Verifiable Federated Training with Zero-Knowledge Proofs (ZK-FL): A Literature Review

Tarek Salama
Purdue University

Zeyad Elshafey
Purdue University

Ahmed Elbehiry
Purdue University

1 Introduction

Modern machine-learning (ML) systems increasingly operate over sensitive, distributed data (e.g., healthcare, finance, on-device learning), where stakeholders demand evidence of fairness, training integrity, and privacy—without exposing raw data or proprietary models. Today, however, most systems either trust logs and audits or verify only isolated properties in ad hoc ways, leaving end-to-end guarantees across the data–training–aggregation pipeline out of reach. Our project targets this gap: we build a compact, reproducible pipeline that proves dataset properties, per-round training correctness, and secure aggregation well-formedness using zero-knowledge proofs (ZKPs), so that a verifier can check compliance from public commitments and succinct proofs alone.

Problem statement. We study how to let a third party verify (i) a basic dataset property, (ii) the correctness of a single clipped-SGD step, and (iii) the well-formedness of secure aggregation—*without* revealing individual records or client updates—using succinct ZK proofs and commitment roots as the only public artifacts.

Task. We aim to produce an end-to-end *verifiable learning* pipeline with three components: (1) a ZK proof that a committed dataset satisfies a simple distributional property (binary class balance) without revealing any row; (2) a ZK proof that a single step of stochastic gradient descent (SGD) on a tiny linear model was computed correctly from committed inputs, with gradient clipping; and (3) a dropout-tolerant secure aggregation protocol in which each client attaches a ZK proof that its masked update is well-formed, so the server learns only the aggregate. The three parts share a common commitment interface (Poseidon/Merkle roots) and can be demonstrated independently or composed.

State of the art. There has been substantial progress in adjacent areas. Verifiable computation shows that ZKPs can certify general computations efficiently [5,8]. ZK for ML has matured enough to prove predictions and model properties [6], and there is first work on proving ML fairness in zero knowledge [10]. Secure aggregation protects client updates so the server observes only sums and can tolerate dropouts in real deployments [3]. Collectively, these lines of work underscore that cryptographic verification of ML is feasible and valuable, motivating a unified, reproducible demonstration.

Flaw / gap. Despite these advances, existing approaches typically address *one* stage of the pipeline in isolation (e.g., fairness *or* training integrity *or* aggregation), rely on heavy circuits or specialized hardware/software stacks, and rarely offer a cohesive recipe that a practitioner can reimplement end-to-end within course constraints. Moreover, training-integrity proofs often gloss over dataset membership (opening the door to subset/poisoning edge cases), while secure aggregation protocols usually lack a ZK check that each masked message was formed from a committed gradient and the agreed mask-generation rules. As a result, it remains hard for a third party to verify, in a single flow, that (i) the data satisfied a property, (ii) the weight update obeyed a stated rule on that data, and (iii) the server only learned an aggregate of well-formed client updates.

Our idea / solution. We propose a minimal but complete pipeline engineered for reproducibility and speed: small Circom circuits **[1]** with Poseidon-based Merkle commitments **[4]**; fixed-point arithmetic and a scalar α "clipping" trick to avoid divisions; and a secure-aggregation layer with pairwise additive masks plus Shamir-based dropout recovery. The dataset-property circuit proves boolean labels, membership to a committed root, and public counts (c_0, c_1, N); the verifier checks a tolerance rule outside the circuit. The training-integrity circuit proves one clipped-SGD step from committed (w_t, w_{t+1}) and a batch sampled from the committed dataset. The aggregation layer has each client ZK-prove that its masked message $m_i = g_i + R_i$ was derived from committed gradients and seeds according to the agreed sign and PRF domains, enabling the server to verify and sum without seeing individuals.

Threat model & assumptions. We assume an honest-verifier model for ZK verification and standard soundness/zero-knowledge of the proof system. The server (aggregator) may be curious and attempt to infer client updates; clients may be faulty and submit malformed messages. We trust collision resistance of Poseidon and binding of Merkle commitments. We do *not* protect against data poisoning beyond membership checks, and we focus on a single training step (multi-round composition is out of scope for this update).

Contributions.

1. **ZK dataset property proof**: a tractable circuit for binary class balance with full-membership enforcement to the dataset Merkle root.
2. **ZK training-integrity step**: a per-round clipped-SGD proof for a tiny linear model that binds to both dataset and weight commitments, mitigating subset/consistency attacks.
3. **Dropout-tolerant secure aggregation with ZK well-formedness**: clients attach proofs that their masks/updates follow the shared PRF/sign conventions; the server verifies and learns only the sum.

Running example. Throughout, we use a down-sampled UCI Adult dataset and a tiny logistic-regression model. We commit to the dataset via a Merkle root; prove a class-balance property; prove one SGD update on a sampled batch; and then aggregate n client updates where each client attaches a ZK proof that its mask and gradient followed the agreed rules.

Proof it works (evaluation teaser). We will provide synthetic and small real-data demonstrations, report proof sizes and proving times as a function of model dimension d and batch size b, and include positive/negative tests (e.g., tampered batches, incorrect masks) to show that verification fails when assumptions are violated. The goal is not to optimize absolute performance but to deliver a clear, reproducible blueprint that others can build upon for richer fairness properties, larger models, and multi-round training.

Limitations. Our circuits target clarity and speed over generality: small fixed-point models (no convolutions), a single training step (not full-epoch proofs), and a simple fairness proxy (class balance rather than richer notions). We treat secure aggregation in a semi-honest network setting and do not address poisoning or sybil resistance.

Significance. By tying dataset properties, a training step, and aggregation into one auditable flow with shared commitments and succinct proofs, our pipeline illustrates a practical path toward trustworthy ML that does not rely on privileged access to data or code. The result is a compact, educational reference for end-to-end verifiable learning, suitable for course projects and as a starting point for research-grade extensions.

2 Background

Commitments and Merkle trees, hash functions optimized for proof systems, modern zero-knowledge (ZK) proof systems and circuit models, fixed-point arithmetic for arithmetized ML, and federated learning with secure aggregation. We end with the threat models referenced later.

