OPSA's server-side time cost is 4∼8 times lower than that of SEAR. Furthermore, it also shows that POT-OPSA is better suited for situations with high-dimensional models and fewer dropped clients. Additionally, our scheme enables clients to verify the codes of SA and the server's hardware it is running through remote attestation. Table V presents the total time required for remote attestation between TEE and the clients, considering different numbers of open threads. This process is generally carried out during the setup phase to ensure the integrity of the code executed by Intel SGX.

We further evaluate the memory usage of enclaves and compare our two constructions with SEAR [11]. As shown in Fig. 10, the memory usage of our enclaves is independent of the model dimension and is linearly related to the number of clients. This is because our scheme only requires processing client key seeds within the enclave, while other operations (e.g., model aggregation and unmasking) are performed outside TEE, unlike SEAR. This design significantly

reduces the enclave memory consumption in our schemes. The experimental machine allocates approximately 94 MB for the Enclave Page Cache (EPC). When the memory usage exceeds the EPC size limit, EPC paging occurs, which involves heavy computation costs due to encryption and leads to substantial slowdown in code execution. This further explains the time cost advantage presented in Table IV.

TABLE IV: Time cost comparison with the TEE-based SA scheme with $m = 500$K and $\delta = 0$.

Scheme	Per client runtime (ms)		Server runtime (ms)	
	$N = 500$	$N = 1000$	$N = 500$	$N = 1000$
SEAR	82.79	82.99	33516.24	105756.45
POT-OPSA	438.65	463.08	5245.55	12163.33
KhPRF-OPSA	2578.46	2580.20	8542.36	13647.76



(a) Memory usage vs. model dimension, with 1K clients and $\delta = 0$. (b) Memory usage vs. number of clients, with $m = 300$K and $\delta = 0$.

Fig. 10: Enclave memory usage for different model dimensions and numbers of clients during an aggregation round.

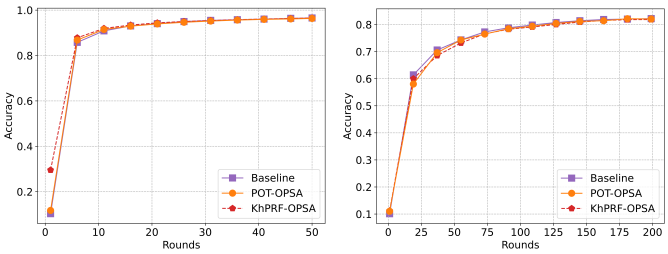
TABLE V: Execution time of remote attestation on the server with multiple threads for different numbers of clients.

Number of threads	Remote attestation (s)		
	$N = 1$K	$N = 2$K	$N = 3$K
20	367.97	737.72	1109.38
30	318.04	634.15	947.06
50	270.78	539.11	809.48

Training accuracy. To evaluate the feasibility of the OPSA framework, we deploy two types of SA protocols in the real model training process. We compare their performance with FedAvg, which serves as the baseline and does not employ any SA scheme. The evaluation is conducted on common datasets MNIST and CIFAR-10, with 100 clients participating in each training session, and its accuracy score is determined by the proportion of correctly predicted samples in the validation set relative to the total number of samples. Similar to Flamingo and LERNA, the primary deviation in our model training process from the baseline lies in the encoding of model floating-point parameters. As shown in Fig. 11, our SA scheme does not impact model performance and accuracy.

VII. Related Work

In this section, we mainly discuss three types of secure aggregation (SA): SA based on Differential Privacy (DP), SA based on Trusted Execution Environment (TEE), and SA based on One-Time Pad (OTP). Moreover, we also discuss some verifiable SA schemes.



(a) MNIST. (b) CIFAR-10.

Fig. 11: Accuracy comparison under different datasets.

DP-based SA. Differential Privacy (DP) is a concept in data privacy that aims to protect the sensitive information of individuals when analyzing or mining data. The fundamental idea behind DP is to add a controlled amount of noise to the data or query results to prevent the disclosure of specific details about any individual's contribution to the data set.

In DP-based SA schemes [43]–[45], each client adds statistical noise (e.g., Gaussian noise) to hide their local models, but this method introduces significant noise during model aggregation, affecting the accuracy of the results. Moreover, the work [30] has proposed an attack against FL protected with distributed differential privacy and secure aggregation.

TEE-based SA. TEE is a secure and isolated environment within a computer system or a microprocessor, providing a protected execution space for sensitive and critical operations. TEE is designed to prevent data and code from unauthorized access, tampering, and other security threats.

Secure aggregation can also be designed based on TEE [11], [46], [47], i.e., the TEE-equipped server decrypts models received from clients and aggregates them within TEE. However, this method may lead to a notable performance degradation when handling high-dimensional models or a large number of clients, due to the TEE's limited memory. Moreover, TEE still has the risk of being attacked, such as side-channel attacks [48].

OTP-based SA. One-time pad (OTP) is a cryptographic technique that employs a randomly generated secret key of the same length as the message to encrypt and decrypt information. It is a type of symmetric encryption algorithm where the key is used only once for a single message.

In OTP-based SA schemes, each client generates one-time pads as single or double masks to protect its local models, and the server then can eliminate the sum of masks to recover a original aggregated model. This approach is introduced by SecAgg [4] and the subsequent work, SecAgg+ [5], applies sub-grouping technique to create random connected graphs to reduce the cost of pairwise masks. FLDP [44] and ACORN [7] optimize one-time masks using the additive homomorphism of (Ring) LWE. As the secret keys are one-time in these schemes, each client is required to regenerate and share a new key with others per FL round, leading to a substantial increase in the communication burden.

To achieve a multi-round setting, MicroFedML [8] intro-

duces a one-time setup phase to reuse secrets. Flamingo [9] designs a multi-round SA protocol integrating threshold decryption with a double masking scheme, while LERNA [10] achieves the same goal integrating flat secret sharing with a key-homomorphic pseudo-random function. Meanwhile, two recent works, OPA [49] and Willow [50], focus on one-shot aggregation, where clients speak only once per aggregation round. Specifically, the former introduces a distributed-key homomorphic PRF, while the latter develops a symmetric encryption scheme that is both key- and message-homomorphic. However, these multi-round SA protocols [9], [10], [49], [50] rely on an additional group, such as decryptors in Flamingo and Willow, or a committee in LERNA and OPA, to improve performance in handling dropout. Even then, their aggregation still require 3 communication round trips. Moreover, none of them consider the verifiability of final aggregation results.

