On the Completeness of Verifiable Secure Aggregation in the Presence of Dropouts

Fatemeh Zarinjouei

December 15, 2023

Abstract

Secure and verifiable aggregation is essential for ensuring the integrity and privacy of Federated Learning (FL). Eltaras et al. (IEEE TIFS 2023) recently proposed an efficient protocol relying on auxiliary nodes to achieve these goals while ostensibly managing client dropouts. In this work, we identify a critical cryptographic flaw in the protocol's verifiability mechanism. We demonstrate that the protocol fails to satisfy completeness in the presence of dropouts, and honest clients are forced to reject correctly aggregated results with overwhelming probability whenever a dropout occurs. To address this, we propose a lightweight modification that realigns the verification key with the set of active participants. This fix restores the completeness of the verifiability guarantee without compromising the protocol's privacy properties.

1 Introduction

Federated learning (FL) enables training models over decentralized data while keeping raw data local [7, 8]. In each round, a server collects client updates and aggregates them into a new global model. This setting raises two central concerns: privacy of individual updates and integrity of the aggregate returned by a potentially untrusted server [2, 9]. Secure aggregation protocols address the first concern by ensuring that the server learns only an aggregate (typically a sum) of client updates [1,6,10], while recent work has sought *verifiable* aggregation, where each client can check that the aggregate is consistent with the set of contributed updates [4,5,11].

Eltaras et al. propose an "Efficient Verifiable Protocol for Privacy-Preserving Aggregation in Federated Learning" (EVP) [3] that aims to provide both properties in cross-silo FL using semi-trusted auxiliary organizations. Their Protocol 1 uses a single-masking mechanism based on pairwise key agreement with auxiliary nodes to achieve secure aggregation, and a lightweight linear "double aggregation" mechanism for verifiability: each client n computes a MAC of the form $\mathsf{MAC}_n = K_n + \alpha x_n$ on its gradient x_n, where K_n is a client-specific key and α is a global scalar derived from the auxiliary nodes. The server aggregates gradients and MACs, and each client later checks whether the relation $\mathsf{MAC} = K + \alpha X$ holds, where X is the aggregate returned by the server and K is a global key derived from auxiliary-node contributions. The protocol is claimed to preserve privacy, tolerate client dropouts, and allow each client to independently verify the correctness of the aggregate.

We revisit EVP from a cryptographic perspective and show that, as specified, its verifiability guarantee is *incomplete* in the presence of dropouts. In the original protocol, the global key K used in the verification equation is fixed during an early key-sharing round as a sum of per-client keys over the user set U_1 that participates in that round. In contrast, the aggregate X and the aggregated MAC are computed later over the (strict) subset $U_2 \subseteq U_1$ of users that actually upload masked gradients in the round. We formally analyze this mismatch and prove that, whenever at least one client drops out between these two rounds, honest clients will reject the correct aggregate with overwhelming probability, even if the server behaves honestly. Thus EVP fails the standard completeness requirement for verifiable computation under the very dropout model it aims to support.

We then ask whether EVP can be repaired without sacrificing its efficiency or privacy guarantees. Our second contribution is a simple modification of Protocol 1 that preserves its architecture and threat model while restoring completeness. The key idea is to decouple the roles of the two key-agreement phases: auxiliary nodes still distribute shares of the global scalar α early, but the global key K is no longer bound to U_1. Instead, after the server determines the actual contributor set U_2, each auxiliary node recomputes a node-level key K_m as a sum of pairwise keys over U_2 and securely sends it to users, who reconstruct $K = \sum_m K_m$. This re-aligns the verification key with the set of users whose gradients are aggregated, leaving the masking layer unchanged and incurring only a small number of additional symmetric encryptions and decryptions. We show that the result-

ing protocol satisfies completeness of verifiability even under arbitrary dropouts between the key-sharing and aggregation rounds.

2 Attack on the Verifiable Secure Aggregation Protocol

In this section we first recall the verifiable secure aggregation protocol of Eltaras et al. [3] (see Protocol 1 in the Appendix), focusing on the parts that are relevant for its verifiability guarantee. We then show that, in the presence of user dropouts, the verification equation used in the protocol can systematically fail even when the server behaves honestly and computes the correct aggregate. This reveals a violation of the completeness aspect of verifiability: there exist honest executions in which honest users always reject a correctly computed result.

2.1 Overview of the Original Protocol

The protocol is executed between a cloud server, a set U of users, and a set M of auxiliary nodes. Each user $n \in U$ holds a local gradient x_n in a ring $\mathbb{Z}_R$. The auxiliary nodes represent organizations (e.g., hospitals or banks) and help to generate randomness and verification keys, but do not contribute gradients. The protocol is instantiated using a key-agreement primitive KA with algorithms $\mathsf{KA.gen}$ and $\mathsf{KA.agree}$, a symmetric authenticated encryption scheme AE with algorithms $\mathsf{AE.enc}$ and $\mathsf{AE.dec}$, and a pseudorandom generator PRG used to derive vector masks from shared seeds. All additions are performed modulo R.

The goal of the protocol is twofold. First, the server must be able to compute the aggregate $X = \sum_{n\in U_2} x_n$ of the gradients of the users that participate in the current training round (denoted U_2). Second, each online user should be able to verify that the aggregate returned by the server is indeed the sum of the gradients of the users who took part in the round. To this end, the protocol uses a single-masking mechanism for privacy and a linear "double aggregation" mechanism for verifiability.