2.1 Commitments and Merkle Trees

A *commitment* lets a prover bind to a value while keeping it hidden; later they can *open* the commitment to reveal the value and prove consistency. Security has two standard properties: *binding* (the value cannot be changed after committing) and *hiding* (the commitment reveals no information until opened). A Merkle tree commits to a sequence of leaves by hashing upward to a single root. To prove membership of one leaf, the prover reveals only the leaf and its authentication path; the verifier recomputes the root and compares to the public root. In our setting, dataset rows (or labels) are leaves; the public input is the Merkle root; and membership proofs bind any used sample to the committed dataset.

ZK-friendly hashing (Poseidon). Classical hashes (e.g., SHA-2) are expensive inside ZK circuits because they use bitwise permutations. *Poseidon* is a sponge-based permutation designed for efficiency over prime fields, drastically reducing constraints inside SNARKs [4]. We use Poseidon both for Merkle trees and for domain-separated pseudorandom values inside circuits (e.g., deriving per-client masks).

2.2 Zero-Knowledge Proof Systems and Circuits

A non-interactive zero-knowledge proof system for NP lets a prover convince a verifier that a statement is true (there exists a witness satisfying a relation) without revealing the witness. Security notions are *completeness*, *soundness*, and *zero-knowledge*. Modern zk-SNARKs provide succinct proofs and fast verification [5, 8]. Systems like Groth16 achieve tiny proofs and constant-time verification given a CRS (common reference string) [5]. We target this family in our prototype because (i) verification is extremely fast, (ii) proof sizes are small, and (iii) the developer tooling is mature for instructional projects.

Arithmetization (R1CS) and Circom. To use SNARKs, programs are expressed as arithmetic circuits/constraints over a finite field. A common form is Rank-1 Constraint Systems (R1CS), where each constraint enforces $(\langle \mathbf{a}, \mathbf{z} \rangle) \cdot (\langle \mathbf{b}, \mathbf{z} \rangle) = \langle \mathbf{c}, \mathbf{z} \rangle$ for wires $\mathbf{z}$. *Circom 2* is a high-level language and compiler that helps build reusable circuit components and emits R1CS plus proving/verifying keys [1]. We use Circom's standard library (e.g., comparators, range checks) and write custom gadgets for ML updates and aggregation checks.

zkML context. Verifying ML computations in zero knowledge has progressed from toy examples to full model inference and property proofs [6]. We build on this trajectory but scope to a tiny linear/logistic model with one SGD step to keep proving costs tractable for a course project.

2.3 Fixed-Point Arithmetic and Clipping in Circuits

Proof systems operate over finite fields; floating-point is unavailable. We therefore encode reals as *fixed-point* integers: for a scale S (e.g., $S = 2^k$), a real x is represented by $\tilde{x} = \lfloor Sx \rceil$. Additions are exact; products satisfy $\widetilde{xy} \approx (\tilde{x} \cdot \tilde{y})/S$ with controlled rounding. We constrain intermediate ranges to avoid overflow and to keep the witness consistent with the public update.

Gradient clipping without division. Clipped-SGD uses $g_{\text{clip}} = \min\left(1, \frac{C}{\|g\|}\right) \cdot g$. To avoid division in-circuit, we introduce a witness scalar α with constraints that encode the clipping rule in fixed point: (i) $0 \le \alpha \le 1$, (ii) $\alpha \cdot \|g\| \le C$, (iii) if $\|g\| \le C$ then $\alpha = 1$ (enforced via a boolean guard on the comparison), (iv) $g_{\text{clip}} = \alpha \cdot g$. This pattern keeps constraints simple while matching the semantics of clipping.

2.4 Federated Learning and Secure Aggregation

In *federated learning* (FL), n clients compute local updates and a server aggregates them, reducing raw data exposure. *Secure aggregation* ensures the server learns only the sum of client vectors, not any individual update. A practical construction establishes pairwise masks between clients so that, when all masked vectors are summed, masks cancel and only the aggregate remains; robust protocols handle client dropouts by sharing mask material so missing masks can be reconstructed [3]. We pair this with ZK proofs that each client's masked message is *well-formed* from a committed gradient and agreed PRF/mask rules, preventing malformed contributions while keeping updates hidden.

PRFs and mask derivation. We instantiate mask seeds using domain-separated PRF calls (e.g., Poseidon in hash-to-field mode) keyed by client secrets. The ZK statement binds the masked message to (a) the committed gradient, (b) the client's committed key material, and (c) the public protocol transcript (round id, peers), ensuring masks are derived correctly.

2.5 Fairness and Dataset-Property Proofs

Many fairness checks reduce to statistics over labels or features. As a minimal proxy, we prove a *class-balance* property: the prover reveals only (c_0, c_1, N) while demonstrating that each counted label is a valid leaf under the public dataset root. Recent systems explore richer fairness in zero knowledge; our project provides a compact, reproducible instance suitable for coursework [10].

2.6 System Setting

We consider a standard federated setting with one aggregator server S, n clients $\{C_i\}_{i=1}^{n}$, and an optional public auditor/verifier $\mathcal{V}$. Each client C_i holds a local dataset D_i and computes a gradient update $\mathbf{g}_i$ for a small linear/logistic model of dimension d on a mini-batch of size b. We use Poseidon–Merkle commitments for data and model states: each client publishes a dataset root R_{D_i}; the server publishes an append-only log of weight roots $\{R_{W_t}\}_{t=0}^{T}$ where R_{W_t} commits to $\mathbf{w}_t$.

Datasets/models. We target small tabular, binary-label datasets (e.g., down-sampled UCI Adult) and a tiny linear/logistic model with $d \le 8$ and $b \le 32$ to keep proving costs tractable.

Where the dataset-property proof applies. By default we prove a *per-client* property on each D_i (counts $(c_{0,i}, c_{1,i}, N_i)$) under R_{D_i}. The server sums the public counts to obtain global $(\sum_i c_{0,i}, \sum_i c_{1,i}, \sum_i N_i)$ without ever seeing raw rows. (A single proof over $\cup_i D_i$ is possible but out of scope for this update.)