Verifiable SA. In SA protocols, a server provides a list of surviving clients who have successfully uploaded their masked models. The verifiability of the aggregation result, indicating that the result is indeed aggregated according to this list, is achieved by the verifiable SA, which ensures that a malicious server can perform aggregation honestly.

VerifyNet [16], based on a double masking scheme, introduces a homomorphic hash function and non-interactive zero-knowledge proofs (NIZK) to verify the aggregation result. VeriFL [17] builds upon this by replacing NIZK with a commitment mechanism. VerSA [18], derived from a double masking scheme, generates two sets of model masks and employs the message authentication code (MAC) concept for result verification. VOSA [19] designs a single mask scheme under the Decisional Composite Residuosity (DCR) assumption and utilizes NIZK to achieve result verification. Eltaras et al. [20] construct a single mask scheme based on key agreement protocols and also utilize the MAC concept for result verification.

In these verifiable SA schemes, all clients are considered to be semi-honest. VerifyNet [16] and VeriFL [17] further assume that the server is semi-honest but capable of forging the aggregation result. VerSA [18] and ESLAM [20], on the other hand, assume that the server is malicious but does not collude with any client at any stage. This assumption is primarily due to the MAC being computed by the same common factor among all clients, which is unknown to the server. VOSA [19] introduces a semi-honest collector for interaction with the server, but there is no guarantee that the collector is always online. Moreover, the client communication overhead in these schemes [18], [20] is twice that of the original SA schemes, which put huge transmission pressure on clients, especially for high-dimensional models.

VIII. Conclusion

In this paper, we proposed OPSA, an efficient and verifiable multi-round one-pass aggregation via hardware-assisted TEE. By offloading computational tasks outside TEE, OPSA efficiently aggregates larger models and handles more clients within TEE's restricted capacity. Furthermore, we presented KhPRF-OPSA and POT-OPSA, which are built upon novel cryptographic primitives specifically designed for OPSA, and developed OPSA-RV, a method that achieves result verifiability under a weaker security assumption, increasing only $O(1)$ communication overhead per client. Experimental results demonstrate significant improvements in efficiency and dropout resilience. In future work, we aim to extend result verification to address combined malicious server-client attacks.

References

[1] P. Kairouz, H. B. McMahan, B. Avent *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1-2, pp. 1–210, 2021.

[2] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *IEEE Symposium on Security and Privacy*, 2019, pp. 691–706.

[3] A. Hatamizadeh, H. Yin, P. Molchanov, A. Myronenko, W. Li *et al.*, "Do gradient inversion attacks make federated learning unsafe?" *IEEE Transactions on Medical Imaging*, vol. 42, no. 7, 2023.

[4] K. Bonawitz, V. Ivanov, B. Kreuter *et al.*, "Practical secure aggregation for privacy-preserving machine learning," in *ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.

[5] J. H. Bell, K. A. Bonawitz, A. Gascón *et al.*, "Secure single-server aggregation with (poly) logarithmic overhead," in *ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 1253–1269.

[6] J. So, B. Güler, and A. S. Avestimehr, "Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning," *IEEE Journal on Selected Areas in Information Theory*, pp. 479–489, 2021.

[7] J. Bell, A. Gascón, T. Lepoint, B. Li, S. Meiklejohn, M. Raykova, and C. Yun, "ACORN: Input validation for secure aggregation," in *32nd USENIX Security Symposium*, 2023, pp. 4805–4822.

[8] Y. Guo, A. Polychroniadou, E. Shi, D. Byrd, and T. Balch, "MicroFedML: Privacy preserving federated learning for small weights," *Cryptology ePrint Archive*, 2022.

[9] Y. Ma, J. Woods, S. Angel *et al.*, "Flamingo: Multi-round single-server secure aggregation with applications to private federated learning," in *IEEE Symposium on Security and Privacy*, 2023, pp. 477–496.

[10] H. Li, H. Lin, A. Polychroniadou, and S. Tessaro, "LERNA: Secure single-server aggregation via key-homomorphic masking," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2023, pp. 302–334.

[11] L. Zhao, J. Jiang, B. Feng *et al.*, "Sear: Secure and efficient aggregation for byzantine-robust federated learning," *IEEE Transactions on Dependable and Secure Computing*, vol. 19, no. 5, pp. 3329–3342, 2021.

[12] F. Kato, Y. Cao, and M. Yoshikawa, "Olive: Oblivious federated learning on trusted execution environment against the risk of sparsification," *Proceedings of the VLDB Endowment*, pp. 2404–2417, 2023.

[13] L. Fowl, J. Geiping, W. Czaja *et al.*, "Robbing the fed: Directly obtaining private data in federated learning with modified models," in *International Conference on Learning Representations*, 2022.

[14] D. Pasquini, D. Francati, and G. Ateniese, "Eluding secure aggregation in federated learning via model inconsistency," in *ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 2429–2443.

[15] J. C. Zhao, A. Sharma, A. R. Elkordy *et al.*, "LOKI: Large-scale data reconstruction attack against federated learning through model manipulation," in *IEEE Symposium on Security and Privacy*, 2024.

[16] G. Xu, H. Li, S. Liu, K. Yang, and X. Lin, "VerifyNet: Secure and verifiable federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 15, pp. 911–926, 2019.

[17] X. Guo, Z. Liu, J. Li *et al.*, "VeriFL: Communication-efficient and fast verifiable aggregation for federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 1736–1751, 2020.

[18] C. Hahn, H. Kim, M. Kim, and J. Hur, "VerSA: Verifiable secure aggregation for cross-device federated learning," *IEEE Transactions on Dependable and Secure Computing*, vol. 20, no. 1, pp. 36–52, 2021.