Setup and key advertising (Round 0). All parties agree on a security parameter λ and public parameters $pp \leftarrow \mathsf{KA.gen}(\lambda)$. Each user and each auxiliary node holds a private authenticated channel with the server. In Round 0, every user n generates three key pairs (pk_n^1, sk_n^1), (pk_n^2, sk_n^2), $(pk_n^3, sk_n^3) \leftarrow \mathsf{KA.gen}(pp)$ and sends its public keys (pk_n^1, pk_n^2, pk_n^3) to the server. Similarly, each auxiliary node m generates three key pairs (pk_m^1, sk_m^1), (pk_m^2, sk_m^2), (pk_m^3, sk_m^3) and sends (pk_m^1, pk_m^2, pk_m^3) to the server. Let $U_1 \subseteq U$ denote the set of users whose keys are received in this round. The server collects all public keys and broadcasts the list of auxiliary-node keys $\{(pk_m^1, pk_m^2, pk_m^3)\}_{m\in M}$ to each user in U_1, and the list of user keys $\{(pk_n^1, pk_n^2, pk_n^3)\}_{n\in U_1}$ to each auxiliary node in M.

Key sharing for verifiability (Round 1). Round 1 sets up the global verification keys and the per-user keys that will be used to compute message authentication codes (MACs) on the gradients. Each auxiliary node $m \in M$ receives $\{(pk_n^1, pk_n^2, pk_n^3)\}_{n\in U_1}$ from the server and, for every user $n \in U_1$, computes a shared verification subkey $K_{n,m} \leftarrow \mathsf{KA.agree}(sk_m^3, pk_n^3)$. It then sums these subkeys into a node-level verification key $K_m \leftarrow \sum_{n\in U_1} K_{n,m}$. Next, m samples a random scalar α_m in a field $\mathbb{F}$ (the "universal" key contribution of node m). For each user $n \in U_1$, the auxiliary node derives an encryption key $k_{n,m}^{(1)} \leftarrow \mathsf{KA.agree}(sk_m^1, pk_n^1)$ and sends to the server the ciphertext $ct_{n,m} \leftarrow \mathsf{AE.enc}\big(k_{n,m}^{(1)}, \alpha_m \,\|\, K_m\big)$, where $\|$ denotes concatenation. The server collects all ciphertexts $\{ct_{n,m}\}_{n\in U_1}$ and forwards, for each user $n \in U_1$, the set $\{ct_{n,m}\}_{m\in M}$ to that user. Each user $n \in U_1$ receives the auxiliary nodes' public keys and ciphertexts, and then, for each $m \in M$, computes $k_{n,m}^{(1)} \leftarrow \mathsf{KA.agree}(sk_n^1, pk_m^1)$ and decrypts $(\alpha_m, K_m) \leftarrow \mathsf{AE.dec}\big(k_{n,m}^{(1)}, ct_{n,m}\big)$. By summing these values over all auxiliary nodes, every user $n \in U_1$ derives the global scalar

$$\alpha \leftarrow \sum_{m\in M} \alpha_m \tag{1}$$

and the global verification key

$$K \leftarrow \sum_{m\in M} K_m\,. \tag{2}$$

In addition, user n computes its own per-user verification key as

$$K_n \leftarrow \sum_{m\in M} K_{n,m} = \sum_{m\in M} \mathsf{KA.agree}(sk_n^3, pk_m^3)\,. \tag{3}$$

For later use, it is convenient to rewrite (2) in terms of the subkeys $K_{n,m}$. Since $K_m = \sum_{n\in U_1} K_{n,m}$, we obtain

$$K = \sum_{m\in M} K_m = \sum_{m\in M}\sum_{n\in U_1} K_{n,m} = \sum_{n\in U_1}\sum_{m\in M} K_{n,m} = \sum_{n\in U_1} K_n. \tag{4}$$

Thus, in the original protocol, the global key K is defined as the sum of all per-user keys K_n over the user set U_1.

Masking and MAC computation (Round 2) In Round 2, only a subset $U_2 \subseteq U_1$ of users is online and sends masked gradients. For each auxiliary node $m \in M$, each user $n \in U_2$ computes a shared seed $s_{n,m} \leftarrow \mathsf{KA.agree}(sk_n^2, pk_m^2)$ and expands it via the pseudorandom generator to obtain a mask vector $P_{n,m} \leftarrow \mathsf{PRG}(s_{n,m})$, with the same dimension as the gradient x_n. The user then masks its local gradient as

$$\hat{x}_n \leftarrow x_n + \sum_{m \in M} P_{n,m} \pmod{R}. \tag{5}$$

To enable verifiability, user $n \in U_2$ uses its per-user key K_n and the global scalar α to compute a linear MAC on its unmasked gradient:

$$\mathsf{MAC}_n \leftarrow K_n + \alpha \cdot x_n \pmod{R}. \tag{6}$$

The user sends the pair $(\hat{x}_n, \mathsf{MAC}_n)$ to the server. After some timeout, the server has collected contributions from the online users $U_2 \subseteq U_1$ and broadcasts the list U_2 (user identifiers) to the auxiliary nodes.