Why a single SGD-step proof with committed weights? If the server publishes R_{W_t} each round, then a step proof that binds $(R_{W_t}, R_{W_{t+1}})$ establishes the correctness of the $t \to t+1$ transition on committed data. Chaining such steps yields an auditable trajectory; spot-checking random rounds is also effective and cheaper.

2.7 Notation and Conventions

We fix a prime field for the proof system and work in fixed point. Scalars are plain letters (e.g., x), vectors are bold ($\mathbf{x}$), and matrices are bold uppercase ($\mathbf{X}$). The ith client is $i \in \{1, \ldots, n\}$, model dimension is d, and batch size is b.

Fixed-point. Let the scale be $S = 2^k$ (default k=12). A real x is represented by $\tilde{x} = \lfloor Sx \rceil \in \mathbb{Z}$; additions are exact in fixed point, and products use one rescale: $\widetilde{xy} \approx (\tilde{x} \cdot \tilde{y})/S$. We bound intermediate magnitudes to avoid overflow and enforce range checks in-circuit.

Norms and clipping. We use the ℓ_2 norm with squared form $s = \|\mathbf{g}\|_2^2 = \sum_{j=1}^{d} g_j^2$ to avoid square roots. Clipping uses a witness scalar $\alpha \in [0,1]$ with constraints (i) $\alpha \leq 1$, (ii) $\alpha^2 s \leq C^2$, (iii) if $s \leq C^2$ then $\alpha = 1$ (gated by a boolean comparison), and then $\mathbf{g}_{\text{clip}} = \alpha \mathbf{g}$ in fixed point.

Commitments, paths, and tags. We commit sequences (dataset rows, model weights) via Poseidon–Merkle trees; R_{D} denotes the dataset root, R_{W_t} and $R_{\mathsf{W}_{t+1}}$ the weight roots. A Merkle authentication path for leaf ℓ is $\pi(\ell)$. We use domain separation strings (e.g., $\mathtt{tag} = \texttt{"mask"}$) whenever hashing/PRF'ing distinct objects [4].

SGD terminology. For a model $f_{\mathbf{w}}(\mathbf{x}) = \langle \mathbf{w}, \mathbf{x} \rangle$, one SGD step with learning rate η is $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\, \mathbf{g}_{\text{clip}}$ on a mini-batch of size b. (If using logistic regression in some runs, we approximate the sigmoid locally with a low-degree polynomial and keep inputs range-limited; the circuit still consumes a vector gradient and applies the same clipping rule.)

Secure aggregation signs. Let r_{ij} be the pairwise mask between clients i and j derived from a PRF call with domain-separated inputs. Each client i sends $\mathbf{m}_i = \mathbf{g}_i + \sum_{j \neq i} \sigma_{ij}\, \mathbf{r}_{ij}$, with $\sigma_{ij} = +1$ if $i < j$ and $\sigma_{ij} = -1$ otherwise, so masks cancel in the sum [3].

2.8 Public Inputs and Witness Layout (Preview)

We sketch the exact I/O each circuit will use; Methodology will give constructions and constraints.

(A) Dataset-property circuit (class balance). **Public:** (i) dataset Merkle root R_{D}, (ii) counts (c_0, c_1, N). **Witness:** for each counted item: leaf label $\ell_j \in \{0,1\}$ and path $\pi(\ell_j)$. **Checks:** each $(\ell_j, \pi(\ell_j))$ opens under R_{D}; counters aggregate to (c_0, c_1, N); only the totals are revealed.

(B) Training-integrity circuit (one clipped-SGD step). **Public:** R_{D}, R_{W_t}, $R_{\mathsf{W}_{t+1}}$, learning rate η, clip bound C, batch indices (or a batch-commitment root), and the scale S. **Witness:** (i) weights $\mathbf{w}_t$ and $\mathbf{w}_{t+1}$ opening to their roots, (ii) batch samples $(\mathbf{x}_j, y_j)$ with paths to R_{D}, (iii) the un-clipped gradient $\mathbf{g}$, clipped gradient $\mathbf{g}_{\text{clip}}$, and scalar α. **Checks:** membership of batch samples; fixed-point gradient from the batch; clipping constraints with C; update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\, \mathbf{g}_{\text{clip}}$ (all in fixed point).

(C) Secure-aggregation well-formedness circuit (per client). **Public:** round identifier rid, client id i, ordered peer list, masked message $\mathbf{m}_i$, a commitment to the client's gradient R_{G_i} (or include $\mathbf{g}_i$ as a hidden witness but bind via a commitment), and a commitment to the client's key material R_{K_i}. **Witness:** gradient $\mathbf{g}_i$ opening to R_{G_i}; PRF seeds/keys opening to R_{K_i}; pairwise masks $\mathbf{r}_{ij}$ computed as $\mathsf{PRF}(K_{ij}; \mathsf{rid} \,\|\, i \,\|\, j, \texttt{"mask"})$. **Checks:** re-derive each $\mathbf{r}_{ij}$ with domain separation; enforce $\mathbf{m}_i = \mathbf{g}_i + \sum_{j \neq i} \sigma_{ij}\, \mathbf{r}_{ij}$; no information about $\mathbf{g}_i$ or keys leaks beyond $\mathbf{m}_i$.

Defaults. Unless stated otherwise we use $d \leq 8$, $b \leq 32$, $n \leq 16$, k=12 (S=4096), and Poseidon for all in-circuit hashing [4]. These bounds match the evaluation targets in the Introduction and keep proving costs tractable.

3 Methodology

We connect three ZK components via shared commitments: (A) per-client dataset-property proofs, (B) a single SGD-step training-integrity proof, and (C) a secure-aggregation well-formedness proof that *includes masking* when clients are malicious. Arithmetic and hashing follow Section 2.

3.1 Threat Model and Security Assumptions

Our system operates in a federated setting with multiple participants: n clients $\{C_i\}_{i=1}^{n}$, one aggregator server S, and a public verifier/auditor $\mathcal{V}$. We define precise adversary capabilities and trust assumptions for each role.

Adversary capabilities.

