IACR Transactions on Cryptographic Hardware and Embedded Systems
ISSN 2569-2925, Vol. 2025, No. 2, pp. 69–93. DOI:10.46586/tches.v2025.i2.69-93

SeaFlame: Communication-Efficient Secure Aggregation for Federated Learning against Malicious Entities

Jinling Tang[1,2], Haixia Xu[1,2](✉), Huimei Liao[1,2] and Yinchang Zhou[1,2]

[1] Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS, Beijing, China
[2] School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
{tangjinling,xuhaixia,liaohuimei,zhouyinchang}@iie.ac.cn

Abstract. Secure aggregation is a popular solution to ensuring privacy for federated learning. However, when considering malicious participants in secure aggregation, it is difficult to achieve both privacy and high efficiency. Therefore, we propose SeaFlame, a communication-efficient secure aggregation protocol against malicious participants. Inspired by the state-of-the-art work, ELSA, SeaFlame also utilizes two non-colluding servers to ensure privacy against malicious entities and provide defenses against boosted gradients. Crucially, to improve communication efficiency, SeaFlame uses arithmetic sharing together with arithmetic-to-arithmetic share conversion to reduce client communication, and uses the random linear combination to reduce server communication.
Security analysis proves that our SeaFlame guarantees privacy against malicious clients colluding with one malicious server. Experimental evaluation demonstrates that, compared to ELSA, SeaFlame optimizes communication by up to 10.5, 6.00, and 8.17 times for a client, a server, and all entities, at the expense of 1.25-1.86 times additional end-to-end runtime.

Keywords: Secure aggregation · communication efficiency · malicious privacy · federated learning

1 Introduction

Federated learning (FL) [MMR+17] has emerged as a promising paradigm for privacy-preserving distributed machine learning (ML). In a typical FL iteration, clients upload gradients to the server after local training, and the server computes the aggregate of gradients received to update the global model and broadcast the fresh global model to clients. It ensures privacy by enabling clients to upload their local gradients instead of exposing sensitive raw data.

However, the discovery of the gradients' vulnerability, which could leak private information about raw data [ZLH19], [YMV+21], highlights the significance of gradient privacy. Therefore, gradients should also be aggregated in a privacy-preserving way. Secure aggregation is one of the most popular solutions. The work [BIK+17] first focuses on mobile devices and presents dropout-robust secure aggregation based on Shamir secret sharing (SSS) and pairwise random masking, such that surviving clients can still finish secure aggregation even if some clients drop out.

Besides dropout, which can also happen to benign parties, many works consider the malicious behaviors of parties. This consideration is essential because, in the real world, malicious adversaries may corrupt both clients and servers. For example, the client or the

Received: 2024-10-15 Accepted: 2024-12-15 Published: 2025-03-04

server could deviate from the protocol specification to obtain information about individual gradients, and the client could intentionally poison its gradients. This promotes two desirable security properties for secure aggregation: malicious privacy and input validation.

Many studies guarantee either the privacy of individual gradients [BIK$^+$17], [TBA$^+$19], [BBG$^+$20], [MWA$^+$23], [TXW$^+$24] or resilience to malformed gradients [BMGS17], [SKSM19], [CFLG21]. Satisfying both of them appears to be a dilemma. Intuitively, the privacy of gradients also provides convertivity for poisonous gradients, thereby increasing the difficulty of detecting malformed gradients. Some works address this dilemma, but they either fall outside the malicious model [AGJ$^+$22], [MMM$^+$22], [PKH22] or are inefficient [CB17], [CGJvdM22], [LBV$^+$23], particularly for large-scale systems.

ELSA [RSWP23] is the state-of-the-art secure aggregation protocol that addresses all these concerns. It focuses on a practical model poisoning attack in the FL setting: malicious clients boost local gradients by scaling up their gradients to a large norm. Such an attack can easily bias the global model. Based on non-colluding two servers, ELSA can filter out boosted gradients and achieves malicious privacy in a lightweight way. To improve efficiency, it lets each client generate cryptographic correlations (i.e., oblivious transfers and square correlations) locally instead of having two servers generate them interactively. Despite these desirable properties, the task of generating both oblivious transfers (OTs) and square correlations brings each client considerable communication overhead, especially when the number of gradients is large. Actually, communication efficiency has always been one of FL's research hotspots [MMR$^+$17], [BBG$^+$20], [KMY$^+$16], [SWMS20], [GLL$^+$21], [KMA$^+$21], and online secure aggregation is often the communication bottleneck for the entire FL system.

Therefore, we propose SeaFlame, a more communication-efficient secure aggregation protocol against malicious entities. We base SeaFlame on ELSA's architecture, replacing key components to reduce communication overhead, as detailed in Section 4. SeaFlame also aims to defend against malicious clients boosting local gradients. Gradient explosions may occur during model training, leading to non-converging or low accuracy. Malicious clients could take advantage of this vulnerability by uploading boosted gradients, which would cause the averaged global gradients to explode based on the aggregation feature. Norm-bounding, which commonly uses 2-norm and infinite norm bounding, is resilient to such an attack. Previous works [SKSM19] and [SHKR22] have pointed out that using a bounded 2-norm to filter out boosted gradients is effective against many model poisoning attacks for practical FL. Because the fraction of malicious clients manipulated by adversaries is small in practical FL, their poisonous local gradients have no clear impact on the averaged global gradients after using norm-bounding. So SeaFlame utilizes norm-bounding to defend against boosted gradients. Note that SeaFlame can't mitigate malformed gradients less than the norm bounds (e.g., shrunk gradients) because they are indistinguishable from the benign gradients in SeaFlame. Defenses against other malformed gradients are of interest for our future research.

Besides, SeaFlame achieves only privacy in the malicious setting, because achieving both privacy and correctness in the malicious setting usually needs heavy cryptographic operations, which is impractical in real-world applications. Malicious privacy is a commonly-studied security property of many previous works (see Table 1).

1.1 Contributions

Our main contributions are summarized as follows:

- We propose SeaFlame, utilizing two techniques to optimize communication. First, SeaFlame consumes fewer OTs by switching from Boolean to arithmetic secret sharing and from Boolean-to-arithmetic share conversion (B2A) to arithmetic-to-arithmetic share conversion (A2A), which reduces the number of OTs by a factor of input bitlength. Both clients and servers benefit from this technique. Then, SeaFlame adopts a random linear combination to batch the second message during square correlation verification (see Section 2.5 for details of square correlation and how to verify it), further reducing server communication overhead.

- We formally prove SeaFlame's malicious privacy, i.e., privacy when malicious clients collude with a malicious server.
- We evaluate SeaFlame using a variety of experiments including performance comparison with state-of-the-art work, overhead breakdown, and combination with the training phase of a concrete FL task. The comparative experiments show that SeaFlame reduces communication by up to 10.5, 6.00, and 8.17 times for a single client, a single server, and all entities, at the expense of 1.25-1.86 times additional computation overhead. It is highly efficient in practical FL scenarios.

1.2 Related Work

Here we focus on secure aggregation based on cryptographic tools. Based on the number of servers participating in aggregation, we can divide these works into two categories: *secure aggregation with a single server* and *secure aggregation with non-colluding two servers.*

1.2.1 Secure aggregation with a single server

Since Google designed a dropout-robust single-server secure aggregation protocol for FL [BIK+17] using pairwise random masks for privacy and SSS for dropout robustness, many follow-up works have built on this idea and looked for improvements. Bell et al. [BBG+20] adopted a k-neighbors communication network to reduce client communication. However, both of them lack aggregate verification. To make aggregates verifiable, VerifyNet [XLL+20] combined zero-knowledge proof (ZKP) with a linear homomorphic hash (LHH), while VeriFL [GLL+21] used Pedersen commitment [Ped91] instead of ZKP to enable more communication-efficient verification, which is independent of the number of gradients. Unfortunately, none of these works can be resilient to poisonous gradients. EIFFeL [CGJvdM22] adopted secret-shared non-interactive proof (SNIP) [CB17] and verifiable secret sharing (VSS) [Fel87] to allow the server to verify whether the gradients inputted are well-formed. RoFL [LBV+23] used Bulletproofs [BBB+18] to enforce gradients within the specified norm bounds. ACORN [BGL+23] claimed that both EIFFeL's and RoFL's methods of input validation blew up communication and achieved communication efficiency by packing a batch of plaintext into a single ciphertext. Additionally, Flamingo [MWA+23] first tailored single-server secure aggregation to the multi-iteration setting and removed the per-iteration setup.

1.2.2 Secure aggregation with non-colluding two servers

Unlike single-server secure aggregation, the approach based on non-colluding two servers offers three inherent benefits:

- Almost full dropout tolerance for clients $(N - 2)$.
- Almost full corruption tolerance for clients $(N - 2)$.
- Client-to-client independence.

Since sharing the gradients to such two servers is able to keep gradients private, there is no need for clients to generate pairwise random masks or even communicate with other clients. Furthermore, without client-to-client communication, each client is independent of others, such that both dropout tolerance and corruption tolerance are almost full, i.e., $N-2$, where N is the number of clients.

Besides these inherent benefits, this type of work explores other properties. Prio [CB17] presented a system to collect aggregate statistics in a privacy-preserving way. It designed a new cryptographic tool, SNIP, to enforce inputs to be well-formed. Its efficiency-optimized work, Prio+ [AGJ+22], replaced SNIP with Boolean secret sharing to ensure inputs within a loose infinite-norm bound for free, but it assumed that all servers are semi-honest. Although Prio and Prio+ were not specifically designed for FL, their sum protocol significantly inspired secure aggregation in this context. ELSA [RSWP23] borrowed the trick of Prio+: Boolean secret sharing supports defense based on infinite norms without any extra overhead. It also added defense based on 2-norms with the help of cryptographic correlations. Without heavy public-key operations and ZKPs, ELSA provided both malicious privacy and input validation in an efficient way, but it also required considerable communication to handle these cryptographic correlations with high demand. Additionally, the work [TXW+24] achieved malicious security, while most of the previous works only achieved malicious privacy. However, it was quite inefficient beacuse of the heavy cryptographic tools and it didn't consider input validation. PINE [ROCT24] achieved optimal communication for 2-norm bounding but needed more operations for clients to generate ZKPs, and it didn't provide the infinite norm bounding.

Tab. 1 provides a qualitative comparison of the related works mentioned above.

Table 1: Qualitative comparison of related works. "One-shot clients" means that clients only send shares of all gradients and cryptographic correlations to the corresponding server as a single message and will not participate in later phases of the secure aggregation protocol.

