Verifiable Federated Training with Zero-Knowledge Proofs (ZK-FL)

Tarek Salama
Purdue University

Zeyad Elshafey
Purdue University

Ahmed Elbehiry
Purdue University

Abstract

Federated learning enables collaborative model training across distributed data sources without centralizing raw data, addressing critical privacy concerns in healthcare, finance, and government domains. However, standard federated learning protocols remain vulnerable to several attack vectors: malicious clients may submit poisoned gradients, curious servers may infer private information from gradient updates, and participants lack mechanisms to verify training correctness. We present a comprehensive cryptographic framework integrating zero-knowledge proofs, Merkle tree commitments, and secure aggregation protocols to address these challenges. Our system comprises three interconnected components: (1) a balance proof circuit verifying dataset properties without revealing individual data points, (2) a training integrity circuit proving gradient computation correctness with norm bounding, and (3) a secure aggregation circuit enabling privacy-preserving gradient aggregation where the server learns only the aggregate update. We implement our framework using Circom and Groth16, achieving proof generation times of approximately 14 seconds per client for a complete training round. Our evaluation demonstrates successful prevention of gradient manipulation attacks while maintaining computational practicality. The cryptographic binding between components ensures end-to-end integrity guarantees throughout the federated learning pipeline, providing a foundation for trustworthy distributed machine learning. GitHub Repo

1 Introduction

Motivation. The proliferation of machine learning applications across sensitive domains has created a fundamental tension between the need for large-scale training data and increasingly stringent privacy regulations. Healthcare institutions developing diagnostic models require access to diverse patient populations, yet regulations such as HIPAA and GDPR impose strict constraints on data sharing. Similarly, financial institutions building fraud detection systems cannot freely exchange customer transaction data due to competitive concerns and regulatory requirements.

Federated learning [1] emerged as a promising paradigm to address this challenge by enabling collaborative model training without centralizing raw data. Rather than transmitting raw data, clients compute gradient updates locally and send only these updates to the server. This approach has been successfully deployed in production systems, including Google's keyboard prediction [2] and Apple's on-device machine learning.

Problem statement. Despite its privacy benefits, federated learning introduces new security challenges. First, gradient updates themselves can leak substantial information—Zhu et al. [3] demonstrated that individual training examples can be reconstructed from gradients. Second, malicious clients can submit carefully crafted gradient updates designed to poison the global model [4]. Third, the server observes individual gradient updates, creating a trusted aggregator assumption that may not hold. Finally, there exists no mechanism for verifying that clients actually performed the claimed computation on legitimate data.

Gap in existing work. Existing approaches typically address one stage of the pipeline in isolation—fairness *or* training integrity *or* aggregation—and rarely offer a cohesive recipe connecting data verification, training correctness, and secure aggregation. Moreover, training-integrity proofs often gloss over dataset membership verification, while secure aggregation protocols typically lack checks that each masked message was formed from a committed gradient following agreed rules.

Our solution. We propose a modular architecture comprising three zero-knowledge proof circuits that work together through shared cryptographic commitments:

Component A (Balance Proof): Clients prove properties of their local datasets without revealing individual data points.

Specifically, clients prove class distribution properties (counts c_0, c_1 for binary labels) while verifying Merkle tree membership for all samples.

Component B (Training Integrity Proof): This circuit proves that a client's submitted gradient satisfies two critical properties: (1) the gradient was computed using samples from the committed dataset, verified through Merkle proofs, and (2) the gradient's squared norm does not exceed a specified threshold τ^2. A key technical contribution is our sign-magnitude decomposition approach that enables sound norm verification in finite field arithmetic.

Component C (Secure Aggregation Proof): Clients mask their gradients using pairwise random values that cancel upon aggregation. The circuit proves that the masked gradient was correctly constructed from the committed gradient and properly derived masking values.

The three components are cryptographically bound through shared Poseidon-Merkle commitments, ensuring that proofs generated for one dataset cannot be reused for another.