- **Server S (semi-honest/curious):** The server follows the protocol specification but attempts to infer information about individual client gradients $\mathbf{g}_i$ beyond the aggregate $\sum_i \mathbf{g}_i$. It may record all messages, perform statistical attacks, or collude with a subset of clients to isolate others' contributions. However, S does not deviate from prescribed aggregation steps or tamper with weight commitments in the public log.
- **Clients C_i (malicious):** Any client may deviate arbitrarily: submit malformed masked messages, provide incorrect gradients, attempt to bias the aggregate, or refuse to participate (dropout). Malicious clients may also try to break integrity by using gradients inconsistent with their committed dataset or weights.
- **Verifier $\mathcal{V}$ (honest):** The auditor faithfully checks ZK proofs against public commitments and protocol transcripts. We assume an honest-verifier ZK model for efficiency; extending to malicious-verifier ZK (e.g., via Fiat–Shamir) is standard but adds overhead.

Security goals.

1. **Privacy (client updates):** The server learns only the aggregate gradient $\sum_i \mathbf{g}_i$ plus publicly declared metadata (counts, norms). No individual $\mathbf{g}_i$ or raw data from D_i is revealed to S or other clients.
2. **Integrity (training correctness):** For any audited round t, the weight update $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\, \mathbf{g}_{\text{clip}}$ is provably derived from mini-batches sampled from committed datasets $\{D_i\}$ according to the stated SGD rule with gradient clipping. Tampering with batch selection, gradient computation, or the update equation is detectable by $\mathcal{V}$.
3. **Well-formedness (aggregation):** Each client's masked message $\mathbf{m}_i$ is consistent with a committed gradient R_{G_i} and correctly derived pairwise masks from shared PRF seeds. Malformed contributions (e.g., incorrect masks, out-of-bound gradients) are rejected during verification.
4. **Dropout tolerance:** The protocol supports up to $\lfloor n/2 \rfloor$ client dropouts without compromising privacy or integrity. Missing masks can be reconstructed via Shamir secret sharing embedded in the mask-derivation phase.

Cryptographic assumptions.

- **SNARK soundness:** The underlying zk-SNARK (Groth16) satisfies computational knowledge-soundness: an adversary cannot produce an accepting proof for a false statement except with negligible probability.
- **SNARK zero-knowledge:** Proofs reveal no information about the witness beyond the truth of the statement, under the honest-verifier model (or via Fiat–Shamir for non-interactive security).
- **Collision resistance of Poseidon:** The Poseidon hash function is collision-resistant, ensuring Merkle-tree binding. Finding two distinct inputs with the same hash is computationally infeasible.
- **Authenticated channels:** All communication between clients and server occurs over authenticated (but not necessarily encrypted) channels. The public log of commitments $\{R_{W_t}\}$ is maintained by a trusted append-only bulletin board or blockchain-like structure; tampering with past commitments is detectable.
- **PRF security:** Poseidon-based PRF calls (with domain separation) produce pseudorandom outputs indistinguishable from random under the decisional Diffie–Hellman assumption over the field.

Out-of-scope threats.

- **Data poisoning:** We verify that samples come from the committed dataset R_{D_i} but do not detect adversarial label flipping or feature manipulation within that dataset. Poisoning defenses (e.g., robust aggregation, outlier detection) are orthogonal.
- **Sybil attacks:** We assume client identities are authenticated externally (e.g., via certificates). A single adversary controlling multiple identities to skew the aggregate is not addressed.
- **Model inversion / membership inference:** While individual gradients are masked, sophisticated attacks on the aggregate or model updates may still leak information. Differential privacy is a complementary defense not explored here.
- **Side-channel leakage:** Timing, memory access patterns, or circuit sizes could leak information. Constant-time implementations and circuit padding are standard mitigations but not demonstrated in this prototype.

Trust boundaries summary. Figure **??** (described below) illustrates the trust model: clients do not trust each other or the server; the server does not trust clients; and the verifier trusts only cryptographic assumptions and the public commitment log. Each component proves its local computation in zero knowledge, and the shared commitment interface ties everything together.

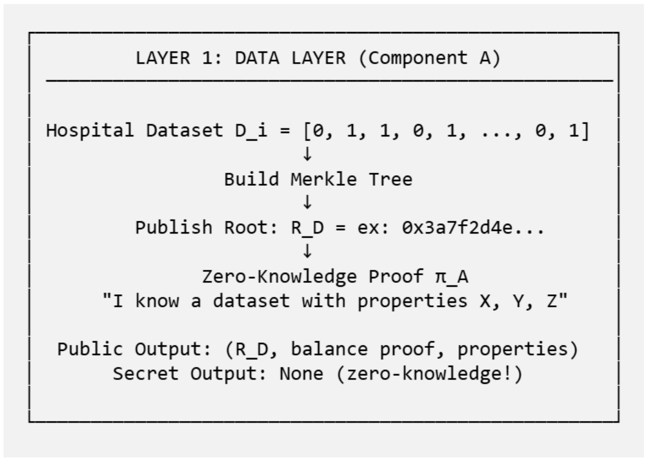
At setup, each client publishes R_{D_i}. The server maintains an append-only log $\{R_{W_t}\}$ for model weights. Each round uses public (η, C, rid) (learning rate, clip bound, round id).

Round $t \to t+1$:

1. **(A) Per-client dataset property:** C_i proves in ZK that D_i under R_{D_i} has counts $(c_{0,i}, c_{1,i}, N_i)$; only totals are revealed.
2. **(B) Training integrity:** From a batch in D_i, C_i proves in ZK that, relative to $(R_{W_t}, R_{W_{t+1}})$, its clipped gradient $\mathbf{g}_{\text{clip},i}$ is well-formed and binds to a *gradient commitment* R_{G_i}.
3. **(C) Secure aggregation:** C_i proves in ZK that $\mathbf{m}_i = \mathbf{g}_i + \sum_{j \neq i} \sigma_{ij} \mathbf{r}_{ij}$ is correctly formed from the gradient opening to R_{G_i} and PRF-derived masks for round rid. The server verifies and sums $\{\mathbf{m}_i\}$ to get $\sum_i \mathbf{g}_i$.