	Single-server				**Non-colluding Two-server**				
	SecAgg [BIK+17], [BBG+20]	VerifyNet [XLL+20], VeriFL [GLL+21]	EIFFeL [CGJvdM22], RoFL [LBV+23]	ACORN [BGL+23]	Prio [CB17]	Prio+ [AGJ+22]	ELSA [RSWP23]	FlexScaAgg [TXW+24]	SeaFlame **(Ours)**
Malicious privacy	√	×	√	√	√	×	√	√	√
Input validation (Poisoning defense)	×	×	√	√	√	√	√	×	√
Almost full dropout tolerance	×	×	×	×	√	√	√	√	√
Almost full corruption tolerance	×	×	×	×	√	√	√	√	√
Client-to-client independence	×	×	×	×	√	√	√	√	√
One-shot clients	×	×	×	×	√	√	√	×	√
Efficient entities	√	√	×	√	×	√	√	×	√
Lightweight communication	√	√	×	√	√	√	×	×	√

2 Preliminaries

This section describes the necessary notations and cryptographic tools in our protocol.

2.1 Notations

We explain the meaning of notations in Tab. 2.

2.2 Arithmetic secret sharing

All arithmetic secret sharing in this paper is additive sharing. SeaFlame primarily utilizes arithmetic sharing between two servers, so we introduce 2-party arithmetic secret sharing. In 2-party arithmetic secret sharing, a value $x \in \mathbb{Z}_p$ will be split into two shares $x^{(0)}$, $x^{(1)}$ such that $x^{(0)} + x^{(1)} = x \pmod{p}$. Furthermore, Boolean sharing is a special instance of arithmetic sharing when $p = 2$.

Table 2: Notations with the corresponding meanings.

Notation	Meaning
S_0, S_1	two non-colluding servers
c	the id of clients
l_2, l_∞	the 2-norm and the infinite norm
N	the total number of clients
$[N]$	the set of integers $\{0, 1, \ldots, N-1\}$
d	the dimension of gradients
λ, κ	the computational and statistical security parameters
$\wedge, \oplus$	the bitwise AND and XOR
$\|$	concatenation of strings
$\mathsf{OTSn}, \mathsf{OTRc}$	OT sender and receiver

2.3 Defenses based on vector norms

This paper focuses on the infinite norm (l_∞ norm) and 2-norm (l_2 norm). The l_∞ norm is defined as the maximum value of all entries of a vector, and the l_2 norm is also known as the Euclidean norm. Given a vector $(x_1, \ldots, x_n)$, its l_2 value is $\sqrt{x_1^2 + \cdots + x_n^2}$. In this paper, the defenses against malicious clients providing boosted gradients are based on these two norms. Moreover, for l_∞ defense, we use a loose upper bound, which is larger than the accurate l_∞ value; for l_2 defense, we compare the square of l_2 value with the square of l_2 bound. Note that we should avoid wrap-arounds when computing l_2 value over cryptographic finite fields. In SeaFlame, if servers find out any gradient vector's norm is larger than the specified bound during secure computation, they will discard the gradient vector.

2.4 Oblivious Transfer (OT)

A "1-out-of-2" OT, $\binom{2}{1}$-OT, means: the sender OTSn holds two messages m_0, m_1, the receiver OTSn holds a choice bit $j \in \mathbb{Z}_2$, and then the sender learns nothing and the receiver only learns the message m_j. Random OT (ROT) is the most general form of OT, where messages m_0, m_1 are randomly chosen. Correlated OT (COT) [ALSZ13] means two messages of the sender are correlated, for example, $m, m+r$, where the offset $r \in \{0,1\}^\lambda$. In SeaFlame, we only use "1-out-of-2" OTs. "1-out-of-2" OTs can be used to construct bit multiplication protocol (Protocol 2 can be an example), which can be further used to convert shares. Unless specifically stated, the OT in the rest of this paper refers to "1-out-of-2" COT, and Δ-COT denotes the COT whose offset is Δ.

2.5 Square correlation

Square correlations are helpful randomness resources for computing squares when computing the value of the l_2 norm. A square correlation over $\mathbb{Z}_q$ is a pair of values formed like (α, α^2), where $\alpha \leftarrow_\$ \mathbb{Z}_q$. Actually, square correlation is a special kind of Beaver triple and a better choice to compute squares than Beaver triple [RSWP23]. A Beaver triple over $\mathbb{Z}_q$ is a tuple of values formed like (a, b, ab), where $a, b \leftarrow_\$ \mathbb{Z}_q$. Beaver triples are used to aid online multiplications in MPC, and square operation is a special kind of multiplication. For computing squares, using square correlations instead of Beaver triples incurs reduction. Likewise, SPDZ sacrifice technique [DPSZ12], [KPR18] can also be used to verify square correlations, the idea of which is sacrificing one square correlation to verify another square correlation. For example, given two correlations $(a, d), (\hat{a}, \hat{d})$ (sacrificing the latter one to verify the former one), server S_b holds $(a^{(b)}, d^{(b)}), (\hat{a}^{(b)}, \hat{d}^{(b)})$. Servers collectively sample a random value $t \in \mathbb{Z}_q$, let $e = ta - \hat{a}$, if $(a, d), (\hat{a}, \hat{d})$ are two well-formed square correlations, then

$$t^2 d - \hat{d} = t^2 a^2 - \hat{a}^2 = (ta - \hat{a})(ta + \hat{a}) = e(e + 2\hat{a}) = 2tea - e^2.$$

After opening e, server S_0 computes $\hat{t}^{(0)} = t^2 d^{(0)} - \hat{d}^{(0)} - 2tea^{(0)} + e^2$, and S_1 computes $\hat{t}^{(1)} = t^2 d^{(1)} - \hat{d}^{(1)} - 2tea^{(1)}$. Therefore, servers can verify the correctness of square correlations by checking whether $\hat{t}^{(0)} + \hat{t}^{(1)} = 0$. If it holds, then $(a, d), (\hat{a}, \hat{d})$ are two well-formed square correlations with overwhelming probability.

3 System Model

3.1 Entity Setup

In SeaFlame, we set up two types of entities:

- **Clients.** There are a total of N clients. Each client may be a company, an organization, or even a person holding private data in the real world. They help with the FL task but don't want to reveal any private information. After local training based on its own dataset, the client shares the model gradients and cryptographic correlations to servers, and doesn't need to participate in subsequent phases until the next iteration.

- **Servers.** A server is someone who collects local gradients and integrates them into the new global model. There are two servers, each of which won't collude with the other server. During each iteration, the servers broadcast the current global model and perform an interactive 2-party secure computation (2PC) protocol to compute the aggregates after receiving messages from the clients. Based on the aggregates, servers update the global model.

Each entity in SeaFlame communicates through secure channels. We assume that servers in SeaFlame never drop out. Our SeaFlame focuses on the secure aggregation phase, excluding local training and updating global models.

3.2 Threat Model

In SeaFlame, we consider malicious adversaries, assuming two servers never collude with each other.

- **Malicious clients.** Since the clients only generate messages at the beginning of SeaFlame, their malicious behaviors can include sharing boosted gradients and incorrect cryptographic correlations. We assume that malicious clients will not provide shrunk gradients because they have little impact on the global model, particularly when the number of clients participating in aggregation is large and can't make the global gradients explode.

- **At most one malicious server.** The malicious server may deviate arbitrarily from SeaFlame's specification to obtain any private information about honest clients. Note that the correctness of the malicious model is beyond SeaFlame, implying that SeaFlame can't prevent the malicious server from manipulating the aggregates.

SeaFlame guarantees privacy against malicious adversaries controlling parts of clients and one server, which can also be viewed as some malicious clients colluding with a malicious server. Formal proofs are given in Section 5.

3.3 Design goals

SeaFlame aims to achieve the following design goals:

- **Privacy of individual gradients.** SeaFlame aims to aggregate individual gradients from clients, and even a malicious adversary who controls parts of clients and one server cannot know any information about honest clients' gradients other than the aggregates and whether their gradients are within the norm bounds.
- **Defenses against boosted gradients.** SeaFlame provides two defenses based on the l_2 norm and the l_∞ norm, which enable detection of boosted gradients beyond the norm bounds. Note that these defenses are powerless for shrunk gradients and "artificial gradients" not generated by local training but within the norm bounds because of indistinguishability from the benign gradients. Defenses against shrunk gradients and gradients within the norm bounds are not SeaFlame's design goals.
- **Communication Efficiency.** SeaFlame expects to optimize communication overhead, because the online secure aggregation protocol often poses the communication bottleneck for the entire FL system. For secure aggregation, computation performance is secondary to communication, because local training also needs heavy operations, such that the impact of computation overhead of secure aggregation on the entire FL system is not as significant as the impact of communication overhead. SeaFlame minimizes communication overhead as much as possible without introducing unaffordable computation overhead.

4 SeaFlame

In this section, we illuminate the details of how we construct SeaFlame.

4.1 Overview

We use the architecture of state-of-the-art work based on two non-colluding servers as our foundation. Clients prepare shares of both gradients and cryptographic correlations (including OTs and square correlations) for two servers, then two servers perform 2PC to check whether gradients are boosted and finish aggregation. More specifically, SeaFlame works as follows:

- **Client.** Each client performs three steps locally: (1) splits gradients into two parts by arithmetic secret sharing; (2) generates enough OTs; and (3) generates enough square correlations. The client sends these three types of messages to the corresponding server in one shot.
- **Servers.** After receiving messages from clients, servers interact with each other to perform six steps: (1) verify OTs with additional OTs [KOS15]; (2) verify square correlations with additional square correlations through SPDZ sacrifice; (3) convert arithmetic shares over the small field to arithmetic shares over the large field with the help of OTs; (4) compute the l_2 value with the help of square correlations; (5) securely compare the l_2 value to the l_2 bound with the help of OTs; and (6) aggregate arithmetic shares from clients and open the final aggregates.

4.2 Reduce OT Usage

As we know, the essence of the l_∞ defense is to enforce each component of input vectors within a certain range. The typical method is to provide a range proof for each component, which brings significant expenses because of the hundreds of thousands of components per vector. Prio+ [AGJ+22] utilizes Boolean shares of inputs to ensure the bitlength of inputs, which can be viewed as a loose l_∞ bound. This method provide a l_∞ defense without extra overhead. To compute aggregates, these Boolean shares should be converted to arithmetic shares. The well-studied technique of Boolean-to-arithmetic share conversion needs one bit-to-arithmetic conversion for each bit of Boolean shares. One bit-to-arithmetic conversion can be performed by consuming one OT, so the required quantity of OTs for each Boolean-to-arithmetic share conversion is equal to the bitlength of Boolean shares.

We find that the required quantity of OTs greatly affects the efficiency of the clients and the entire protocol. To reduce the required number of OTs, our main goal is to find a solution to lifting shares over a small field to ones over a larger field, which consumes fewer OTs. As we know, Boolean secret sharing views each gradient as a bitstring and performs bitwise sharing, while arithmetic sharing views each gradient as one element in a finite field. The intuition is that using arithmetic sharing may benefit from the algebraic structure or properties of number theory to reduce OT's bitlength-dependent quantity. Thus, we choose arithmetic secret sharing rather than Boolean sharing to split gradients and explain how to convert the shares to ones over a larger field. It is fortunate that this alternative indeed reduces the usage of OTs. Next, we explain the technical details.

