

Securing Private Federated Learning in a Malicious Setting: A Scalable TEE-Based Approach with Client Auditing

Shun Takagi*
LY Corporation
Japan
shutakag@lycorp.co.jp

Satoshi Hasegawa
LY Corporation
Japan
satoshi.hasegawa@lycorp.co.jp

Abstract

In cross-device private federated learning, differentially private follow-the-regularized-leader (DP-FTRL) has emerged as a promising privacy-preserving method. However, existing approaches assume a semi-honest server and have not addressed the challenge of securely removing this assumption. This is due to its statefulness, which becomes particularly problematic in practical settings where clients can drop out or be corrupted. While trusted execution environments (TEEs) might seem like an obvious solution, a straightforward implementation can introduce forking attacks or availability issues due to state management. To address this problem, our paper introduces a novel server extension that acts as a trusted computing base (TCB) to realize maliciously secure DP-FTRL. The TCB is implemented with an ephemeral TEE module on the server side to produce verifiable proofs of server actions. Some clients, upon being selected, participate in auditing these proofs with small additional communication and computational demands. This extension solution reduces the size of the TCB while maintaining the system's scalability and liveness. We provide formal proofs based on interactive differential privacy, demonstrating privacy guarantee in malicious settings. Finally, we experimentally show that our framework adds small constant overhead to clients in several realistic settings.

Keywords

differential privacy, federated learning, malicious security

1 Introduction

Cross-device Private Federated Learning (PFL) has recently garnered significant attention as a privacy-preserving learning framework across distributed clients' data [22, 43, 84]. In a typical cross-device PFL setting, a single central server (referred to simply as a server hereafter) aims to train a unified machine learning model by collecting model updates instead of raw data from numerous lightweight clients. To provide a provable guarantee that privacy leakage from model updates is bounded, a differential privacy (DP) [31] mechanism injects noise into the (aggregated) updates.

The FL systems should be designed to uphold key privacy principles [14, 39]. DP supports this goal by enforcing data minimization

*Corresponding author.

This work is licensed under the Creative Commons Attribution 4.0 International License. To view a copy of this license visit <https://creativecommons.org/licenses/by/4.0/> or send a letter to Creative Commons, PO Box 1866, Mountain View, CA 94042, USA.



Proceedings on Privacy Enhancing Technologies 2026(1), 486–508

© 2026 Copyright held by the owner/author(s).
<https://doi.org/10.56553/poops-2026-0025>

and anonymization. However, many current DP FL methods lack mechanisms to ensure transparency, verifiability, and auditability from the clients' perspective [25]. That is, they heavily rely on the assumption that the server behaves *honestly*. A dishonest server or an external attacker (i.e., an adversary) might gain sensitive information more than clients expect, without their awareness, causing the DP guarantee to fail.

To address this gap, we adopt a *malicious* adversarial model (i.e., the Dolev-Yao model [29]) rather than the weaker semi-honest model. A *semi-honest* adversary correctly follows the protocol specifications but attempts to glean sensitive information from the execution. A *malicious* adversary, however, poses a greater threat as they are not bound by the protocol; they can deviate from it arbitrarily, which includes corrupting the server or a subset of clients.

Among the various DP approaches, methods based on differentially private stochastic gradient descent (DP-SGD) [1], such as DP-FedAvg [59] and DP-SCAFFOLD [67], have been widely used. Extensive research has focused on making these methods at least partially secure in settings with an untrusted server. Secure Multiparty Computation (SMPC) [13] can ensure the secure aggregation of updates and can be adapted for PFL, such as through distributed differential privacy [2, 45]. However, these methods assume the honesty of clients, and corrupted clients—such as in Sybil attacks [30]¹—can reduce the noise added for privacy. Moreover, achieving essential privacy amplification with subsampling [1] requires complex random device sampling protocols, which are challenging to securely implement in a distributed environment (e.g., relying on non-colluding servers [76]). Another approach, shuffling [36], involves client-side perturbation and anonymization [34] of updates. Shuffling is also susceptible to Sybil attacks, compromising the privacy of remaining clients' updates if a large number of clients are corrupted [47]. Additionally, shuffling often results in a worse utility-privacy trade-off or higher communication costs compared to other methods [8].

That is, these approaches require subsampling [1] or shuffling [34] of updates, which are difficult to implement effectively in malicious settings. Even in semi-honest settings, these approaches are impractical in a cross-device environment since the server has limited control over the subset of training data it encounters at any time [47]. In this context, DP-FTRL (Differentially Private Follow-the-Regularized-Leader) [46] has emerged as an alternative to DP-SGD-based methods, eliminating the need for such techniques. Moreover, recent studies have demonstrated, both theoretically and

¹In this paper, we define Sybil attacks as scenarios where an adversary controls multiple client identities (i.e., Sybils) by possessing their secret keys, which allows a portion of the clients to collude under the adversary's direction.

empirically, that DP-FTRL may offer a better privacy-utility trade-off than DP-SGD [18, 19]. Therefore, DP-FTRL appears promising for PFL in malicious settings.

However, implementing DP-FTRL in malicious environments introduces new challenges due to its stateful nature [25]. Unlike DP-SGD, which aggregates updates with DP independently in each round, DP-FTRL requires maintaining state throughout the training process, leading to a more complex server implementation that is difficult to verify for clients. Specifically, DP-FTRL requires two steps for planning and aggregation that takes previous rounds into account. During the planning phase, the server selects a new set of participants, referred to as a cohort, based on the clients' participation history to manage privacy leakage by bounding contribution. In the aggregation phase, the server integrates the cohort's updates with noise that is correlated with noise used in earlier rounds, reducing the total noise required.

Recent work [7, 12] has attempted to bridge this gap by proposing SMPC for such noise, but this relies on the assumption that the planning phase honestly determines the majority cohort. These approaches do not effectively protect against a malicious server that can freely select corrupted clients (i.e., Sybils) during the planning phase to conduct Sybil attacks. Notably, a malicious server could exploit the planning phase with Sybils to repeat an iteration with different cohorts to uncover previous noise from correlations, thus recovering raw past updates. Since clients cannot detect repeated iterations due to changes in cohorts, preventing such attacks is impossible without communication. Consequently, achieving DP-FTRL in a malicious setting remains a challenging issue.

Contributions

Problem Formulation. To address this challenge, we begin research on developing a maliciously secure system long-running FL with DP-FTRL that supports adaptive² client participation. We pose the following research question:

What constitutes ideal security for a maliciously secure DP-FTRL system for long-running FL involving lightweight devices that can drop out, and how can we achieve it at a low cost?

To formalize this problem, we adopt interactive DP [79] in a distributed manner. The (ϵ, δ) -interactive DP framework assumes only predefined client algorithms, while the server protocol can be arbitrary. This ensures that any server protocol concurrently interacting with client algorithms (i.e., star topology) cannot infer distributed data beyond what is guaranteed by (ϵ, δ) -DP. Therefore, this formalization accurately reflects our malicious setting.

As described above, DP-FTRL needs to manage its state throughout the execution process. Therefore, we envision the set of client algorithms as a concurrent system that maintains a shared object with all clients, representing the state, such as participation history. In this framework, we view DP-FTRL as a series of sequential operations composed of two key steps on the concurrent system with consistent state: planning with *consistent* participation history and

²Adaptivity is crucial because a static client selection strategy risks losing large amounts of data due to client dropouts, which can severely degrade model quality. Furthermore, in applications where users adaptively gather data, those who have collected data should participate [43].

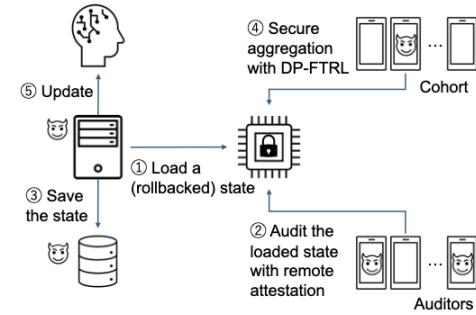


Figure 1: The process overview of each round of our system.

secure aggregation with *consistent* noise. Within the concurrent system, these two steps constitute a process initiated by a (potentially malicious) server, which updates the shared object.

Since breaking consistency can lead to privacy leaks as mentioned above, a necessary condition for the concurrent system to satisfy interactive DP is to maintain process consistency with respect to the shared object, ensuring that a malicious server cannot complete any process that compromises this consistency. Thus, we focus on ensuring the consistency of the concurrent system. Specifically, the system must uphold:

- **Integrity:** The process loads the shared object and adhere to the intended protocol (i.e., secure aggregation with the correct clients and correct noise).
- **Linearizability [38]:** The history of concurrent processes is equivalent to a history of sequential processes with respect to the shared object (i.e., participation history and noise added in the past rounds).

Implementing a linearizable system in a malicious environment requires global consensus (involving all clients) at each iteration, which is prohibitively expensive for large-scale FL. Without global consensus, fork linearizability is the strongest consistency that can be achieved [58], but it is not sufficient for DP-FTRL because it still permits the aforementioned Sybil attack. Instead, we rely on a probabilistic linearizability guarantee, which is sufficient for our purpose.

Proposed Method. To address the aforementioned problem, we incorporate Trusted Execution Environments (TEEs), which are gaining attention as future solutions for secure infrastructure [5, 33, 72] in private data analytics, including FL infrastructure³ [25, 87]. Although this paper focuses on Intel SGX [21] as a concrete example of a TEE, our method is applicable to any TEE that supports attested execution [70]. While running the entire FL logic within a TEE could serve as a potential solution, functioning as a trusted third party, naive designs are susceptible to forking attacks on the TEE-managed state [82], which breaks the consistency.

To tackle this issue, we propose a novel system that integrates TEEs with client auditing. Figure 1 illustrates the overview of our system during each update. (1) For each round, the server feeds

³<https://research.google/blog/advances-in-private-training-for-production-on-device-language-models/>

Table 1: Comparison of TEE-based systems for stateful applications. Our approach achieves consistency and liveness guarantees with a small TCB and without requiring external trusted hardware.

Approach	Consistency	Liveness	No External Trusted Hw.	Formal Verification	TCB Size
TEEs without sealing	✓	✗	✓	n/a	Small
TEEs with sealing	✗	✓	✓	n/a	Small
SMR on TEEs without sealing [4, 40, 57]	✓	✗ [†]	✓	n/a	Medium
SMR on TEEs with sealing [66, 81]	✗ [‡]	✓	✓	n/a	Large
TEEs + external trusted hardware [75]	✓	✓	✗	✓	Small
Ours	✓ [*]	✓ [*]	✓	✓	Small

^{*} Ours provides probabilistic guarantees; These properties hold with high probability, failing only with a small chance (e.g., 10^{-8}).

[†] Liveness is achieved under the assumption that a majority of TEEs do not experience faults (i.e., disaster).

[‡] Consistency is achieved under the assumption that a majority of TEEs remain uncompromised by adversaries.

n/a: Not addressed or not available in the literature.

the current system state, along with the arguments for secure aggregation, into the TEE. (2) Upon successful client auditing of the state within the TEE with remote attestation, the TEE generates proofs of integrity and linearizability. (3) These proofs, bundled with the newly validated state, are then saved to untrusted storage. (4) Then, the secure aggregation process within the TEE with the input arguments is completed only after clients have successfully verified it. Should an adversary attempt a fork or rollback attack by inputting a fraudulent state, the client-side verification would fail, thus preventing the secure aggregation from executing.

However, guaranteeing linearizability poses a significant hurdle, as it would still necessitate consensus among all clients. A malicious server could otherwise exploit any set of corrupted auditors to approve a proof that violates consistency. Our system mitigates this vulnerability by employing randomized auditing. In each round, a random subset of clients is tasked with auditing the server’s proof, which makes it infeasible for an adversary to ensure its colluders are selected. As a result, the probability of a forged proof being successfully validated becomes small (e.g., 10^{-8}). Consequently, our protocol forgoes a linearizability guarantee in favor of a probabilistic one, with a failure rate small enough for practical applications.

This system acts as an extension to the existing FL system, where the server merely appends proof to an original message, which each client verifies. Moreover, the additional computation and communication costs for the client are constant and small, enabling the system to preserve the scalability and liveness of the original arrangement.

Related Work. While TEEs provide a powerful primitive for attested execution—conceptually abstracted as $\mathcal{G}_{\text{att}}^{\text{rollback}}$ [11], they are not known to offer a self-contained solution for preventing rollback attacks. One strategy to this problem avoids TEE sealing capabilities, reducing the TEE to a stateless primitive (i.e., \mathcal{G}_{att} [70]) that ensures consistency at the cost of liveness. Another major strategy augments the TEE with an external source of trust, such as an external hardware monotonic counter like Trusted Platform Module (TPMs). In Table 1, we compare our proposed system against these existing approaches based on five key properties important for secure, long-running FL with DP-FTRL on TEEs in a malicious setting.

First, the most straightforward TEE-based implementations either forgo state persistence or use native sealing. An implementation without sealing maintains consistency and a small trusted computing base (TCB) (i.e., the small size of code in the TEE, which reduces the need to implicitly trust its correctness) but fails to provide liveness; the state is irrevocably lost if the TEE instance crashes. Conversely, an implementation with sealing ensures liveness by persisting the state to untrusted memory, but it becomes vulnerable to forking and rollback attacks [82], thus failing to guarantee consistency.

Recent work has sought to address both consistency and liveness by implementing State Machine Replication (SMR) across multiple TEEs. Systems like Engraft [81] and Narrator [66] achieve consistency, but their guarantees rely on an honest majority of TEE replicas. This assumption does not hold in our threat model, where an adversary may control (e.g., rollback) the majority of nodes.

Nimble [4], CCF [40], and ROTE [57] operate under a threat model similar to ours. They provide consistency even in the presence of a malicious majority, not by relying solely on sealing APIs for node recovery, but by leveraging consensus among the live nodes. However, this design choice means they cannot guarantee liveness in the face of catastrophic failures (e.g., a crash of the majority of TEEs). Furthermore, these systems inherit the complexity of SMR protocols inside TEEs, leading to a large TCB, which is undesirable according to the principles of TEEs [6]. As a result, formally verifying their security is challenging⁴, and complex in-enclave implementations are more susceptible to bugs. Indeed, vulnerabilities related to consistency have been discovered in some of these systems [66, 81].

Another line of work combines TEEs with additional trusted hardware components (e.g., TPMs) external to the TEE to enforce state integrity [69, 75]. These solutions are often simple, formally verifiable, and feature a small TCB. However, they depend on the availability of specialized hardware or a trusted monotonic counter, and securely integrating it with a TEE can be challenging. For example, the SGX-native monotonic counter is now deprecated, and combining SGX with an external TPM is susceptible to man-in-the-middle attacks like the Cuckoo attack [68], where a TEE is tricked

⁴Nimble [4] made considerable efforts in this area but did not fully verify functions such as reconfiguration. <https://github.com/microsoft/Nimble/issues/6>

into communicating with a remote, adversary-controlled TPM instead of the local one, allowing forking attacks. The vulnerability arises because, as explicitly described in the paper [75], the method requires the TEE to be locally bound (i.e., on the same machine) to the TPM to prevent forking attacks. However, an enclave cannot verify this local binding due to the threat of the Cuckoo attack. A proposed countermeasure, verifying adjacency by measuring communication timing as proposed by Fink et al. [35], is ineffective in our setting, which may involve high-speed networks like a LAN. Therefore, establishing a truly secure and trusted channel between a TEE and a TPM has another challenging problem.

Conceptually, our method for achieving consistency is analogous to TPM-based solutions. However, instead of relying on external trusted hardware, our system leverages client auditing. This approach inherits the benefits of a simple and small TCB, enabling formal verification. It provides recovery from fault (i.e., liveness) and requires only a single TEE-enabled server, thereby reducing management overhead. More general related work is provided in Appendix C.

Contributions. In summary, our proposed system realizes DP-FTRL with the following desired characteristics:

- **Privacy:** Our approach preserves interactive DP (i.e., DP under the malicious setting). Smaller reliance on TEE (i.e., small TCB size) reduces security risks and enhances verifiability.
- **Availability:** Our system retains its standard resilience to dropouts—training can continue even if some devices become nonresponsive, and it is possible to recover from crashes (i.e., liveness). A malicious client cannot halt the system via fraudulent audits.
- **Scalability:** Clients perform only constant lightweight auditing, incurring small computation and communication overhead. The system operates with large-scale clients that can drop out and progresses multiple processes concurrently.

Outline. The remainder of this paper is structured as follows. In Section 2, we provide the necessary background on key concepts, including TEEs, DP-FTRL, and the security models we employ. We then formalize our problem in Section 3, defining the requirements for a maliciously secure DP-FTRL system in terms of integrity and linearizability, and introduce our simulation-based proof framework. Section 4 details the core design of our proposed system. We provide a thorough analysis of our system in Section 5, covering its privacy guarantees, liveness under client dropouts, scalability, and guidance on parameter selection. In Section 6, we present experimental results that demonstrate the practical performance of our system, focusing on the trade-offs between communication cost, security, and liveness, as well as the concrete overhead of our implementation. We discuss side-channel attacks and implementation challenges in Appendix A and situate our work within the existing literature in Appendix C.