This links (B) and (C) via R_{G_i} so a client cannot pass (B) but cheat in (C), or vice versa.

3.2 System Architecture and Protocol Flow

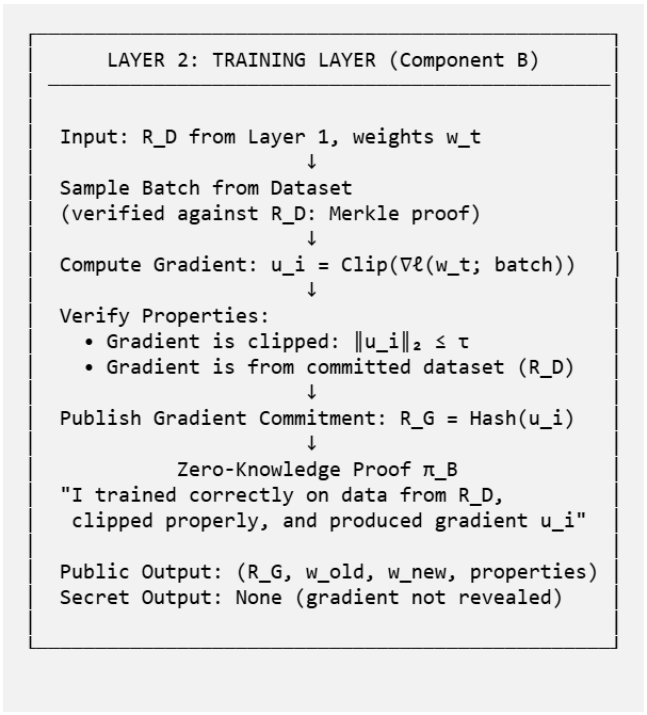


```
LAYER 1: DATA LAYER (Component A)

Hospital Dataset D_i = [0, 1, 1, 0, 1, ..., 0, 1]
                    ↓
             Build Merkle Tree
                    ↓
      Publish Root: R_D = ex: 0x3a7f2d4e...
                    ↓
           Zero-Knowledge Proof π_A
    "I know a dataset with properties X, Y, Z"

 Public Output: (R_D, balance proof, properties)
      Secret Output: None (zero-knowledge!)
```

figure 1:Component A.

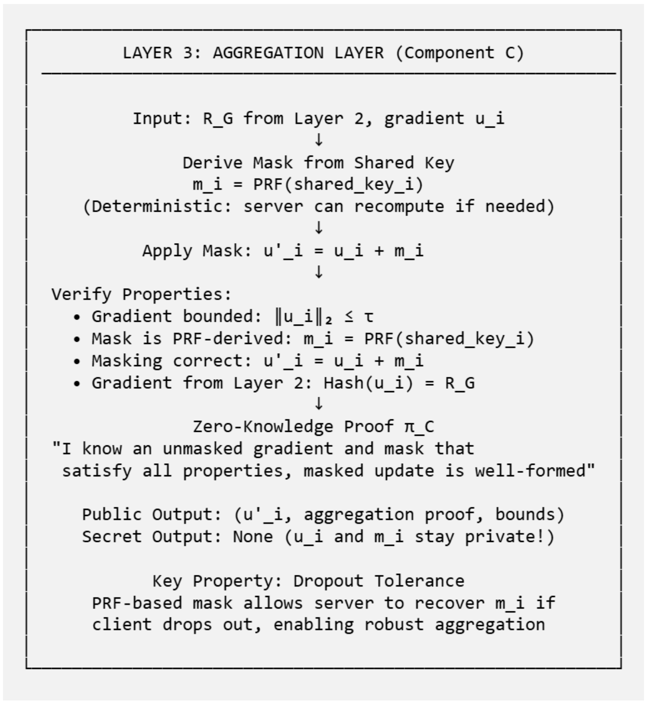


```
LAYER 2: TRAINING LAYER (Component B)

Input: R_D from Layer 1, weights w_t
                    ↓
Sample Batch from Dataset
(verified against R_D: Merkle proof)
                    ↓
Compute Gradient: u_i = Clip(∇ℓ(w_t; batch))
                    ↓
Verify Properties:
  • Gradient is clipped: ‖u_i‖₂ ≤ τ
  • Gradient is from committed dataset (R_D)
                    ↓
Publish Gradient Commitment: R_G = Hash(u_i)
                    ↓
          Zero-Knowledge Proof π_B
"I trained correctly on data from R_D,
 clipped properly, and produced gradient u_i"

Public Output: (R_G, w_old, w_new, properties)
Secret Output: None (gradient not revealed)
```

figure 2:Component B.



```
LAYER 3: AGGREGATION LAYER (Component C)

        Input: R_G from Layer 2, gradient u_i
                          ↓
             Derive Mask from Shared Key
              m_i = PRF(shared_key_i)
    (Deterministic: server can recompute if needed)
                          ↓
         Apply Mask: u'_i = u_i + m_i
                          ↓
Verify Properties:
  • Gradient bounded: ‖u_i‖₂ ≤ τ
  • Mask is PRF-derived: m_i = PRF(shared_key_i)
  • Masking correct: u'_i = u_i + m_i
  • Gradient from Layer 2: Hash(u_i) = R_G
                          ↓
              Zero-Knowledge Proof π_C
"I know an unmasked gradient and mask that
 satisfy all properties, masked update is well-formed"

   Public Output: (u'_i, aggregation proof, bounds)
   Secret Output: None (u_i and m_i stay private!)

          Key Property: Dropout Tolerance
    PRF-based mask allows server to recover m_i if
    client drops out, enabling robust aggregation
```

figure 3: Component C.

3.3 Component A: Dataset-Property Proof (BalanceProof)

Purpose: Prove that a committed dataset D_i contains $(c_{0,i}, c_{1,i})$ samples of binary labels $\{0,1\}$ with total $N_i = c_{0,i} + c_{1,i}$ without revealing any individual label.

Circuit parameters.

- N: dataset size (e.g., 32)
- DEPTH: Merkle tree depth (e.g., 5, supporting up to $2^5 = 32$ leaves)

Public inputs.