Contributions. 25em

1. **Integrated Framework Design:** We present the first complete framework integrating dataset property verification, training integrity proofs, and secure aggregation within a unified zero-knowledge proof system for federated learning.
2. **Sound Gradient Norm Verification:** We identify and resolve a critical vulnerability in naive approaches to gradient clipping verification within finite field arithmetic, providing a provably sound construction based on sign-magnitude decomposition.
3. **Cryptographic Binding Mechanism:** We develop a commitment-based binding approach ensuring all proof components reference the same underlying dataset and gradient.
4. **Complete Implementation and Evaluation:** We provide a fully functional implementation using Circom and snarkjs, demonstrating practical feasibility with proof generation times of $\sim$14 seconds per client and verification times of $\sim$25 milliseconds.

Results preview. Our experimental evaluation on a consumer-grade machine (Intel Core i7, 16GB RAM) demonstrates that the framework successfully blocks all tested attack vectors including dataset substitution, gradient inflation, and mask manipulation. Proof sizes remain constant at 192 bytes per component regardless of circuit complexity, enabling efficient communication.

Limitations. Our circuits target clarity and reproducibility over generality: small linear models (4 dimensions), limited dataset sizes (8 samples), and fixed circuit parameters requiring recompilation for different configurations. We do not address differential privacy or multi-round composition in this work.

2 Background

2.1 Finite Field Arithmetic

Zero-knowledge proof systems operate over finite fields rather than integers or real numbers. A prime field $\mathbb{F}_p$ is the set of integers $\{0, 1, \dots, p-1\}$ with addition and multiplication modulo p. Our implementation uses the BN254 scalar field with a 254-bit prime providing approximately 128 bits of security.

A critical observation is that finite fields have no native notion of "negative" numbers. The value $-x$ in $\mathbb{F}_p$ is represented as $p-x$. This creates challenges for operations depending on sign, such as computing squared norms of vectors with negative components, which we address through sign-magnitude decomposition in Section **??**.

2.2 Poseidon Hash Function

Standard hash functions like SHA-256 are expensive within ZK circuits due to bitwise operations. Poseidon [5] is designed for efficient arithmetic circuit implementation, operating directly on field elements using only field additions and multiplications. The result is a hash function requiring approximately 250 constraints per invocation, compared to over 25,000 for SHA-256—a $100\times$ improvement critical for practical proof generation.

2.3 Merkle Trees

A Merkle tree over dataset $D = \{d_0, \dots, d_{n-1}\}$ is a complete binary tree where each leaf contains $H(d_i)$ and each internal node contains the hash of its children's concatenation. The root provides a single commitment to the entire dataset. Membership proofs require only $O(\log n)$ sibling hashes, enabling efficient verification within ZK circuits.

2.4 Zero-Knowledge Proofs and Groth16

A zero-knowledge proof system enables a prover to convince a verifier that a statement is true without revealing any information beyond the statement's validity. We employ the Groth16 [6] proving system, which achieves optimal proof size among pairing-based zk-SNARKs: three elliptic curve group elements totaling approximately 192 bytes, with verification requiring only three bilinear pairings completing in approximately 10 milliseconds.

Groth16 operates on Rank-1 Constraint Systems (R1CS), where each constraint enforces $(\mathbf{a}_i \cdot \mathbf{s}) \times (\mathbf{b}_i \cdot \mathbf{s}) = (\mathbf{c}_i \cdot \mathbf{s})$ for witness vector $\mathbf{s}$. We use Circom [7] to compile high-level circuit descriptions to R1CS.

2.5 Federated Learning and Secure Aggregation

In federated learning [1], clients compute gradient updates locally and a server aggregates them. Secure aggregation [8] uses pairwise random masks that cancel upon aggregation: client i submits $m_i = g_i + \sum_{j \neq i} \sigma_{ij} r_{ij}$ where $\sigma_{ij} = +1$ if $i < j$ and $\sigma_{ij} = -1$ otherwise. When summing across all clients, each mask appears with opposite signs and cancels, leaving $\sum_i m_i = \sum_i g_i$.

3 Methodology

3.1 Threat Model

We consider a federated learning system with n clients and a central aggregation server.

Malicious clients. Up to $n-1$ clients may be fully malicious, capable of submitting arbitrary gradient updates not computed from claimed training data, attempting model poisoning, colluding with other malicious clients, or claiming false dataset properties.

Curious server. The aggregation server follows the protocol correctly (semi-honest) but attempts to learn private information from observed messages, including recording and analyzing all gradient updates and attempting gradient inversion attacks.