2 Preliminaries

In this section, we review some key concepts and frameworks relevant to our proposed method, including Trusted Execution Environments (TEEs), Differentially Private Follow-the-Regularized-Leader (DP-FTRL), the concurrent system model, and security notions.

2.1 Trusted Execution Environments (TEEs)

Although this paper focuses on Intel SGX [21] as a concrete example of a TEE, our method is applicable to any TEE that provides attested execution [11, 70] like AMD SEV-SNP [3] (i.e., confidentiality and integrity of execution). Below, we offer a high-level overview and refer the reader to [24] for more intricate details.

2.1.1 Enclaves. TEEs utilize trusted hardware to provide isolated environments known as enclaves. Enclaves protect memory by restricting access, thereby ensuring data confidentiality from unauthorized processes, including the operating system. Enclaves can produce an attestation report, which we call evidence. This report typically includes a hash of the running code (to verify its integrity), additional data related to the enclave’s execution, and a signature using a key endorsed by the TEE manufacturer. In this paper, an enclave can be seen as an entity with functions that generate evidence. Due to confidentiality guarantees, only the final evidence is visible. Moreover, enclaves can generate a secure nonce valid for the duration of the function’s execution to thwart replay attacks.

Sealing. Enclaves are volatile but can be made persistent through sealing. Specifically, they can securely encrypt and store states outside the enclave, allowing replication by an enclave with the same binary.

2.1.2 Remote Attestation. The signature in the evidence allows a remote party to verify that the evidence was generated by a genuine enclave running on authentic TEE hardware. A client can authenticate this evidence through various methods, including certificate chains or contacting an attestation service. The included hash of the code and computation results enable clients to affirm integrity.

2.1.3 Secure Aggregation. TEEs can perform secure aggregation of distributed data x_i [16, 41, 48], enabling computation of $\sum_{i=1}^n x_i$ without exposing individual values x_i to a potentially malicious server. Compared to SMPC [13], which provides similar functionality, TEE-based secure aggregation requires smaller client-side operations, maintains constant complexity, and offers the flexibility to add complex noise without Sybil attacks. The core idea is using an enclave as a trusted computing entity through remote attestation. Each client sets up a secure channel to the enclave using the Diffie-Hellman key exchange protocol. Client i generates a symmetric key to encrypt x_i and then transmits this symmetric key securely to the enclave. Since loading large data (i.e., encrypted model parameters) into an enclave can be time-intensive, methods such as stream ciphers [48] like homomorphic one-time pad [41] can enhance efficiency. See Section B.3 of [41] for a detailed methodology.

2.2 Threat Model

We consider a system comprising $n + 1$ parties: one server and n clients. The server has significant computational resources. Each client holds private input data x and a pair of private and public keys. Clients can only communicate with the server, reflecting a star network topology. These clients are lightweight and can drop out, meaning they may have limitations in connectivity, bandwidth, and computational power.

An adversary aims to compromise the privacy of clients’ sensitive values. We adopt the Dolev-Yao model [29] as our model of the

malicious adversary. That is, an adversary can eavesdrop on all communication, inject messages, and corrupt parties by obtaining their private keys. However, the adversary cannot break cryptographic primitives, in this paper, digital signatures, nonces, Diffie-Hellman key exchanges, and the symmetric encryption.

In our threat model, we include a fixed server and clients listed in a public key list, assuming the following conditions with $\gamma, \kappa, \beta \in [0, 1]$:

- The adversary can corrupt the server and a fraction γ of **all clients**. The compromised clients are fixed.
- At any given time, at least a fraction κ of **all clients** are available, with this availability being variable.
- A fraction β of **all available clients** may drop out, failing to fulfill their roles in the protocol.

This model specializes the (γ, β) secure federated MPC paradigm [7], tailoring it to more realistically account for the fraction of all clients rather than just any single cohort. No party, apart from an adversary, can distinguish corrupted clients. No party can discern which clients will drop out.

We presuppose a trusted Public Key Infrastructure (PKI). While clients do not communicate with each other directly, they leverage the PKI to validate the public key list. Specifically, a client sends a hash of a client list to the PKI, and the PKI certifies whether the list is valid. A valid list is one where the proportion of malicious clients is at most γ and the total number of clients is above a certain threshold. This circumscribed role for the PKI reduces our dependency on it compared to other SMPC frameworks [7, 13].

We assume that enclaves provide confidentiality and integrity as described in Section 2.1.1. Thus, vulnerabilities against TEEs, such as side-channel attacks, are out of scope; however, a detailed discussion about side-channel attacks on our system is provided in Appendix A.1.

2.3 Concurrent System

In the threat model outlined above, we consider a distributed concurrent system of clients working cooperatively with a server, all operating on a single object with one type of operation. Generally, an operation is denoted as $\langle q \text{ op(args)} \rangle \text{OK(response)} \rangle$, where A is the process name, op is the type of operation, $args$ is the list of arguments, and q is the object. OK indicates the completion of process A and outputs the response.

For example, let q be an ordered list, and consider an operation that appends an element a to this list, outputting the result of $f(q, args)$. This could be denoted as $\langle q \text{ append}(a, args) \rangle \text{OK}(f(q, args)) \rangle$ with some given function f and arguments $args$. In the *concurrent* system, processes are invoked concurrently, meaning that process B can complete before process A , even if process A was invoked prior to process B .

2.3.1 Integrity and Linearizability. The properties of the concurrent system that we focus on are **integrity** and **linearizability**.

Integrity of execution. Integrity is the property that each individual operation behaves according to its specification. For the above example, given a object q , element a , and arguments $args$ for a function f , a system with integrity correctly appends a to q and computes and returns the result of $f(q, args)$. This implies that

the object's state, the arguments, and the operation itself are not tampered with, and the computation is performed as intended.

Linearizability [38]. Linearizability is a correctness condition for concurrent objects which ensures that each process appears to take effect instantaneously at some point between its invocation and response. If a concurrent system respects linearizability, it ensures that the result of concurrent processes is equivalent to a result of sequential processes that are produced by the system that satisfies integrity and respects real-time precedence ordering. See [38] for the more formal definition. For instance, consider f as a function that outputs q :

- If a system produces $\langle q \text{ append}(a_1) \rangle \text{OK}(a_1) \rangle \text{OK}(a_2) \rangle$, it respects linearizability; this is consistent with $\langle q \text{ append}(a_1) \rangle \text{OK}(a_1) \rangle \text{OK}(a_2) \rangle$.
- However, a system that produces $\langle q \text{ append}(a_1) \rangle \text{OK}(a_1) \rangle \text{OK}(a_2) \rangle$ does not, because the output of process B does not account for the update made by A , and thus it is not equivalent to a result of any sequential processes.
- Also, a system that produces $\langle q \text{ append}(a_2) \rangle \text{OK}(a_2) \rangle \text{OK}(a_1) \rangle$ does not respect linearizability. Here, B completes before A invokes, establishing a real-time order where B precedes A . However, B 's response depends on A 's input, requiring A to precede B in any sequential explanation. This contradiction violates the real-time precedence condition.

Unfortunately, under the threat model described in Section 2.2, achieving linearizability is infeasible without the agreement of all clients in each process. The strongest consistency notion achievable under these conditions is captured by fork-linearizability [53, 58]. However, this is insufficient for DP-FTRL as it remains vulnerable to the Sybil attack mentioned in Introduction. Instead, we rely on a probabilistic linearizability guarantee, which is sufficient for our purpose.

2.4 Interactive Differential Privacy (Interactive DP)

DP provides a privacy framework for randomized mechanisms that release outputs [32, 77]. It measures how close the probability distributions of a mechanism's outputs are when applied to two adjacent datasets. The definition of adjacency is crucial. While it can be broadly defined by the addition/removal of a single user's record, in the context of FL where the participants are fixed, we adopt a more specific definition known as *zero-out adjacency* [19, 46]. In this setting, two datasets D and D' , which are multisets over a data universe \mathcal{X} , are called adjacent if they differ in the *participation* of a single client. Here, the *participation* means containing the client's true contribution, while in the other, that contribution is replaced with a vector of zeros to represent the client's non-participation.

Definition 2.1 (Differential Privacy [31]). For $\epsilon, \delta \geq 0$, a randomized mechanism $M : \mathcal{X}^n \rightarrow \mathcal{Z}$ is (ϵ, δ) -differentially private if for every pair of adjacent datasets $D, D' \in \mathcal{X}^n$ and all subsets $T \subseteq \mathcal{Z}$, the following holds:

$$\Pr[M(D) \in T] \leq e^\epsilon \cdot \Pr[M(D') \in T] + \delta$$

where the randomness is over the internal coin flips of the algorithm M .

Recently, the concept of interactive DP [78] was developed to handle scenarios where multiple DP mechanisms operate concurrently. Originally, this was explored in a centralized DP context, where a single party holds all sensitive data, to examine properties like the concurrent composition theorem [79]. In our setting, we consider the distributed setting of DP, where each client manages their own data [9]. Note that we do not require each mechanism to meet the criteria of local DP [49], instead focusing on interactive DP.

Here we present interactive DP applicable to our context. We focus on an interactive protocol S for the concurrent composition of client mechanisms M_1, M_2, \dots, M_n , where M_i has sensitive input x_i for $i \in [n]$, denoted by $M = \text{CONCOMP}(M_1, M_2, \dots, M_n)$ [79]. Within this interactive protocol, S can send messages to any client (i.e., $i \in [n]$) and receive responses from M_i . Each party can store received messages for crafting future responses. The view of the interactive protocol, $\text{View}(S, M(D))$, is defined as the sequence of messages that S receives from M_1, M_2, \dots, M_n , where $D = (x_1, x_2, \dots, x_n) \in \mathcal{X}^n$. For a more formal definition, see [78].

Interactive DP is defined as follows:

Definition 2.2 (Interactive Differential Privacy [78]). A randomized algorithm M is an (ϵ, δ) -differentially private interactive mechanism if, for every pair of adjacent datasets $D, D' \in \mathcal{X}^n$, for every adversary algorithm A , and for every possible output set $T \subseteq \text{Range}(\text{View}(A, M(\cdot)))$, the following holds:

$$\Pr[\text{View}(A, M(D)) \in T] \leq e^\epsilon \Pr[\text{View}(A, M(D')) \in T] + \delta$$

where the randomness is over the internal coin flips of both the algorithm M and the adversary A .

Simulation Based Proof. Proving interactive DP is not straightforward because we cannot assume a semi-honest server-side protocol (e.g., the server can invoke processes concurrently). To address this, we adopt an approach similar to Braun et al. [15] and Ball et al. [7], using a simulation-based method [54] to prove the interactive DP of $\text{CONCOMP}(M_1, M_2, \dots, M_n)$. This proof technique compares our real-world protocol to an idealized version. The proof framework consists of two main steps: defining the ideal functionality and proving the emulation of the ideal functionality by the real-world protocol.

First, we define an *ideal functionality* that is executed by a trusted party in a hypothetical ideal world. This functionality specifies the perfect, secure execution of DP-FTRL. It interacts with a *Simulator*, which represents the adversary in this ideal world. By design, the view of the simulator in this interaction is guaranteed to be DP. The ideal functionality is designed to allow the simulator to influence the protocol on behalf of corrupted parties, for example, by providing malicious inputs or choosing the order of round completion, thus modeling the power of a real-world adversary.

Second, we demonstrate that our proposed concurrent system, $\text{CONCOMP}(M_1, M_2, \dots, M_n)$, emulates this ideal functionality when interacting with any real-world adversary A . This is shown by constructing a simulator, Sim , that can produce a view computationally indistinguishable from the real adversary's (i.e., A) view, using only its interaction with the ideal functionality. This ensures that any

view created by A matches the DP-compliant view in the ideal functionality, thereby ensuring compliance with interactive DP.

2.5 Differentially Private Follow-the-Regularized-Leader (DP-FTRL)

Kairouz et al. [46] proposed DP-FTRL as a DP optimization mechanism that avoids the need for subsampling [1] and shuffling [34] which are difficult due to client dropouts. Subsequent research expanded DP-FTRL with the matrix mechanism [28], called MF-DP-FTRL. This approach enables useful techniques such as multi-epoch training, amplification, and runtime efficiency due to its flexibility [19, 20, 60]. Recent studies indicate that MF-DP-FTRL can generalize DP-SGD and demonstrate its superiority in terms of the utility-privacy trade-off [18, 19].

We employ MF-DP-FTRL in our research to align our method with those operating within this framework. The pseudocode for the MF-DP-FTRL framework is located between Line 6 and Line 17 (where the simulator assigns k as i , C_{cohort} as valid client indices, and θ as $\hat{\theta}_i$) in Algorithm 1. While resembling DP-SGD with gradient computation, clipping, and gradient-noise summation for updates, it deviates in the following key ways:

- (1) Participation schema: client indices are arbitrarily chosen from those meeting the participation schema based on historical participation with an option for sampling.
- (2) Stateful aggregation: noise is correlated with noise used in previous rounds.

Due to these differences, unlike DP-SGD, each iteration in DP-FTRL cannot be treated as an independent DP mechanism. Thus, the server must handle participation history and noise management as detailed below.

Participation Schema [19]. Participation history H is a sequence of sets, each representing a client's participation record. Specifically, H_i is a set representing client i 's participation pattern $\pi \subseteq [n_{\text{round}}]$, where n_{round} is the number of total rounds. For instance, $H = (\{0, 2\}, \{1, 3\})$ indicates that client 0 participates in the 0th and 2nd rounds, while client 1 participates in the 1st and 3rd rounds. The participation schema is defined as follows:

Definition 2.3. A participation schema Π is defined as the set of possible participation patterns $\pi \subseteq [n_{\text{round}}]$. A participation history H adheres to a participation schema Π if for all $i \in [n]$, there exists $\pi \in \Pi$ such that $H_i \subseteq \pi$.

Updating H with $C \subseteq [n]$ and index i involves adding i to H_j for each $j \in C$. In this context, the current participation history H is used to determine which clients are eligible to participate in the i th round by using the function $f_{\text{qualify}}(\Pi, H, i)$, defined below. The function $f_{\text{qualify}}(\Pi, H, i)$ returns the set $C_{\text{qualify}} \subseteq [n]$ such that even if H is updated with C_{qualify} and i , H continues to adhere to the participation schema Π .

For example, if a client's participation is limited to just once, $\Pi = \{\{1\}, \{2\}, \dots, \{n_{\text{iter}}\}\}$. Recent works have demonstrated that the Π_b schema is both practical and attains utility comparable to DP-SGD by permitting multi-epoch training and amplification [19]. This schema permits participation at intervals of b between submissions.

Stateful Aggregation. In stateful aggregation, a noise matrix $Z \in \mathbb{R}^{n_{\text{round}} \times d}$ is employed throughout the DP-FTRL execution, with its elements independently drawn from a Gaussian distribution.⁵ The noise added in the i -th execution relies on Z_{jk} for $j \in [i]$ and $k \in [d]$. This method, which depends on noise from prior rounds, is termed correlated noise. The particular approach for calculating this noise is given by C . Specifically, the noise for i th round is $\zeta(C^{-1}Z)[i, :]$ where ζ is the clipping bound.

This method of noise integration is called the matrix mechanism in the adaptive streaming [28], and in the context of DP-FTRL, C is optimized according to a participation schema. Intuitively, this optimization minimizes the introduced error by controlling the sensitivity of participating clients based on the schema. Refer to [19, 28] for more details. Therefore, if clients deviate from the participation schema, their sensitivity may surpass allowable thresholds, risking privacy breaches.

3 Problem Statement

This section aims to formalize the problem of implementing DP-FTRL in our malicious setting.

3.1 DP-FTRL in the Malicious Setting

In our threat model (see Section 2.2), the server interacts with client algorithms M_1, M_2, \dots, M_n to update the model. The server is malicious, so it can deviate from a predefined protocol, making it impractical to presume any specific server behavior. This situation corresponds to concurrent composition in interactive DP (see Section 2.4). Therefore, our goal is to construct a concurrent composition mechanism $M = \text{ConComp}(M_1, M_2, \dots, M_n)$ that can execute DP-FTRL with an appropriate server protocol, while also ensuring that M maintains interactive DP to provide robustness against a malicious server.

3.1.1 Strawman Approaches. It might seem that a straightforward implementation of DP-FTRL within a TEE could serve as a solution, as the TEE can act as a trusted third party to correctly manage the state. However, naive implementations fail to provide both liveness and the interactive DP guarantee. We illustrate this by presenting two strawman approaches.

With Sealing. Since an enclave is volatile, its state is lost when it crashes. To prevent this, the server needs to *seal* the enclave's state after each round is completed, allowing the state to be recovered even after a crash. However, this approach introduces a fatal vulnerability: it does not provide interactive DP because the server can clone (i.e., fork) the sealed enclave state at will [82]. If an adversary can fork the process, they can observe outcomes related to step i 's noise more than once using the Sybil clients. This leads to a critical privacy breach even if the i th input does not include the target client's values. This is because the noise added at step i is correlated with the noise from earlier steps. This correlation can leak information about the noise at earlier steps and, consequently, the earlier private inputs that include the target value.