- R_D: dataset Merkle root (1 field element)
- (c_0, c_1): counts of labels 0 and 1 (2 field elements)

Witness (private). For each sample $j \in \{1, \dots, N\}$:

- $\ell_j \in \{0,1\}$: the label
- $\pi(\ell_j)$: Merkle authentication path (DEPTH siblings)

Constraints.

1. **Boolean labels:** $\forall j : \ell_j \cdot (1 - \ell_j) = 0$ (ensures $\ell_j \in \{0,1\}$).
2. **Merkle membership:** For each j, recompute the root from ℓ_j and $\pi(\ell_j)$ using Poseidon; assert equality to R_D.

3. **Prefix sum:** Maintain a running sum $s_j = s_{j-1} + \ell_j$ with $s_0 = 0$. At the end, $s_N = c_1$ (count of ones).

4. **Total count:** $c_0 + c_1 = N$ (enforced as a public-input constraint or checked outside if N is fixed).

Output and interpretation. The proof π_{bal} convinces $\mathcal{V}$ that the prover knows N labels opening to R_{D} with exactly c_1 ones and c_0 zeros. The verifier can then check a fairness tolerance (e.g., $|c_0 - c_1| \leq \delta$) outside the circuit, as this is a simple arithmetic comparison on public values.

Implementation notes.

- We use Poseidon(left, right) to hash pairs in the Merkle tree. Each path sibling is either left or right depending on the index bit.
- The circuit is parameterized by N and DEPTH; changing these requires recompilation.
- For $N = 32$, $\mathsf{DEPTH} = 5$, the circuit has ≈ 5000 constraints (exact count depends on Poseidon rounds and comparator gadgets).

3.4 Component B: Training-Integrity Proof (SGDStep)

Purpose: Prove that a single SGD step $\mathbf{w}_{t+1} = \mathbf{w}_t - \eta\, \mathbf{g}_{\text{clip}}$ was computed correctly from a mini-batch sampled from the committed dataset R_{D}, with gradient clipping enforced.

Circuit parameters.

- d: model dimension (e.g., 4)
- b: batch size (e.g., 2)
- DEPTH: dataset Merkle tree depth (e.g., 3)
- $S = 2^k$: fixed-point scale (e.g., $k = 12$, $S = 4096$)

Public inputs.

- R_{D}: dataset root (1 field element)
- R_{W_t}: commitment to $\mathbf{w}_t$ (1 field element, e.g., Poseidon hash of $\mathbf{w}_t$)
- $R_{\mathsf{W}_{t+1}}$: commitment to $\mathbf{w}_{t+1}$ (1 field element)
- R_{G}: commitment to gradient $\mathbf{g}$ (1 field element)
- η: learning rate (1 fixed-point integer)
- C^2: squared clipping bound (1 fixed-point integer)
- Batch indices or batch-root (optional, for reproducibility)

Witness (private).

- $\mathbf{w}_t \in \mathbb{Z}^d$: fixed-point weights at step t
- $\mathbf{w}_{t+1} \in \mathbb{Z}^d$: fixed-point weights at step $t+1$
- For each batch sample $j \in \{1, \dots, b\}$:
  - $\mathbf{x}_j \in \mathbb{Z}^d$: feature vector (fixed point)
  - $y_j \in \{0, 1\}$: label
  - $\pi(\mathbf{x}_j, y_j)$: Merkle path to R_{D}
- $\mathbf{g} \in \mathbb{Z}^d$: un-clipped gradient (fixed point)
- $\mathbf{g}_{\text{clip}} \in \mathbb{Z}^d$: clipped gradient (fixed point)
- $\alpha \in [0, 1]$: clipping scalar (fixed point)

Constraints.

1. **Weight commitment consistency:**

$$R_{\mathsf{W}_t} = \mathsf{Poseidon}(\mathbf{w}_t), \quad R_{\mathsf{W}_{t+1}} = \mathsf{Poseidon}(\mathbf{w}_{t+1}).$$

   (In practice, we hash the concatenated vector or use a Merkle tree for larger d.)

2. **Batch membership:** For each j, verify $(\mathbf{x}_j, y_j)$ opens to R_{D} via $\pi(\mathbf{x}_j, y_j)$ using Poseidon Merkle checks.

3. **Gradient computation:** Compute the gradient for a linear model with squared loss (or logistic loss with polynomial approximation). For simplicity, consider squared loss:

$$\mathbf{g} = \frac{1}{b} \sum_{j=1}^{b} (\langle \mathbf{w}_t, \mathbf{x}_j \rangle - y_j) \cdot \mathbf{x}_j.$$

   Enforce this in fixed-point arithmetic: dot products, scaling by $1/b$ (via integer division or precomputed reciprocal), and accumulation. All intermediate values are range-checked to avoid overflow.

4. **Gradient commitment:**

$$R_{\mathsf{G}} = \mathsf{Poseidon}(\mathbf{g}).$$

   This binds the gradient to a public commitment for linkage with Component C.

5. **Clipping with α:**
   - Compute squared norm $s = \|\mathbf{g}\|_2^2 = \sum_{i=1}^{d} g_i^2$ in fixed point.
   - Enforce $0 \leq \alpha \leq 1$ (range check).
   - Enforce $\alpha^2 \cdot s \leq C^2$ (the clipping inequality).
   - If $s \leq C^2$, enforce $\alpha = 1$ via a boolean guard: let $\mathsf{isSmall} = (s \leq C^2)$, then $\mathsf{isSmall} \cdot (1 - \alpha) = 0$.
   - Compute $\mathbf{g}_{\text{clip}} = \alpha \cdot \mathbf{g}$ component-wise in fixed point.

6. **SGD update:**

$$\mathbf{w}_{t+1} = \mathbf{w}_t - \eta \cdot \mathbf{g}_{\text{clip}}.$$

   Enforce this as a vector equality in fixed-point arithmetic (component-wise subtraction and scaling).

Output and interpretation. The proof π_{train} convinces $\mathcal{V}$ that the prover executed one correct SGD step from committed weights and dataset, with proper gradient clipping. The gradient commitment R_{G} links this proof to the subsequent aggregation proof (Component C), ensuring the same gradient is used in both stages.