[19] Y. Wang, A. Zhang, S. Wu, and S. Yu, "VOSA: Verifiable and oblivious secure aggregation for privacy-preserving federated learning," *IEEE Transactions on Dependable and Secure Computing*, 2022.

[20] T. Eltaras, F. Sabry, W. Labda, K. Alzoubi, and Q. Malluhi, "Efficient verifiable protocol for privacy-preserving aggregation in federated learning," *IEEE Transactions on Information Forensics and Security*, 2023.

[21] N. C. Will and C. A. Maziero, "Intel software guard extensions applications: A survey," *ACM Computing Surveys*, vol. 55, no. 14s, 2023.

[22] S. Van Schaik, A. Seto, T. Yurek, A. Batori, B. AlBassam, D. Genkin, A. Miller, E. Ronen, Y. Yarom, and C. Garman, "Sok: Sgx. fail: How stuff gets exposed," in *IEEE Symposium on Security and Privacy*, 2024, pp. 4143–4162.

[23] D. Boneh, K. Lewi, H. Montgomery, and A. Raghunathan, "Key homomorphic prfs and their applications," in *Annual Cryptology Conference*. Springer, 2013, pp. 410–428.

[24] A. Banerjee, C. Peikert, and A. Rosen, "Pseudorandom functions and lattices," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012, pp. 719–737.

[25] S. G. Choi, J. Katz, R. Kumaresan, and C. Cid, "Multi-client non-interactive verifiable computation," in *Theory of Cryptography Conference*. Springer, 2013, pp. 499–518.

[26] T. P. Pedersen, "Non-interactive and information-theoretic secure verifiable secret sharing," in *Annual international cryptology conference*. Springer, 1991, pp. 129–140.

[27] R. Pass, E. Shi, and F. Tramer, "Formal abstractions for attested execution secure processors," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 260–289.

[28] J. So, R. E. Ali, B. Güler *et al.*, "Securing secure aggregation: Mitigating multi-round privacy leakage in federated learning," in *AAAI Conference on Artificial Intelligence*, 2023, pp. 9864–9873.

[29] Z. Liu, H.-Y. Lin, and Y. Liu, "Long-term privacy-preserving aggregation with user-dynamics for federated learning," *IEEE Transactions on Information Forensics and Security*, vol. 18, pp. 2398–2412, 2023.

[30] F. Boenisch, A. Dziedzic, R. Schuster *et al.*, "Reconstructing individual data points in federated learning hardened with differential privacy and secure aggregation," in *IEEE European Symposium on Security and Privacy*, 2023, pp. 241–257.

[31] J. Z. Yu, S. Shinde, T. E. Carlson, and P. Saxena, "Elasticlave: An efficient memory model for enclaves," in *31st USENIX Security Symposium*, 2022, pp. 4111–4128.

[32] Y. Xu, J. Pangia, C. Ye, Y. Solihin, and X. Shen, "Data enclave: A data-centric trusted execution environment," in *IEEE International Symposium on High-Performance Computer Architecture*, 2024, pp. 218–232.

[33] A. Ahmad, A. Schultz, B. Lee, and P. Fonseca, "An extensible orchestration and protection framework for confidential cloud computing," in *17th USENIX Symposium on Operating Systems Design and Implementation*, 2023, pp. 173–191.

[34] Y. Bai, P. Li, Y. Huang, M. C. Huang, S. Zhao, L. Zhao, F. Zhang, D. Meng, and R. Hou, "HyperTEE: A decoupled tee architecture with secure enclave management," in *57th IEEE/ACM International Symposium on Microarchitecture*. IEEE, 2024, pp. 105–120.

[35] F. Toffalini, M. Payer, J. Zhou, and L. Cavallaro, "Designing a provenance analysis for sgx enclaves," in *Proceedings of the 38th Annual Computer Security Applications Conference*, 2022, pp. 102–116.

[36] Y. Jia, S. Liu, W. Wang, Y. Chen, Z. Zhai, S. Yan, and Z. He, "HyperEnclave: An open and cross-platform trusted execution environment," in *USENIX Annual Technical Conference*, 2022, pp. 437–454.

[37] J. Protzenko, B. Parno, A. Fromherz, C. Hawblitzel, M. Polubelova, K. Bhargavan, B. Beurdouche, J. Choi, A. Delignat-Lavaud, C. Fournet *et al.*, "Evercrypt: A fast, verified, cross-platform cryptographic provider," in *IEEE Symposium on Security and Privacy*, 2020, pp. 983–1002.

[38] J. Wichelmann, A. Rabich, A. Pätschke, and T. Eisenbarth, "Obelix: Mitigating side-channels through dynamic obfuscation," in *IEEE Symposium on Security and Privacy*, 2024, pp. 189–189.

[39] X. Cao, M. Fang, J. Liu, and N. Z. Gong, "FLTrust: Byzantine-robust federated learning via trust bootstrapping," in *Annual Network and Distributed System Security Symposium*, 2021.

[40] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Journal of Mathematical Cryptology*, vol. 9, no. 3, pp. 169–203, 2015.

[41] D. Byrd, M. Hybinette, and T. H. Balch, "ABIDES: Towards high-fidelity multi-agent market simulation," in *ACM SIGSIM Conference on Principles of Advanced Discrete Simulation*, 2020, pp. 11–22.

[42] B. Biggio, B. Nelson, and P. Laskov, "Poisoning attacks against support vector machines," *arXiv preprint arXiv:1206.6389*, 2012.

[43] N. Agarwal, A. T. Suresh, F. X. X. Yu, S. Kumar, and B. McMahan, "cpSGD: Communication-efficient and differentially-private distributed sgd," *Advances in Neural Information Processing Systems*, vol. 31, 2018.

[44] T. Stevens, C. Skalka, C. Vincent *et al.*, "Efficient differentially private secure aggregation for federated learning via hardness of learning with errors," in *31st USENIX Security Symposium*, 2022, pp. 1379–1395.

[45] W.-N. Chen, C. A. C. Choo *et al.*, "The fundamental price of secure aggregation in differentially private federated learning," in *International Conference on Machine Learning*, 2022, pp. 3056–3089.