Even if the sequence of noise values is predetermined (e.g., from a fixed seed), a malicious server could perform the following forking

⁵Not all elements of Z need to be sampled initially; they can be incrementally sampled as required.

attack. Suppose the server forks the computation at step i . It first executes step i on one fork with a cohort that includes a target client's value alongside values from corrupted clients (e.g., all zeros). It then executes step i on a second fork using a different cohort containing only corrupted clients (all zeros). By comparing the outputs from these two forks, the server can isolate the contribution of the target client. The difference between the outputs effectively removes the common, correlated noise component, thus breaking the privacy guarantee.

While combining a TEE's sealing capabilities with a trusted hardware like TPM can enforce state integrity [75], this integration creates its own security challenges. Securely linking the two trusted components expands the attack surface and opens the door to man-in-the-middle attacks. A prime example is the integration of Intel SGX with a TPM, as explored by Strackx et al. [75]. However, this method is vulnerable to the Cuckoo attack [68] as mentioned in Introduction.

Without Sealing. To circumvent forking attacks, one might consider an approach without sealing, where the server restarts the entire process from its initial state if a crash occurs. However, this strategy faces a different, equally critical problem in a malicious server model.

The core issue is that honest clients cannot distinguish whether the server has genuinely crashed or is merely *pretending* to have crashed. A malicious server can declare a crash to honest clients while secretly continuing to run the remaining planned iterations with only the corrupted clients it controls. This raises the same issue as the case with sealing. That is, because DP-FTRL uses correlated noise, running these extra, secret iterations leaks further information about the inputs used in the earlier iterations that involved honest clients.

Consequently, the privacy analysis must account for the worst-case leakage. This forces a conservative accounting of the privacy budget for the maximum number of iterations the server could potentially run with its corrupted clients after the apparent stop. This results in an inherent and significant privacy loss. Furthermore, DP-FTRL's correlated noise mechanism is optimized based on the total number of iterations planned in advance. An early stop and restart breaks the optimality of this noise structure, potentially leading to a much worse total privacy loss than originally intended. Even using SMR [4, 40, 57] without sealing suffers from the same problem if a majority of TEEs experience a fault (i.e., disaster).

3.1.2 Ideal Functionality. As discussed in Section 2.4, we adopt a simulation-based proof framework to prove interactive DP of our system. To do so, we first define an ideal functionality that acts as a trusted party and is designed to avoid the issues of the strawman approaches. We then show that our system emulates this ideal functionality with some simulator.

Ideal Functionality. Algorithm 1 outlines the pseudocode for the trusted party in our ideal model. Two main differences exist when compared to a standard DP-FTRL system, both tailored to the practical needs of a real model implementation. First, in Lines 2-5, the ideal functionality allows for a small probability of failure that is independent of D , indicating a small but non-zero of catastrophic privacy leakage, contributing to δ in (ϵ, δ) -DP. This adjustment

Algorithm 1 Ideal Functionality By a Trusted Party

Require: Participation schema Π , the number of rounds n_{round} , matrix $C \in \mathbb{R}^{n_{\text{round}} \times n_{\text{round}}}$, clipping parameter $\zeta \in \mathbb{R}$, the number of clients n , indices of corrupted clients $C_{\text{corrupted}}$.

- 1: Receive D from clients.
- 2: $l \leftarrow$ a sample from a Bernoulli distribution with success probability δ_{privacy} .
- 3: **if** $l = 1$ **then**
- 4: **send** D to Sim. // Leakage
- 5: **else**
- 6: Initialize the participation history H .
- 7: Initialize model parameters $\theta_0 \in \mathbb{R}^d$.
- 8: Make the matrix with all zeros $\theta_{1:n_{\text{round}}} \in \mathbb{R}^{n_{\text{round}} \times d}$.
- 9: Sample $Z[i, j] \sim N(0, \sigma^2)$ for $i \in [n_{\text{round}}]$, $j \in [d]$.
- 10: **for** $i \in [n_{\text{round}}]$ **do**
- 11: Receive $C_{\text{cohort}} \subseteq [n]$, $k \in [n_{\text{round}}]$, $\tilde{\theta} \in \mathbb{R}^d$ from Sim.
- 12: Restart from LINE 10 if any input is invalid. // Valid if $C_{\text{cohort}} \subseteq f_{\text{qualify}}(\Pi, H, k)$ and if $k \in [n_{\text{round}}]$ is not in H yet.
- 13: Update H with C_{cohort} and k .
- 14: Compute $g_j = \text{clip}(\nabla_{\theta} \text{loss}(X_j, \tilde{\theta}), \zeta)$ for $j \in C_{\text{cohort}}$ where $\text{clip}(d, \zeta) = \min(1, \zeta / |d|)d$
- 15: For $j \in C_{\text{cohort}} \cap C_{\text{corrupted}}$, set g_j as received from Sim
- 16: Set $\tilde{\theta}_k := \sum_{j \in C_{\text{cohort}}} g_j + \zeta(C^{-1}Z)[k, :]$.
- 17: **send** $\tilde{\theta}_{1:i}$ to Sim // Leakage.
- 18: **end for**
- 19: **end if**

enables the real model to be designed without requiring exhaustive client interactions for each update (i.e., probabilistic relaxation of linearizability), as discussed in Section 2.3.1. Second, in Line 11, the simulator (i.e., an adversary) can modify indices during secure aggregation. This is due to the non-blocking attribute of linearizability [38], which permits concurrent execution. Also, the adversary can modify the values received from the corrupted clients (Line 15).

Despite these variations, the following theorem confirms that the original DP guarantee of DP-FTRL is preserved:

THEOREM 3.1. *Assume that if SIM sets k to i , C_{cohort} to be a subset of $f_{\text{qualify}}(\Pi, H, k)$, $\tilde{\theta}$ to $\tilde{\theta}_{i-1}$, and g_j as defined in Algorithm 1 (i.e., the original DP-FTRL), the view of SIM satisfies $(\epsilon, \delta + \delta_{\text{privacy}})$ -DP. Then, the view of any other SIM' satisfies $(\epsilon, \delta + \delta_{\text{privacy}})$ -DP.*

The proof is detailed in Appendix D.1. Therefore, our goal is to create a concurrent system capable of emulating the ideal functionality within our threat model, whose δ_{privacy} is a sufficiently small probability. In parallel, the system should be designed to offer availability with small communication and computation overhead for clients.

3.2 Overview of Our Approach

We model CONCOMP(M_1, M_2, \dots, M_n) as a distributed concurrent system (see Section 2.3) to formulate the problem. In this model, the concurrent system maintains a state object q , which includes iteration number i , participation history H and noise matrix Z explained in Section 2.5. This state is shared among all clients,

Algorithm 2 The Process

- 1: The server initializes an enclave or replicates the enclave with q , C_{cohort} , and i .
- 2: The enclave sends evidence to the clients.
- 3: (Audit request) Clients verify the integrity and linearizability using the evidence and send their signatures to the enclave.
- 4: **if** the enclave was initialized **then** the enclave validates these and, if it is the initial enclave, the enclave makes $q.Z$ is sealed, and stops this process.
- 5: **end if**
- 6: The enclave verifies C_{cohort} and sends evidence to the C_{cohort} . The server updates $q.H$ with C_{cohort} and i .
- 7: (SecAgg request) Clients in C_{cohort} verify the evidence and send encrypted data with MAC to the enclave.
- 8: The enclave verifies these and outputs the result of $\text{SECAGG}_{q.Z}$.

and processes are concurrently invoked by the server. The process with $C_{\text{cohort}} \subseteq [n]$ and $\text{args}_{\text{secagg}}$ (i.e., indices of participating clients and settings such as current model parameters $\tilde{\theta}$ respectively) increments $q.i$, updates $q.H$ with $C_{\text{cohort}} \subseteq [n]$ and $q.i$, and outputs the result of stateful aggregation $\text{SECAGG}_{q.Z}$ only if $C_{\text{cohort}} \subseteq f_{\text{qualify}}(\Pi, q.H, q.i)$, denoted by $\langle q \text{ update}(C_{\text{cohort}}, q.i, \text{args}_{\text{secagg}}) \text{ A } \rangle \text{OK}(\text{SecAgg}_{q.Z}(C_{\text{cohort}}, q.i, \text{args}_{\text{secagg}})) \text{ A } \rangle$. $\text{SECAGG}_{q.Z}$ is a function that computes the summation with noise $\zeta(C^{-1}(q.Z))[q.i, :]$ for C_{cohort} using the settings specified by $\text{args}_{\text{secagg}}$.

Assume that given Π , a system executes the process update with integrity and linearizability. That is, due to the integrity, each process does not break the participation schema Π and get the correct result of $\text{SECAGG}_{q.Z}(C_{\text{cohort}}, q.i, \text{args}_{\text{secagg}})$. Due to the linearizability, the result of the processes is equivalent to a sequential processes that respect real-time precedence ordering (i.e., the order of i). That is, an adversary cannot get multiple $\tilde{\theta}_i$ with the same i and cannot define i th C_{cohort} and $\text{args}_{\text{secagg}}$ after knowing $\tilde{\theta}_k$ for $i \leq k$. Thus, by achieving integrity and linearizability, the system emulates the ideal functionality in Algorithm 1 with a simulator. Therefore, integrity and linearizability of the update process are sufficient conditions for the system to achieve interactive DP.

Here, we present the high-level concept of our approach in the real world to achieve integrity and linearizability. The pseudocode for the process is provided in Algorithm 2. The server issues two types of the requests to clients: audit request and secure aggregation request. Intuitively, before outputting the result of $\text{SECAGG}_{q.Z}$ —which involves private data and may thus compromise privacy—the server must provide proof of linearizability and integrity with enclaves in the audit request. Clients must verify this during the audit request, and only upon successful verification can the server complete the process to obtain the result. Due to the linearizability and integrity checks enforced by clients, a malicious server cannot access any information beyond the output.

For verification purposes, a client requires remote attestation and validation of certain values. The number of clients needed for auditing is modest (e.g., 129 for $\beta, \gamma = 0.1$) as demonstrated in Section 6.1. Hence, the additional computational and communication

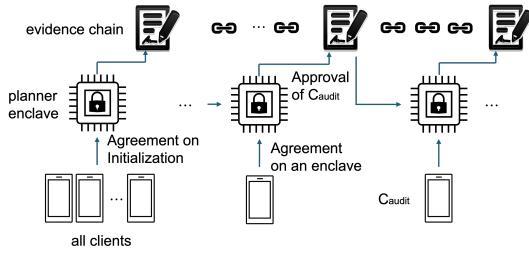


Figure 2: Overview of the core idea behind planner enclaves.

overhead is both low and constant. Moreover, since a different enclave is used for each process, it allows for concurrent execution, enhancing scalability. Also, even if an enclave fails, the entire DP-FTRL execution does not halt because the server can make a new process with a new enclave, thereby providing liveness.

This is achieved by enforcing integrity and linearizability through TEEs combined with client auditing. The challenge lies in proving integrity and linearizability to the clients, a topic that is thoroughly explained in Section 4.

4 Design of Planner

In this section, we explain how to ensure integrity and linearizability in the presence of enclaves and client auditing. We start by detailing a system that achieves integrity and linearizability under the assumption that all clients are honest and available (i.e., $(\gamma = 0, \kappa = 1, \beta = 0)$), to highlight the core of our proposed method. Then, in Section 4.2, we propose modifications to the system to accommodate scenarios where $(\gamma > 0, \kappa \neq 1, \beta > 0)$.

4.1 Design Under $(\gamma = 0, \kappa = 1, \beta = 0)$

Figure 2 illustrates the basic concept of the system. We employ an enclave that deploys our code, which we call a planner enclave. The planner enclave initially secures consensus from all clients on the shared object q and seals (see Section 2.1) the states. In subsequent processes, replicated planner enclaves load the shared object q from an *evidence chain*, after which a client audits whether the correct shared object was loaded. If verified by the client, the planner enclave continues the process and generates evidence designating the next client to audit the subsequent planner enclave. This iterative process ensures linearizability, as planner enclaves always load the latest state due to designated client audits (a roll-backed enclave is rejected), and integrity is maintained through the TEE characteristics.

4.1.1 Server-side Algorithm. Algorithm 3 provides pseudocode for the server and planner enclave protocols. At each process (i.e., iteration), the server-side algorithm establishes proofs of linearizability and integrity in Lines 11-16 (i.e., Lines 1-3 of Algorithm 2) to conduct secure aggregation in Line 19 (i.e., Lines 4 and 6 of Algorithm 2).

Here, $f(\text{args}_{\text{selection}})$ is a function that outputs $\text{args}_{\text{selection}} \subseteq [n]$ when $|\text{args}_{\text{selection}}| \geq 1$; otherwise, it aborts, and we set $\tau = |\text{args}_{\text{selection}}|$. We use this function as a placeholder for now and will change it later to accommodate the case where $\beta > 0$ and $\gamma > 0$ (i.e., Section 4.2).

Algorithm 3 Algorithm for the Server-side and the Planner Enclave

- 1: **Initialization**
- 2: The server initializes a planner enclave with the public keys of all clients $C_{\text{all}} = [n]$, and sets $\text{args}_{\text{selection}} \subseteq C_{\text{all}}$.
- 3: The enclave generates random seeds for Z and a nonce as ID.
- 4: **if** the enclave verifies agreement on C_{all} **then**
- 5: The enclave computes $C_{\text{audit}} = f_{\text{select}}(\text{args}_{\text{selection}})$.
- 6: The enclave seals the states.
- 7: The enclave generates evidence including C_{audit} and ID.
- 8: **end if**
- 9: The server initializes an evidence chain (EC) with the generated evidence.
- 10: **Iteration (concurrent)**
- 11: The server selects a cohort $C_{\text{cohort}} \subseteq C_{\text{all}}$.
- 12: The server replicates the sealed planner enclave and inputs EC , C_{cohort} , $\text{args}_{\text{secagg}}$, and $\text{args}_{\text{selection}}$.
- 13: **if** the enclave verifies EC , C_{cohort} , and $\text{args}_{\text{selection}}$ **then**
- 14: The enclave retrieves C_{audit} from the latest evidence.
- 15: The enclave computes $C_{\text{next}} = f_{\text{select}}(\text{args}_{\text{selection}})$.
- 16: **if** the enclave verifies agreement on τ clients in C_{audit} **then**
- 17: The enclave generates evidence including C_{next} as auditors, C_{cohort} , the hash of $\text{args}_{\text{secagg}}$, and the digest of EC .
- 18: The server appends the new evidence to the EC .
- 19: The enclave does SECAGG_Z on C_{cohort} with $\text{args}_{\text{secagg}}$.
- 20: The server updates the global model.
- 21: **end if**
- 22: **end if**

Planner enclaves require the server to pass three types of verifications to ensure integrity and linearizability, which are as follows:

- (1) *Initialization* (Line 4) verifies that all clients have reached consensus on a shared object q . This ensures that clients can deny planner enclaves that load a different shared object q' for future auditing.
- (2) *Approval of auditors* C_{audit} (Line 13) verifies that clients C_{audit} have been approved as auditors by a planner enclave. This means auditors can verify whether a planner enclave loads the correct shared object and will correctly perform secure aggregation (i.e., integrity).
- (3) *Agreement on a planner enclave* (Line 16) verifies that the enclave is the planner enclave agreed upon by auditors C_{audit} . This ensures the enclave maintains integrity and is the sole enclave loading the current shared object (i.e., linearizability).

Next, we discuss the evidence chain that manages a shared object q , followed by detailed explanations of the three verifications, illustrated in Figure 3.

Evidence Chain. We introduce the evidence chain as the state management method for our concurrent system. It consists of a sequence of evidence pieces generated by planner enclaves, with evidences linked via hashes like blockchain [64] (i.e., evidence in the evidence chain has the hash of the previous evidence). The digest of the evidence chain is the hash of the latest evidence. The ID of the evidence chain is the ID included in the initial evidence.

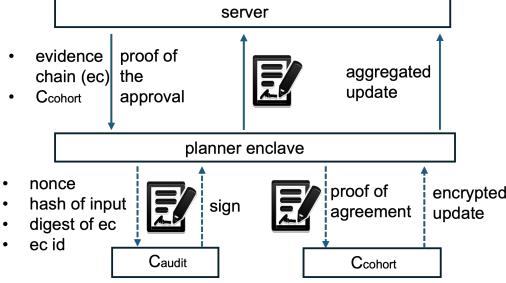


Figure 3: The flow of evidence-related proofs by the server. A dotted line indicates indirect communication via the server.

Each piece of evidence includes updates to $q.H$ (in our context, C_{cohort}, i).

Verification of the evidence chain involves confirming that the latest evidence remains untampered (verified via remote attestation) and that the hash chain is valid.