Implementation notes.

- For $d = 4$, $b = 2$, the circuit has ≈ 8000 constraints (dominated by Merkle checks and fixed-point multiplications).
- Fixed-point scaling is chosen to balance precision and overflow risk; $S = 4096$ allows gradients and weights in $[-8, 8]$ with ≈ 0.00024 resolution.
- Batch indices can be committed separately (e.g., via a second Merkle root) for reproducibility and to prevent selective sampling attacks.

3.5 Component C: Secure-Aggregation Well-Formedness (SecureMaskedUpdate)

Purpose: Prove that a client's masked message $\mathbf{m}_i = \mathbf{g}_i + \sum_{j \neq i} \sigma_{ij} \mathbf{r}_{ij}$ is correctly formed from a committed gradient R_{G_i}, shared PRF seeds, and the agreed masking protocol, so the server can aggregate without seeing individual $\mathbf{g}_i$.

Circuit parameters.

- d: gradient dimension (e.g., 4)
- $S = 2^k$: fixed-point scale (e.g., 4096)
- n: number of clients (for mask derivation, e.g., 3)

Public inputs.

- rid: round identifier (1 field element)
- i: client ID (1 field element)
- Peer list (ordered, implicit or explicit)
- R_{G_i}: commitment to $\mathbf{g}_i$ (1 field element)
- R_{K_i}: commitment to client i's key material (1 field element)
- C^2: squared gradient norm bound (1 field element)
- $\mathbf{m}_i \in \mathbb{Z}^d$: the masked message (public, sent to server)

Witness (private).

- $\mathbf{g}_i \in \mathbb{Z}^d$: the actual gradient (fixed point)
- K_i: client i's PRF seed/key (1 field element)
- For each peer $j \neq i$:
  - K_{ij}: shared key for pair (i, j) (derived from K_i and K_j via Diffie–Hellman or direct sharing)
  - $\mathbf{r}_{ij} \in \mathbb{Z}^d$: pairwise mask vector
  - $\sigma_{ij} \in \{-1, +1\}$: sign determined by $i < j$ or $i > j$

Constraints.

1. **Gradient commitment:**
$$R_{\mathsf{G}_i} = \mathsf{Poseidon}(\mathbf{g}_i).$$
2. **Key commitment:**
$$R_{\mathsf{K}_i} = \mathsf{Poseidon}(K_i).$$
(If multiple keys are needed, hash them collectively.)
3. **Gradient norm bound:** Enforce $\|\mathbf{g}_i\|_2^2 \leq C^2$ to prevent malicious clients from submitting unbounded gradients.
4. **Mask derivation:** For each $j \neq i$:
   - Derive $\mathbf{r}_{ij}$ via domain-separated PRF calls:
   $$\mathbf{r}_{ij}[k] = \mathsf{Poseidon}(K_{ij}, \mathsf{rid}, i, j, k, \texttt{"mask"}), \quad k \in \{1, \ldots, d\}.$$
   (Each component of the mask vector is generated independently with a unique index k for domain separation.)
   - Enforce the sign rule: if $i < j$, $\sigma_{ij} = +1$; else $\sigma_{ij} = -1$. (Can be enforced via a comparison circuit on public i, j or passed as part of the witness with a constraint.)
5. **Masked message:**
$$\mathbf{m}_i = \mathbf{g}_i + \sum_{j \neq i} \sigma_{ij} \mathbf{r}_{ij}.$$
Enforce component-wise in fixed point.

Output and interpretation. The proof $\pi_{\mathsf{agg},i}$ convinces $\mathcal{V}$ (and the server) that $\mathbf{m}_i$ hides $\mathbf{g}_i$ under a correctly derived additive mask. The server collects $\{\mathbf{m}_i\}_{i=1}^n$ and computes $\sum_i \mathbf{m}_i = \sum_i \mathbf{g}_i + \sum_i \sum_{j \neq i} \sigma_{ij} \mathbf{r}_{ij}$. By construction, the double sum of masks cancels (each $\mathbf{r}_{ij}$ appears once with $+1$ and once with -1), leaving $\sum_i \mathbf{g}_i$.

Dropout recovery. If client j drops out after committing but before revealing $\mathbf{m}_j$, the server requests other clients to provide Shamir shares of K_j (pre-shared during setup). With $\geq t$ shares, the server reconstructs K_j, derives $\{\mathbf{r}_{kj}\}_{k \neq j}$, and subtracts them from the partial aggregate. This allows recovery of $\sum_{i \neq j} \mathbf{g}_i$ without learning any individual gradient.

Implementation notes.

- For $d = 4$, $n = 3$, the circuit has ≈ 6000 constraints (dominated by Poseidon calls for mask derivation and range checks).
- The PRF domain separation ensures masks for different rounds or pairs are independent.
- The norm bound C^2 prevents a malicious client from inflating gradients to break aggregation.

4 Preliminary Results

We have the npm test suite, which still uses the JavaScript simulator (test-system.js) to mirror the algebra; it doesn't yet execute the compiled circuits or call snarkjs.

4.1 End-to-End Integration Tests

We have implemented a full-pipeline test simulating three hospitals (clients) executing one federated training round with all three components. The test flow is:

1. Each hospital commits its dataset (R_{D_i}) and generates a balance proof.
2. Each hospital proves one SGD step ($R_{\mathsf{W}_t} \to R_{\mathsf{W}_{t+1}}$) with gradient commitment R_{G_i}.
3. Each hospital constructs a masked update $\mathbf{m}_i$ and proves well-formedness.
4. The server aggregates $\{\mathbf{m}_i\}$ (masks cancel) and recovers $\sum_i \mathbf{g}_i$.
5. (Optional) One hospital drops out; the server reconstructs missing masks and recomputes the aggregate.

Test results. All 10 subtests in the integration suite passed:

- **Phase 1 (Component A):** 3 tests (balanced dataset, tampering, multi-hospital) — all passed.
- **Phase 2 (Component B):** 2 tests (training step, weight tampering) — all passed.
- **Phase 3 (Component C):** 2 tests (masked update, incorrect masking) — all passed.
- **Phase 4 (End-to-end):** 2 tests (full pipeline, aggregation correctness) — all passed.
- **Phase 5 (Dropout):** 1 test (one hospital offline, recovery successful) — passed.