[46] Y. Zhang, Z. Wang, J. Cao *et al.*, "ShuffleFL: Gradient-preserving federated learning using trusted execution environment," in *ACM International Conference on Computing Frontiers*, 2021, pp. 161–168.

[47] A. P. Kalapaaking, I. Khalil, M. S. Rahman, M. Atiquzzaman *et al.*, "Blockchain-based federated learning with secure aggregation in trusted execution environment for internet-of-things," *IEEE Transactions on Industrial Informatics*, vol. 19, no. 2, pp. 1703–1714, 2022.

[48] C. Liu, A. Chakraborty, N. Chawla, and N. Roggel, "Frequency throttling side-channel attack," in *ACM SIGSAC Conference on Computer and Communications Security*, 2022, pp. 1977–1991.

[49] H. Karthikeyan and A. Polychroniadou, "OPA: one-shot private aggregation with single client interaction and its applications to federated learning," *Cryptology ePrint Archive*, 2024.

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

APPENDIX A
PROXY OBLIVIOUS TRANSFER

Here, we review the notion of non-interactive proxy oblivious transfer (POT_2^1) defined by Choi et al. [25]. A POT_2^1 protocol involves three parties: a sender, a chooser, and a proxy. The sender knows two inputs m_0 and m_1, and the chooser holds a choice $b \in \{0,1\}$. At the end of the protocol, the proxy learns m_b, while the two other parties learn nothing. Based on [25], a proxy OT protocol requires both *sender privacy* and *chooser privacy*. *Sender privacy* denotes that the proxy learns only the value of the sender input that corresponds to the chooser's input sequence. *Chooser privacy*, on the other hand, means that the proxy learns no information about the chooser's input sequence.

Now, we extend POT_2^1 to *k-out-of-n proxy oblivious transfer* (POT_n^k). First, we introduce the syntax of our POT_n^k protocol, which consists of the following algorithms:

- $A \leftarrow \mathsf{Send}(r, sk_S, pk_C, \{m_i\}_{i\in[n]})$: in the r-th POT_n^k execution, the sender takes as inputs its private key sk_S, the chooser's public key pk_C, and a plaintext set $\{m_i\}_{i\in[n]}$. It outputs an encoded message set $A = \{ct_i\}_{i\in[n]}$ to be sent to the proxy.
- $B \leftarrow \mathsf{Choose}(r, sk_C, pk_S, \{b_j\}_{j\in[k]})$: in the r-th POT_n^k protocol, the chooser takes as inputs its private key sk_C, the sender's public key pk_S, and a choice set $\{b_j\}_{j\in[k]}$. It outputs an auxiliary set $B = \{aux_j\}_{j\in[k]}$ to be sent to the proxy.
- $M \leftarrow \mathsf{Proxy}(r, pk_S, pk_C, A, B)$: in the r-th POT_n^k protocol, the proxy takes as inputs the sender's public key pk_S, the chooser's public key pk_C, the sender message A, and the chooser message B. It outputs a selected plaintext set $M = \{m_{b_j}\}_{j\in[k]}$.

APPENDIX B
SECURITY PROOFS

B.1 Proof of Theorem 1

Proof. The proof of dropout resilience is rather simple: in each round of the training phase, δ fraction of clients can drop out, i.e., $|\mathcal{D}| \leq \lfloor \delta N \rfloor$. Since $|\mathcal{V}| \geq \lceil (1-\delta)N \rceil$ (Fig. 3 and Fig. 4), the TEE instance can always help the server to reveal the secret keys. Therefore, the dropout resilience

property directly follows that of the underlying key agreement protocol instantiated with Equation 1. For each client $u \in \mathcal{U}$, there are two primary approaches to mask individual models.

For KhPRF-OPSA, client u masks its individual model as $\mathbf{y}_u^{(r)} = \Delta \cdot \mathbf{x}_u^{(r)} + \mathbf{mask}_u$ where $\mathbf{mask}_u = \mathrm{KF}(F_1(k_{u,e}, r), \mathbf{b})$. Here, F_1 is a secure pseudo-random function invoked on a round $r \in [1, R]$ and a shared key $k_{u,e}$ between client u and TEE instance M_e, and KF is a 1-almost key-homomorphic PRF invoked on a temporary key $\mathbf{k}_u = F_1(k_{u,e}, r)$ and a public parameter $\mathbf{b}$. For the surviving list $\mathcal{V}^{(r)}$, TEE computes $\mathbf{k}_{\mathcal{V}^{(r)}} = \sum_{u \in \mathcal{V}^{(r)}} F_1(k_{e,u}, r)$. Now, given $k_{u,e} = k_{e,u}$, we can observe that $\sum_{u \in \mathcal{V}^{(r)}} \mathrm{KF}(F_1(k_{u,e}, r), \mathbf{b}) = \mathrm{KF}(\mathbf{k}_{\mathcal{V}^{(r)}}, \mathbf{b}) + \mathbf{e}$ where $|\mathbf{e}_i| < \Delta$ for all $i \in [1, m]$, and obtain an aggregated model as follows:

$$\begin{aligned}
\sum_{u \in \mathcal{V}^{(r)}} \mathbf{x}_u^{(r)} &= \frac{1}{\Delta} \left(\sum_{u \in \mathcal{V}^{(r)}} \mathbf{y}_u^{(r)} - \mathrm{KF}(\mathbf{k}_{\mathcal{V}^{(r)}}, \mathbf{b}) \right) \\
&= \frac{1}{\Delta} \left(\sum_{u \in \mathcal{V}^{(r)}} \Delta \cdot \mathbf{x}_u^{(r)} + \sum_{u \in \mathcal{V}^{(r)}} \mathbf{mask}_u - \mathrm{KF}(\mathbf{k}_{\mathcal{V}^{(r)}}, \mathbf{b}) \right) \\
&= \frac{1}{\Delta} \left(\Delta \sum_{u \in \mathcal{V}^{(r)}} \mathbf{x}_u^{(r)} + \mathbf{e} \right) \\
&= \sum_{u \in \mathcal{V}^{(r)}} \mathbf{x}_u^{(r)} + \mathbf{0}
\end{aligned}$$