Unmasking and aggregation (Round 3). In Round 3, each auxiliary node m receives the set U_2 of online users and reconstructs the masks needed to unmask their contributions. For each $n \in U_2$, it computes $s_{n,m} \leftarrow \mathsf{KA.agree}(sk_m^2, pk_n^2)$, $P_{n,m} \leftarrow \mathsf{PRG}(s_{n,m})$, and sums them to obtain

$$P_m \leftarrow \sum_{n \in U_2} P_{n,m}. \tag{7}$$

The auxiliary node sends P_m to the server. The server receives all P_m values and computes the aggregate

$$X \leftarrow \sum_{n \in U_2} \hat{x}_n - \sum_{m \in M} P_m \pmod{R}. \tag{8}$$

By construction, the masks cancel pairwise and we indeed have $X = \sum_{n \in U_2} x_n$. At the same time, the server aggregates the users' MACs as

$$\mathsf{MAC} \leftarrow \sum_{n \in U_2} \mathsf{MAC}_n. \tag{9}$$

The server broadcasts the pair (X, MAC) to all users that are online in this round, denoted $U_3 \subseteq U_2$.

Verification (Round 4). In the final round, each online user $n \in U_3$ receives the aggregate X and the aggregated MAC MAC. Using the global values α and K computed in Round 1, the user recomputes

$$\mathsf{MAC}' \leftarrow K + \alpha \cdot X \pmod{R} \tag{10}$$

and accepts the server's output if and only if

$$\mathsf{MAC}' = \mathsf{MAC}. \tag{11}$$

Intuitively, if the server modifies X while keeping MAC unchanged, the linear relation (11) should be violated with overwhelming probability, because the MACs are computed using a secret linear functional determined by α and the K_n. Eltaras et al. claim that this enables each user to verify the correctness of the aggregated result while the protocol "handles dropouts by default".

2.2 Attack on Verifiability in the Presence of Dropouts

We now show that the above protocol does not always satisfy the stated verifiability requirement. Specifically, we consider executions in which some users drop out between Round 1 and Round 2, and we show that, even if the server behaves honestly, the remaining users will reject the correct aggregate because the verification equation (11) fails. This reveals a flaw in the way the global key K is defined and used when dropouts occur.

Verifiability requirement. The security analysis in [3] informally states that each participant should be able to verify the correctness of the aggregated result returned by the server, and that an honest server should always be accepted by honest users. Formally, if the server follows the protocol and computes $X = \sum_{n \in U_2} x_n$ and $\mathsf{MAC} = \sum_{n \in U_2} \mathsf{MAC}_n$ as specified, then every honest user that completes Round 4 should accept. This is the standard completeness condition for verifiable computation.

Dropout scenario. Consider an execution in which Round 0 and Round 1 complete with user set U_1, and the global values α and K are computed as in (1) and (4). In Round 2, suppose that only a strict subset $U_2 \subset U_1$ of users are online and send masked gradients and MACs. This corresponds precisely to the dropout pattern that the protocol claims to handle: users who do not send gradients in the current round are excluded from the aggregation.

Assume that all users in U_2 and all auxiliary nodes follow the protocol honestly, and that the server is honest as well: it computes X and MAC exactly as in (8) and (9), without tampering. We show that in this honest execution, the verifying users in $U_3 \subseteq U_2$ do *not* accept, except with negligible probability.

The value of K used in verification. Recall from (4) that, in the original protocol, K is defined in terms of U_1, the users active in Round 1:

$$K = \sum_{n \in U_1} K_n = \sum_{n \in U_1} \sum_{m \in M} K_{n,m}. \tag{12}$$

This equality continues to hold in our dropout scenario, because K is computed *once* in Round 1 and never updated. In particular, the global key K used in the verification equation (10) includes contributions from all users in U_1, including those who later drop out and are not part of U_2.

MACs and aggregate when the server is honest. Let us expand the expression for the aggregated MAC MAC when all parties behave honestly. For each user $n \in U_2$, the MAC is $\mathsf{MAC}_n = K_n + \alpha x_n = \sum_{m \in M} K_{n,m} + \alpha x_n$. Summing over U_2, we obtain

$$\begin{aligned}\mathsf{MAC} &= \sum_{n \in U_2} \mathsf{MAC}_n = \sum_{n \in U_2} (K_n + \alpha x_n) \\ &= \sum_{n \in U_2} \sum_{m \in M} K_{n,m} + \alpha \sum_{n \in U_2} x_n \pmod R. \end{aligned} \tag{13}$$

By the correctness of the masking/unmasking procedure, the aggregate X computed in (8) is

$$X = \sum_{n \in U_2} x_n. \tag{14}$$

Substituting (14) into (13), we can rewrite the aggregated MAC as

$$\mathsf{MAC} = \sum_{n \in U_2} \sum_{m \in M} K_{n,m} + \alpha X \pmod R. \tag{15}$$

User-side verification computation. On the user side, the verification procedure uses the global values α and K fixed in Round 1, together with the aggregate X received from the server, to compute

$$\mathsf{MAC}' = K + \alpha X \pmod R. \tag{16}$$

Replacing K by its definition (12) and X by (14), we obtain

$$\begin{aligned}\mathsf{MAC}' &= \left(\sum_{n \in U_1} \sum_{m \in M} K_{n,m} \right) + \alpha \sum_{n \in U_2} x_n \pmod R \\ &= \sum_{n \in U_1} \sum_{m \in M} K_{n,m} + \alpha X \pmod R. \end{aligned} \tag{17}$$