Arithmetic-to-arithmetic share conversion (A2A). Each client splits its input $x \in \mathbb{Z}_p$ into a pair of shares $(x^{(0)}, x^{(1)})$ such that $x^{(0)} + x^{(1)} = x \pmod p$. Servers can trust that the client's input is less than p by the shares. Then each server S_b converts its share $x^{(b)}$ to $y^{(b)}$ over $\mathbb{Z}_{\widetilde{p}}$, where $\widetilde{p} > p$, such that $y^{(0)} + y^{(1)} = x \pmod{\widetilde{p}}$. Unfortunately, such a conversion is rather expensive. To address this problem, we use quotient transfer (QT) [KIM+18], inspired by Precio [DWA+21]. Now we describe the details of this technique.

For simplicity, we assume that $p, \widetilde{p}$ are two distinct odds. From $x^{(0)} + x^{(1)} = x \pmod p$, we can know that

$$x = x^{(0)} + x^{(1)} - \alpha \cdot p, \tag{1}$$

where $\alpha \in \{0, 1\}$. Further, when x is an even value, the parity of $x^{(0)} + x^{(1)}$ will be the same as the parity of α. It implies that α is equal to the XOR value of the least significant bit (LSB) of $x^{(0)}$ and $x^{(1)}$, i.e.,

$$\begin{aligned} \alpha &= \mathsf{LSB}(x^{(0)}) \oplus \mathsf{LSB}(x^{(1)}) \\ &= \mathsf{LSB}(x^{(0)}) + \mathsf{LSB}(x^{(1)}) - 2 \cdot \mathsf{LSB}(x^{(0)}) \cdot \mathsf{LSB}(x^{(1)}). \end{aligned} \tag{2}$$

According to Eq. (1), to get $y^{(0)}$ and $y^{(1)}$, we also need to share α such that $\alpha = \alpha^{(0)} + \alpha^{(1)} \pmod{\widetilde{p}}$. $\forall b \in \{0, 1\}$, each server S_b can update its share by computing $y^{(b)} = x^{(b)} - \alpha^{(b)} \cdot p \pmod{\widetilde{p}}$, such that $x = y^{(0)} + y^{(1)} \pmod{\widetilde{p}}$.

To get $\alpha^{(0)}$ and $\alpha^{(1)}$, S_0 and S_1 need to run a bit multiplication protocol to compute the shares of $\mathsf{LSB}(x^{(0)}) \cdot \mathsf{LSB}(x^{(1)})$. Here we use an aligned bit multiplication protocol [RSWP23], which requires less communication.

To use the technique above, we must ensure that x is an even value. It is naive to consider $2x$. Although $2x$ is an even value, $2x \bmod p$ is uncertain. Thus, we consider two cases: $x < \frac{p}{2}$ and $x > \frac{p}{2}$.

Case 1: $x < \frac{p}{2}$. In this case, $2x < p$, so $2x$ is an even element in $\mathbb{Z}_p$. Similar to Eq. (1), we have

$$2x = \overline{x}^{(0)} + \overline{x}^{(1)} - \beta_1 \cdot p, \tag{3}$$

where $\overline{x}^{(0)} + \overline{x}^{(1)} = 2x \pmod p$ and $\beta_1 \in \{0, 1\}$. Each server S_b can obtain its share of $2x \bmod p$ by computing $\overline{x}^{(b)} = 2x^{(b)} \pmod p$. Then $\beta_1 = \mathsf{LSB}(\overline{x}^{(0)}) \oplus \mathsf{LSB}(\overline{x}^{(1)})$ so that we can use the technique above:

$$\overline{y}^{(b)} = \overline{x}^{(b)} - \beta_1^{(b)} \cdot p \pmod{\widetilde{p}}, \tag{4}$$

such that $\beta_1 = \beta_1^{(0)} + \beta_1^{(1)} \pmod{\widetilde{p}}$. Then $y^{(b)} = 2^{-1} \cdot \overline{y}^{(b)} \pmod{\widetilde{p}}$, since 2 is invertible over $\mathbb{Z}^*_{\widetilde{p}}$.

Case 2: $x > \frac{p}{2}$. In this case, $p < 2x < 2p$, so $2x - p$ is an odd element in $\mathbb{Z}_p$. Similar to Eq. (1), we have

$$2x - p = \overline{x}^{(0)} + \overline{x}^{(1)} - \beta_2 \cdot p, \tag{5}$$

where $\overline{x}^{(0)} + \overline{x}^{(1)} = 2x \pmod p$ and $\beta_2 \in \{0,1\}$. Each server S_b can obtain its share of $2x \bmod p$ by computing $\overline{x}^{(b)} = 2x^{(b)} \pmod p$.

$$2x = \overline{x}^{(0)} + \overline{x}^{(1)} + (1 - \beta_2) \cdot p, \tag{6}$$

We also have $1 - \beta_2 = \mathsf{LSB}(\overline{x}^{(0)}) \oplus \mathsf{LSB}(\overline{x}^{(1)})$. Let $\overline{\beta}_2 = 1 - \beta_2$. Note that $\overline{\beta}_2 \in \{0,1\}$, because $\beta_2 \in \{0,1\}$. We also use the technique above:

$$\overline{y}^{(b)} = \overline{x}^{(b)} + \overline{\beta}_2^{(b)} \cdot p \pmod{\widetilde{p}}, \tag{7}$$

such that $\overline{\beta}_2 = \overline{\beta}_2^{(0)} + \overline{\beta}_2^{(1)} \pmod{\widetilde{p}}$. Then $y^{(b)} = 2^{-1} \cdot \overline{y}^{(b)} \pmod{\widetilde{p}}$.

According to Eq. (4) and (7), such a separate discussion above causes a difference in the protocol process. A naive attempt is to perform equivalent deformation on Eq. (6):

$$2x = \overline{x}^{(0)} + \overline{x}^{(1)} - (\beta_2 - 1) \cdot p. \tag{8}$$

Unfortunately, $\beta_2 - 1 = \mathsf{LSB}(\overline{x}^{(0)}) \oplus \mathsf{LSB}(\overline{x}^{(1)})$ is incorrect, because $\beta_2 - 1 \in \{-1, 0\}$ instead of $\{0,1\}$.

Due to this negative result, we choose **Case 2** to construct our A2A protocol, so the required input is $\frac{p}{2} < x < p$. We assume that malicious clients will not provide input $x < \frac{p}{2}$. In FL, malicious clients prefer to input boosted gradients to manipulate global models because the global models are derived from the average of aggregates; boosted gradients may lead to the average's explosion that results in the global model's not converging or low accuracy, and shrunk gradients often have little impact on global models. Moreover, one of the main goals of our protocol is to filter out boosted gradients, other forms of malicious gradients are beyond the scope of this work and left for future research.

The detailed description of our optimized A2A protocol is shown in Protocol 1. If we use B2A like previous work, each server will consume $\lceil \log p \rceil$ OTs per client gradient. Our optimized A2A consumes only one OT for each gradient; therefore, the number of OTs consumed has decreased by $\lceil \log p \rceil$ times.

Protocol 1 Optimized A2A $\Pi_{\mathsf{A2A}}^{p,\widetilde{p}}$

Input: Arithmetic shares $x^{(b)} \in \mathbb{Z}_p$ to convert, where $b \in \{0,1\}$ and $x > \frac{p}{2}$. As a OT sender OTSn, needs additional inputs $\Delta, W \in \mathbb{F}_{2^\lambda}$. As a OT receiver OTRc, needs additional inputs $T \in \mathbb{F}_{2^\lambda}$. Require that p and $\widetilde{p}$ are odd.

Output: $y^{(b)} \in \mathbb{Z}_{\widetilde{p}}$ for each $b \in \{0,1\}$, such that

$$y^{(0)} + y^{(1)} \pmod{\widetilde{p}} = x^{(0)} + x^{(1)} \pmod p.$$

Steps: Between servers (fixing S_0 as OTSn).

```
1: $\forall b \in \{0,1\}$, $S_b$ computes $\overline{x}^{(b)} = 2x^{(b)} \bmod p$.
2: $S_0$ sets $m^{(0)} \leftarrow \Pi_{\mathsf{AliBitMult}}^{\widetilde{p}}(\mathsf{LSB}(\overline{x}^{(0)}), \Delta, W)$.
3: $S_1$ sets $m^{(1)} \leftarrow \Pi_{\mathsf{AliBitMult}}^{\widetilde{p}}(\mathsf{LSB}(\overline{x}^{(1)}), T)$.
4: $\forall b \in \{0,1\}$, $S_b$ computes $z^{(b)} = \mathsf{LSB}(\overline{x}^{(b)}) - 2m^{(b)} \pmod{\widetilde{p}}$.
5: $\forall b \in \{0,1\}$, $S_b$ computes $\overline{y}^{(b)} = \overline{x}^{(b)} + z^{(b)}p \pmod{\widetilde{p}}$.
6: $\forall b \in \{0,1\}$, $S_b$ computes $y^{(b)} = 2^{-1} \cdot \overline{y}^{(b)} \bmod \widetilde{p}$.
```

Protocol 2 Aligned Bit Multiplication $\Pi^{\widetilde{p}}_{\mathsf{AliBitMult}}$

Input: Bit shares $x^{(b)} \in \mathbb{Z}_2$ to multiply, where $b \in \{0,1\}$. As a OT sender OTSn, needs additional inputs $\Delta, w \in \mathbb{F}_{2^\lambda}$. As a OT receiver OTRc, needs additional inputs $t \in \mathbb{F}_{2^\lambda}$. Let $H : [N] \times \mathbb{F}_{2^\lambda} \to \mathbb{Z}_{2^\lambda}$ be a hash function in the random oracle model.
Output: $y^{(b)} \in \mathbb{Z}_{\widetilde{p}}$ for each $b \in \{0,1\}$, such that

$$y^{(0)} + y^{(1)} \pmod{\widetilde{p}} = x^{(0)} \wedge x^{(1)}.$$

Steps: Between servers (fixing S_0 as OTSn).

```
1: $S_0$ computes $v_0 = H(c\|w), v_1 = H(c\|w + \Delta)$, where $c$ is a global counter.
2: $S_0$ sets $y^{(0)} \leftarrow -v_0 \pmod{\widetilde{p}}$.
3: $S_0$ sends $u = v_0 + v_1 + x^{(0)} \pmod{\widetilde{p}}$ to $S_1$.
4: $S_1$ computes $v = H(c\|t)$.
5: $S_1$ sets $y^{(1)} \leftarrow x^{(1)}u + (-1)^{x^{(1)}}v \pmod{\widetilde{p}}$.
```

Protocol 3 Local OT Correlation Generation $\mathsf{LocalOT}$

Input: Choice bits $x \in \mathbb{Z}_2^n$ and an offset $\Delta \in \mathbb{F}_{2^\lambda}$.
Output: W and T such that $W = T + x \cdot \Delta \in \mathbb{F}_{2^\lambda}^n$.
Steps: Locally at each client.

```
1: for $j \in [n]$ do
2:     Sample $w_j \in \mathbb{F}_{2^\lambda}$.
3:     Set $t_j \leftarrow w_j + x_j \cdot \Delta$.
   end for
4: Set $W \leftarrow \{w_0, \ldots, w_{n-1}\}$ and $T \leftarrow \{t_0, \ldots, t_{n-1}\}$.
```

4.3 Square Correlation

Square correlation is the optimal resource for helping compute squares during the l_2 norm computation, as opposed to the Beaver triple. Next, we analyze key issues regarding square correlations. Let all of the square correlations be over $\mathbb{Z}_q$.