For POT-OPSA, each client $u \in \mathcal{U}$ masks its individual model as $\mathbf{y}_u^{(r)} = \mathbf{x}_u^{(r)} + \mathbf{mask}_u$ where $\mathbf{mask}_u = G(\widetilde{k}_u) + \sum_{v \in \mathcal{U}, u<v} G(F_0(k_{u,v}, r)) - \sum_{v \in \mathcal{U}, u>v} G(F_0(k_{u,v}, r))$. Here, F_0 is a secure pseudo-random function invoked on a round $r \in [1, R]$ and a shared key $k_{u,v}$ between client u and its neighbor v, and G is a secure pseudo-random generator invoked on a temporary individual key $\widetilde{k}_u \xleftarrow{\$} \{0,1\}^\lambda$ or a temporary pairwise key $\widetilde{k}_{u,v} = F_0(k_{u,v}, r)$. For the client list $\mathcal{U} = \mathcal{V}^{(r)} \cup \mathcal{D}^{(r)}$, the TEE instance helps the server to recover $\{\widetilde{k}_{v,u}\}_{v \in \mathcal{D}^{(r)}, u \in \mathcal{V}^{(r)}}$ for dropped clients and $\{\widetilde{k}_u\}_{u \in \mathcal{V}^{(r)}}$ for surviving clients. Then, the server computes $\mathbf{mask}_{\mathcal{U}}^{(r)} = \sum_{u \in \mathcal{V}^{(r)}, v \in \mathcal{D}^{(r)}} \left(G(\widetilde{k}_u) + \sum_{u<v} G(\widetilde{k}_{u,v}) - \sum_{u>v} G(\widetilde{k}_{u,v}) \right)$. Now, given $k_{u,v} = k_{v,u}$ for all client u and v, we can observe that $\sum_{u,v \in \mathcal{L}, u<v} G(\widetilde{k}_{u,v}) = \sum_{u,v \in \mathcal{L}, u>v} G(\widetilde{k}_{u,v})$ for all $u, v \in \mathcal{L}$ and $\mathcal{L} \subseteq \mathcal{U}$, and obtain an aggregated model as follows:

$$\begin{aligned}
\sum_{u \in \mathcal{V}^{(r)}} \mathbf{x}_u^{(r)} &= \sum_{u \in \mathcal{V}^{(r)}} \mathbf{y}_u - \mathbf{mask}_{\mathcal{U}}^{(r)} \\
&= \sum_{u \in \mathcal{V}^{(r)}} \mathbf{x}_u^{(r)} + \sum_{u \in \mathcal{V}^{(r)}, v \in \mathcal{U}} \mathbf{mask}_u - \mathbf{mask}_{\mathcal{U}}^{(r)} \\
&= \sum_{u \in \mathcal{V}^{(r)}} \mathbf{x}_u^{(r)} + \sum_{u,v \in \mathcal{V}^{(r)}} \left(\sum_{u<v} G(\widetilde{k}_{u,v}) - \sum_{u>v} G(\widetilde{k}_{u,v}) \right) \\
&= \sum_{u \in \mathcal{V}^{(r)}} \mathbf{x}_u^{(r)} + 0
\end{aligned}$$

This is satisfied when the number of surviving clients with $|\mathcal{V}| \geq \lceil (1-\delta)N \rceil$ and $(1-\delta) \geq \alpha$, where α is defined in Definition 4. □

B.2 Proof of Theorem 2

Proof. The proof is based on a standard hybrid argument using the simulator SIM. For any round $r \in [1, R]$, all input vectors $\mathcal{X}^{(r)}$, and all sets of corrupted clients $\mathcal{C} \subseteq \mathcal{U}$ of size $|\mathcal{C}| \leq \lfloor \gamma N \rfloor$, we prove the output of SIM is computationally indistinguishable from the view of a malicious adversary $\mathcal{A}$, i.e., $\mathsf{REAL}_{\mathcal{C},r}^{\mathcal{U},S,R}(\mathcal{A}, \{\mathbf{x}_u\}_{u \in \mathcal{U} \setminus \mathcal{C}}) \approx_c \mathsf{SIM}_{\mathcal{C},r}^{\mathcal{U},S,R,F_{\alpha,\{\mathbf{x}_u^{(r)}\}_{u \in \mathcal{U} \setminus \mathcal{C}}}}(\mathcal{A})$ where $F_{\alpha,\mathcal{X}^{(r)}}$ is α-summation ideal functionality defined in Definition 4.

Following the MicroFedML [8] and Flamingo [9], we first define the behavior of our simulator SIM for KhPRF-OPSA:

- Setup phase:
  (1) Each honest client u and TEE instance M_e follow the protocol description in one-time setup.
  (2) For each corrupt client v, TEE instance M_e computes and stores $k_{e,v} = \mathsf{KA}(sk_e, pk_v)$.
  (3) For each honest clients u, the simulator chooses $k_a \xleftarrow{\$} \{0,1\}^\lambda$ and sets $k_{u,e}^* = k_{e,u}^* = k_a$.
- The r-th round of the Training phase:
  (1) Each honest client u randomly chooses a random vector $\mathbf{y}_u^*$ and sends it the server.
  (2) The server S first adds the client u to the surviving list $\mathcal{V}^{(r)}$, and then the simulator makes a query to $F_{\alpha,\mathcal{X}^{(r)}}(\mathcal{V}^{(r)} \setminus \mathcal{C})$ to get $\mathbf{x}^{(r)}$.
  (3) For all honest clients $u \in \mathcal{V}^{(r)} \setminus \mathcal{C}$, the simulator randomly samples $\mathbf{x}_u^* \xleftarrow{\$} \mathcal{M}$ under the restriction $\sum_{u \in \mathcal{V}^{(r)} \setminus \mathcal{C}} \mathbf{x}_u^* = \mathbf{x}^{(r)}$, and then computes $\mathbf{mask}_u^* = \mathbf{y}_u^* - \mathbf{x}_u^*$ by programming the random oracle to set the output of KhPRF. The simulator sets $\mathbf{k}_u^* = \mathrm{PRF}(k_{u,e}^*, r)$ and samples a random vector $\mathbf{b}$ such that $\mathbf{mask}_u^* = \mathcal{R}_{\mathsf{KhPRF}}(\mathbf{k}_u^*, \mathbf{b})$ for each $u \in \mathcal{V}^{(r)} \setminus \mathcal{C}$, reveals masked keys $\mathbf{k}_{\mathcal{V}^{(r)}}^* = \sum_{u \in \mathcal{V}^{(r)}} \mathbf{k}_u^*$ for TEE instance M_e, and computes $\mathbf{mask}_{\mathcal{V}^{(r)}}^* = \mathcal{R}_{\mathsf{KhPRF}}(\mathbf{k}_{\mathcal{V}}^*, \mathbf{b})$ for the server.