When a planner enclave successfully verifies an evidence chain and is agreed upon by the auditors C_{audit} (explained later) documented in the latest evidence, it implies that the enclave reads the latest state by sequentially updating $q.H$ with the evidence chain. This occurs because if consensus on C_{audit} is reached, it is concluded that the current auditors are the most recently approved.

Approval of Auditors (verified in Line 13). "Approval of auditors" is a process where a planner enclave designates clients C_{audit} as auditors just once. As shown in Figure 3, the server validates this approval through evidence generated by a previous planner enclave (i.e., the latest evidence of the evidence chain). A planner enclave is designed to designate clients and produce evidence just once only after it has obtained consensus from the designated auditors. Therefore, verification of code via remote attestation establishes that the clients listed as auditors were indeed approved.

Agreement on a Planner Enclave (verified in Line 16). "Agreement on a planner enclave" is the consensus process where auditors designate a specific planner enclave. This implies:

- Any planner enclave not agreed upon by auditors is invalid.
- The planner enclave correctly loads the shared object and will perform secure aggregation without disrupting the participation schema.

As illustrated in Figure 3, the server first generates evidence using a planner enclave, containing a nonce, the hash of inputs (i.e., the hash of C_{cohort} and $\text{args}_{\text{secagg}}$), the digest of the evidence chain, and the ID of the evidence chain. This evidence is provided to the approved auditors retrieved from the evidence chain (i.e., C_{audit}). Approved auditors perform remote attestation to verify the integrity of the evidence and the code. If auditors verify the ID and that the digest has not been seen before (to prevent replay attacks), they sign the nonce in response. The server sends all signatures of C_{audit} (i.e., τ signatures) to the planner enclave to generate further evidence (Lines 17 and 18), which proves the "agreement on a planner enclave".

Initialization (verified in Line 4). "Initialization" is a process by which a planner enclave designates the shared object q to all clients. Specifically, it implies:

- Consensus on the shared object and initial auditors by all clients.
- Correct loading of the public key list by the planner enclave.

The process is similar to the "agreement on a planner enclave," though the evidence now includes the digest of the public key list, nonce, and the ID of the enclave. Each client checks the digest against their known hash of the public key list through PKI, as described in Section 2.2, before registering the nonce as the evidence chain ID and responding with a signature.

Note that "initialization" means all clients agree on $q.Z$ because a replicated planner enclave possessing the ID as the evidence chain ID also possesses the random seeds for Z .

Secure Stateful Aggregation. Here, we explain the secure aggregation of Line 19 (i.e., $\text{SECAGG}_{q,Z}$). Due to three prior verifications conducted before Line 19 of Algorithm 3, the planner enclave loads the shared object correctly and thus possesses the correct Z , i , and C_{cohort} , where i is the number of entries in the evidence chain. This allows precise computation of Line 16 of Algorithm 1.

We primarily adhere to the secure aggregation method outlined by Huba et al. [41]. However, as discussed in Section 2.1.3, given that the original formulation didn't include noise, additional modifications are required to incorporate stateful noise. A simplistic extension suffices; while carrying out secure aggregation method from Huba et al., the enclave adds $\zeta(C^{-1}Z)[i, :]$. Then, aggregated data with the noise is decrypted only when data from C_{cohort} is collected with verified MAC.

4.1.2 Client-side Algorithm. The client-side algorithm is depicted in Algorithm 4. The client-side algorithm is stateful, storing `evidence_chain_id` and `signed_digests`. It employs two functions: `AUDIT` and `SECUREAGGREGATION`.

`AUDIT` is involved in Lines 4 and 16 of Algorithm 3 (i.e., *Initialization* and *Agreement on a planner enclave*). Acting as an auditor, a client verifies whether the planner enclave establishes or reads the shared object. During Line 3 of Algorithm 4, a client performs remote attestations to verify that the evidence originated from a planner enclave. The evidence may encompass the ID of the evidence chain, a nonce, and the digest of the evidence chain. The client verifies whether this matches the stored `evidence_chain_id` (Line 10). If not previously registered (i.e., `evidence_chain_id=None`), it is stored accordingly (Line 8). Then, after ensuring that `signed_digests` does not contain the digest to prevent replay attacks (Lines 13), the client responds with the signature to the nonce.

`SECUREAGGREGATION` is initiated in Line 19 of Algorithm 3. First, the client performs remote attestation to verify the integrity of the evidence (i.e., to ensure that it is generated by the enclave that has passed the three verifications). The client then extracts the peer's public key for the Diffie-Hellman (DH) key exchange from the evidence. Using this key, the client sends the update encrypted by a symmetric key, the symmetric key encrypted with the DH key, and the MAC generated from the DH key (see Section 2.1.3 for more details).

Algorithm 4 Client-side Algorithm

```

1: Initialize evidence_chain_id ← None and signed_digests ← ∅
2: procedure AUDIT(evidence)
3:   If REMOTEATTESTATION(evidence) fails Then abort
4:   Extract nonce, digest_of_evidence_chain, evidence_chain_id from evidence
5:   if self.evidence_chain_id = None then
6:     self.evidence_chain_id ← evidence_chain_id
7:   end if
8:   If evidence_chain_id ≠ self.evidence_chain_id Then abort
9:   If digest_of_evidence_chain ∈ self.signed_digests Then
10:    abort
11:    APPEND(self.signed_digests, digest_of_evidence_chain)
12:  return SIGN(nonce)
13: end procedure
14: procedure SECUREAGGREGATION(evidence)
15:   If REMOTEATTESTATION(evidence) fails Then abort
16:   Extract nonce, argssecagg, ec_peer_pub_key from evidence
17:   update ← COMPUTELOCALUPDATE(argssecagg)
18:   encrypted_update, encrypted_decryption_key,
19:     ec_pub_key, MAC ←
20:     MAKESECUREAGGREGATIONMESSAGE(update, nonce,
21:       ec_peer_pub_key)
22:   return encrypted_update, encrypted_decryption_key,
23:     ec_pub_key, MAC
24: end procedure

```

4.1.3 Property of The System. In our concurrent system, each process involves an object q virtually shared with all clients. The singular operation applicable to this object is "update" with secure aggregation output, denoted by

$$\langle q \text{ update}(C_{\text{cohort}}, i) \text{ A} \rangle \langle \text{OK}(\text{SECAGG}_{q,Z}(C_{\text{cohort}}, i, \text{aux}_{\text{secagg}})), \text{ A} \rangle.$$

While producing $\text{SECAGG}_{q,Z}(C_{\text{cohort}}, i, \text{aux}_{\text{secagg}})$ as output, the process updates H with C_{cohort} and i . This means that Algorithm 3 concurrently performs the process described in Algorithm 2.

THEOREM 4.1. *Algorithms 3 (server-side) and 4 (client-side) achieve both integrity and linearizability with respect to the process described in Algorithm 2 when $\gamma = 0$.*

The proof is presented in Appendix D.2. Also, we formally verified this by the Tamarin prover. See Appendix E for details.

4.1.4 Vulnerability of the System Under ($\gamma > 0, \beta > 0$). Theorem 4.1 demonstrates integrity and linearizability, but its assumptions overlook any client dropouts or dishonest behavior (i.e., $(\beta = 0, \gamma = 0)$). Here, we evaluate the more realistic scenario where $\beta, \gamma > 0$, and $\kappa \neq 1$, outlining the vulnerabilities introduced under these conditions.

Sybil Selection. The inability to differentiate between malicious and honest clients enables Sybil attacks, where Sybils break linearizability. A malicious server might select $\text{args}_{\text{selection}}$ composed entirely of corrupted clients, allowing corrupted auditors to deviate from the protocol, which leads to linearizability violations. Specifically, in Line 16 of Algorithm 3, corrupted auditors could agree on multiple enclaves, causing arbitrary state forks.

Interruptions Caused by Dropouts. In Line 16, the planner enclave requires the collection of signatures from all auditors C_{audit} to validate and proceed with secure aggregation, as $\tau = |C_{\text{audit}}|$. However, an auditor dropping out before signing disrupts the entire process outlined in Algorithm 3. The secure aggregation phase in Line 19 is similarly susceptible to disruptions.

4.2 Adaptation for ($\gamma > 0, \kappa \neq 1, \beta > 0$)

To address the aforementioned vulnerabilities, we propose necessary modifications to Algorithm 3.

4.2.1 Preventing Interruptions by Dropout. As outlined in Section 4.1.4, client dropouts can halt a process at Line 16 of Algorithm 3. To alleviate this issue, we modify the value of τ . As discussed in Section 4.1.1, the process of "agreement on a planner enclave" on C_{audit} must ensure that no alternative planner enclave gains agreement on C_{audit} . To meet this requirement, we can set $\tau > |C_{\text{audit}}|/2$, requiring agreement from the majority of C_{audit} . By setting a larger $n_{\text{audit}} = |C_{\text{audit}}|$ and a smaller τ , the probability of interruptions occurring can be minimized, as described in Section 5.2.2.

Interruptions may occur during the secure aggregation process (i.e., Line 19 of Algorithm 3) as well. However, these interruptions are not catastrophic since they occur post-Line 17, where evidence for the next process is generated. This is a feature of linearizability known as non-blocking characteristics [38]. The process, however, loses utility as it fails to output results. To tackle this issue, we can utilize an overselection strategy. Overselection involves selecting a large pool of candidate clients for secure aggregation, allowing available clients from these candidates to participate.

Remark. The overselection strategy could result in a disparity between C_{cohort} and the clients that actually participate, possibly causing data loss since participation schemes are based on C_{cohort} . For simplicity, this paper does not consider this utility loss. However, this problem is not insurmountable; it can be resolved by creating evidence that includes C_{part} at Line 19, where C_{part} is the set of clients that actually partake in secure aggregation. By adding this evidence to the evidence chain, linked to the evidence generated in Line 18, participation scheme verification can be employed to avert data loss while accommodating client dropouts.

4.2.2 Preventing Sybil Selection. Within Algorithm 3, corrupted clients can only influence the agreement parts at Lines 4 and 16 of Algorithm 3. These clients could potentially send signatures without conducting verification during AUDIT, permitting multiple enclaves to exist carrying the same shared object, disrupting linearizability. To counter this, it is crucial to establish that the majority of **honest** clients in the auditors have reached an agreement. However, since corrupted clients cannot be differentiated from honest ones, proving this becomes impossible when the number of corrupted clients exceeds the cohort size.

To avoid this issue, we restrict the server's ability to arbitrarily choose auditors. The server is allowed to choose a sufficient number of candidates for auditors, but the actual selection of auditors is done randomly by the enclave. Specifically, the enclave instead employs $f_{\text{selection}}(\text{args}_{\text{selection}})$, which is a randomized function that returns a random subset C_{audit} of $\text{args}_{\text{selection}}$ such that $|C_{\text{audit}}| =$

n_{audit} where $n_{\text{audit}} \in [n]$ is an adjustable parameter and aborts if $|\text{args}_{\text{selection}}| < n\kappa$.

Despite the random selection, auditors may still include some corrupted clients, making it impossible to unequivocally prove that a majority of **honest** clients are in agreement. Therefore, we set τ to be a value larger than $n_{\text{audit}}/2$, ensuring that the probability of honest clients being in the majority becomes sufficiently high. Note, however, that setting a higher τ increases the likelihood of interruptions, presenting a trade-off, which is optimized in Section 5.4.

Assuming the server sets $\text{args}_{\text{selection}}$ as the available clients $C_{\text{available}}$ (i.e., $|C_{\text{available}}| = n\kappa$ as per Section 2.2), and considering that up to a ratio of γ of all clients may be corrupt, the probability of having a certain number of corrupted clients in C_{audit} can be calculated. By combining this with τ and n_{audit} , the probability of a Sybil attack occurring can be estimated, and this is the probability that breaks integrity and linearizability.

THEOREM 4.2. *When $\gamma > 0$, Algorithm 3 maintains linearizability and integrity with a probability of $1 - \delta_{\text{privacy}} =$*

$$1 - \left(1 - \sum_{i=1}^{2(n_{\text{audit}} - \tau)} \binom{n\gamma}{2\tau - n_{\text{audit}} + i} \binom{n(\kappa - \gamma)}{2n_{\text{audit}} - 2\tau - i} / \binom{\kappa n}{n_{\text{audit}}} \right)^{n_{\text{round}}}.$$

In other words, with probability $1 - \delta_{\text{privacy}}$, our protocol maintains Theorem 4.1 that is formally verified. The proof can be found in Appendix D.3. The relationship between δ_{privacy} and parameters is shown in 6.1.

5 Analysis of Our System

In this section, we analyze our system from the perspectives of privacy, correctness, liveness, and scalability.

5.1 Privacy and Correctness

We first present the main theorem of this paper, which clarifies the privacy and utility aspects of our system.

THEOREM 5.1. *Let M_i be Algorithm 4 for $i \in [n]$. The interactive protocol $\text{CONCOMP}(M_1, \dots, M_n)$, when used with any server protocol, emulates the ideal model (i.e., Algorithm 1) with some simulator.*

The proof is presented in Appendix D.4. From this theorem, the server protocol can derive the output of MF-DP-FTRL from $\text{CONCOMP}(M_1, \dots, M_n)$, since Algorithm 1 reflects the MF-DP-FTRL approach by setting $k = i$ and $\hat{\theta} = \hat{\theta}_i$. $M = \text{CONCOMP}(M_1, \dots, M_n)$ meets the criteria of $(\epsilon, \delta + \delta_{\text{privacy}})$ -interactive DP. This is because any server protocol interacting with M fulfills $(\epsilon, \delta + \delta_{\text{privacy}})$ -DP, given that the corresponding simulator itself adheres to $(\epsilon, \delta + \delta_{\text{privacy}})$ -DP, as concluded in Theorem 3.1.

5.2 Liveness

In our context, liveness refers to the capability to complete a process (i.e., Algorithm 2). We analyze three potential issues that might stop the process: crashes, interruptions, and attacks.

5.2.1 Crashes. If an enclave crashes for any reason, there is no straightforward way to recover the process. This is because allowing such recovery would enable an adversary to fork the enclave's state. Maintaining linearizability would require verification, which is not possible. However, it is important to note that while a single

enclave crash may disrupt a process (i.e., Algorithm 2), it does not necessarily mean that the entire system (i.e., Algorithm 3) will come to a halt. As described in Appendix B, even if an enclave crashes, the system can initiate another process to recover the state. This is an advantage of our system compared to systems relying on SMR that cannot securely recover from disasters [4, 40].

5.2.2 Interruptions by Dropouts. In our system, auditors may drop out, which could potentially cause the system to halt as discussed in 4.1.4. Here, we analyze the probability associated with client dropouts.

PROPOSITION 5.2. *The probability $\delta_{\text{interrupt}}$ of interruption of auditing is $1 - \left(1 - \sum_{i=1}^{\tau} \binom{n\kappa\beta}{n_{\text{audit}} - \tau + i} \binom{n\kappa(1-\beta)}{\tau - i} / \binom{n\kappa}{n_{\text{audit}}} \right)^{n_{\text{round}}}.$*

The proof is presented in Appendix D.5. As shown in Section 6.1, the probability can be small if we set large n_{audit} .

5.2.3 Attacks. We consider scenarios where a malicious client attempts to disrupt the process. In our system, a malicious client might send invalid values. However, this does not halt the system since the enclave can disregard invalid inputs, similar to handling dropouts. Given our system's resilience to dropouts, a single malicious client lacks the capability to stop the process.

5.3 Scalability

We evaluate the scalability of our concurrent system from two perspectives. First, we examine server-side scalability concerning throughput [65]. Second, we analyze client-side costs, considering that clients are lightweight and can drop out.

We evaluate the additional communication and computation costs incurred by clients due to our planner. In both AUDIT and SECUREAGGREGATION, the communication cost is constant, influenced only by security parameters such as the size of the signature. This is discussed in detail in Section 6.2. The computation cost is also constant; the encryption method used is part of the standard communication process and does not necessitate additional computational resources. Although the frequency of AUDIT depends on parameters like β , κ , and γ , it remains low, as demonstrated in Section 6.1. On the contrary, if n becomes large, the frequency of AUDIT decreases. Consequently, the costs incurred by clients remain constant even as the system scales.

5.4 Parameter Selection

In our system, we have two adjustable parameters: τ and n_{audit} . Here, we discuss how to determine these parameters. The goal is to minimize n_{audit} to reduce communication costs (i.e., frequency of AUDIT) while ensuring $\delta_{\text{interrupt}} \leq p_1 \in [0, 1]$ and $\delta_{\text{privacy}} \leq p_2 \in [0, 1]$ for liveness and security requirements. This can be formulated as the following optimization problem:

$$\min_{\tau, n_{\text{audit}}} n_{\text{audit}} \text{ s.t. } \delta_{\text{privacy}}(n_{\text{audit}}, \tau) \leq p_1 \text{ and } \delta_{\text{interrupt}}(n_{\text{audit}}, \tau) \leq p_2,$$

where $\delta_{\text{privacy}}(n_{\text{audit}}, \tau)$ and $\delta_{\text{interrupt}}(n_{\text{audit}}, \tau)$ represent the probabilities of $\delta_{\text{interrupt}}$ and δ_{privacy} when τ and n_{audit} are provided. Since $n_{\text{audit}} \leq n$ and $\tau \leq n_{\text{audit}}$ are integers, and the probabilities can be calculated from Proposition 5.2 and Theorem 4.2, this problem is solvable using binary search.