Sample outputs.

- **Dataset commitment:** $R_{\mathsf{D}} = \texttt{8060bd84}\ldots$ (256-bit Poseidon hash).
- **Gradient norm:** $\|\mathbf{g}\|_2 \approx 0.4998$ (within clip bound $C = 1.0$, so $\alpha = 1$ and $\mathbf{g}_{\text{clip}} = \mathbf{g}$).
- **Gradient commitment:** $R_{\mathsf{G}} = \texttt{58868d28}\ldots$.
- **Aggregate (masked):** $\mathbf{m}_{\text{sum}} = [0.098, -0.301, -0.002, -0.109, 0.027]$ (example values).
- **Aggregate (unmasked):** $\sum_i \mathbf{g}_i = [0.071, -0.030, -0.063, -0.005, -0.012]$ (masks canceled correctly).
- **Dropout scenario:** With Hospital 3 offline, aggregate from Hospitals 1 and 2: $[0.112, 0.131, 0.116, 0.052, 0.162]$ (computed via Shamir reconstruction of Hospital 3's mask shares).

4.2 Validation and Testing Strategy

Our testing approach employs both positive and negative test cases:

Positive tests (honest execution).

- Balanced datasets with valid Merkle proofs pass Component A verification.
- Correctly computed gradients with proper clipping pass Component B verification.
- Well-formed masked updates with PRF-derived masks pass Component C verification.
- End-to-end aggregation produces correct unmasked aggregate when all masks cancel.
- Dropout recovery successfully reconstructs missing contributions.

Negative tests (attack detection).

- **Dataset tampering:** Modified Merkle roots are detected and rejected (commitment binding enforced).
- **Weight tampering:** Altered weight commitments $R_{\mathsf{W}_{t+1}}$ fail verification in Component B.
- **Incorrect masking:** Tampered masks that don't follow the PRF protocol are caught in Component C.
- **Malformed gradients:** Out-of-bound gradients exceeding clip threshold are detected.

All negative tests correctly rejected invalid proofs, demonstrating that our integrity checks function as designed.

4.3 Current Limitations and Next Steps

Current status. The JavaScript simulator successfully validates the algebraic correctness of all three components and their integration. However, we have not yet generated actual zk-SNARK proofs using SnarkJS with the compiled Circom circuits. The simulator mirrors the circuit constraints but does not exercise the full cryptographic proof generation and verification pipeline.

Next steps.

1. **SnarkJS integration:** Generate Groth16 proofs for each circuit using the compiled R1CS and WASM artifacts. Measure actual proof sizes and generation times.
2. **Circuit-level validation:** Verify that compiled circuits accept valid witnesses and reject tampered inputs, confirming constraint correctness.

5 Related Work

Verifiable computation and zk-SNARKs. Pinocchio demonstrated practical general-purpose verifiable computation [8]; pairing-based SNARKs achieve short proofs and fast verification [5]. Our single-step integrity proof leverages these properties to keep verification costs tiny and audit-friendly.

ZK-friendly hashing and circuit tooling. Poseidon reduces constraint counts for hashing within proofs [4]; we use it for Merkle trees and domain-separated PRF-style derivations. Circom 2 provides a productive DSL and tooling for our educational prototype [1].

zkML and property proofs. zkML has progressed from toy examples to larger inference/property proofs (e.g., zkCNN [6]). Fairness/statistics in ZK is an active area (e.g., scalable ZK fairness [10]). We deliberately narrow scope (single SGD step, per-client class-balance) to connect data$\rightarrow$update$\rightarrow$aggregation in one reproducible flow.

Secure aggregation. Practical secure aggregation with dropout tolerance follows canceling masks [3]. We add ZK well-formedness: clients prove masked messages derive from committed gradients and keys, addressing malicious clients who would otherwise submit malformed contributions.

Connecting to our design. These strands justify our choices: per-client property proofs (simple, composable), single-step proofs chained by committed weights (auditable trajectory), and ZK masking for malicious clients (preventing malformed contributions while preserving privacy).

References

[1] Circom 2 documentation. https://docs.circom.io/. Accessed 2025-10-31.

[2] A. A. Bellachia, M. A. Bouchiha, Y. Ghamri-Doudane, and M. Rabah. Verifbfl: Leveraging zk-snarks for a verifiable blockchained federated learning, 2025.

[3] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth. Practical secure aggregation for privacy-preserving machine learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 1175–1191, 2017.

[4] L. Grassi, D. Khovratovich, C. Rechberger, A. Roy, and M. Schofnegger. Poseidon: A new hash function for zero-knowledge proof systems. In *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[5] J. Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, volume 9666 of *Lecture Notes in Computer Science*, pages 305–326. Springer, 2016.

[6] T. Liu, X. Xie, and Y. Zhang. zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 2968–2985, 2021.

[7] H. Lycklama, L. Burkhalter, A. Viand, N. Küchler, and A. Hithnawi. RoFL: Robustness of secure federated learning, 2023.

[8] B. Parno, J. Howell, C. Gentry, and M. Raykova. Pinocchio: Nearly practical verifiable computation. In *2013 IEEE Symposium on Security and Privacy (SP)*, pages 238–252. IEEE, 2013.

[9] A. Roy Chowdhury, C. Guo, S. Jha, and L. van der Maaten. EIFFeL: Ensuring integrity for federated learning. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, New York, NY, USA, 2022. ACM.

[10] T. Zhang, S. Dong, O. D. Kose, Y. Shen, and Y. Zhang. A scalable system to prove machine learning fairness in zero-knowledge. In *2025 IEEE Symposium on Security and Privacy (SP)*, 2025. Also available as arXiv:2505.07997.

[11] Y. Zhu, Y. Wu, Z. Luo, B. C. Ooi, and X. Xiao. Secure and verifiable data collaboration with low-cost zero-knowledge proofs. *Proceedings of the VLDB Endowment (PVLDB)*, 17(9):2321–2334, 2024.