Choice of q. Note that square correlations are defined over $\mathbb{Z}_q$ but used to compute the l_2 value over $\mathbb{Z}_{\widetilde{p}}$. For simplicity, we only consider one square correlation here. Given a well-formed square correlation (a, d), i.e., $d \equiv a^2 \bmod q$, our goal is to find the relation between q and $\widetilde{p}$, such that $d \equiv a^2 \bmod \widetilde{p}$. According to the property of congruences, we find that $\widetilde{p} \mid q$ can satisfy this requirement.

Square correlation verification. According to the technique of SPDZ sacrifice [DPSZ12], each square correlation is verified by sacrificing another square correlation. For each pair of square correlations $(a, d), (\hat{a}, \hat{d})$, each server S_b holds $(a^{(b)}, d^{(b)}), (\hat{a}^{(b)}, \hat{d}^{(b)})$. Servers collectively sample a random value $t \in \mathbb{Z}_q$ and then open $e = ta - \hat{a}$, i.e., $\forall b \in \{0,1\}$, S_b sends $e^{(b)} = ta^{(b)} - \hat{a}^{(b)}$ to peer S_{1-b}, which is the first message to be sent during square correlation verification. Then S_0 sends $\hat{t}^{(0)} = t^2 d^{(0)} - \hat{d}^{(0)} - 2tea^{(0)} + e^2$ to S_1, S_1 sends $\hat{t}^{(1)} = t^2 d^{(1)} - \hat{d}^{(1)} - 2tea^{(1)}$ to S_0, so that each server can check the correctness of (a, d) by checking whether $\hat{t}^{(0)} + \hat{t}^{(1)} = 0 \pmod q$. $\hat{t}^{(b)}$ is the second message to be sent. Therefore, to verify d square correlations, each server needs to send $2d$ elements over $\mathbb{Z}_q$.

To further reduce server communication, we use the technique of random linear combinations to batch the second message during square correlation verification. Instead of sending $\hat{t}^{(b)}$ for per-correlation verification to peer server, this technique packs all $\hat{t}^{(b)}$s into a single element by using randomness. The randomness for a random linear combination can be generated by a common random tape of two servers (two party multiple coins flipping protocol). As a result, server-to-server communication during square correlation verification is reduced by about two times, from $2d$ elements to $d+1$ elements, where d denotes the number of square correlations to be verified.

4.4 Full Protocol

We put the previous parts all together, thus forming the full protocol in Protocol 4. Note that the expected inputs for SeaFlame are larger than $\frac{p}{2}$ and less than p. It is not difficult to encode the gradients generated from local training to this range.

Protocol 4 Secure Aggregation Π_{Agg}

Input: Input vectors of dimension d and each input is larger than $\frac{p}{2}$ and less than p. Two different big primes p, η and a big odd $\widetilde{p}$ such that $\widetilde{p} \gg Np$ and $q = \eta\widetilde{p}$. For the l_2 defense, μ is the upper bound to enforce. For the l_∞ defense, p is the upper bound to enforce. We fix the server S_0 as the OTSn and S_1 as the OTRc. Let $n = d + 2 \cdot \lceil \log \widetilde{p} \rceil$. λ is the computational security parameter.

Output: Aggregate vector over $\mathbb{Z}_{\widetilde{p}}^d$.

Input Sharing: Locally at each client.

1: For each client c, let x denote the input of this client and x_i is its i-th entry, where $i \in [d]$.
2: Generate shares $x^{(0)}, x^{(1)}$ of x over $\mathbb{Z}_p^d$ and send $x^{(b)}$ to S_b.

OT Generation: Locally at each client.

1: Each client c samples $\Delta \in \mathbb{F}_{2^\lambda}$ and $r \in \mathbb{Z}_2^{2\cdot\lceil \log \widetilde{p} \rceil}$.
2: Each client c samples $\overline{x}^{(1)} = 2x^{(1)} \bmod p$.
3: Each client c generates OT correlations W, T and W', T' by calling the $\mathsf{LocalOT}$ as sub-protocol on inputs $\mathsf{LSB}(\overline{x}^{(1)}) \in \mathbb{Z}_2^d$, r and Δ:

$$\begin{aligned}(W, T) &\leftarrow \mathsf{LocalOT}(\mathsf{LSB}(\overline{x}^{(1)}), \Delta),\\ (W', T') &\leftarrow \mathsf{LocalOT}(r, \Delta).\end{aligned} \tag{9}$$

4: Each client c sends Δ and $W \leftarrow (W \parallel W')$ to S_0, and r and $T \leftarrow (T \parallel T')$ to S_1.

Square Correlation Generation: Locally at each client.

1: Each client c samples $\{a_i\}_{i=1}^{2d} \in \mathbb{Z}_q^{2d}$.
2: Each client c generates arithmetic shares $M^{(0)}, M^{(1)}$ of $\{(a_i, d_i)\}_{i=1}^{2d}$, where $d_i = a_i^2 \bmod q$.
3: Each client c sends $M^{(b)}$ to S_b, where $b \in \{0, 1\}$.

OT Verification: Between servers.

Servers perform the following steps for each client c:

1: Servers S_0, S_1 collectively sample randomness $\{\chi_i\}_{i=1}^n \in \mathbb{F}_{2^\lambda}^n$.
2: S_1 sets $\hat{r} \leftarrow (\mathsf{LSB}(\overline{x}^{(1)}) \parallel r)$ and computes:

$$\tilde{r} = \sum_{j=1}^{n} \hat{r}_j \cdot \chi_j \text{ and } \tilde{t} = \sum_{j=1}^{n} T_j \cdot \chi_j, \tag{10}$$

where r_j is the j-th bit of $\hat{r}$ and $T_j \in \mathbb{F}_{2^\lambda}$ is the j-th correlation in T. S_1 sends $\tilde{r}, \tilde{t}$ to S_0.
3: S_0 computes:

$$\tilde{w} = \sum_{j=1}^{n} W_j \cdot \chi_j, \tag{11}$$

where $W_j \in \mathbb{F}_{2^\lambda}$ is the j-th correlation in W.
4: S_0 checks as follows:

```
5: **if** $\tilde{t} = \tilde{w} + \tilde{r} \cdot \Delta$ **then**
6:     $S_0, S_1$ continue the protocol.
7: **else**
8:     $S_0$ rejects this client.
9: $S_0, S_1$ split OTs into two sets $(W^A, T^A), (W^B, T^B)$ with $d$ correlations in the first set.
```

Square Correlation Verification: Between servers.

Servers perform the following steps for each client c:

```
1: $S_b$ sets $\hat{M}^{(b)} \leftarrow \{\}$.
2: **for** each pair of correlations $(a^{(b)}, d^{(b)}), (\hat{a}^{(b)}, \hat{d}^{(b)}) \in M^{(b)}$ **do**
3:     Collectively sample a random value $t \in \mathbb{Z}_q$.
4:     Servers open $e = ta - \hat{a}$.
5:     $S_0$ computes $\hat{t}^{(0)} = t^2 d^{(0)} - \hat{d}^{(0)} - 2tea^{(0)} + e^2$.
6:     $S_1$ computes $\hat{t}^{(1)} = t^2 d^{(1)} - \hat{d}^{(1)} - 2tea^{(1)}$.
   **end for**
7: $S_0, S_1$ collectively sample randomness $\{\phi_i\}_{i=1}^d \in \mathbb{Z}_q^d$.
8: $S_b$ computes the random linear combination, $\bar{t}^{(b)}$, of $\hat{t}^{(b)}$s by using $\phi_i$s.
9: Servers check whether $\bar{t}^{(0)}$, $\bar{t}^{(1)}$ are the shares of zero:
10: **if** yes **then**
11:     $S_b$ stores $\{(a_i^{(b)}, d_i^{(b)})\}_{i=1}^d$ as $\hat{M}^{(b)}$.
12: **else**
13:     Servers reject this client.
```

A2A Conversion: Between servers.

Servers perform the following steps for each client c:

```
1: **for** $i \in [d]$ **do**
2:     $S_0$ sets $z_i^{(0)} \leftarrow \Pi_{\mathsf{A2A}}^{p,\widetilde{p}}(x_i^{(0)}, \Delta, W_i^A)$.
3:     $S_1$ sets $z_i^{(1)} \leftarrow \Pi_{\mathsf{A2A}}^{p,\widetilde{p}}(x_i^{(1)}, T_i^A)$.
   **end for**
```

l_2 **Computation:** Between servers.

Servers perform the following steps for each client c:

```
1: $S_b$ sets $\tilde{z}^{(b)} \leftarrow 0 \in \mathbb{Z}_{\widetilde{p}}$.
2: **for** $i \in [d]$ **do**
3:     Servers open $e = z_i - a_i$.
4:     $S_0$ sets $\tilde{z}^{(0)} \leftarrow \tilde{z}^{(0)} + d_i^{(0)} + 2ez_i^{(0)} - e^2 \bmod \widetilde{p}$.
5:     $S_1$ sets $\tilde{z}^{(1)} \leftarrow \tilde{z}^{(1)} + d_i^{(1)} + 2ez_i^{(1)} \bmod \widetilde{p}$.
   **end for**
```

l_2 **Enforcement:** Between servers.

Servers perform the following steps for each client c:

```
1: $S_0$ sets $\tilde{z}^{(0)} \leftarrow \tilde{z}^{(0)} - \mu^2$.
2: $S_0, S_1$ extract the sign bit of $\tilde{z}$ by securely computing an adder [DSZ15] on inputs
   $\tilde{z}^{(b)}$. For each AND, $S_0$ calls $\Pi_{\mathsf{AND}}$ [RSWP23] with additional inputs $\Delta$ and fresh
   $w, w' \in W^B$, $S_1$ calls $\Pi_{\mathsf{AND}}$ with additional inputs $r$ and corresponding $t, t' \in T^B$.
3: **if** the sign bit is zero **then**
4:     Reject this client.
5: **else**
6:     Pass the check.
```

Aggregation: Between servers.

```
1: For $i \in [d]$, $S_b$ adds $z_i^{(b)}$ of all clients who pass the checks above together into $y_i^{(b)}$.
2: Servers open $y = \{y_i\}_{i=1}^d$.
```

5 Security Analysis

In this section, we prove SeaFlame's privacy against the malicious adversary corrupting parts of clients and one server.

Here we consider that the malicious adversary $\mathcal{A}$ controls parts of clients and one server. First, we give the following definitions and a theorem.