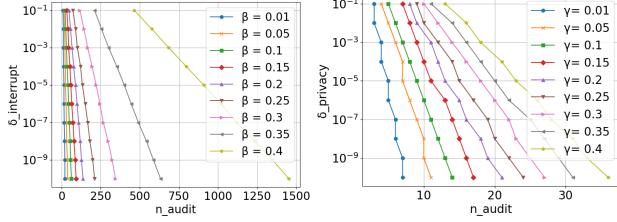


Figure 4: The trade-off between communication cost and privacy for various γ with $\beta = 0$ and $\kappa = 1$ (left) and various γ with $\beta = 0$ and $\kappa = 1$ (right).

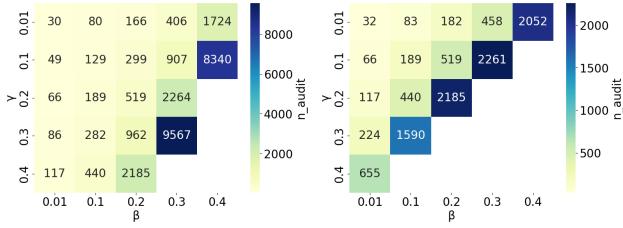


Figure 5: Communication cost for various settings of (β, γ) with the constraint of $(\delta_{\text{privacy}}, \delta_{\text{interrupt}}) = (10^{-8}, 10^{-8})$, $\kappa = 1$ (left), and $\kappa = 0.5$ (right). Results larger than 10000 are omitted.

6 Experiments

In this section, we first examine the effects of various parameters. Then, we present the TCB size and actual communication overhead associated with the implementation.

6.1 Communication, Liveness, and Privacy Trade-offs

In this section, we analyze the trade-offs among communication cost, liveness, and privacy. Here, n_{audit} represents the minimum number of auditors required for each process derived in Section 5.4, which in turn correlates with the entire communication cost. Liveness is quantified by $\delta_{\text{interrupt}}$, as described in Proposition 5.2; it represents the probability of system failure in conducting DP-FTRL. Privacy is measured by δ_{privacy} , as defined in Theorem 4.2; it denotes the additional overhead to δ for approximate DP. By varying n_{audit} , we explore the balance among these factors. Here, we set $n_{\text{round}} = 10000$ and $n = 10^7$, and the server is required to set $|\text{args}_{\text{selection}}|$ as the available clients.

Initially, we illustrate the balance between communication cost and liveness and between communication cost and privacy in Figure 4. For these analyses, we fix $\gamma = 0, \kappa = 1$ or $\beta = 0, \kappa = 1$ to focus on liveness or privacy, respectively. The results indicate that by slightly increasing n_{audit} , the system can significantly lower both $\delta_{\text{interrupt}}$ and δ_{privacy} , implying a communication overhead to achieve sufficient liveness and privacy.

Next, we evaluate the communication cost across various settings of (β, γ) under the constraints $(\delta_{\text{privacy}}, \delta_{\text{interrupt}}) = (10^{-8}, 10^{-8})$ with $\kappa = 1$ or $\kappa = 0.5$, as depicted in Figure 5. The plots illustrate the communication cost as γ and β vary. Compared to earlier scenarios where either γ or β is zero, achieving robustness requires greater

communication costs. This is because robustness for liveness requires a low value of τ , while robustness for privacy necessitates a high value. To increase the value of feasible τ s to simultaneously satisfy both requirements, n_{audit} must be increased.

The value of κ is also crucial as it quickly increases the value of n_{audit} . This occurs because the candidates for auditors are selected by the server (i.e., adversary), so the ratio of corrupted clients among the candidates rises quickly if the number of candidates (i.e., n_k) is small. Even when κ is small, if β is also low, our framework can still function effectively with high γ and a small n_{audit} .

We observed that the value of n does not significantly impact the results. Thus, by increasing n , the communication cost for each client can be reduced because it decreases the probability of being chosen as auditors.

6.2 Implementations

We implemented the planner enclave using Intel SGX with the OpenEnclave SDK⁶. For both signing and key exchange, we use ECDSA and ECDH with P-256, respectively, as implemented by OpenSSL. In this section, we examine the implementation from two angles: the TCB size and the message size required for a client.

TCB size. We assess the TCB size without the secure aggregation component to compare it with systems focused on achieving linearizability, such as ROTE [57] and Nimble [4], which, to the best of our knowledge, has the smallest TCB size for this purpose. Our TCB size is 0.7K excluding libraries, which is more compact than ROTE’s 1.1K and Nimble’s 2.3K as reported in the paper, as our approach relies on client auditing rather than SMR.

Communication Overhead. We evaluated the additional communication overhead involved with AUDIT and SECUREAGGREGATION according to the client-side algorithm. For AUDIT, the evidence size is 5038 bytes, while for SECUREAGGREGATION, it is 5194 bytes. In terms of responses, AUDIT sends 64 bytes, and SECUREAGGREGATION sends 144 bytes. Note that we intentionally excluded the size of model updates to concentrate solely on our additional communication costs. These communication sizes are fixed and do not vary with FL parameters such as n and d .

7 Conclusion

Our paper considers malicious settings in private FL to bridge the gap in privacy principles concerning transparency and verifiability [25]. Our approach achieves this using interactive DP, relying exclusively on TEEs and thereby avoiding additional trusted hardware such as a TPM [75], which is vulnerable to the Cuckoo attack [68]. Also, our design follows the fundamental principles of TEE applications [4, 6]; that is, it has a small TCB size. Furthermore, our system is resilient to disasters [4]—meaning it is capable of recovery—and requires only a single TEE-enabled server, thereby reducing management costs. As for future work, a key direction is to investigate the trade-off between our approach of minimizing trust in the server (i.e., bringing it closer to a zero-trust model) and the implementation costs discussed in Appendix A.2. Exploring simpler client auditing mechanisms with appropriate cost of trust will be crucial for broader practical deployment.

⁶<https://github.com/openenclave/openenclave>

Acknowledgments

This research received no specific grant from any funding agency in the public, commercial, or not-for-profit sectors. The authors used ChatGPT4o to revise the texts throughout this paper to correct any typos, grammatical errors, and awkward phrasing.

References

- [1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 308–318.
- [2] Naman Agarwal, Peter Kairouz, and Ziyu Liu. 2021. The skellam mechanism for differentially private federated learning. *Advances in Neural Information Processing Systems* 34 (2021), 5052–5064.
- [3] AMD. 2016. AMD Secure Encrypted Virtualization (SEV). <https://www.amd.com/en/developer/sev.html> (2016).
- [4] Sebastian Angel, Aditya Basu, Weidong Cui, Trent Jaeger, Stella Lau, Srinath Setty, and Sudheesh Singanamalla. 2023. Nimble: Rollback protection for confidential cloud services. In *17th USENIX Symposium on Operating Systems Design and Implementation (OSDI 23)*. 193–208.
- [5] Apple. 2024. *Private Cloud Compute: A new frontier for AI privacy in the cloud*, <https://security.apple.com/blog/private-cloud-compute/>. Technical Report. Apple.
- [6] Sergei Arnaudov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O’keeffe, Mark L Stillwell, et al. 2016. {SCONE}: Secure linux containers with intel {SGX}. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 689–703.
- [7] Marshall Ball, James Bell-Clark, Adria Gascon, Peter Kairouz, Sewoong Oh, and Zhiye Xie. 2024. Secure Stateful Aggregation: A Practical Protocol with Applications in Differentially-Private Federated Learning. *arXiv preprint arXiv:2410.11368* (2024).
- [8] Borja Balle, James Bell, Adria Gascón, and Kobbi Nissim. 2020. Private summation in the multi-message shuffle model. In *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*. 657–676.
- [9] Amos Beimel, Kobbi Nissim, and Eran Omri. 2008. Distributed private data analysis: Simultaneously solving how and what. In *Advances in Cryptology—CRYPTO 2008: 28th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 17–21, 2008. Proceedings 28*. Springer, 451–468.
- [10] James Henry Bell, Kallista Bonawitz, Adrià Gascón, Tancrède Lepoint, and Mariana Raykova. 2020. Secure single-server aggregation with (poly) logarithmic overhead. In *Proceedings of the 2020 ACM SIGSAC conference on computer and communications security*. 1253–1269.
- [11] Pramod Bhatotia, Markulf Kohlweiss, Lorenzo Martinico, and Yiannis Tselekounis. 2021. Steel: composable hardware-based stateful and randomised functional encryption. In *IACR International Conference on Public-Key Cryptography*. Springer, 709–736.
- [12] Alexander Bienstock, Antigoni Polychroniadi, and Ujjwal Kumar. 2024. DMM: Distributed Matrix Mechanism for Differentially-Private Federated Learning using Packed Secret Sharing. In *International Workshop on Federated Foundation Models in Conjunction with NeurIPS*.
- [13] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical secure aggregation for privacy-preserving machine learning. In *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 1175–1191.
- [14] Kallista Bonawitz, Peter Kairouz, Brendan McMahan, and Daniel Ramage. 2021. Federated learning and privacy: Building privacy-preserving systems for machine learning and data science on decentralized data. *Queue* 19, 5 (2021), 87–114.
- [15] Lennart Braun, Adrià Gascón, Mariana Raykova, Philipp Schoppmann, and Karn Seth. 2024. Malicious security for sparse private histograms. *Cryptography ePrint Archive* (2024).
- [16] Javad Ghareh Chamani and Dimitrios Papadopoulos. 2020. Mitigating leakage in federated learning with trusted hardware. In *Advances in Neural Information Processing Systems, Privacy Preserving Machine Learning - PriML and PPML Joint Edition*.
- [17] Zitai Chen, Georgios Vasilakis, Kit Murdock, Edward Dean, David Oswald, and Flavio D Garcia. 2021. {VoltPillager}: Hardware-based fault injection attacks against intel {SGX} enclaves using the {SVID} voltage scaling interface. In *30th USENIX Security Symposium (USENIX Security 21)*. 699–716.
- [18] Christopher A Choquette-Choo, Krishnamurthy Dij Dvijotham, Krishna Pillutla, Arun Ganesh, Thomas Steinke, and Abhradeep Guha Thakurta. 2024. Correlated Noise Provably Beats Independent Noise for Differentially Private Learning. In *The Twelfth International Conference on Learning Representations*.
- [19] Christopher A Choquette-Choo, Arun Ganesh, Ryan McKenna, H Brendan McMahan, John Rush, Abhradeep Guha Thakurta, and Zheng Xu. 2024. (Amplified) Banded Matrix Factorization: A unified approach to private training. *Advances in Neural Information Processing Systems* 36 (2024).
- [20] Christopher A Choquette-Choo, H Brendan McMahan, Keith Rush, and Abhradeep Thakurta. 2022. Multi-epoch matrix factorization mechanisms for private machine learning. *arXiv preprint arXiv:2211.06530* (2022).
- [21] INTEL CORP. 2015. Product Change Notification 114074-00. <https://qdms.intel.com/dm/i.aspx/5A160770-FC47-47A0-BF8A-062540456F0A/PCN114074-00> (2015).
- [22] LINE Corporation. 2023. *Differential Privacy in LINE Federated Learning*. Technical Report. <https://linecorp.com/en/security/article/461>.
- [23] Henry Corrigan-Gibbs and Dan Boneh. 2017. Prio: Private, robust, and scalable computation of aggregate statistics. In *14th USENIX symposium on networked systems design and implementation (NSDI 17)*. 259–282.
- [24] Victor Costan and Srinivas Devadas. 2016. Intel SGX explained. *Cryptography ePrint Archive* (2016).
- [25] Katharine Daly, Hubert Eichner, Peter Kairouz, H Brendan McMahan, Daniel Ramage, and Zheng Xu. 2024. Federated learning in practice: reflections and projections. In *2024 IEEE 6th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*. IEEE, 148–156.
- [26] Research Scientist Daniel Ramage and Stefano Mazzocchi. 2020. Federated Analytics: Collaborative Data Science without Data Collection. <https://research.google/blog/federated-analytics-collaborative-data-science-without-data-collection/> (2020).
- [27] Poulam Das, Lisa Eckey, Tommaso Frassetto, David Gens, Kristina Hostáková, Patrick Jauernig, Sebastian Faust, and Ahmad-Reza Sadeghi. 2019. {FastKitten}: Practical smart contracts on bitcoin. In *28th USENIX Security Symposium (USENIX Security 19)*. 801–818.
- [28] Sergey Denisov, H Brendan McMahan, John Rush, Adam Smith, and Abhradeep Guha Thakurta. 2022. Improved differential privacy for sgd via optimal private linear operators on adaptive streams. *Advances in Neural Information Processing Systems* 35 (2022), 5910–5924.
- [29] Danny Dolev and Andrew Yao. 1983. On the security of public key protocols. *IEEE Transactions on information theory* 29, 2 (1983), 198–208.
- [30] John R Douceur. 2002. The sybil attack. In *International workshop on peer-to-peer systems*. Springer, 251–260.
- [31] Cynthia Dwork, Krishnaram Kenthapadi, Frank McSherry, Ilya Mironov, and Moni Naor. 2006. Our data, ourselves: Privacy via distributed noise generation. In *Advances in Cryptology-EUROCRYPT 2006: 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, St. Petersburg, Russia, May 28–June 1, 2006. Proceedings 25*. Springer, 486–503.
- [32] Cynthia Dwork, Aaron Roth, et al. 2014. The algorithmic foundations of differential privacy. *Foundations and Trends® in Theoretical Computer Science* 9, 3–4 (2014), 211–407.
- [33] Hubert Eichner, Daniel Ramage, Kallista Bonawitz, Dzmitry Huba, Tiziano Santoro, Brett McLaren, Timon Van Overveldt, Nova Fallen, Peter Kairouz, Albert Cheu, et al. 2024. Confidential federated computations. *arXiv preprint arXiv:2404.10764* (2024).
- [34] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. 2019. Amplification by shuffling: From local to central differential privacy via anonymity. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2468–2479.
- [35] Russell A Fink, Alan T Sherman, Alexander O Mitchell, and David C Challener. 2011. Catching the Cuckoo: Verifying TPM Proximity Using a Quote Timing Side-Channel (Short Paper). In *Trust and Trustworthy Computing: 4th International Conference, TRUST 2011, Pittsburgh, PA, USA, June 22–24, 2011. Proceedings 4*. Springer, 294–301.
- [36] Antonios Gligis, Deepesh Data, Suhas Diggavi, Peter Kairouz, and Ananda Theertha Suresh. 2021. Shuffled model of differential privacy in federated learning. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 2521–2529.
- [37] Anonymization Team Google. 2019. Delta for thresholding. https://github.com/google/differential-privacy/blob/main/common_docs/Delta_For_Thresholding.pdf (2019).
- [38] Maurice P Herlihy and Jeannette M Wing. 1990. Linearizability: A correctness condition for concurrent objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)* 12, 3 (1990), 463–492.
- [39] White House. 2012. Consumer data privacy in a networked world: A framework for protecting a privacy and promoting innovation in the globaeconom. <http://www.whitehouse.gov/pnY/sites/default/files/privac> (2012).
- [40] Heidi Howard, Fritz Alder, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Antoine Delignat-Lavaud, Cédric Fournet, Andrew Jeffery, Matthew Kerner, et al. 2023. Confidential consortium framework: Secure multiparty applications with confidentiality, integrity, and high availability. *arXiv preprint arXiv:2310.11559* (2023).
- [41] Dzmitry Huba, John Nguyen, Kshitiz Malik, Ruiyu Zhu, Mike Rabbat, Ashkan Yousefpour, Carole-Jean Wu, Hongyuan Zhan, Pavel Ustинov, Harish Srinivas, et al. 2022. Papaya: Practical, private, and scalable federated learning. *Proceedings*

of Machine Learning and Systems 4 (2022), 814–832.

[42] Mohit Kumar Jangid, Guoxing Chen, Yinqian Zhang, and Zhiqiang Lin. 2021. Towards formal verification of state continuity for enclave programs. In *30th USENIX Security Symposium (USENIX Security 21)*, 573–590.

[43] An Ji, Bortik Bandyopadhyay, Congzheng Song, Natarajan Krishnaswami, Prabal Vashish, Rigel Smiroldo, Isabel Litton, Sayantan Mahinder, Mona Chitnis, and Andrew W Hill. 2025. Private Federated Learning In Real World Application–A Case Study. *arXiv preprint arXiv:2502.04565* (2025).

[44] Jiankai Jin, Chitchanok Chuengsatiansup, Toby Murray, Benjamin IP Rubinstein, Yuval Yarom, and Olga Ohrimenko. 2024. Elephants Do Not Forget: Differential Privacy with State Continuity for Privacy Budget. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 1909–1923.