Comparing MAC' and MAC. We can now compare MAC' from (17) with MAC from (15) Taking the difference, we obtain

$$\mathsf{MAC}' - \mathsf{MAC} \tag{18}$$

$$= \left(\sum_{n \in U_1} \sum_{m \in M} K_{n,m} + \alpha X \right) \tag{19}$$

$$\begin{aligned} &- \left(\sum_{n \in U_2} \sum_{m \in M} K_{n,m} + \alpha X \right) \pmod R \\ &= \sum_{n \in U_1} \sum_{m \in M} K_{n,m} - \sum_{n \in U_2} \sum_{m \in M} K_{n,m} \pmod R \\ &= \sum_{n \in U_1 \setminus U_2} \sum_{m \in M} K_{n,m} \pmod R. \end{aligned} \tag{20}$$

Therefore, unless

$$\sum_{n \in U_1 \setminus U_2} \sum_{m \in M} K_{n,m} \equiv 0 \pmod R, \tag{21}$$

we have $\mathsf{MAC}' \neq \mathsf{MAC}$, and the verification equation (11) fails. The sum in (20) ranges over all users who participated in Round 1 (and contributed to K) but did not participate in Round 2 (and therefore did not contribute to MAC or X). Since the subkeys $K_{n,m}$ are derived via key agreement and modeled as pseudorandom values in $\mathbb{Z}_R$, the probability that the non-trivial sum (21) accidentally equals zero is negligible in the security parameter. In a typical execution with at least one dropout between Round 1 and Round 2, we therefore have $\mathsf{MAC}' \neq \mathsf{MAC}$ with overwhelming probability.

We have exhibited a class of honest executions—namely, any execution in which at least one user drops out between Round 1 and Round 2, while all remaining users, auxiliary nodes, and the server behave honestly—in which the verification condition (11) fails and all honest users reject the correct aggregate. In other words, the protocol as specified in [3] does not satisfy completeness for verifiability under the very dropout model it claims to handle: whenever $U_2 \subset U_1$, the global key K used in the verification step is inconsistent with the set of users whose gradients are actually aggregated, and the equality $\mathsf{MAC}' = \mathsf{MAC}$ no longer characterizes correct behavior of the server.

This attack does not rely on any adversarial action by the users or the server; it arises purely from the interaction between the key-sharing phase (which defines K over U_1) and the aggregation phase (which operates over U_2). As a result, the protocol fails to achieve the stated property that "each user can validate the result by verifying" the equation $\mathsf{MAC} - K - \alpha X = 0$ in the presence of dropouts.

3 Fixing the Verifiable Secure Aggregation Protocol

In this section we present a concrete modification of Protocol I in Eltaras et al. [3] (detailed in Protocol 2) that restores completeness of the verifiability mechanism in the presence of user dropouts, while preserving the original privacy and efficiency goals. The key idea is simple: instead of binding the global verification key K to the user set U_1 that participates in the key-sharing round, we recompute and distribute K *after* the server has determined the actual set of users U_2 whose gradients are aggregated. The global scalar α remains a universal key shared in the first round, but the dependency of K on users who may later drop out is removed.

We deliberately keep the structure and primitives of the original protocol—a single-masking scheme for privacy, auxiliary nodes as organizations, and a lightweight linear "double aggregation" for verifiability—and change only the rounds that interact with the verification keys. We use the same notation as in the original paper: KA denotes a key-agreement primitive with algorithms $\mathsf{KA.gen}$ and $\mathsf{KA.agree}$, AE denotes a symmetric authenticated-encryption scheme with algorithms $\mathsf{AE.enc}$ and $\mathsf{AE.dec}$, and PRG is a pseudorandom generator used to expand seeds into vector masks. All additions are modulo a ring $\mathbb{Z}_R$. We also follow the same convention for user sets: U_0 for users that complete Round 0, $U_1 \subseteq U_0$ for users that complete Round 1, $U_2 \subseteq U_1$ for users that upload masked gradients, $U_3 \subseteq U_2$ for users that receive the aggregate, and $U_4 \subseteq U_3$ for users that actually perform the verification.

Setup and Round 0 (keys advertising). The setup and key-advertising phase remain unchanged. All parties agree on a security parameter λ and public parameters $pp \leftarrow \mathsf{KA.gen}(\lambda)$. Each user and each auxiliary node has a private authenticated channel with the server.

In Round 0, every user n generates three key pairs (pk_n^1, sk_n^1), (pk_n^2, sk_n^2), $(pk_n^3, sk_n^3) \leftarrow \mathsf{KA.gen}(pp)$ and sends (pk_n^1, pk_n^2, pk_n^3) to the server. Each auxiliary node m similarly generates (pk_m^1, sk_m^1), (pk_m^2, sk_m^2), and (pk_m^3, sk_m^3) and sends (pk_m^1, pk_m^2, pk_m^3) to the server. Let U_0 be the set of users for which the server receives public keys in this round, and M the set of auxiliary nodes. The server then broadcasts to each user in U_0 the list of auxiliary-node public keys $\{(pk_m^1, pk_m^2, pk_m^3)\}_{m\in M}$, and to each auxiliary node in M the list of user public keys $\{(pk_n^1, pk_n^2, pk_n^3)\}_{n\in U_0}$.