Ideal Functionality $\hat{\mathcal{F}}$. $\hat{\mathcal{F}}$ works as follows:

- Receives gradients from clients.
- Checks whether both the l_∞ and l_2 values of each gradient vector are within the required bound. All of unsatisified submissions are rejected.
- Receives the exclusion list specified by adversary $\mathcal{A}$, which denotes a set of clients, whose gradients will not be involved in the subsequent computation and the final aggregate.
- Discards the gradients of clients in the exclusion list and computes the aggregate of surviving gradients.
- Receives the output message specified by adversary $\mathcal{A}$, and outputs it.

Note that the exclusion list captures the power of $\mathcal{A}$ to force some clients to abort, which implies that a malicious server can falsely report the inputs of some clients as malformed. Furthermore, ideal functionality $\hat{\mathcal{F}}$ is corruptible, which allows $\mathcal{A}$ to specify the final output. The formal definition of corruptible ideal functionality can be found in [RSWP23].

Notations. We use $\mathcal{S}_H$ and $\mathcal{S}_M$ to denote the honest server and the malicious server, respectively. Similarly, $\mathcal{C}_H$ and $\mathcal{C}_M$ represent honest clients and malicious clients, respectively. Let $p_0 = \lceil \frac{p}{2} \rceil$ and N_H denote the number of honest clients involved in final aggregation.

Theorem 1. *For every PPT malicious adversary $\mathcal{A}$ controlling clients and at most one server, there exists a PPT simulator* SIM *in the ideal world such that the distributions*

$$\left\{\mathsf{Ideal}_{\hat{\mathcal{F}},\mathsf{SIM}(\mathsf{aux})}(\{x_i\}_{i=1}^{|\mathcal{C}|}, \lambda, \kappa)\right\}_{\{x_i\}_{i=1}^{|\mathcal{C}|},\mathsf{aux},\lambda,\kappa},$$

$$\left\{\mathsf{Real}_{\Pi,\mathcal{A}(\mathsf{aux})}(\{x_i\}_{i=1}^{|\mathcal{C}|}, \lambda, \kappa)\right\}_{\{x_i\}_{i=1}^{|\mathcal{C}|},\mathsf{aux},\lambda,\kappa}$$

are computationally indistinguishable in the $(\mathcal{F}_{\mathsf{CoinFlip}}, \mathcal{F}_{\mathsf{RO}})$-hybrid model, where aux *is an auxiliary input, $\mathcal{C}$ is the set of all clients with gradient inputs $\{x_i\}_{i=1}^{|\mathcal{C}|}$, and λ, κ are computational and statistical security parameters. $\mathsf{Ideal}_{\hat{\mathcal{F}},\mathsf{SIM}(\mathsf{aux})}(\{x_i\}_{i=1}^{|\mathcal{C}|}, \lambda, \kappa)$ and $\mathsf{Real}_{\Pi,\mathcal{A}(\mathsf{aux})}(\{x_i\}_{i=1}^{|\mathcal{C}|}, \lambda, \kappa)$ are output pairs of honest parties and the adversary in the ideal and real world, respectively.*

Proof. We start by defining a PPT simulator, which calls adversary $\mathcal{A}$ as a subroutine in a black-box manner.

Simulator SIM. SIM works as follows:

- Input Sharing, OT and Square Correlation Generation Phases:
  1. Generates dummy gradient inputs of all honest clients ($\mathcal{C}_H$) by setting them as vectors where all components are p_0, and then generates the corresponding shares of these inputs.
  2. Generates OT and square correlation shares according to the protocol specification.

3. Sends the corresponding (for $\mathcal{S}_M$) shares of inputs, OTs and square correlations above to $\mathcal{A}$.
4. Receives the shares (for $\mathcal{S}_H$) of inputs, OTs and square correlations from $\mathcal{A}$ as messages from malicious clients. If any share of a malicious client is missing, or if any input share of a malicious client is out of the l_∞ bound, ignores the corresponding client and adds it into the exclusion list $\mathcal{L}$, which captures the ability of $\mathcal{A}$ to cause malicious clients to abort early.
5. If $\mathcal{A}$ aborts, outputs whatever $\mathcal{A}$ outputs and sends $\perp$ to $\hat{\mathcal{F}}$ as the final output.

- OT and Square Correlation Verification Phases:
  1. Relays the calls to $\mathcal{F}_{\mathsf{CoinFlip}}$ from $\mathcal{S}_M$ and stores the outputs of these calls, i.e., common random challenges $\{\chi_i\}_{i=1}^n$, t and $\{\phi_i\}_{i=1}^d$, which are used to verify OTs and square correlations. Responds the outputs of $\mathcal{F}_{\mathsf{CoinFlip}}$ to $\mathcal{S}_M$.
  2. Detects if $\mathcal{A}$ causes $\mathcal{S}_M$ to send wrong random challenges about honest clients to $\mathcal{S}_H$. If yes, stops simulating any more messages from these clients and adds these clients into the exclusion list $\mathcal{L}$.
  3. Simulates messages acting like $\mathcal{S}_H$ to $\mathcal{S}_M$ during verification. Upon detecting any failure of verification for clients, adds these clients into the exclusion list $\mathcal{L}$ and stops simulating messages from honest clients among them.
  4. If $\mathcal{A}$ aborts, outputs whatever $\mathcal{A}$ outputs and sends $\perp$ to $\hat{\mathcal{F}}$ as the final output.

- A2A Conversion, l_2 Computation and l_2 Enforcement Phases:
  1. Follows the protocol specification and simulates messages from $\mathcal{S}_H$ to $\mathcal{S}_M$. Relays the calls to $\mathcal{F}_{\mathsf{RO}}$ from $\mathcal{S}_M$ when it invokes hash function in $\widetilde{\Pi}^p_{\mathsf{AliBitMult}}$ or Π_{AND}, and stores the output of these calls. Responds the outputs of $\mathcal{F}_{\mathsf{RO}}$ to $\mathcal{S}_M$.
  2. Upon detecting any noncompliance of l_2, adds the corresponding clients into the exclusion list $\mathcal{L}$ and stops simulating messages from honest clients among them. This also considers the situations where $\mathcal{S}_M$ stops sending messages for processing some clients' data.
  3. If $\mathcal{A}$ aborts, outputs whatever $\mathcal{A}$ outputs and sends $\perp$ to $\hat{\mathcal{F}}$ as the final output.

- Aggregation Phase:
  1. Sends zero vectors to $\hat{\mathcal{F}}$ as inputs of clients in $\mathcal{C}_M$.
  2. Sends the exclusion list $\mathcal{L}$ to $\hat{\mathcal{F}}$.
  3. Receives the aggregate of inputs of the surviving honest clients, A_H, which are not in $\mathcal{L}$.
  4. Follows the protocol to simulate the share of the aggregate from $\mathcal{S}_H$, and adds $A_H - N_H \cdot p_0$ (all components of A_H subtract $N_H \cdot p_0$) to the share to correct it. Sends the corrected share to $\mathcal{S}_M$.
  5. Receives the final message from $\mathcal{S}_M$ and adds it to the corrected share above to compute the final aggregate A.
  6. If $\mathcal{A}$ aborts, outputs whatever $\mathcal{A}$ outputs and sends $\perp$ to $\hat{\mathcal{F}}$ as the final aggregate.
  7. Sends A to $\hat{\mathcal{F}}$.
  8. Outputs whatever $\mathcal{A}$ outputs.

Next we prove the indistinguishability between the real and ideal worlds by defining a sequence of hybrids.

Hybrid 0 $\mathcal{H}_0$**.** This is actually the real world.

Hybrid 1 $\mathcal{H}_1$**.** Instead of letting honest clients generate OTs and square correlations, we have SIM generate the corresponding correlations, which is the only difference from $\mathcal{H}_0$. $\mathcal{A}$'s view is identically distributed to the previous one because both honest clients and SIM generate correlations uniformly at random.

If $\mathcal{S}_M$ is S_0 (OTSn)

Hybrid 2 $\mathcal{H}_2$**.** In this hybrid, SIM sends random share of vectors (where all components are p_0) to $\mathcal{S}_M$ as $x^{(0)}$ instead of honest clients randomly sharing their gradient vectors. All subsequent steps in the protocol execution use shares of such vectors, which will normally result in an inconsistent output. So SIM inputs zero vectors and $\mathcal{L}$ to $\hat{\mathcal{F}}$ to obtain the sum of honest clients' gradients, A_H, and adds $A_H - N_H \cdot p_0$ (all components of A_H subtract $N_H \cdot p_0$) back into the last message $y^{(1)}$ to obtain the final aggregate A from $\mathcal{A}$. SIM sends A back to $\hat{\mathcal{F}}$. This is actually the ideal world. $\mathcal{A}$'s view is computationally indistinguishable from the previous one. Because messages to $\mathcal{S}_M$ are masked by random correlations, and the vectors where all components are p_0 always satisfy the norm bound.

If $\mathcal{S}_M$ is S_1 (OTRc)

Hybrid 2 $\mathcal{H}_2$**.** In this hybrid, SIM sends random share of vectors (where all components are p_0) to $\mathcal{S}_M$ as $x^{(1)}$ instead of honest clients randomly sharing their gradient vectors. This is actually the ideal world. Similar to the previous case, $\mathcal{A}$'s view is computationally indistinguishable from $\mathcal{H}_1$.

□

6 Evaluation

We implement SeaFlame in the Rust language. The key libraries and cryptographic settings are as follows:

- For the end-to-end communication, we use Tokio [tok].
- For the security parameter, we choose 128 as our computational security parameter and vary the statistical security parameter according to subsequent specific experiments.
- For the OT, we use EMP Toolkit [emp].
- For the hash function, we use miTCCR [GKW+20].
- For the pseudorandom generator (PRG), we use hardware-accelerated AES.
- For arithmetic operations, we use Rug [rug], which supports operations for arbitrary-precision numbers.
- For the training phase of FL, we use tch-rs [tch], which provides Rust bindings for the C++ API of PyTorch.

First, we focus on secure aggregation and provide a performance comparison of SeaFlame with the state-of-the-art work, ELSA. Then we split these individual phases of SeaFlame and ELSA and analyze the impact of different bitlengths of gradients, i.e., input sizes, on our optimization. Finally, to simulate practical FL scenarios, we combine our secure aggregation with the FL local training phase.

6.1 Comparison to ELSA

Setup. We use an Alibaba Cloud instance with 512 GB of RAM and the CPU (Intel Xeon (Ice Lake) Platinum 9369B, 64 vCPUs) to simulate all clients in multi-threaded mode. We also deploy servers on two Alibaba Cloud instances with 256 GB of RAM and the CPU (Intel Xeon (Emerald Rapids) Platinum 8575C, 32 vCPUs). The machine settings are similar to those in ELSA. We deploy our experiments on both LANs and WANs (100 Mbps and 1 Gbps bandwidth for clients and servers, respectively). Unless otherwise specified, we fix the bitlength of gradients to 32 and carry out l_2 computation and aggregation over 64 bits. Thus, we set the prime $p = 2^{32} - 5$, and $\widetilde{p} = p^2$, and choose the prime $\eta = 2^{31} + 11$ to achieve the security level almost equivalent to the statistical security parameter $\kappa = 61$.