[45] Peter Kairouz, Ziyu Liu, and Thomas Steinke. 2021. The distributed discrete gaussian mechanism for federated learning with secure aggregation. In *International Conference on Machine Learning*. PMLR, 5201–5212.

[46] Peter Kairouz, Brendan McMahan, Shuang Song, Om Thakkar, Abhradeep Thakurta, and Zheng Xu. 2021. Practical and private (deep) learning without sampling or shuffling. In *International Conference on Machine Learning*. PMLR, 5213–5225.

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

[48] Ryan Karl, Jonathan Takeshita, and Taeho Jung. 2021. Cryptonite: A framework for flexible time-series secure aggregation with non-interactive fault recovery. In *Security and Privacy in Communication Networks: 17th EAI International Conference, SecureComm 2021, Virtual Event, September 6–9, 2021, Proceedings, Part I* 17. Springer, 311–331.

[49] Shiva Prasad Kasiviswanathan, Homin K Lee, Kobbi Nissim, Sofya Raskhodnikova, and Adam Smith. 2011. What can we learn privately? *SIAM J. Comput.* 40, 3 (2011), 793–826.

[50] Fumiuyuki Kato, Yang Cao, and Masatoshi Yoshikawa. 2023. Olive: Oblivious Federated Learning on Trusted Execution Environment against the Risk of Specification. *Proceedings of the VLDB Endowment* 16, 10 (2023), 2404–2417.

[51] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2020. Spectre attacks: Exploiting speculative execution. *Commun. ACM* 63, 7 (2020), 93–101.

[52] Hanjun Li, Huijia Lin, Antigoni Polychroniadou, and Stefano Tessaro. 2023. LERNA: secure single-server aggregation via key-homomorphic masking. In *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 302–334.

[53] Jinyuan Li, Maxwell N Krohn, David Mazieres, and Dennis E Shasha. 2004. Secure Untrusted Data Repository (SUNDR).. In *Osdi*, Vol. 4. 9–9.

[54] Yehuda Lindell. 2017. How to simulate it—a tutorial on the simulation proof technique. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich* (2017), 277–346.

[55] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, et al. 2020. Meltdown: Reading kernel memory from user space. *Commun. ACM* 63, 6 (2020), 46–56.

[56] Yiping Ma, Jess Woods, Sebastian Angel, Antigoni Polychroniadou, and Tal Rabin. 2023. Flamingo: Multi-round single-server secure aggregation with applications to private federated learning. In *2023 IEEE Symposium on Security and Privacy (SP)*. IEEE, 477–496.

[57] Sinisa Matetic, Mansoor Ahmed, Kari Kostiainen, Aritra Dhar, David Sommer, Arthur Gervais, Ari Juels, and Srdjan Capkun. 2017. {ROTE}: Rollback protection for trusted execution. In *26th USENIX Security Symposium (USENIX Security 17)*. 1289–1306.

[58] David Mazieres and Dennis Shasha. 2002. Building secure file systems out of Byzantine storage. In *Proceedings of the twenty-first annual symposium on Principles of distributed computing*. 108–117.

[59] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. In *International Conference on Learning Representations*.

[60] H Brendan McMahan, Zheng Xu, and Yanxiang Zhang. 2024. A Hassle-free Algorithm for Private Learning in Practice: Don't Use Tree Aggregation, Use BLTs. *arXiv preprint arXiv:2408.08868* (2024).

[61] Simon Meier, Benedikt Schmidt, Cas Cremers, and David Basin. 2013. The TAMARIN prover for the symbolic analysis of security protocols. In *Computer Aided Verification: 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13–19, 2013. Proceedings* 25. Springer, 696–701.

[62] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: Privacy-preserving federated learning with trusted execution environments. In *Proceedings of the 19th annual international conference on mobile systems, applications, and services*. 94–108.

[63] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based fault injection attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1466–1482.

[64] Satoshi Nakamoto. 2008. Bitcoin: A peer-to-peer electronic cash system. <https://bitcoin.org/bitcoin.pdf> (2008).

[65] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. 2022. Federated learning with buffered asynchronous aggregation. In *International conference on artificial intelligence and statistics*. PMLR, 3581–3607.

[66] Jianyu Niu, Wei Peng, Xiaokuan Zhang, and Yinqian Zhang. 2022. Narrator: Secure and practical state continuity for trusted execution in the cloud. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2385–2399.

[67] Maxence Noble, Aurélien Bellet, and Aymeric Dieuleveut. 2022. Differentially private federated learning on heterogeneous data. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 10110–10145.

[68] Bryan Parno. 2008. Bootstrapping trust in a "trusted" platform. In *Proceedings of the 3rd conference on Hot topics in security*. 1–6.

[69] Bryan Parno, Jacob R Lorch, John R Douceur, James Mickens, and Jonathan M McCune. 2011. Memoir: Practical state continuity for protected modules. In *2011 IEEE Symposium on Security and Privacy*. IEEE, 379–394.

[70] Rafael Pass, Elaine Shi, and Florian Tramer. 2017. Formal abstractions for attested execution secure processors. In *Advances in Cryptology—EUROCRYPT 2017: 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30–May 4, 2017, Proceedings, Part I* 36. Springer, 260–289.

[71] Frank Piessens and Paul C van Oorschot. 2024. Side-channel attacks: A short tour. *IEEE Security & Privacy* 22, 2 (2024), 75–80.

[72] Mark Russinovich. 2023. Confidential computing: Elevating cloud security and privacy. *Commun. ACM* 67, 1 (2023), 52–53.

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

[74] Harish Srinivas, Graham Cormode, Mehrdad Honarkhah, Samuel Lurye, Jonathan Hehir, Lunwen He, George Hong, Ahmed Magdy, Dzmitry Huba, Kaikai Wang, et al. 2024. Federated Analytics in Practice: Engineering for Privacy, Scalability and Practicality. *arXiv preprint arXiv:2412.02340* (2024).

[75] Raoul Strackx and Frank Piessens. 2016. Ariadne: A minimal approach to state continuity. In *25th USENIX Security Symposium (USENIX Security 16)*. 875–892.

[76] Kunal Talwar, Shan Wang, Audra McMillan, Vitaly Feldman, Pansy Bansal, Bailey Basile, Aine Cahill, Yi Sheng Chan, Mike Chatzidakis, Junye Chen, et al. 2024. Samplable anonymous aggregation for private federated data analysis. In *Proceedings of the 2024 on ACM SIGSAC Conference on Computer and Communications Security*. 2859–2873.

[77] Salil Vadhan. 2017. The complexity of differential privacy. *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich* (2017), 347–450.

[78] Salil Vadhan and Tianhao Wang. 2021. Concurrent composition of differential privacy. In *Theory of Cryptography: 19th International Conference, TCC 2021, Raleigh, NC, USA, November 8–11, 2021, Proceedings, Part II* 19. Springer, 582–604.

[79] Salil Vadhan and Wanrong Zhang. 2023. Concurrent composition theorems for differential privacy. In *Proceedings of the 55th Annual ACM Symposium on Theory of Computing*. 507–519.

[80] Stephan van Schaik, Marina Minkin, Daniel Genkin, Yuval Yarom, and Andrew Kwong. 2021. CacheOut and SG Axe: How SGX Fails in Practice. Video at <https://youtu.be/kO-3Uh7tq60?t=1806>.

[81] Weili Wang, Sen Deng, Jianyu Niu, Michael K Reiter, and Yinqian Zhang. 2022. Engraft: Enclave-guarded raft on byzantine faulty nodes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2841–2855.

[82] Annika Wilde, Tim Niklas Gruel, Claudio Soriente, and Ghassan Karame. 2024. The Forking Way: When TEEs Meet Consensus. *arXiv preprint arXiv:2412.00706* (2024).

[83] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 640–656.

[84] Zheng Xu, Yanxiang Zhang, Galen Andrew, Christopher Choquette, Peter Kairouz, Brendan McMahan, Jesse Rosenstock, and Yuanbo Zhang. 2023. Federated Learning of Gboard Language Models with Differential Privacy. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 5: Industry Track)*. Sunayana Sitaram, Beata Beigman Klebanov, and Jason D Williams (Eds.). Association for Computational Linguistics, Toronto, Canada, 629–639. <https://doi.org/10.18653/v1/2023.acl-industry.60>

[85] Xinying Yang, Yuan Zhang, Sheng Wang, Benquan Yu, Feifei Li, Yize Li, and Wenyuan Yan. 2020. LedgerDB: A centralized ledger database for universal audit and verification. *Proceedings of the VLDB Endowment* 13, 12 (2020), 3138–3151.

[86] Cong Yue, Gang Chen, Tien Tuan Anh Dinh, Beng Chin Ooi, Zhongle Xie, Xiaokui Xiao, and Meihui Zhang. 2023. GlassDB: An Efficient Verifiable Ledger Database System Through Transparency. *Proceedings of the VLDB Endowment* 16, 6 (2023), 1359–1371.

[87] Yuanbo Zhang, Daniel Ramage, Zheng Xu, Yanxiang Zhang, Shumin Zhai, and Peter Kairouz. 2023. Private Federated Learning in Gboard. *arXiv e-prints* (2023), 500

A Discussions

In this section, we explore the vulnerabilities of our approach, discuss practical implementation challenges.

A.1 Side-Channel Attacks

Our protocol's security guarantees depend on the security of the underlying TEEs. However, TEEs themselves can be susceptible to various side-channel attacks [71]. We analyze our protocol's vulnerabilities to such attacks.

Our component for achieving linearizability (i.e., planner) is independent of the secure aggregation mechanism. This design offers the flexibility to employ different secure aggregation methods: SMPC [7] or TEE-based approaches [41]. We now analyze the side-channel vulnerabilities of our system in two scenarios: using TEE-based secure aggregation and using SMPC-based secure aggregation.

Side-channel attacks that break confidentiality. One well-known category of side-channel attacks is Controlled-Channel Attacks (CCA) [83], where an adversary can infer sensitive input by observing an application's input-dependent execution flow through architectural side-effects like page faults. Our linearizability component can be input-oblivious with respect to the execution flow and is therefore not vulnerable to CCA. However, a TEE-based secure aggregation protocol can be vulnerable if it performs input-dependent operations, such as data sparsification [50]. Consequently, the secure aggregation protocol must be carefully designed to avoid such vulnerabilities.

Next, we consider microarchitectural side-channel attacks (e.g., transient execution attacks such as Spectre [51] and Meltdown [55]) that leak memory content from within an enclave. The recovery protocol for our planner mechanism depends on enclave memory confidentiality. However, this dependency can be removed at some cost, as shown in Appendix B.2. TEE-based secure aggregation is inherently vulnerable to this attack class because it processes raw data in plaintext inside the enclave. An alternative is to use an SMPC-based secure aggregation protocol [7], which does not require enclave confidentiality. This approach, however, introduces a different vulnerability: it is susceptible to Sybil attacks. Since SMPC requires clients to collaboratively add noise, an adversary controlling a set of corrupted clients could instruct them to omit their noise contributions, thereby breaking the DP guarantee of the final output. This implies the need for a mechanism that can securely add noise without relying on enclave confidentiality.

Side-channel attacks that break integrity. Our protocol's linearizability guarantee is critically dependent on the integrity of the enclave. If an attacker compromises the enclave's integrity, the entire protocol becomes vulnerable. For instance, fault injection attacks like Plundervolt [63] or Voltpillager [17] could be used to bypass integrity and linearizability checks. In such a scenario, the protocol would fail, allowing an adversary to control the process, extract noise information, and ultimately breach interactive DP. Such side-channel attacks typically necessitate either physical access to the hardware [17] or kernel-level privileges [63] for execution.

Within a robust cloud VM infrastructure, direct physical access is inherently prevented. Moreover, the hypervisor acts as a critical defense layer, blocking privileged operations that attempt to manipulate the physical hardware state—for example, by intercepting writes to MSRs [17]. Consequently, leveraging a cloud-based VM is a valid strategy for mitigating the risks posed by this class of side-channel attacks

Furthermore, microarchitectural attacks like SGAXe [80] could potentially leak the attestation key, compromising integrity. Such vulnerabilities must be promptly addressed by TEE manufacturers, so we can mitigate the risk by performing frequent TCB updates, and clients can verify that the server is using a secure and up-to-date version of the TEE.

A.2 Implementation Challenges

Our system's security guarantee does not rely on additional trusted hardware like TPMs or complex mechanisms like SMR, but rather on honest clients performing auditing. This design choice enhances transparency. However, it also introduces new practical implementation challenges, particularly concerning the deployment of auditing functions on client devices. A flawed or unavailable implementation of this client auditing is critical, as it could halt the entire FL protocol. Here, we discuss the specific challenges of our implementation and potential mitigation strategies.

A.2.1 Remote Attestation on Heterogeneous Clients. A core requirement of our protocol is that clients perform remote attestation (RA) to verify the server-side enclave's quote. Implementing this verification process on client devices presents a challenge, especially in a typical cross-device FL environment with heterogeneous clients.

Verifying an RA quote can be a complex task. For instance, with Intel SGX, this may involve processing collateral information and requires the integration of specific SDKs and cryptographic libraries. This increases the size and complexity of the client-side application, potentially leading to instability and harming the availability of client auditing.

A practical solution is to employ a third-party verifier service like Intel Tiber Trust Authority⁷. Instead of performing the complex verification locally, the client application can forward the enclave's quote to this trusted service, which then returns a simple pass/fail result. This approach offloads the complexity from the diverse client devices, simplifying the client application and improving overall system stability and availability.

A.2.2 TCB Update and Management. In a real-world deployment, the TCB will require updates to address bugs or respond to newly discovered side-channel vulnerabilities. This necessity poses a challenge: clients must have a secure mechanism to learn the correct hash of the new, updated enclave binary.

This is a common challenge for TEE-based systems. For example, Papaya [41] addresses this by using a verifiable log, managed by a trusted party, to record valid enclave hashes. Our system can address this challenge in a more integrated and simpler manner by leveraging our existing **evidence chain**.

The evidence chain in our protocol is already an externally verifiable mechanism for managing state. We can extend its use to

⁷<https://www.intel.com/content/www/us/en/security/trust-authority.html>

manage TCB updates. The FL provider appends a special entry to the evidence chain containing the hash of the new valid enclave version. Clients, who already verify the evidence chain, can securely learn the new TCB hash from this entry.

This approach seamlessly integrates TCB management into our existing state-management protocol, eliminating the need for a separate verifiable log system. This highlights a key advantage of our system's design: the secure and verifiable state management provided by the evidence chain offers a simple and robust solution for TCB updates.

B Recovery

This appendix details two proposed recovery mechanisms designed to handle a critical failure scenario: an enclave crash occurring during the state transition process. Our protocol's liveness depends on the successful generation of a new evidence block that designates the next set of auditors. A crash during this critical phase (specifically, within Line 16 of Algorithm 3 after auditor signatures have been collected but before the next state is committed) could halt the entire DP-FTRL execution.

To solve this, we propose two distinct solutions, each tailored to a different threat model regarding the security guarantees of the underlying TEE.

B.1 Recovery under a Confidentiality Assumption

This first solution operates under the confidentiality assumption that the TEE provides robust confidentiality for enclave memory, protecting it from both software and hardware-based attacks, including side-channel attacks.

Mechanism. The protocol is augmented with the following steps:

- (1) **Key Initialization:** During the initial setup (Line 2 of Algorithm 3), the planner enclave generates a symmetric `recovery_key`. This key is immediately sealed and persisted as part of the enclave's initial state, never being exposed outside the trusted boundary.
- (2) **State Commitment:** In a regular update process, before requesting auditor signatures, the planner enclave randomly determines the next set of auditors, C_{next} . It then encrypts this selection with the `recovery_key` and includes this encrypted blob within the evidence presented to the current auditors.
- (3) **Recovery Trigger and Execution:** If the enclave crashes during Line 16 of Algorithm 3 after collecting a sufficient number of signatures from C_{audit} , the server initiates a recovery. It launches a new planner enclave instance, invoking a dedicated recovery function. The server provides this function with the evidence from the crashed enclave (containing the encrypted C_{next}) and the collected auditor signatures.
- (4) **State Restoration:** The recovery enclave unseals the `recovery_key`, decrypts C_{next} , and verifies the validity of the provided signatures against the auditor set specified in the evidence. If successful, it deterministically regenerates the evidence

block that the previous enclave failed to produce, thus restoring the continuity of the evidence chain and allowing the overall FL process to continue.

Limitations and Trade-offs. While this approach effectively recovers from a crash, it has two important limitations. First, the utility of the crashed round is partially lost, as the secure aggregation result itself cannot be recovered; only the protocol's state continuity is restored.

Second, and more critically, the security of this mechanism is entirely dependent on the confidentiality of the `recovery_key`. As discussed in Appendix A.1, if a side-channel attack were to leak this key, an adversary could decrypt C_{next} prematurely. This would create a strategic vulnerability. By revealing the upcoming random selection of auditors, it allows the adversary to repeatedly force a re-selection by inducing crashes until a favorable set (i.e., one including many corrupted clients) is chosen.