Round 1: establishing the universal scalar α. The first modification is to restrict Round 1 to the construction of the universal scalar α, without committing to a global verification key K. This round uses key pair (pk^1, sk^1) for encrypted point-to-point communication between users and auxiliary nodes, but postpones any operation involving (pk^3, sk^3) to later rounds.

Each auxiliary node m receives $\{(pk_n^1, pk_n^2, pk_n^3)\}_{n\in U_0}$ from the server and samples a fresh random element α_m from a field $\mathbb{F}$. For every user $n \in U_0$, it derives a symmetric key $k_{n,m}^{(1)} \leftarrow \mathsf{KA.agree}(sk_m^1, pk_n^1)$ and computes a ciphertext $a_{n,m} \leftarrow \mathsf{AE.enc}\big(k_{n,m}^{(1)}, \alpha_m\big)$. The auxiliary node sends the collection $\{a_{n,m}\}_{n\in U_0}$ to the server.

The server defines $U_1 \subseteq U_0$ as the set of users that are online in this round and, for each $n \in U_1$, collects $\{a_{n,m}\}_{m\in M}$ and forwards them to user n.

Each user $n \in U_1$ receives the auxiliary nodes' public keys and ciphertexts, and for each $m \in M$ derives the same symmetric key $k_{n,m}^{(1)} \leftarrow \mathsf{KA.agree}(sk_n^1, pk_m^1)$ and decrypts $\alpha_m \leftarrow \mathsf{AE.dec}(k_{n,m}^{(1)}, a_{n,m})$. By summing these local values, every user $n \in U_1$ obtains the same global scalar

$$\alpha \leftarrow \sum_{m\in M} \alpha_m \,. \tag{22}$$

Note that, compared to the original protocol, users *do not* learn any K_m values in Round 1, and no global key K is derived yet. The only global verification parameter fixed at this stage is α, which is independent of user dropouts in later rounds.

Round 2: masking and local MAC computation. Round 2 retains the same masking mechanism as in the original protocol and moves the computation of the per-user verification keys K_n to this stage, restricted to users that actually upload gradients.

Let $U_2 \subseteq U_1$ denote the subset of users that are online in this round and wish to participate in the current aggregation. Each such user $n \in U_2$ uses the key pair (pk^2, sk^2) to derive mask seeds with every auxiliary node: $s_{n,m} \leftarrow \mathsf{KA.agree}(sk_n^2, pk_m^2)$ for all $m \in M$. These seeds are expanded via the pseudorandom generator to obtain mask vectors $P_{n,m} \leftarrow \mathsf{PRG}(s_{n,m})$, with the same dimension as the gradient x_n. The masked gradient is then computed as

$$\hat{x}_n \leftarrow x_n + \sum_{m\in M} P_{n,m} \pmod R. \tag{23}$$

For verifiability, each user $n \in U_2$ now uses the verification key pair (pk^3, sk^3) to compute its per-user verification key

$$K_n \leftarrow \sum_{m\in M} K_{n,m} = \sum_{m\in M} \mathsf{KA.agree}(sk_n^3, pk_m^3). \tag{24}$$

This is identical to the per-user key in the original design, but, crucially, it is now computed only for users that will actually contribute gradients in this round.

Given K_n and the global scalar α from (22), user $n \in U_2$ computes its MAC on the *unmasked* gradient:

$$\mathsf{MAC}_n \leftarrow K_n + \alpha \cdot x_n \pmod{R}. \tag{25}$$

The user sends the pair $(\hat{x}_n, \mathsf{MAC}_n)$ to the server. After a timeout, the server has received contributions from all users in U_2 and broadcasts the list U_2 to the auxiliary nodes, exactly as in the original protocol.

Round 3: unmasking and constructing a consistent global key K. Round 3 is where the essential fix is applied. The masking part remains unchanged: each auxiliary node uses (pk^2, sk^2) to regenerate the masks corresponding to the users in U_2 and sends their sum to the server. In addition, auxiliary nodes now recompute their verification contributions *only over* U_2, encrypt them for each user, and send them to the server so that users can reconstruct a global verification key K that matches the aggregated gradients.

More concretely, each auxiliary node $m \in M$ receives the set U_2 from the server. Using the key pair (pk^2, sk^2), it computes for each $n \in U_2$ the seed $s_{n,m} \leftarrow \mathsf{KA.agree}(sk^2_m, pk^2_n)$, $P_{n,m} \leftarrow \mathsf{PRG}(s_{n,m})$, and then sums the masks to obtain

$$P_m \leftarrow \sum_{n \in U_2} P_{n,m}. \tag{26}$$

In parallel, using the verification key pair (pk^3, sk^3), auxiliary node m computes for each $n \in U_2$ the verification subkey $K_{n,m} \leftarrow \mathsf{KA.agree}(sk^3_m, pk^3_n)$ and aggregates them into a node-level key *restricted to* U_2:

$$K_m \leftarrow \sum_{n \in U_2} K_{n,m}. \tag{27}$$

Notice the difference from the original protocol: here the sum is taken over U_2, not U_1. This is what realigns the global key with the set of users whose gradients are actually aggregated. For every user $n \in U_2$, auxiliary node m then derives an encryption key $k^{(1)}_{n,m} \leftarrow \mathsf{KA.agree}(sk^1_m, pk^1_n)$ and encrypts its node-level key: $ct_{n,m} \leftarrow \mathsf{AE.enc}\big(k^{(1)}_{n,m}, K_m\big)$. It sends P_m and the set $\{ct_{n,m}\}_{n \in U_2}$ to the server.