We start our performance comparison of SeaFlame with ELSA by varying the number of clients and gradients, which are denoted by N and d, respectively, in the previous text. We measure the outgoing communication and runtime of each client and server under groups of parameter settings. Note that we use a weaker machine with 32 GB of RAM and the CPU (Intel Core i9-12900H, 8 cores) when measuring the runtime of a single client to simulate a practical scenario, and the other indicators are still measured using the previous setup. We implement two servers in a balanced manner, i.e., each server serves as the OT sender for one half of clients and the OT receiver for the other half of clients, respectively.

Tab. 3 reports the detailed results of our comparison. The outgoing communication of each client in SeaFlame is about 10.5 times less than ELSA, and each server is about 5.97-6.00 times less. Moreover, the total outgoing communication of SeaFlame is about 8.16-8.18 times less than ELSA. With regard to runtime, when without bandwidth limits, each client in SeaFlame is about 3.68-3.98 times more than ELSA, and each server is about 1.56-2.45 times more. If network latency is not considered, the end-to-end runtime almost entirely depends on the servers' runtime. Therefore, the ratio of SeaFlame's end-to-end runtime to ELSA is close to the one for each server.

When over WANs setting limited bandwidth, the runtime of each client in ELSA increases to about 13.1-15.3 times than those without bandwidth limits, and the runtime of each server in ELSA increases to about 1.21-1.44 times. While the runtime of each client in SeaFlame increases more slowly and the runtime of each server is almost unaffected. In general, when bandwidth is limited, the runtime of each client in ELSA is about 2.36-2.58 times longer than SeaFlame's, and each server is 1.22-1.87 times shorter. Therefore, SeaFlame is more friendly to bandwidth-limited devices, especially for clients. Additionally, missing values of ELSA in Table 3 mean out of memory when running under the corresponding settings, and SeaFlame's runnability for large parameters also implies its efficiency optimization.

Table 3: Performance Comparison of SeaFlame with ELSA. Values denote outgoing communication (MB) and runtime (s; in parenthesis, the left one over LANs and right one over WANs with limited bandwidth) of each client and server.

#Cli.	#Grad.	**ELSA**		**SeaFlame**	
		Client	Server	Client	Server
50	100k	52.27 (0.044, 0.616)	801.85 (2.251, 2.958)	4.96 (0.167, 0.242)	134.28 (5.088, 4.995)
50	500k	261.31 (0.207, 3.195)	4009.25 (14.14, 17.16)	24.80 (0.804, 1.356)	671.39 (24.56, 24.22)
50	1M	522.62 (0.417, 6.402)	8018.50 (31.04, 39.79)	49.60 (1.614, 2.683)	1342.78 (48.31, 48.49)
100	100k	52.27 (0.043, 0.617)	1602.94 (4.016, 5.383)	4.96 (0.169, 0.239)	267.80 (9.828, 9.660)
100	500k	261.31 (0.218, 3.198)	8014.68 (24.81, 31.43)	24.80 (0.834, 1.352)	1338.97 (46.70, 46.51)
100	1M	522.62 (0.431, 6.403)	16029.36 (50.12, 72.23)	49.60 (1.631, 2.695)	2677.92 (95.75, 94.23)
500	100k	52.27 (0.047, 0.616)	8011.65 (19.58, 23.77)	4.96 (0.173, 0.242)	1335.94 (45.14, 44.40)
500	500k	261.31 (0.213, 3.203)	40058.16 (118.84, 136.35)	24.80 (0.845, 1.356)	6679.57 (228.19, 228.32)
500	1M	-	-	49.60 (1.635, 2.699)	13359.10 (463.53, 465.26)
1k	100k	52.27 (0.044, 0.623)	16022.54 (40.62, 47.97)	4.96 (0.170, 0.246)	2671.12 (89.31, 88.21)
1k	500k	-	-	24.80 (0.842, 1.367)	13355.32 (447.73, 449.91)
1k	1M	-	-	49.60 (1.667, 2.706)	26710.58 (905.95, 905.67)

Moreover, we consider the special settings of gradient dimension to fit four popular ML models: CNN, LeNet-5 [LeC15], ResNet-18 [HZRS16] and LSTM [HS97]. This is quite similar to RoFL [LBV+23] and ELSA [RSWP23]. We fix the bitlength of gradients and the number of clients to 32 and 100, respectively. Tab. 4 shows the concrete comparison of both the total outgoing communication and the end-to-end runtime. SeaFlame achieves about 8.16-8.17 times communication optimization at the expenses of 2.25-2.86 times more runtime than ELSA.

Table 4: Comparison of total outgoing communication (GB) and end-to-end runtime (s) when setting the number of gradients to fit four popular ML models. Besides, we fix the number of clients to 100.

Model	#Gradients	**ELSA**		**SeaFlame**	
		Comm.	Runtime	Comm.	Runtime
CNN	19k	1.5650	0.9163	0.1919	2.4400
LeNet-5 [LeC15]	62k	5.1058	2.7165	0.6250	7.7817
ResNet-18 [HZRS16]	273k	22.480	14.642	2.7505	32.909
LSTM [HS97]	818k	67.358	47.688	8.2404	112.32

6.2 Comparison to PINE

In this section, we provide a theoretical and practical comparison with PINE [ROCT24], which is one of the state-of-the-art works. Both PINE and SeaFlame are based on non-colluding two servers. In PINE, clients share gradients over the aggregation field, providing zero-knowledge proof within the l_2 bound; servers serve as verifiers. In SeaFlame, clients share gradients over a smaller field, providing cryptographic correlations; servers use correlations to lift shares from a smaller field to the larger aggregation field and compute the l_2 value. Moreover, SeaFlame also provides l_∞ defense by letting clients share gradients over a smaller field. Different from the l_2 defense, the l_∞ defense defines a component-wise upper bound. PINE directly shares gradients over the aggregation field, so it makes no sense to discuss the l_∞ defense over the same field as l_2.

For the theoretical analysis of communication, each client in PINE and SeaFlame needs $O(d)$ communication to share gradients, where d is the dimension of the gradient vector. For the remaining communication overhead, each client in PINE only needs $O(\sqrt{d})$ to generate zero-knowledge proof, while SeaFlame's client needs $O(d)$ to generate OTs and $O(d)$ to generate square correlations. For each client, servers in PINE only need $O(1)$ communication to verify the proof, while servers in SeaFlame need $O(1)$ to verify OTs, $O(d)$ to verify square correlations, $O(d)$ to run the A2A protocol, $O(d)$ to compute the l_2 value, and $O(1)$ to securely compare the l_2 value with the bound. Therefore, for each client's uploading, the asymptotical communication complexity of both PINE and SeaFlame is $O(d)$.

For the practical analysis of performance, we compare the concrete instances for 64-bit-aggregation field and 10^4-dimensional gradient vectors of PINE (statistical ZK with 2^{-50} zero-knowledge error and 2^{-50} soundness error) and SeaFlame (32-bit gradients, setting statistical security parameter as 61 and computational security parameter as 128). The detailed values are shown in Tab. 5. Due to PINE's not providing the specific runtime, we also evaluate the runtime by the number of additions and multiplications in the same way. For each client's uploading, PINE runs $306.7d$ additions and $92.5d$ multiplications, while SeaFlame runs $30d$ additions and $33d$ multiplications. In terms of communication, PINE requires $64d$ bits for sharing and 22% additional communication for ZK proof, $14.1d$ bits; SeaFlame needs $32d$ bits for sharing, $128d$ bits for OT generation (128 bits per OT), $192d$ bits for square correlation generation (96 bits per square correlation), $96d$ bits for square correlation verification (one 96-bit element per square correlation), $64d$ bits for A2A (one

64-bit element per bit multiplication), and $64d$ bits for l_2 computation (one 64-bit element per square correlation consumed). In total, compared to PINE, SeaFlame's communication overhead increases by about 7.4 times, and the number of additions and multiplications reduces by about 10.2 and 2.8 times, respectively. SeaFlame improves computational efficiency by generating cryptographic correlations to aid computation, thereby increasing communication overhead, while PINE achieves optimized communication by generating ZK proofs, thereby increasing computational operations.

Table 5: Practical Performance Comparison of SeaFlame with PINE. Values in parenthesis mean the field sizes of operations. Almost all additions and multiplications in PINE are over 64-bit fields, as marked one time; while those in SeaFlame are over fields of different sizes. For simplicity, we roughly estimate the total number by adding them up. Missing values in the table indicate none or negligible.

	PINE				SeaFlame			
	Steps	Comm. (bits)	Runtime		Steps	Comm. (bits)	Runtime	
			#Add	#Mult			#Add	#Mult
Each Client	Sharing	$64d$	d (64-bit)	-	Sharing	$32d$	d (32-bit)	-
					OT Gen	$128d$	d (128-bit)	d (128-bit)
	ZK Gen	$14.1d$	$153.4d$	$69.6d$	SqCorr Gen	$192d$	$2d$ (96-bit)	d (96-bit)
	Total	$78.1d$	$154.4d$	$69.6d$	Total	$352d$	$4d$	$2d$
Two Servers					OT Vrf	-	$3d$ (128-bit)	$3d$ (128-bit)
					SqCorr Vrf	$96d$	$8d$ (96-bit)	$13d$ (96-bit)
	ZK Vrf	-	$152.3d$	$22.9d$	A2A	$64d$	$8d$ (64-bit)	$10d$ (64-bit)
					L2 Comp	$64d$	$7d$ (64-bit)	$5d$ (64-bit)
					L2 Enf	-	-	-
	Total	-	$152.3d$	$22.9d$	Total	$224d$	$26d$	$31d$
Total	-	$78.1d$	$306.7d$	$92.5d$	-	$576d$	$30d$	$33d$

6.3 Overhead Breakdown

Here we split phases of SeaFlame and ELSA to quantify our optimization of these phases. Meanwhile, we demonstrate the impact of different gradient bitlengths on our optimization. We set 8, 16, 32 and 64-bit gradients, and fix the number of clients and gradients to 100 and 10000, respectively. We run this group of experiments on a laptop with 32 GB of RAM and 8 cores in multi-threaded mode, and average measurements of 100 runs.