B.2 Recovery Resilient to Side-Channel Attacks

To address the limitations of the confidentiality-dependent approach, we propose a second mechanism designed for a weaker threat model that assumes a *transparent enclave* [70]. This model presumes that an adversary may be able to read the enclave's memory, rendering secret-based commitments insecure.

Mechanism. This protocol avoids long-term secrets and instead relies on a modification to the protocol logic:

- (1) **No Secret Key:** This mechanism entirely avoids the use of a secret `recovery_key`. The commitment to the next state is purely logic-based.
- (2) **Recovery Flag:** The evidence is extended to include a boolean `recovery_flag`. In a normal, non-failed operation, this flag is always set to false.
- (3) **Controlled Fork on Post-Signature Crash:** The protocol explicitly permits a controlled, temporary fork to handle a specific failure scenario: a crash that occurs after the auditors for a given round (e.g., $(i-1)$ -th C_{audit}) have signed for a normal transition (to state S_i with new randomly chosen auditors i -th C_{audit}), but before that state is finalized. In this case, $(i-1)$ -th C_{audit} are permitted to sign a second time for a recovery transition (to state S'_i), provided the server's request is for a recovery process and the corresponding enclave sets the `recovery_flag` to true. In this recovery transition, i -th C_{audit} is set to the same set as $(i-1)$ -th C_{audit} , and secure aggregation is not performed. This allows two potentially valid successor states, S_i and S'_i , to co-exist, representing a deliberate fork in the state history.
- (4) **Fork Resolution via Interactive Verification:** To resolve this fork and ensure the system converges back to a single history, the subsequent planner enclave is tasked with additional verification before it can proceed. To validate a transition from either S_i or S'_i , the planner enclave must query the auditors of the previous round ($(i-1)$ -th C_{audit}). Each member of $(i-1)$ -th C_{audit} , based on their local knowledge of whether they participated in a recovery signature, provides a signed message to the planner enclave endorsing only one of the two branches. The planner enclave must

collect a sufficient quorum of these endorsements to prove that a single, unambiguous branch has been chosen.

Limitations and Trade-offs. The key advantage of this design is its resilience to confidentiality-breaking attacks like side-channels. However, this robustness comes at a price. The protocol becomes significantly more complex. More critically, it introduces new liveness dependencies that may, paradoxically, reduce overall system availability. The success of a recovery now hinges on the availability of *two* consecutive sets of auditors (the current and the previous). A failure or mass dropout in either of these sets would cause the auditing itself to fail, making the system potentially more brittle than the single-dependency model in Section B.1.

C Related Work

The most relevant recent works include those by Bienstock et al. [12] and Ball et al. [7], which attempt to realize DP-FTRL in a (partially) malicious setting. These works utilize SMPC and re-share secrets between cohorts to achieve correlated noise. However, they assume that Sybil attacks do not occur, which means that they partially rely on the honesty of the central server during the planning phase to select the correct C_{cohort} . To the best of our knowledge, we are the first to propose a maliciously secure DP-FTRL that includes planning. In the following, we discuss related works that share similar objectives or approaches.

C.1 Secure FL

There is a vast body of work focusing on secure FL under untrusted server conditions. The foundational study was conducted by Bonawitz et al. [13], which proposed using SMPC within a cohort for aggregating updates in FL such as FedAvg to prevent untrusted servers from accessing individual updates. Some research has extended this to PFL setting that adds noise [2, 45]. Subsequent work [10, 52, 56, 73] focused on optimizing the number of intermediate helper users or minimizing communication rounds, but this requires additional computation, communication, and synchronization from clients. Moreover, corrupted clients could refuse to add distributed noise, undermining the intended privacy guarantees.

SMPC among non-colluding servers [23, 76] may overcome these challenges but requires non-collusion assumptions, which are unrealistic for a single FL conductor.

There is also literature on TEE-based secure aggregation for FL [16, 41, 62]. However, these studies focus only on stateless and noiseless secure aggregation methods like FedAvg, and the challenge of achieving DP-FTRL using TEEs has not yet been addressed.

C.2 Maliciously Secure Federated Analytics

As a broader concept of secure FL, (maliciously) secure federated analytics for DP has become a recent trend [26]. One of the major challenges is conducting stateful workloads on TEEs.

Confidential Federated Computation (CFC) [33], although not specifically designed for FL, shares similar motivations with our work. It addresses rollback attacks using SMR through multiple TEE. Therefore, it is susceptible to the problems mentioned in Introduction 1. Also, the application of this general system to DP-FTRL remains unclear. Srinivas et al. [74] proposed a federated analytics system that utilizes TEEs optimized for one-shot analytics to avoid

the challenge of the state management, but this approach sacrifices state management, making DP-FTRL unattainable.

The SMPC-based approach for sparse histogram [15], while not explicitly for federated learning, shares a similar environment. They secure the sparse histogram mechanism with interactive DP, while our approach secures DP-FTRL.

C.3 Maliciously Secure State Management System

A crucial aspect of our system is realizing a linearizable concurrent system under malicious conditions, a concept explored in previous research. Generally, two methods achieve this: using TEEs and utilizing auditors.

C.3.1 TEE Based Approach. Many TEEs provide sealing capabilities to offload encrypted and signed state to disk, which, combined with monotonic hardware counters, could be used to implement rollback protection [75]. However, this approach suffers from low performance [57], weak security [68], and quick exhaustion of hardware counters, hence recent Intel SGX versions lack monotonic counters. Systems using SMR for linearizability [4, 57] or Raft consensus for fault tolerance [40, 81] on TEEs might mitigate the problem. Elephants DP [44] utilizes similar systems for DP budget management, but it assumes a trusted data curator or does not address FL. Replacing our planner's linearizability component with these systems could work but would increase the TCB size significantly, complicating verification. Furthermore, these solutions cannot recover from disasters [4, 40], making them costly to manage. Our planner can operate on a single machine and recover.

C.3.2 Auditor Based Approach. In auditor-based methods, systems provide external state verifiability [85, 86]. The main challenge is establishing trusted auditors and process integrity. Combining TEEs might offer some integrity, but corrupted auditors could lead to faults and privacy leaks. Integrating blockchain [27, 66] with TEEs is another way to maintain consistent state. However, managing a blockchain is required, and in a single FL conductor setting, adopting a permissioned blockchain (i.e., internal verifiability) is infeasible, while managing a permissionless blockchain poses incentive issues or is vulnerable to a 51% attack.

D Missing Proofs

THEOREM D.1 (THEOREM 3.1). *Assume that if SIM sets k to i , C_{cohort} to be a subset of $f_{\text{qualify}}(\Pi, H, k)$, $\tilde{\theta}$ to $\tilde{\theta}_{i-1}$, and g_j as defined in Algorithm 1 (i.e., the original DP-FTRL), the view of SIM satisfies $(\epsilon, \delta + \delta_{\text{privacy}})$ -DP. Then, the view of any other SIM' satisfies $(\epsilon, \delta + \delta_{\text{privacy}})$ -DP.*

PROOF. This proof uses the "add the delta" technique [37] (i.e., joint convexity). We interpret the mechanism $M(D)$ as choosing either $M'(D)$ with probability $1 - \delta_{\text{privacy}}$ or $M''(D)$ with probability δ_{privacy} , where M'' is defined by Line 4 of Algorithm 1 and M' by Lines 6-17. If M' satisfies (ϵ, δ) -DP, then M satisfies $(\epsilon, \delta + \delta_{\text{privacy}})$.

We will demonstrate that M' satisfies (ϵ, δ) -DP. When setting k to i , C_{cohort} to be a subset of $f_{\text{qualify}}(\Pi, H, k)$, $\tilde{\theta}$ to $\tilde{\theta}_{i-1}$, and g_j as defined in Algorithm 1, this aligns with the matrix mechanism in adaptive streaming [28]. We show that the privacy guarantee

holds even if k , C_{cohort} , $\tilde{\theta}$, and g_j are chosen adaptively, using the method described in Theorem 2.1 of Denisov et al. [28], which equates the matrix mechanism to the adaptive Gaussian mechanism. Given that the sensitivity of M' is bounded by the participation schema due to Line 10 and SIM receives $\tilde{\theta}_{1:i}$ at round i (i.e., $C\theta_{1:i} + Z[:i]$ where $\theta_{1:i} \in \mathbb{R}^{i \times d}$) instead of $\tilde{\theta}_k$, our mechanism remains equivalent to the adaptive Gaussian mechanism. Consequently, the folklore technique used in [28] implies that the privacy of the adaptive Gaussian mechanism is equivalent to that of the non-adaptive Gaussian mechanism. Therefore, the privacy guarantee is maintained just as in that of Denisov et al. [28]. Thus, we conclude that M' satisfies (ϵ, δ) -DP, and consequently, M satisfies $(\epsilon, \delta + \delta_{\text{privacy}})$ -DP.

□

THEOREM D.2 (THEOREM 4.1). *Algorithms 3 (server-side) and 4 (client-side) achieve both integrity and linearizability with respect to the process described in Algorithm 2 when $\gamma = 0$.*

PROOF. This derives from the three verifications outlined in Section 4.1. Here's a brief review:

Integrity: We demonstrate that each process (i.e., Algorithm 2) loads the shared object q and correctly executes $\text{SECAGG}_{q,Z}$ on C_{cohort} with index i only when C_{cohort} adheres to the participation schema with $q.H$ and i , and correctly updates $q.H$ in the process.

Firstly, we show that a completed process loads the shared object q due to the evidence chain for $q.H$ and the sealing for $q.Z$. Initially, all clients store the evidence chain ID corresponding to $q.Z$ (Line 4 of Algorithm 3 in the verification of "Initialization"). An enclave loads $q.Z$ by unsealing with the evidence chain ID and generates evidence, including it. During AUDIT, clients verify that the evidence chain ID inside the evidence matches theirs. Thus, the enclave can pass the "agreement on an enclave" stage only when $q.Z$ is correctly loaded with the evidence chain ID by unsealing. The enclave retrieves $q.H$ from the evidence chain, and the validity of this chain is verified by remote attestation to extract correct auditors ("the approval of auditors") and AUDIT to verify the evidence chain's digest. Thus, the enclave can pass the "agreement on an enclave" stage only when $q.H$ is correctly loaded from the evidence chain.

Next, we ensure the correct execution of $\text{SECAGG}_{q,Z}$. In AUDIT, auditors agree only when they verify, through remote attestation, that C_{cohort} is confirmed to adhere to the participation schema and that $\text{SECAGG}_{q,Z}$ will be executed correctly. Therefore, passing the "agreement on an enclave" stage indicates the correct execution of $\text{SECAGG}_{q,Z}$.

Linearizability: In this section, to show linearizability, we demonstrate that the result of concurrent processes is equivalent to the result produced by processes that are sequentially (and not concurrently) invoked and completed based on the length of their loaded evidence chain. This equivalence is due to the auditors' approval and the the agreement on an enclave. To update $q.H$ (i.e., to create new evidence for the evidence chain as the enclave loads $q.H$ from the evidence chain), the evidence chain requires the signatures of the auditors recorded in the latest evidence. The auditors sign only once, indicating that only one enclave, which has loaded the evidence chain, can pass the verification. That is, other enclaves that load the same evidence chain (i.e., q) will never complete their

process. From the integrity, an enclave that loads q completes the process using the loaded q . Thus, the result of concurrent processes aligns with that of processes sequenced according to the order defined by the length of the loaded evidence chain.

□

THEOREM D.3 (THEOREM 4.2). *When $\gamma > 0$, Algorithm 3 maintains linearizability and integrity of Theorem 4.1 with a probability of $1 - \delta_{\text{privacy}}$ =*

$$1 - \left(1 - \sum_{i=1}^{2(n_{\text{audit}} - \tau)} \binom{ny}{2\tau - n_{\text{audit}} + i} \binom{n(\kappa - \gamma)}{2n_{\text{audit}} - 2\tau - i} / \binom{n\kappa}{n_{\text{audit}}} \right)^{n_{\text{round}}}.$$

PROOF. In each completed process, the system satisfies Theorem 4.1 as long as there is a majority agreement among honest clients within C_{audit} (i.e., reaching consensus). Therefore, the probability we are interested in calculating is the probability that at least one round fails to reach consensus among the n_{round} processes completed.

First, let's calculate the probability that a single round fails to reach a majority agreement among honest clients in C_{audit} . This situation occurs when the collected τ signatures do not include a majority of honest clients' signatures within C_{audit} . Define a as the random variable representing the number of corrupted clients among those in C_{audit} . When τ signatures are collected, they include at most $\tau - a$ signatures of honest auditors. If this number is less than the number of honest clients in C_{audit} who have not agreed, i.e., $n_{\text{audit}} - \tau$ where $n_{\text{audit}} = |C_{\text{audit}}|$, then a majority agreement is not reached. Hence, the probability is given by:

$$\Pr[\tau - a < n_{\text{audit}} - \tau] = \Pr[a > 2\tau - n_{\text{audit}}].$$

Here, a follows a hypergeometric distribution because it counts the number of corrupted clients among a randomly selected n_{audit} from the $n_{\text{available}} = n\kappa$ candidates, which includes at most $n \times \beta$ corrupted clients. Therefore, the probability p we want to find is:

$$p = \Pr[a > 2\tau - n_{\text{audit}}] = \sum_{i=1}^{2(n_{\text{audit}} - \tau)} \Pr[a = 2\tau - n_{\text{audit}} + i].$$

$$= \sum_{i=1}^{2(n_{\text{audit}} - \tau)} \binom{ny}{2\tau - n_{\text{audit}} + i} \binom{n(\kappa - \gamma)}{2n_{\text{audit}} - 2\tau - i} / \binom{n\kappa}{n_{\text{audit}}}.$$

This occurs independently across n_{round} rounds. Let b be the random variable that represents the number of rounds where consensus was not reached. Then, the probability of interest is:

$$1 - \Pr[b = 0].$$

Since b follows a binomial distribution with probability p and n_{round} trials, the probability we are looking for is:

$$1 - (1 - p)^{n_{\text{round}}}.$$

This expression gives us the probability that at least one of the rounds fails to reach a consensus.

□

THEOREM D.4 (THEOREM 5.1). *$M = \text{ConCOMP}(M_1, \dots, M_n)$, when used with any server interactive protocol, emulates the ideal model (i.e., Algorithm 1) with some simulator.*

PROOF. Let $T(D)$ denote the ideal functionality executed by a trusted party as defined in Algorithm 2. Here, we show that for any interactive protocol A in the real world, there exists a simulator Sim in the ideal world such that the view of the adversary, $\text{View}(A, M(D))$, is computationally indistinguishable from the view of the simulator, $\text{View}(\text{Sim}, T(D))$.

First, let us analyze the adversary's view in the real world, $\text{View}(A, M(D))$. This view consists of all messages that the adversary A receives during the execution of the protocol $M(D)$. Clients perform remote attestation to verify the integrity of the enclave code and its parameters. They reject any messages not originating from a legitimate enclave, ensuring that any data they accept is a valid output of a process defined in Algorithm 2. Although the adversary A controls when enclaves are executed, it cannot access their internal memory due to the confidentiality guarantees of the TEE. Consequently, A can only observe the inputs and outputs of the enclave for each process. Crucially, according to Theorem 4.2, the system maintains linearizability with a probability of $1 - \delta_{\text{privacy}}$. This means that with overwhelming probability, the sequence of outputs generated by the enclaves is equivalent to a sequential history where each process

$\langle q \text{ update}(C_{\text{cohort}}, i) \text{ } P \rangle \langle \text{OK}(\text{SECAGG}_{q, Z}(C_{\text{cohort}}, i, \text{aux}_{\text{secagg}}*)) \text{ } P \rangle$, completes in an ordered manner, with i incrementing sequentially. With the remaining probability of δ_{privacy} , linearizability may be violated, potentially exposing information about the dataset D to the adversary.

Now, let's construct the simulator Sim for the ideal world. Sim must generate a view, $\text{View}(\text{Sim}, T(D))$, that is computationally indistinguishable from $\text{View}(A, M(D))$. Sim can simulate the messages sent from clients to the server (e.g., signatures, encrypted updates) by generating random values. This is justified because, under standard cryptographic assumptions, these messages are computationally indistinguishable from random numbers to the adversary. The simulator Sim can mimic the messages that A receives from $M(D)$ by generating them randomly. To simulate the outputs from the enclaves, Sim interacts with the trusted party $T(D)$. Due to the integrity property of the enclaves in the real world, the i -th output from a real enclave is identical to the i -th output provided by the ideal functionality $T(D)$. Therefore, Sim can simply forward the output from $T(D)$ to the simulated adversary. Sim must also simulate the adversary's ability to influence the protocol. The ideal functionality $T(D)$ is explicitly designed to allow Sim to provide inputs on behalf of corrupted clients (e.g., malicious updates g_j where $j \in C_{\text{cohort}}$) and to choose the completion order of rounds by setting the index k . This allows Sim to perfectly replicate any malicious actions A could take regarding corrupted client inputs and process scheduling. Due to the linearizability of the system, the inputs for a given round k are independent of the outputs of any other concurrently executing but not-yet-completed round i (where $i < k$). This non-blocking nature allows Sim to simulate the adversary's reordering of round completions. Finally, we consider the failure case, which occurs with probability δ_{privacy} . In the ideal world, the trusted party $T(D)$ leaks the entire dataset D to Sim . With this information, Sim can perfectly simulate any possible view for the adversary, even one resulting from a catastrophic failure in the real protocol.