Below, we present a series of subsequent hybrids to the real execution REAL of our KhPRF-OPSA protocol.

Hyb_0: This random variable is the joint views of all parties in $\mathcal{C}$ in the real execution.

Hyb_1: In this hybrid, the simulator, which knows all secrets of honest parties in every round, runs the execution of the KhPRF-OPSA protocol with $\mathcal{A}$. Therefore, the view of the adversary is the same as the previous hybrid.

Hyb_2: This hybrid is identical to Hyb_1 except that the simulator substitutes the shared key $k_{u,e} = \mathsf{KA}(sk_u, pk_e)$ between honest client u and TEE M_e with a random key $k_a \xleftarrow{\$} \{0,1\}^\lambda$. The 2ODH assumption [4] guarantees this hybrid is indistinguishable from the previous one.

Hyb_3: This hybrid is same as Hyb_2 except that the values of $\mathbf{y}_u$ computed by SIM on behalf of the honest client u and sent to the server, are substituted with uniformly sampled values $\mathbf{y}_u^*$. Since $\mathcal{A}$ does not know the secrets of honest parities and $\mathbf{y}_u$ has the same distribution as $\mathbf{y}_u^*$, the view of the adversary in this hybrid is statistically indistinguishable from the previous one.

Hyb_4: This hybrid is same as Hyb_3 except that the simulator replaces the revealed keys outputted by TEE instance M_e with $\mathbf{k}^*_{\mathcal{V}}$ and computes the aggregated mask $\mathbf{mask}^*_{\mathcal{V}}$ (as in Behavior 3 of Training phase of SIM) by querying the $F_{\alpha,\mathcal{X}^{(r)}}(\mathcal{V}^{(r)} \setminus \mathcal{C})$ and utilizing a random oracle $\mathcal{R}_{\mathsf{KhPRF}}$. Since $\mathcal{A}$ does not know the secrets of honest parities and the aggregated mask in SIM have the same distribution as the one in REAL, this hybrid is indistinguishable from the previous one.

Hyb_5: This hybrid is same as Hyb_4 except one modification: SIM sets the output of the ideal functionality $F_{\alpha,\mathcal{X}^{(r)}}(\mathcal{V}^{(r)} \setminus \mathcal{C})$ (as in Behavior 2 of Training phase of SIM) as the aggregated model $\mathbf{x}^{(r)}$. Note that the ideal functionality will not return $\perp$ in this case. This modification does not change the view seen by the adversary, and hence this hybrid is indistinguishable from the previous one.

After completing these hybrids, SIM simulates REAL without relying on inputs from the honest parties, indicating that the joint view of $\mathcal{A}$ in the real execution (Hyb_0) is computationally indistinguishable from the output of SIM (Hyb_5).

Here, we redefine the behavior of our simulator SIM for POT-OPSA:

- Setup phase:
  (1) Each honest client u and TEE instance M_e follow the protocol description in one-time setup.
  (2) For each corrupt client v, an honest client u computes and stores $k_{u,v} = \mathsf{KA}(sk_u, pk_v)$, and TEE instance M_e computes and stores $k_{e,v} = \mathsf{KA}(sk_e, pk_v)$.
  (3) For each honest clients u, the simulator chooses $k_a \xleftarrow{\$} \{0,1\}^\lambda$ and sets $k^*_{u,e} = k^*_{e,u} = k_a$. For each pair of honest clients u, v, the simulator chooses $k_b \xleftarrow{\$} \{0,1\}^\lambda$ and sets $k^*_{u,v} = k^*_{v,u} = k_b$.
- The r-th round of the Training phase:
  (1) Each honest client u randomly chooses a random vector $\mathbf{y}^*_u$ and sends it the server. For each corrupt client v, Each honest client u randomly chooses $ct^*_{u,v}$ as a ciphertext and sends it to the server.
  (2) The server S first adds the client u to the surviving list $\mathcal{V}^{(r)}$, and then the simulator makes a query to $F_{\alpha,\mathcal{X}^{(r)}}(\mathcal{V}^{(r)} \setminus \mathcal{C})$ to get $\mathbf{x}^{(r)}$.
  (3) For all honest clients $u \in \mathcal{V}^{(r)} \setminus \mathcal{C}$, the simulator randomly samples $\mathbf{x}^*_u \xleftarrow{\$} \mathcal{M}$ under the restriction $\sum_{u \in \mathcal{V}^{(r)} \setminus \mathcal{C}} \mathbf{x}^*_u = \mathbf{x}^{(r)}$, and then computes $\mathbf{mask}^*_u = \mathbf{y}^*_u - \mathbf{x}^*_u$ by programming the random oracle to set the output of PRG. The simulator sets $\{\widetilde{\mathbf{k}}^*_{u,v}\}_{v \in \mathcal{U}} = \{\mathcal{R}_{\mathsf{PRG}}(\widetilde{k}^*_{u,v})\}_{v \in \mathcal{U}}$ where $\{\widetilde{k}^*_{u,v}\}_{v \in \mathcal{U}} = \{\mathrm{PRF}(k^*_{u,v}, r)\}_{v \in \mathcal{U}}$, and samples a random $\widetilde{k}^*_{u}$ to set $\widetilde{\mathbf{k}}^*_u = \mathcal{R}_{\mathsf{PRG}}(\widetilde{k}^*_u)$ such that $\mathbf{mask}^*_u = \widetilde{\mathbf{k}}^*_u + \sum_{u<v} \widetilde{\mathbf{k}}^*_{u,v} - \sum_{u>v} \widetilde{\mathbf{k}}^*_{u,v}$ for each $u \in \mathcal{V}^{(r)} \setminus \mathcal{C}$ and $v \in \mathcal{U}$. The simulator computes $\{\widetilde{\mathbf{k}}^*_u, \widetilde{\mathbf{k}}^*_{u,v}\}_{u \in \mathcal{V}^{(r)} \setminus \mathcal{C}, v \in \mathcal{D}^{(r)}}$ for M_e, and computes $\mathbf{mask}^*_{\mathcal{U}} = \sum \widetilde{\mathbf{k}}^*_u + \sum \sum_{u<v} \widetilde{\mathbf{k}}^*_{u,v} - \sum \sum_{u>v} \widetilde{\mathbf{k}}^*_{u,v}$ where $u \in \mathcal{V}^{(r)} \setminus \mathcal{C}$ and $v \in \mathcal{D}^{(r)}$ for the server.