The server receives all $\{P_m, \{ct_{n,m}\}_{n \in U_2}\}_{m \in M}$ and computes the aggregate

$$X \leftarrow \sum_{n \in U_2} \hat{x}_n - \sum_{m \in M} P_m \pmod{R}, \tag{28}$$

which still satisfies $X = \sum_{n \in U_2} x_n$ because the masks cancel as before. It also computes the aggregated MAC

$$\mathsf{MAC} \leftarrow \sum_{n \in U_2} \mathsf{MAC}_n. \tag{29}$$

Let $U_3 \subseteq U_2$ denote the users that are online at this point. For each $n \in U_3$, the server sends (X, MAC) together with the ciphertexts $\{ct_{n,m}\}_{m \in M}$.

Round 4: user-side reconstruction of K and verification. In the final round, each user reconstructs a global key K from the auxiliary-node contributions corresponding to U_2 and uses it together with α to verify the aggregated result. Each user $n \in U_3$ receives (X, MAC) and the ciphertexts $\{ct_{n,m}\}_{m \in M}$. For every auxiliary node $m \in M$, user n computes $k^{(1)}_{n,m} \leftarrow \mathsf{KA.agree}(sk^1_n, pk^1_m)$ and decrypts $K_m \leftarrow \mathsf{AE.dec}(k^{(1)}_{n,m}, ct_{n,m})$. By summing over all auxiliary nodes, each user obtains

$$K \leftarrow \sum_{m \in M} K_m. \tag{30}$$

Let $U_4 \subseteq U_3$ be the set of users that actually perform the verification. For each $n \in U_4$, the verification computation is

$$\mathsf{MAC}' \leftarrow K + \alpha \cdot X \pmod{R}, \tag{31}$$

and user n accepts X if and only if $\mathsf{MAC}' = \mathsf{MAC}$.

Completeness under dropouts. The only semantic difference from the original protocol is that the global key K used in (31) is now defined as a function of U_2, the set of users that actually contributed gradients and MACs in the current round. Using the symmetry of the key agreement, we have $K_{n,m} = \mathsf{KA.agree}(sk^3_n, pk^3_m) = \mathsf{KA.agree}(sk^3_m, pk^3_n)$, and therefore, from (24) and (27), $K_n = \sum_{m \in M} K_{n,m}$ and $K_m = \sum_{n \in U_2} K_{n,m}$. Summing (27) over all auxiliary nodes yields

$$K = \sum_{m \in M} K_m = \sum_{m \in M} \sum_{n \in U_2} K_{n,m} \tag{32}$$

$$= \sum_{n \in U_2} \sum_{m \in M} K_{n,m} = \sum_{n \in U_2} K_n. \tag{33}$$

On the other hand, for an honest execution, (28) and (29) give

$$X = \sum_{n \in U_2} x_n, \qquad \mathsf{MAC} = \sum_{n \in U_2} \big(K_n + \alpha x_n\big) = \sum_{n \in U_2} K_n + \alpha X.$$

Combining this with (33), we obtain $\mathsf{MAC} = K + \alpha X \pmod{R}$, so that $\mathsf{MAC}' = \mathsf{MAC}$ in (31) for all honest users that complete Round 4, regardless of how many users dropped out between Rounds 1 and 2. This restores the completeness of the verification mechanism: as long as the server aggregates the masked gradients and MACs correctly over U_2, all honest users that verify will accept the result.

Importantly, the privacy guarantees and communication/computation costs of the original protocol are preserved. The masking structure and the sets of seeds $\{s_{n,m}\}$ are unchanged, and the auxiliary nodes still only

hold column-wise views of the mask matrix. The only additional operations are a small number of symmetric encryptions and decryptions to send K_m to users in Round 3 and reconstruct K in Round 4. Thus, the modified protocol retains the lightweight nature of the original scheme while fixing the inconsistency between the definition of K and the set of users whose gradients are aggregated.

4 Conclusion

In this work, we identified a critical flaw in the verifiability mechanism of the "Efficient Verifiable Protocol" (EVP) by Eltaras et al., demonstrating that it fails to satisfy the standard completeness requirement in the presence of user dropouts. We showed that the protocol's original design, which binds the global verification key to the initial set of participants, causes honest users to reject valid server aggregates whenever a client drops out. To resolve this, we proposed a modification that dynamically realigns the global verification key with the set of active contributors during the unmasking phase. Our formal analysis confirms that this fix restores robust verifiability under arbitrary dropouts without sacrificing the privacy guarantees.

References

[1] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. Practical secure aggregation for federated learning on user-held data. *arXiv preprint arXiv:1611.04482*, 2016.

[2] Nader Bouacida and Prasant Mohapatra. Vulnerabilities in federated learning. *IEEe Access*, 9:63229–63249, 2021.