Since the views are computationally indistinguishable in both the success case (with probability $1 - \delta_{\text{privacy}}$) and the failure case (with probability δ_{privacy}), we conclude that $\text{View}(A, M(D))$ is computationally indistinguishable from $\text{View}(\text{Sim}, T(D))$. This holds under the standard security assumptions for the underlying cryptographic primitives and the TEE's remote attestation mechanism. \square

PROPOSITION D.5 (PROPOSITION 5.2). *The probability $\delta_{\text{interrupt}}$ of interruption of auditing is*

$$1 - \left(1 - \sum_{i=1}^{\tau} \binom{nk\beta}{n_{\text{audit}} - \tau + i} \binom{nk(1 - \beta)}{\tau - i} \right) \binom{nk}{n_{\text{audit}}}^{n_{\text{round}}}.$$

PROOF. The method for computing the probability is almost identical to that in Theorem 4.2. In each process, the interruption of auditing occurs when more than $n_{\text{audit}} - \tau$ auditors drop out. Since n_{audit} auditors are selected from nk clients, which includes $nk\beta$ dropout clients as assumed in Section 2.2, the probability of this happening is given by $p = \Pr[a > n_{\text{audit}} - \tau]$, where a is a random variable following a hypergeometric distribution with parameters nk and n_{audit} . Specifically,

$$p = \sum_{i=1}^{\tau} \binom{nk\beta}{n_{\text{audit}} - \tau + i} \binom{nk(1 - \beta)}{\tau - i} \binom{nk}{n_{\text{audit}}}.$$

This trial independently iterates across n_{round} rounds. In the same way as Theorem 4.2, we get the probability as

$$1 - (1 - p)^{n_{\text{round}}}.$$

\square

E Formal Verification

This appendix details the formal verification of the core security properties of our proposed planner enclave protocol. The goal is to formally prove its **linearizability** and integrity under a malicious server model (i.e., Theorem 4.1) while ensuring **liveness**. We use the symbolic verification tool **Tamarin Prover** to model the protocol and verify its security against a powerful adversary. This formal proof provides strong evidence for the claims made in Theorem 4.1 (linearizability) and Section 5.2.1 (liveness).

Remark: To simplify the model, here, we represent auditor consensus as the signature of a single, representative auditor who correctly follows the protocol (i.e., $\gamma = 0$). This abstraction is sufficient to demonstrate the linearizability and integrity, as long as at least one honest auditor is present. Even in the case where $\gamma > 0$, Theorem 4.2 shows that by collecting τ signatures, it is highly probable that honest clients form a majority. This justifies modeling the behavior of the quorum as that of a single, representative honest auditor. Therefore, this simplification effectively demonstrates the essential security properties of our system.

E.1 Modeling Framework: Tamarin Prover

Tamarin Prover is a tool for the symbolic modeling and analysis of security protocols [61]. It models protocols using *Multiset Rewriting (MSR)* rules and assumes a Dolev-Yao adversary, which aligns with our threat model. Modeling enclave program logic with Tamarin has been shown to be an effective approach for verifying state

continuity properties, as demonstrated in prior work [42]. See the details of modeling enclave programs in Tamarin in [42].

E.2 Modeling Our Protocol in Tamarin

We translated our protocol's core logic into a set of Tamarin MSR rules, modeling the interactions between the planner enclave, the clients (auditors), and the adversary.

E.2.1 TEE Primitives.

- **Planner Enclave and Remote Attestation:** An enclave is modeled by rules that use a unique hardware-derived attestation key ('!AttestationSecretKey'). Remote attestation is modeled by having the enclave sign a message containing its identity ('mrenclave'), a fresh nonce, and other relevant data, which is called a quote. The **integrity** of the enclave's execution is guaranteed by Tamarin's assumption that an entity faithfully follows its specified rules, which are verified by other parties through this quote.
- **Ecall/Ocall Flow:** The sequence of an Ecall, a subsequent Ocall to a client, and the final processing step inside the enclave is modeled using a **Linear Fact**. This fact can only be consumed once, ensuring that the enclave's internal state is securely passed from one step to the next in a sequential manner.
- **Confidentiality:** Sensitive data, such as a committed random number, is modeled as a fact whose contents are inaccessible to the adversary. The adversary can see the fact exists but cannot learn the value of the input, thus modeling confidentiality.

E.2.2 Protocol Logic.

Evidence Chain: An evidence chain is modeled as a sequence of EvidenceChain(. . .) facts, each of which has a hash of the previous block (block_hash). Crucially, it contains not only the block hash but also a TEE-generated quote over that hash. In our model, this quote is represented as a signature by the enclave's attestation key (~attestation_sk) over a tuple containing the block_hash, the enclave's identity ('mrenclave'), and a nonce. This design ensures the integrity and authenticity of the chain's progression. When a subsequent process begins, the enclave must verify this quote before it can proceed. While the EvidenceChain facts themselves are passed through the public channel (making them subject to replay attacks by the adversary), the cryptographic link provided by the quote prevents the adversary from forging a new, invalid state or tampering with an existing one without being detected.

The Ecall/Ocall Flow. The core of our model (i.e., Algorithm 3) lies in the Ecall/Ocall interactions between the planner enclave and the clients (auditors) using the evidence chain, as shown in Figure 6. By modeling this flow in detail, we enable Tamarin to verify the linearizability of all state transitions. The following explains the key rules for the two main phases of the protocol.

Initial State Generation (Genesis Block): The protocol begins by generating the first block of the evidence chain, which we refer to as the genesis block. This corresponds to Line 2-7 of Algorithm 3. This process is modeled by the following Tamarin rules:

- **EC_Initialization (Ecall):** This rule models the planner enclave initiating the creation of the evidence chain. The enclave generates a unique nonce_for_init for the entire chain and a nonce_for_enclave_thread for this specific transaction. It then produces a quote—a signature using its hardware-derived !AttestationSecretKey—over its identity ('mrenclave'), the operation type ('init'), and these nonces. This quote, along with the nonces, is sent to the auditors via an Ocall (Out(. . .)). The linear fact EnclaveThreadForInitialization ensures state continuity within the enclave for the subsequent Sealing step.
- **OC_AuditorSignForInitialization (Ocall):** An auditor handles the Ocall from EC_Initialization with this rule. It first performs remote attestation by verifying the quote against the enclave's public key (!AttestationPublicKey). Upon successful verification, the auditor signs the nonce_for_enclave_thread to signal its approval of the initialization and returns this signature to the enclave. This represents the initial consensus on the genesis block.
- **Sealing:** The enclave receives the auditor's signature and finalizes the genesis block by sealing it with the EvidenceChain fact. This fact includes the block_hash of the genesis block, which is derived from the enclave's identity, the nonces, and the auditor's signature. The enclave then commits this state to the evidence chain, ensuring that it is now part of a verifiable history.

State Update Process: Once initialized, the chain is extended through subsequent update processes. This corresponds to Lines 11-20 of Algorithm 3. This process is modeled by the following Tamarin rules:

- **EC_InitProcess (Ecall):** This rule is invoked for subsequent updates, modeling the planner enclave loading a previously committed state (EvidenceChain(. . .)). A critical step here is the verification of the quoted_block_hash from the loaded EvidenceChain, ensuring the enclave is starting from a legitimate, previously attested state. The enclave then prepares a new state transition based on a new input and generates a new quote. This new quote attests to the proposed update, binding the old block_hash and the new input together. The new quote is then sent to the auditors via an Ocall, and the EnclaveThreadForUpdate linear fact maintains the enclave's state for the final Update rule.
- **OC_AuditorSign (Ocall):** This rule models the core auditing function. An auditor receives the new quote from the EC_InitProcess Ocall. It verifies this quote, confirming that the enclave is proposing a valid state transition. By signing the quote, the auditor approves the update, as enforced by the AuditorSignOnlyOnce restriction (see Listing 1 below). This signature is the crucial element that allows the Update rule inside the enclave to finalize the new state and append it to the evidence chain.
- **Update:** Finally, the enclave processes the auditor's signature and updates the evidence chain with the new state. It generates a new EvidenceChain fact that includes the new block_hash, which is derived from the previous block's

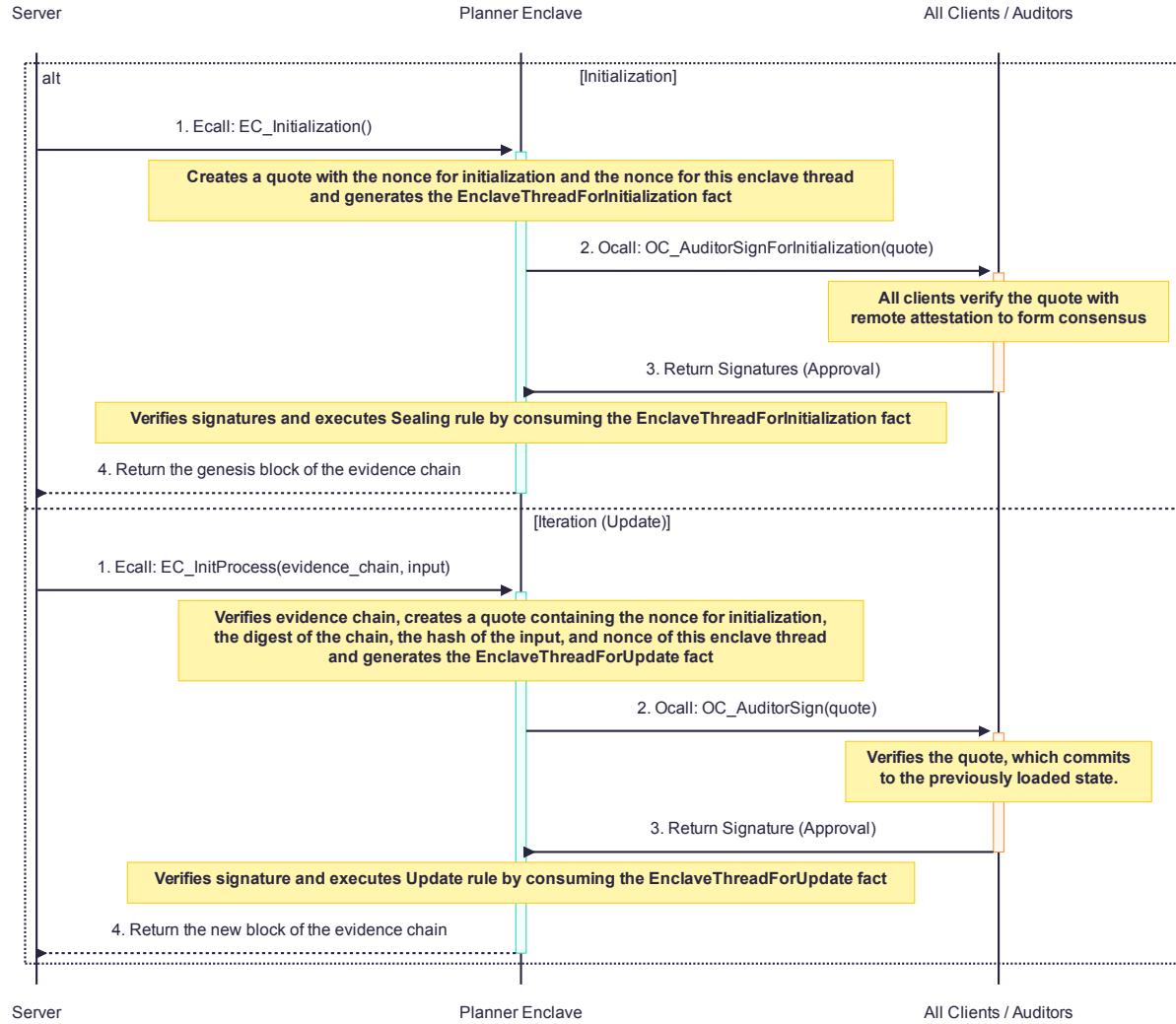


Figure 6: Sequence diagram of the state transition protocol between the Server, Planner Enclave, and Auditors.

hash, the new input. This new block is then appended to the evidence chain, completing the state transition.

This detailed modeling of the Ecall/Ocall flow allows Tamarin to verify that every state transition is correctly attested by the TEE and validated by an honest auditor. This provides the formal proof for the linearizability and integrity properties outlined in the main paper, demonstrating resilience against the rollback attacks modeled in `GenerateRollbackEvidenceChain` (see Listing 2 below).

E.2.3 Auditors and Adversary Model.

Honest Auditors: The honest behavior of auditors is enforced using a ‘restriction’. This restriction ensures that an auditor cannot approve two different histories branching from the same state. Specifically, an auditor can only sign once for any given `block_hash`.

```

1 restriction AuditorSignOnlyOnce:
2   "
3     All nonce_for_init block_hash input1 input2 #t1 #t2.
4     AuditorSignLabel(nonce_for_init, block_hash, input1)
5     ↙ @t1
6     &
7     AuditorSignLabel(nonce_for_init, block_hash, input2)
8     ↙ @t2
9     &
10    not (input1 = input2)
11    ==>
12    #t1 = #t2
13    "

```

Listing 1: Restriction enforcing that an auditor signs only once per state.

Adversary: We explicitly model the adversary's primary attack vector: the rollback attack. The rule `GenerateRollbackedEvidenceChain` allows the adversary to take any previously observed `EvidenceChain` from the public channel and re-inject it into the system as a valid input for a new update process. This directly models the forking threat described in this paper.

```

1 rule GenerateRollbackedEvidenceChain:
2   [
3     // Adversary takes a previously seen
4     ↪ EvidenceChain from the network
5     In(<..., block_hash, ..., quoted_block_hash>),
6     // Verifies its quote to make it seem legitimate
7     !AttestationPublicKey(attestation_pk),
8     _restrict( verify(quoted_block_hash, ...,
9     ↪ attestation_pk) = true )
10    ]
11   -->
12   [
13     // Re-creates the EvidenceChain to feed into the
14     ↪ Ecall
15     EvidenceChain(~nonce_for_init, ..., block_hash,
16     ↪ ...)
17   ]

```

Listing 2: Rule explicitly modeling the adversary's rollback attack capability by replaying a past state.

E.3 Security Properties and Verification Results

We defined two key lemmas in Tamarin to represent our desired security properties and used the prover to check if they hold true against our adversary model.

E.3.1 Proving Linearizability. To prove that our system achieves linearizability, we defined the following lemma, which asserts that for any given state (identified by `block_hash`), only one unique subsequent state can be committed.

```

1 lemma UpdateOnlyOnceForAllBlockHash:
2   "
3     All nonce_for_init ... block_hash input1 input2 #t1 #
4     ↪ t2.
5     UpdateLabel(nonce_for_init, ..., block_hash, input1)
6     ↪ @t1
7     &
8     UpdateLabel(nonce_for_init, ..., block_hash, input2)
9     ↪ @t2
10    &
11    not (input1 = input2)
12    ==>
13    #t1 = #t2
14    "

```

Listing 3: Lemma for proving linearizability.

This lemma states that it is impossible for two update operations with different inputs ($\text{input1} \neq \text{input2}$) to successfully complete from the same state (`block_hash`). This directly prevents the system from being forked into two different valid histories from a single point in time. An adversary attempting a rollback attack (modeled by `GenerateRollbackedEvidenceChain`) will fail to create a divergent history if a valid update has already been committed from that state.

Tamarin Prover successfully proved this lemma. This provides formal evidence that our protocol maintains a single, linearizable history and is resilient to rollback and forking attacks.

E.3.2 Proving Liveness. Next, we wanted to ensure that our protocol not only is secure but also guarantees **liveness**, allowing it to make progress even in the presence of crashes.

```

1 lemma UpdateAvailability: exists-trace
2   "
3     Ex nonce_for_init block_hash input #t1 #t2.
4     UpdateLabel(..., block_hash, input) @t1
5     &
6     UpdateLabel(..., block_hash, input) @t2
7     &
8     #t1 < #t2
9   "

```

Listing 4: Lemma for proving availability/executability.

The primary purpose of this lemma is to demonstrate the protocol's crash recovery capability. It proves that a valid execution trace exists, which implies that if an update process initiated at time $t1$ were to fail (or "crash") before completion, a subsequent process at a later time $t2$ can successfully execute the *exact same* update using the same input. State consistency is critical during this recovery. As proven by the `UpdateOnlyOnceForAllBlockHash` lemma, any attempt to recover the process with a *different* input would be rejected. Therefore, this lemma, in conjunction with the linearizability proof, formally shows that our protocol supports consistent recovery from failures without compromising the integrity of the state.

Tamarin Prover successfully found a valid execution trace, proving the lemma. This confirms that our protocol has **liveness**.