Below, we present a series of subsequent hybrids to the real execution REAL of our POT-OPSA protocol.

Hyb_0: This random variable is the joint views of all parties in $\mathcal{C}$ in the real execution.

Hyb_1: In this hybrid, the simulator, which knows all secrets of honest parties in every round, runs the execution of the POT-OPSA protocol with $\mathcal{A}$. Therefore, the view of the adversary is the same as the previous hybrid.

Hyb_2: This hybrid is identical to Hyb_1 except that the simulator substitutes the shared key $k_{u,e} = \mathsf{KA}(sk_u, pk_e)$ between honest client u and TEE M_e with a random key $k_a \xleftarrow{\$} \{0,1\}^\lambda$, and replaces the shared key $k_{u,v} = \mathsf{KA}(sk_u, pk_v)$ between honest client u and v with a random key $k_b \xleftarrow{\$} \{0,1\}^\lambda$. The 2ODH assumption [4] guarantees this hybrid is indistinguishable from the previous one.

Hyb_3: This hybrid is same as Hyb_2 except that the values of $(\mathbf{y}_u, \{ct_{u,v}\}_{v \in \mathcal{U}})$ computed by SIM on behalf of the honest client u and sent to the server, are substituted with uniformly sampled values $(\mathbf{y}^*_u, \{ct^*_{u,v}\}_{v \in \mathcal{U}})$ at random. Since $\mathcal{A}$ does not know the secrets of honest parities and $(\mathbf{y}_u, \{ct_{u,v}\}_v)$ has the same distribution as $(\mathbf{y}^*_u, \{ct^*_{u,v}\}_v)$, the view of the adversary in this hybrid is statistically indistinguishable from the previous one.

Hyb_4: This hybrid is same as Hyb_3 except that the simulator replaces the revealed keys outputted by TEE instance M_e with $\{\widetilde{\mathbf{k}}^*_u, \widetilde{\mathbf{k}}^*_{u,v}\}_{u,v}$ and computes the aggregated mask $\mathbf{mask}^*_{\mathcal{U}}$ (as in Step 3 of Training phase behavior of SIM) by querying the $F_{\alpha,\mathcal{X}^{(r)}}(\mathcal{V}^{(r)} \setminus \mathcal{C})$ and utilizing a random oracle $\mathcal{R}_{\mathsf{PRG}}$. Since $\mathcal{A}$ does not know the secrets of honest parities and the aggregated mask in SIM have the same distribution as the one in REAL, this hybrid is indistinguishable from the previous one.

Hyb_5: This hybrid is same as Hyb_4 except one modification: SIM sets the output of the ideal functionality $F_{\alpha,\mathcal{X}^{(r)}}(\mathcal{V}^{(r)} \setminus \mathcal{C})$ (as in Step 2 of Training phase behavior of SIM) as the aggregated model $\mathbf{x}^{(r)}$. Note that the ideal functionality will not return $\perp$ in this case. This modification does not change the view seen by the adversary, and hence this hybrid is indistinguishable from the previous one.

After completing these hybrids, SIM simulates REAL without relying on inputs from the honest parties, indicating that the joint view of $\mathcal{A}$ in the real execution (Hyb_0) is computationally indistinguishable from the output of SIM (Hyb_5).

For KhPRF-OPSA and POT-OPSA, SIM can simulate REAL without knowledge of the inputs from the honest parties, indicating the security of our OPSA framework in the malicious setting. Note that our malicious setting includes the semi-honest setting, thereby extending the scope of our security proof to encompass the latter.

□

B.3 Proof of Theorem 3

Proof. We first prove the security of individual models in the construction instantiated with Construction 1 or 2 providing result verification (subsection IV-C), following the proof of Theorem 2. As aforementioned, we assume that all clients are semi-honest.

Now, we define how our simulator SIM behaves during result verification of KhPRF-OPSA or POT-OPSA. In fact, SIM behaves similarly to that in in the proof of Theorem 2, with the following new additions and modifications:

- The r-th round of the Training phase:
  (1) Each honest client u samples com_u^* at random. It then computes a signature $\sigma_u^* = \mathsf{Sign}(sk_u, \mathsf{com}_u^*)$ and sends $(\mathsf{com}_u^*, \sigma_u^*)$ to the server.
  (3) For all honest clients $u \in \mathcal{V}^{(r)} \setminus \mathcal{C}$, the simulator samples $\mathsf{rand}_u^* = \mathsf{PRF}(k_{u,e}^*, 0||r)$ such that $\mathsf{com}_u^* = \mathsf{Commit}(\mathbf{x}_u^*, \mathsf{rand}_u^*)$. For TEE instance M_e, the simulator computes $\mathsf{rand}^* = \sum_u \mathsf{rand}_u^*$ and $\mathsf{com}^* = \prod_u \mathsf{com}_u^*$ for all $u \in \mathcal{V}^{(r)} \setminus \mathcal{C}$ and generates a signature $\sigma_e^* := \mathsf{Sign}(sk_e, \mathsf{com}^*||\mathsf{rand}^*||r)$. The server sends $(\mathbf{x}^r, \mathsf{rand}^*, \sigma_e^*)$ to all clients.