[3] Tamer Eltaras, Farida Sabry, Wadha Labda, Khawla Alzoubi, and Qutaibah Ahmedeltaras. Efficient verifiable protocol for privacy-preserving aggregation in federated learning. *IEEE Transactions on Information Forensics and Security*, 18:2977–2990, 2023.

[4] Xiaojie Guo, Zheli Liu, Jin Li, Jiqiang Gao, Boyu Hou, Changyu Dong, and Thar Baker. V eri fl: Communication-efficient and fast verifiable aggregation for federated learning. *IEEE Transactions on Information Forensics and Security*, 16:1736–1751, 2020.

[5] Changhee Hahn, Hodong Kim, Minjae Kim, and Junbeom Hur. Versa: Verifiable secure aggregation for cross-device federated learning. *IEEE Transactions on Dependable and Secure Computing*, 20(1):36–52, 2021.

[6] Swanand Kadhe, Nived Rajaraman, O Ozan Koyluoglu, and Kannan Ramchandran. Fastsecagg: Scalable secure aggregation for privacy-preserving federated learning. *arXiv preprint arXiv:2009.11248*, 2020.

[7] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Kallista Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. Advances and open problems in federated learning. *Foundations and trends® in machine learning*, 14(1–2):1–210, 2021.

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

[9] Viraaji Mothukuri, Reza M Parizi, Seyedamin Pouriyeh, Yan Huang, Ali Dehghantanha, and Gautam Srivastava. A survey on security and privacy of federated learning. *Future Generation Computer Systems*, 115:619–640, 2021.

[10] Jinhyun So, Başak Güler, and A Salman Avestimehr. Turbo-aggregate: Breaking the quadratic aggregation barrier in secure federated learning. *IEEE Journal on Selected Areas in Information Theory*, 2(1):479–489, 2021.

[11] Guowen Xu, Hongwei Li, Sen Liu, Kan Yang, and Xiaodong Lin. Verifynet: Secure and verifiable federated learning. *IEEE Transactions on Information Forensics and Security*, 15:911–926, 2019.

Protocol 1: Original Verifiable Secure Aggregation Protocol (Eltaras et al.)

Setup:

- Parties agree on security parameter λ and public parameters $pp \leftarrow \mathsf{KA.gen}(\lambda)$.
- All users and auxiliary nodes have private authenticated channels with the server.

Round 0 (Key Advertisement):

- **User** n**:** Generates $(pk_n^1, sk_n^1), (pk_n^2, sk_n^2), (pk_n^3, sk_n^3) \leftarrow \mathsf{KA.gen}(pp)$ and sends public keys to server.
- **Aux Node** m**:** Generates $(pk_m^1, sk_m^1), (pk_m^2, sk_m^2), (pk_m^3, sk_m^3) \leftarrow \mathsf{KA.gen}(pp)$ and sends public keys to server.
- **Server:**
  - Collects keys from users (U_1) and auxiliary nodes (M).
  - Broadcasts $\{(pk_m^1, pk_m^2, pk_m^3)\}_{m \in M}$ to users in U_1.
  - Broadcasts $\{(pk_n^1, pk_n^2, pk_n^3)\}_{n \in U_1}$ to auxiliary nodes in M.

Round 1 (Key Sharing):

- **Aux Node** m**:**
  - Computes shared keys $K_{n,m} \leftarrow \mathsf{KA.agree}(sk_m^3, pk_n^3)$ for all $n \in U_1$.
  - Computes node key $K_m \leftarrow \sum_{n \in U_1} K_{n,m}$.
  - Samples $\alpha_m \leftarrow \mathbb{F}$.
  - Encrypts $ct_{n,m} \leftarrow \mathsf{AE.enc}(\mathsf{KA.agree}(sk_m^1, pk_n^1), \alpha_m \| K_m)$.
  - Sends $\{ct_{n,m}\}_{n \in U_1}$ to the server.
- **User** n**:**
  - Decrypts $\alpha \| K \leftarrow \sum_{m \in M} \mathsf{AE.dec}(\mathsf{KA.agree}(sk_n^1, pk_m^1), ct_{n,m})$.
  - Computes personal key $K_n \leftarrow \sum_{m \in M} \mathsf{KA.agree}(sk_n^3, pk_m^3)$.
- **Server:** Forwards ciphertexts $\{ct_{n,m}\}$ to respective users.

Round 2 (Masking Submission):

- **User** n**:**
  - Derives masks $s_{n,m} \leftarrow \mathsf{KA.agree}(sk_n^2, pk_m^2)$.
  - Masks gradient: $\hat{x}_n \leftarrow x_n + \sum_{m \in M} \mathsf{PRG}(s_{n,m}) \pmod R$.
  - Computes MAC: $\mathsf{MAC}_n = K_n + \alpha \cdot x_n \pmod R$.
  - Sends $\hat{x}_n$ and MAC_n to server.
- **Server:** Receives data from online users $U_2 \subseteq U_1$. Broadcasts U_2 to auxiliary nodes.

Round 3 (Unmasking Aggregation):

- **Aux Node** m**:**
  - Reconstructs masks for U_2: $P_{n,m} \leftarrow \mathsf{PRG}(\mathsf{KA.agree}(sk_m^2, pk_n^2))$.
  - Sums masks: $P_m \leftarrow \sum_{n \in U_2} P_{n,m}$. Sends P_m to server.
- **Server:**
  - Aggregates gradients: $X = \sum_{n \in U_2} \hat{x}_n - \sum_{m \in M} P_m \pmod R$.
  - Aggregates MACs: $\mathsf{MAC} = \sum_{n \in U_2} \mathsf{MAC}_n$.
  - Broadcasts (X, MAC) to users U_3.