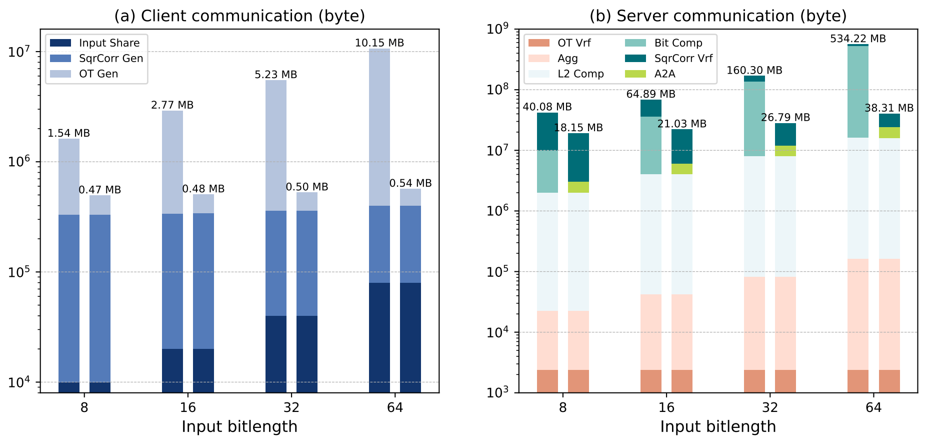

Figure 1: The communication breakdown of phases in SeaFlame (right one) and ELSA (left one) varying with different input bitlength. Values above the bars denote the total communication. (a) For a single client. (b) For a single server. The communication of l_2 enforcement is not shown in the subfigure because it is negligible.

First, we begin with the outgoing communication breakdown of a single client and server, as depicted in Fig. 1. Recall that our presented techniques are devoted to saving communication. More specifically, SeaFlame reduces client communication during the OT generation phase and server communication during both the A2A phase (bit composition for ELSA) and the square correlation verification phase. The communication of other phases in SeaFlame is equal to ELSA. So we put the stacked bar of those phases with equal communication at the bottom, which can be found intuitively from two subfigures. Overall, the client communication of SeaFlame is 3.28 (5.77, 10.46, 18.80) times less than ELSA for 8-bit (16-bit, 32-bit, 64-bit) gradients, and the server communication is 2.21 (3.09, 5.98, 13.94) times less for 8-bit (16-bit, 32-bit, 64-bit) gradients. The larger of the bitlength, the greater of the degree of our communication optimization. This is consistent with our previous theoretical analysis in Section 4.2 that the reduction of OT usage is the factor of $\lceil \log p \rceil$, i.e., the gradient bitlength.

In terms of the computation overhead, we consider the end-to-end runtime instead of the runtime of a single client and server. We show the runtime breakdown in sequence of protocol execution in Fig. 2. We view the client runtime as a whole phase, because a single part of it, like input sharing, OT generation, and square correlation generation, has too small values, and we simulate all clients almost simultaneously on one machine. The server preparation phase includes servers receiving messages from clients and extending values using PRG seeds. We can see that the OT verification in SeaFlame is faster than ELSA due to the reduction in the number of OTs. More specifically, the runtime of OT verification in SeaFlame is about 1.19 (2.68, 4.66, 4.89) times faster than ELSA for 8-bit (16-bit, 32-bit, 64-bit) gradients. We find that other phases of SeaFlame are slower than ELSA, and we think the main reason is that arithmetic operations are slower than bitwise operations. The end-to-end runtime of SeaFlame is about 4.2 (3.15, 2.62, 1.65) times slower than ELSA for 8-bit (16-bit, 32-bit, 64-bit) gradients. With the bitlength increasing, the proportion of additional computation overhead brought by SeaFlame decreases.

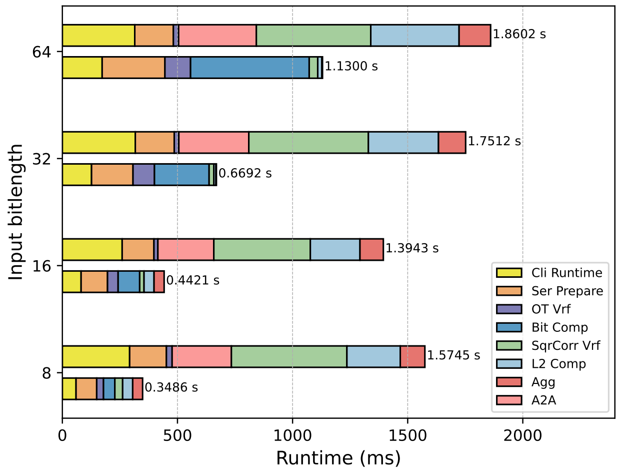

Figure 2: The runtime breakdown of phases in SeaFlame (upper one) and ELSA (nether one) varying with different input bitlength. Values on the right of the bars denote the total end-to-end runtime. The runtimes of phases not shown in the figure are negligible.

6.4 Combination with FL

In this section, we combine our SeaFlame with the training phase of a specific FL image classification task. We aim to train a CNN model from [TXW+24] based on the MNIST dataset. This model has two convolution layers with a 5×5 kernel and two fully connected layers with 128 and 10 output neurons, respectively. Each convolution layer is followed by a ReLU activation function and a 2×2 max pooling layer in sequence. The first fully connected layer is followed by a ReLU activation function and a dropout layer with the probability 0.5. We choose Adam optimization algorithm and FedAvg algorithm [MMR+17]. In each iteration, each client obtains 80,202 gradients after local training; all clients and servers execute our SeaFlame once to compute aggregates; and servers broadcast the averaged aggregates as the fresh global model.

We deploy this group of experiments on a laptop with 32 GB of RAM and 8 cores. We fix the bitlength of gradients, the number of clients and gradients to 32, 100 and 80202, respectively. And we assume that each client holds independent-and-identically-distributed (IID) data. Fig. 3 illustrates the detailed experimental results. The left subfigure shows that the model accuracy varies with iterations under the settings of different hyperparameters. Among these eight settings, when the local epoch, the batch size and the learning rate are 2, 50 and 0.001, the model converges the fastest. Thus, we measure the accumulated overhead of this setting per 20 iterations, as depicted in the right subfigure. According to the workflow of the local training phase with SeaFlame, the newly added overhead includes the clients' local training time and the servers' communication overhead of broadcasting global models. Experiments show the validity and efficiency of SeaFlame when applied to practical FL tasks.

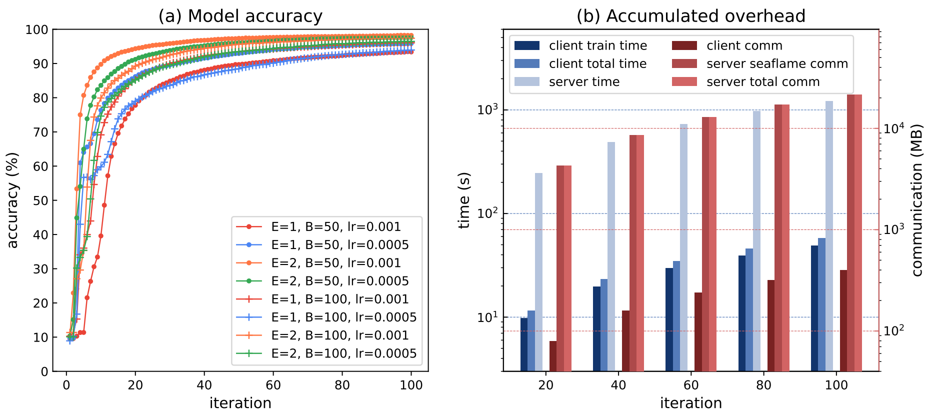

Figure 3: (a) IID, $N = 100$, $d = 80202$, model accuracy varies with iterations under settings of different hyperparameters. E, B and lr denote the local epoch, batch size and learning rate, respectively. (b) IID, $N = 100$, $d = 80202$, $E = 2$, $B = 50$, $lr = 0.001$, accumulated computation and communication overhead per 20 iterations of each client and server. The bars in blue and red refer to time and communication, respectively, corresponding with the left and right y-axis.

7 Conclusion and Future Work

SeaFlame uses arithmetic sharing with A2A conversion and random linear combination to achieve better communication efficiency than the state-of-the-art work. Besides, SeaFlame only achieves malicious privacy and requires gradients larger than $\frac{p}{2}$. For our future research, we will try to achieve full malicious security, which indicates both privacy and correctness [Lin17], [TXW+24], in a practical way and remove the constraint of gradients.

References

[AGJ+22] Surya Addanki, Kevin Garbe, Eli Jaffe, Rafail Ostrovsky, and Antigoni Polychroniadou. Prio+: Privacy preserving aggregate statistics via boolean shares. In *SCN*, volume 13409 of *Lecture Notes in Computer Science*, pages 516–539. Springer, 2022.

[ALSZ13] Gilad Asharov, Yehuda Lindell, Thomas Schneider, and Michael Zohner. More efficient oblivious transfer and extensions for faster secure computation. In *CCS*, pages 535–548. ACM, 2013.

[BBB+18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Gregory Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *IEEE Symposium on Security and Privacy*, pages 315–334. IEEE Computer Society, 2018.

[BBG+20] James Henry Bell, Kallista A. Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. Secure single-server aggregation with (poly)logarithmic overhead. In *CCS*, pages 1253–1269. ACM, 2020.

[BGL+23] James Bell, Adrià Gascón, Tancrède Lepoint, Baiyu Li, Sarah Meiklejohn, Mariana Raykova, and Cathie Yun. ACORN: input validation for secure aggregation. In *USENIX Security Symposium*, pages 4805–4822. USENIX Association, 2023.

[BIK+17] Kallista A. Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for privacy-preserving machine learning. In *CCS*, pages 1175–1191. ACM, 2017.

[BMGS17] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In *NIPS*, pages 119–129, 2017.

[CB17] Henry Corrigan-Gibbs and Dan Boneh. Prio: Private, robust, and scalable computation of aggregate statistics. In *NSDI*, pages 259–282. USENIX Association, 2017.

[CFLG21] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Zhenqiang Gong. Fltrust: Byzantine-robust federated learning via trust bootstrapping. In *NDSS*. The Internet Society, 2021.

[CGJvdM22] Amrita Roy Chowdhury, Chuan Guo, Somesh Jha, and Laurens van der Maaten. Eiffel: Ensuring integrity for federated learning. In *CCS*, pages 2535–2549. ACM, 2022.

[DPSZ12] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In *CRYPTO*, volume 7417 of *Lecture Notes in Computer Science*, pages 643–662. Springer, 2012.

[DSZ15] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS*. The Internet Society, 2015.

[DWA+21] F. Betül Durak, Chenkai Weng, Erik Anderson, Kim Laine, and Melissa Chase. Precio: Private aggregate measurement via oblivious shuffling. Cryptology ePrint Archive, Paper 2021/1490, 2021.

[emp] Emp toolkit.

[Fel87] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In *FOCS*, pages 427–437. IEEE Computer Society, 1987.

[GKW$^+$20] Chun Guo, Jonathan Katz, Xiao Wang, Chenkai Weng, and Yu Yu. Better concrete security for half-gates garbling (in the multi-instance setting). In *CRYPTO (2)*, volume 12171 of *Lecture Notes in Computer Science*, pages 793–822. Springer, 2020.

[GLL$^+$21] Xiaojie Guo, Zheli Liu, Jin Li, Jiqiang Gao, Boyu Hou, Changyu Dong, and Thar Baker. Verifl: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Trans. Inf. Forensics Secur.*, 16:1736–1751, 2021.