Below, we present a series of subsequent hybrids to the real execution REAL of our KhPRF-OPSA protocol with result verification.

Hyb_0: This random variable is the joint views of all parties in $\mathcal{C}$ in the real execution.

Hyb_1: In this hybrid, the simulator, which knows all secrets of honest parties in every round, runs the execution of the KhPRF-OPSA protocol with result verification with $\mathcal{A}$. Therefore, the view of the adversary is the same as the previous hybrid.

Hyb_2: This hybrid is identical to Hyb_1 except that the simulator substitutes the shared key $k_{u,e} = \mathsf{KA}(sk_u, pk_e)$ between honest client u and TEE M_e with a random key $k_a \xleftarrow{\$} \{0,1\}^\lambda$. The 2ODH assumption [4] guarantees this hybrid is indistinguishable from the previous one.

Hyb_3: This hybrid is same as Hyb_2 except that the values of $(\mathbf{y}_u, \mathsf{com}_u, \sigma_u)$ computed by SIM on behalf of the honest client u and sent to the server, are substituted with uniformly sampled values $(\mathbf{y}_u^*, \mathsf{com}_u^*, \sigma_u^*)$. Since $\mathcal{A}$ does not know the secrets of honest parities and $(\mathbf{y}_u, \mathsf{com}_u, \sigma_u)$ has the same distribution as $(\mathbf{y}_u^*, \mathsf{com}_u^*, \sigma_u^*)$, the view of the adversary in this hybrid is statistically indistinguishable from the previous one.

Hyb_4: This hybrid is same as Hyb_3 except that the simulator replaces the output of TEE instance M_e with $(\mathbf{k}_{\mathcal{V}}^*, \mathsf{com}^*, \mathsf{rand}^*, \sigma_e^*)$ and computes the aggregated mask $\mathbf{mask}_{\mathcal{V}}^*$ (as in Behavior 3 of Training phase of SIM) by querying the $F_{\alpha,\mathcal{X}^{(r)}}(\mathcal{V}^{(r)} \setminus \mathcal{C})$ and utilizing a random oracle $\mathcal{R}_{\mathsf{KhPRF}}$. Since $\mathcal{A}$ does not know the secrets of honest parities and the aggregated mask in SIM have the same distribution as the one in REAL, this hybrid is indistinguishable from the previous one. Note that the indistinguishability of this hybrid relies on the hiding property of the underlying vector Pedersen commitment scheme and the security property of the signature schemes.

Hyb_5: This hybrid is same as Hyb_4 except one modification: SIM sets the output of the ideal functionality $F_{\alpha,\mathcal{X}^{(r)}}(\mathcal{V}^{(r)} \setminus \mathcal{C})$ (as in Behavior 2 of Training phase of SIM) as the aggregated model $\mathbf{x}^{(r)}$. The server sends $(\mathbf{x}^r, \mathsf{rand}^*, \sigma_e^*)$ to all clients. Note that the ideal functionality will not return $\perp$ in this case. This modification does not change the view seen by the adversary, and hence this hybrid is indistinguishable from the previous one.

After completing these hybrids, SIM simulates REAL without relying on inputs from the honest parties, and we can see that the joint view of $\mathcal{A}$ in the real execution (Hyb_0) is computationally indistinguishable from the output of SIM (Hyb_5).

Here, we omit the hybrids to the real execution REAL of our POT-OPSA protocol with result verification, as they undergo the same modifications as described above.

Next, we give the proof for result authenticity against a malicious server. At the end of round r, the server sends $(\mathbf{x}^{(r)}, \mathsf{com}^{(r)}, \mathsf{rand}^{(r)}, \sigma_e^{(r)})$ to all clients. Here, $\mathbf{x}^{(r)} = \sum_{u \in \mathcal{V}^{(r)}} \mathbf{x}_u^{(r)}$ represents the sum of honest survivors (i.e, $\mathbf{z}_{\mathcal{H}}$), $\mathsf{com}^{(r)} = \mathsf{Commit}(\mathbf{x}^{(r)}, \mathsf{rand}^{(r)})$ denotes a valid commitment with a random number $\mathsf{rand}^{(r)}$, and $\sigma_e^{(r)} := \mathsf{Sign}(sk_e, \mathsf{com}^{(r)}||\mathsf{rand}^{(r)}||r)$ is a signature generated by TEE instance M_e. To forge the aggregation output, $\mathcal{A}$ can either tamper with an individual model from a specific client or generate random vectors as output results. Without loss of generality, let $\mathbf{z}_{\mathcal{A}} = \mathbf{z}_{\mathcal{H}} + \widetilde{\mathbf{x}_{\mathcal{A}}} + \widetilde{\mathbf{e}}$ be a tampered global model, where $\widetilde{\mathbf{x}_{\mathcal{A}}}$ denotes a tampered individual model and $\widetilde{\mathbf{e}}$ is an error vector introduced by the malicious server. There are two situations where a semi-honest client might accept $\mathbf{z}_{\mathcal{A}}$: one happens if the server can generate $\mathbf{z}_{\mathcal{A}}$ based on the same $\mathsf{com}^{(r)}$ and $\mathsf{rand}^{(r)}$, and the other happens if the server can forge the signature of TEE instance M_e for the new $\widetilde{\mathsf{com}}^{(r)}$ and $\widetilde{\mathsf{rand}}^{(r)}$ corresponding to $\mathbf{z}_{\mathcal{A}}$. Clearly, the former contradicts the binding property of vector Pedersen commitment scheme, and the latter violates the EUF-CMA secure signature scheme.

Therefore, for KhPRF-OPSA and POT-OPSA, we prove the security and result authenticity of our OPSA framework against the malicious server. □