Round 4 (Verification):

- **User** n**:**
  - Receives (X, MAC).
  - Computes $\mathsf{MAC}' = K + \alpha X$.
  - Accepts if $\mathsf{MAC}' = \mathsf{MAC}$.

Protocol 2: Corrected Verifiable Secure Aggregation Protocol

Setup:

- All parties agree on security parameter λ and public parameters $pp \leftarrow \mathsf{KA.gen}(\lambda)$.
- All users and auxiliary nodes have private authenticated channels with the server.

Round 0: Key Advertisement

- **User n:** Generates key pairs $(pk_n^1, sk_n^1), (pk_n^2, sk_n^2), (pk_n^3, sk_n^3) \leftarrow \mathsf{KA.gen}(pp)$ and sends public keys to the server.
- **Auxiliary Node m:** Generates key pairs $(pk_m^1, sk_m^1), (pk_m^2, sk_m^2), (pk_m^3, sk_m^3) \leftarrow \mathsf{KA.gen}(pp)$ and sends public keys to the server.
- **Server:**
  - Collects keys from users (U_0) and auxiliary nodes (M).
  - Broadcasts $\{(pk_m^1, pk_m^2, pk_m^3)\}_{m \in M}$ to all users in U_0.
  - Broadcasts $\{(pk_n^1, pk_n^2, pk_n^3)\}_{n \in U_0}$ to all auxiliary nodes in M.

Round 1: Establishing Universal Scalar α

- **Auxiliary Node m:**
  - Samples random $\alpha_m \leftarrow \mathbb{F}$.
  - For each $n \in U_0$, computes $a_{n,m} \leftarrow \mathsf{AE.enc}(\mathsf{KA.agree}(sk_m^1, pk_n^1), \alpha_m)$ and sends $\{a_{n,m}\}_{n \in U_0}$ to the server.
- **Server:** Forwards $\{a_{n,m}\}_{m \in M}$ to each online user $n \in U_1$.
- **User n ($\in U_1$):**
  - Decrypts $\alpha_m \leftarrow \mathsf{AE.dec}(\mathsf{KA.agree}(sk_n^1, pk_m^1), a_{n,m})$.
  - Computes global scalar $\alpha \leftarrow \sum_{m \in M} \alpha_m$.

Round 2: Masking and Gradient Submission

- **User n ($\in U_2$):**
  - Derives shared seeds $s_{n,m} \leftarrow \mathsf{KA.agree}(sk_n^2, pk_m^2)$ and masks gradient: $\hat{x}_n \leftarrow x_n + \sum_{m \in M} \mathsf{PRG}(s_{n,m}) \pmod R$.
  - Computes local verification key: $K_n \leftarrow \sum_{m \in M} \mathsf{KA.agree}(sk_n^3, pk_m^3)$.
  - Computes MAC: $\mathsf{MAC}_n \leftarrow K_n + \alpha \cdot x_n \pmod R$.
  - Sends $(\hat{x}_n, \mathsf{MAC}_n)$ to the server.
- **Server:** Receives contributions from U_2. Broadcasts the identity set U_2 to all auxiliary nodes.

Round 3: Unmasking and Key Alignment (The Fix)

- **Auxiliary Node m:**
  - Reconstructs masks for U_2: $P_{n,m} \leftarrow \mathsf{PRG}(\mathsf{KA.agree}(sk_m^2, pk_n^2))$. Computes sum $P_m \leftarrow \sum_{n \in U_2} P_{n,m}$.
  - **Key Realignment:** For each $n \in U_2$, computes verification subkey $K_{n,m} \leftarrow \mathsf{KA.agree}(sk_m^3, pk_n^3)$.
  - Aggregates verification key over U_2: $K_m \leftarrow \sum_{n \in U_2} K_{n,m}$.
  - Encrypts for users: $ct_{n,m} \leftarrow \mathsf{AE.enc}(\mathsf{KA.agree}(sk_m^1, pk_n^1), K_m)$.
  - Sends P_m and $\{ct_{n,m}\}_{n \in U_2}$ to the server.
- **Server:**
  - Aggregates result: $X \leftarrow \sum_{n \in U_2} \hat{x}_n - \sum_{m \in M} P_m \pmod R$.
  - Aggregates MACs: $\mathsf{MAC} \leftarrow \sum_{n \in U_2} \mathsf{MAC}_n$.
  - Broadcasts (X, MAC) and ciphertexts $\{ct_{n,m}\}_{m \in M}$ to online users U_3.

Round 4: Verification

- **User n ($\in U_4$):**
  - Decrypts node keys: $K_m \leftarrow \mathsf{AE.dec}(\mathsf{KA.agree}(sk_n^1, pk_m^1), ct_{n,m})$.
  - Reconstructs global key: $K \leftarrow \sum_{m \in M} K_m$.
  - Computes $\mathsf{MAC}' \leftarrow K + \alpha \cdot X \pmod R$.
  - **Verification:** Accepts aggregate X if and only if $\mathsf{MAC}' = \mathsf{MAC}$.