[HS97] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, 1997.

[HZRS16] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *CVPR*, pages 770–778. IEEE Computer Society, 2016.

[KIM$^+$18] Ryo Kikuchi, Dai Ikarashi, Takahiro Matsuda, Koki Hamada, and Koji Chida. Efficient bit-decomposition and modulus-conversion protocols with an honest majority. In *ACISP*, volume 10946 of *Lecture Notes in Computer Science*, pages 64–82. Springer, 2018.

[KMA$^+$21] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G. L. D'Oliveira, Hubert Eichner, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaïd Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Hang Qi, Daniel Ramage, Ramesh Raskar, Mariana Raykova, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. Advances and open problems in federated learning. *Found. Trends Mach. Learn.*, 14(1-2):1–210, 2021.

[KMY$^+$16] Jakub Konečný, H. Brendan McMahan, Felix X. Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. Federated learning: Strategies for improving communication efficiency. *CoRR*, abs/1610.05492, 2016.

[KOS15] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Actively secure OT extension with optimal overhead. In *CRYPTO (1)*, volume 9215 of *Lecture Notes in Computer Science*, pages 724–741. Springer, 2015.

[KPR18] Marcel Keller, Valerio Pastro, and Dragos Rotaru. Overdrive: Making SPDZ great again. In *EUROCRYPT (3)*, volume 10822 of *Lecture Notes in Computer Science*, pages 158–189. Springer, 2018.

[LBV+23] Hidde Lycklama, Lukas Burkhalter, Alexander Viand, Nicolas Küchler, and Anwar Hithnawi. Rofl: Robustness of secure federated learning. In *SP*, pages 453–476. IEEE, 2023.

[LeC15] Yann LeCun. *LeNet-5, convolutional neural networks*, 2015.

[Lin17] Yehuda Lindell. How to simulate it - A tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer International Publishing, 2017.

[MMM+22] Zhuoran Ma, Jianfeng Ma, Yinbin Miao, Yingjiu Li, and Robert H. Deng. Shieldfl: Mitigating model poisoning attacks in privacy-preserving federated learning. *IEEE Trans. Inf. Forensics Secur.*, 17:1639–1654, 2022.

[MMR+17] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Agüera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *AISTATS*, volume 54, pages 1273–1282. PMLR, 2017.

[MWA+23] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *SP*, pages 477–496. IEEE, 2023.

[Ped91] Torben P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *CRYPTO*, volume 576 of *Lecture Notes in Computer Science*, pages 129–140. Springer, 1991.

[PKH22] Krishna Pillutla, Sham M. Kakade, and Zaïd Harchaoui. Robust aggregation for federated learning. *IEEE Trans. Signal Process.*, 70:1142–1154, 2022.

[ROCT24] Guy N. Rothblum, Eran Omri, Junye Chen, and Kunal Talwar. PINE: Efficient verification of a euclidean norm bound of a Secret-Shared vector. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 6975–6992. USENIX Association, 2024.

[RSWP23] Mayank Rathee, Conghao Shen, Sameer Wagh, and Raluca Ada Popa. ELSA: secure aggregation for federated learning with malicious actors. In *SP*, pages 1961–1979. IEEE, 2023.

[rug] Rug.

[SHKR22] Virat Shejwalkar, Amir Houmansadr, Peter Kairouz, and Daniel Ramage. Back to the drawing board: A critical evaluation of poisoning attacks on production federated learning. In *SP*, pages 1354–1371. IEEE, 2022.

[SKSM19] Ziteng Sun, Peter Kairouz, Ananda Theertha Suresh, and H. Brendan McMahan. Can you really backdoor federated learning? *CoRR*, abs/1911.07963, 2019.

[SWMS20] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. Robust and communication-efficient federated learning from non-i.i.d. data. *IEEE Trans. Neural Networks Learn. Syst.*, 31(9):3400–3413, 2020.

[TBA+19] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. A hybrid approach to privacy-preserving federated learning - (extended abstract). *Inform. Spektrum*, 42(5):356–357, 2019.

[tch] tch-rs.

[tok] Tokio.

[TXW+24] Jinling Tang, Haixia Xu, Mingsheng Wang, Tao Tang, Chunying Peng, and Huimei Liao. A flexible and scalable malicious secure aggregation protocol for federated learning. *IEEE Trans. Inf. Forensics Secur.*, 19:4174–4187, 2024.

[XLL+20] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. Verifynet: Secure and verifiable federated learning. *IEEE Trans. Inf. Forensics Secur.*, 15:911–926, 2020.

[YMV+21] Hongxu Yin, Arun Mallya, Arash Vahdat, José M. Álvarez, Jan Kautz, and Pavlo Molchanov. See through gradients: Image batch recovery via gradinversion. In *CVPR*, pages 16337–16346. IEEE, 2021.

[ZLH19] Ligeng Zhu, Zhijian Liu, and Song Han. Deep leakage from gradients. In *NeurIPS*, pages 14747–14756, 2019.

A Correctness in the Semi-honest Model

We consider that all parties are semi-honest and state the correctness of SeaFlame's key components.

Correctness of $\Pi^{\widetilde{p}}_{\mathsf{AliBitMult}}$. (Protocol 2) Recall that S_0 and S_1 are the OT sender and the OT receiver respectively, so we just consider the two cases of the input of S_1, i.e., $x^{(1)}$. If $x^{(1)} = 0$, then

$$\begin{aligned} y^{(0)} + y^{(1)} &= -H(c\|w) + H(c\|t) \pmod{\widetilde{p}} \\ &= -H(c\|w) + H(c\|w + x^{(1)}\Delta) \pmod{\widetilde{p}} \\ &= -H(c\|w) + H(c\|w) \pmod{\widetilde{p}} \\ &= 0 = x^{(0)} \wedge x^{(1)}. \end{aligned}$$

If $x^{(1)} = 1$, then

$$\begin{aligned} y^{(0)} + y^{(1)} &= -v_0 + u - v \pmod{\widetilde{p}} \\ &= -v_0 + v_0 + v_1 + x^{(0)} - v \pmod{\widetilde{p}} \\ &= H(c\|w + \Delta) + x^{(0)} - H(c\|t) \pmod{\widetilde{p}} \\ &= H(c\|w + \Delta) + x^{(0)} - H(c\|w + \Delta) \pmod{\widetilde{p}} \\ &= x^{(0)} = x^{(0)} \wedge x^{(1)}. \end{aligned}$$

Correctness of $\Pi^{p,\widetilde{p}}_{\mathsf{A2A}}$. (Protocol 1) Now we explain the correctness of A2A conversion. Since all clients are semi-honest, their inputs are larger than $\frac{p}{2}$ and less than p. For $\Pi^{p,\widetilde{p}}_{\mathsf{A2A}}$, $\frac{p}{2} < x < p$, so $p < 2x < 2p$ and $2x \pmod{p} = 2x - p$ is an odd element in $\mathbb{Z}_p$. $\forall b \in \{0, 1\}$, $\overline{x}^{(b)} = 2 \cdot x^{(b)} \pmod{p}$, then we have

$$\overline{x}^{(0)} + \overline{x}^{(1)} - \alpha p = 2x - p,$$

where $\alpha \in \{0, 1\}$. Since $2x - p$ is odd, we have

$$\mathsf{LSB}(\overline{x}^{(0)}) \oplus \mathsf{LSB}(\overline{x}^{(1)}) = 1 - \alpha.$$

Therefore, the output of $\Pi^{p,\widetilde{p}}_{\mathsf{A2A}}$ is

$$\begin{aligned} y^{(b)} &= 2^{-1} \cdot \overline{y}^{(b)} \pmod{\widetilde{p}} \\ &= 2^{-1} \cdot (\overline{x}^{(b)} + z^{(b)}p) \pmod{\widetilde{p}}. \end{aligned}$$

Note that

$$\begin{aligned} z^{(0)} + z^{(1)} \pmod{\widetilde{p}} &= \mathsf{LSB}(\overline{x}^{(0)}) + \mathsf{LSB}(\overline{x}^{(1)}) - 2 \cdot (m^{(0)} + m^{(1)}) \\ &= \mathsf{LSB}(\overline{x}^{(0)}) + \mathsf{LSB}(\overline{x}^{(1)}) - 2 \cdot \mathsf{LSB}(\overline{x}^{(0)}) \wedge \mathsf{LSB}(\overline{x}^{(1)}) \\ &= \mathsf{LSB}(\overline{x}^{(0)}) \oplus \mathsf{LSB}(\overline{x}^{(1)}) \\ &= 1 - \alpha, \end{aligned}$$

where $m^{(b)}$ is the output of $\Pi^{\widetilde{p}}_{\mathsf{AliBitMult}}$ on input $\mathsf{LSB}(\overline{x}^{(b)})$. Finally, we have

$$\begin{aligned} y^{(0)} + y^{(1)} \pmod{\widetilde{p}} &= 2^{-1} \cdot (\overline{x}^{(0)} + \overline{x}^{(1)} + z^{(0)}p + z^{(1)}p) \\ &= 2^{-1} \cdot (\overline{x}^{(0)} + \overline{x}^{(1)}) + (z^{(0)} + z^{(1)})p \\ &= 2^{-1} \cdot [2x + (\alpha - 1)p + (1 - \alpha)p] \\ &= x = x^{(0)} + x^{(1)} \pmod{p}. \end{aligned}$$

Correctness of l_2 Computation. Since all clients are semi-honest, all square correlations will be correct. Given a well-formed square correlation (a, d) with shares $(a^{(b)}, d^{(b)})$, where $b \in \{0, 1\}$, the corresponding share of z^2 (denoted by $\widetilde{z}^{(b)}$) can be computed by:

$$\begin{aligned} \widetilde{z}^{(b)} &= d^{(b)} + 2 \cdot e \cdot z^{(b)} - b \cdot e^2 \\ &= d^{(b)} + 2(z - a)z^{(b)} - b(z - a)^2 \\ &= d^{(b)} + 2zz^{(b)} - 2az^{(b)} - bz^2 + 2abz - ba^2, \end{aligned}$$

where $e = z - a$. Since $d \equiv a^2 \bmod q$ and $q = \eta\widetilde{p}$, then $d \equiv a^2 \bmod \widetilde{p}$ holds. We further have

$$\begin{aligned} \widetilde{z}^{(0)} + \widetilde{z}^{(1)} &= d + 2z^2 - 2az - z^2 + 2az - a^2 \\ &= z^2 \pmod{\widetilde{p}}. \end{aligned}$$

To sum up, all clients are semi-honest, so all OTs are correct. Then the correctness of $\Pi^{\widetilde{p}}_{\mathsf{AliBitMult}}$ and further $\Pi^{p,\widetilde{p}}_{\mathsf{A2A}}$, i.e., A2A conversion, are guaranteed. The correctness of aggregation phase is obvious, thus SeaFlame's correctness is guaranteed.