Security goals. Our framework provides: (1) *training integrity*—gradients are provably derived from committed datasets using stated algorithms; (2) *gradient privacy*—the server learns only the aggregate; (3) *dataset property verification*—class balance is verified without revealing data; (4) *gradient norm bounding*—cryptographic enforcement of clipping thresholds.

Cryptographic assumptions. We assume Groth16 soundness and zero-knowledge, Poseidon collision resistance, and an honest trusted setup ceremony. We do *not* protect against data poisoning within committed datasets, side-channel attacks, or attacks on the final trained model.

3.2 System Architecture

Figure 1 illustrates our three-component architecture. All components share cryptographic commitments: `root_D` (dataset), `root_G` (gradient), and `root_W` (weights).

```
+-----------------------------------+
|           CLIENT i                |
|  +--------+  +--------+  +-----+  |
|  | Comp A |  | Comp B |  |Comp |  |
|  |Balance |  |Training|  |  C  |  |
|  | Proof  |  | Proof  |  |SecAg|  |
|  +---+----+  +---+----+  +--+--+  |
|      |           |          |     |
|      +-----+-----+-----+----+     |
|            | root_D, root_G       |
+------------|----------------------+
             v
     +---------------+
     |    SERVER     |
     | Verify proofs |
     | Check binding |
     | Aggregate sum |
     +---------------+
```

Figure 1: System architecture showing three proof components bound through shared commitments.

Protocol flow. A complete federated learning round proceeds in five phases: (1) Setup: server distributes model weights and parameters; (2) Balance Verification: clients prove dataset properties; (3) Local Training: clients compute gradients and generate training proofs; (4) Secure Aggregation: clients generate masked updates with proofs; (5) Aggregation: server verifies all proofs, checks binding, and computes aggregate.

3.3 Cryptographic Binding Mechanism

The security of our framework relies on binding the three proof components through shared commitments. The dataset commitment is computed as:

$$\texttt{root\_D} = \text{MerkleRoot}\left(\{H(x_i \| y_i)\}_{i=0}^{N-1}\right)$$

where H is Poseidon and both Components A and B compute leaves identically. The gradient commitment binds values to client identity and round number:

$$\texttt{root\_G} = H(\texttt{client\_id}, \texttt{round}, H(\mathbf{g}))$$

Both Components B and C verify against this commitment.

3.4 Component A: Balance Proof Circuit

Component A proves that a committed dataset contains specified counts (c_0, c_1) of binary labels without revealing individual labels or features.

Public inputs. Client ID, Merkle root `root`, counts c_0, c_1, and total N.

Algorithm 1 Balance Proof Circuit

```
Require: Features $\mathbf{x}[N]$, labels $y[N]$, siblings, pathIndices
Ensure: Valid proof that dataset has $c_0$ zeros and $c_1$ ones
 1: $sum \leftarrow 0$
 2: for $i = 0$ to $N-1$ do
 3:     // Constraint 1: Label booleanity
 4:     assert $y[i] \cdot (y[i]-1) = 0$
 5:     // Constraint 2: Merkle membership
 6:     $leaf \leftarrow \text{Poseidon}(\mathbf{x}[i] \| y[i])$
 7:     $computed \leftarrow \text{MerkleVerify}(leaf, \text{siblings}[i], \text{pathIndices}[i])$
 8:     assert $computed = \texttt{root}$
 9:     // Accumulate label count
10:     $sum \leftarrow sum + y[i]$
11: end for
12: // Constraint 3: Count accuracy
13: assert $sum = c_1$
14: // Constraint 4: Total consistency
15: assert $c_0 + c_1 = N$
```

Private witness. Feature vectors, labels, and Merkle authentication paths for all N samples.

Algorithm 1 presents the constraint logic.

3.5 Component B: Training Integrity Circuit

Component B proves that the submitted gradient was correctly computed from the committed dataset and satisfies the clipping bound $\|\mathbf{g}\|^2 \leq \tau^2$.

Public inputs. Client ID, round number, dataset root `root_D`, gradient commitment `root_G`, weight commitment `root_W`, and clipping threshold τ^2.

Private witness. Weights $\mathbf{w}$, batch features and labels with Merkle paths, gradient components, and sign-magnitude decomposition.

Key technical challenge: signed arithmetic. Zero-knowledge circuits operate over finite fields where subtraction of a larger value from a smaller produces $p - |a-b|$ rather than $-|a-b|$. A naive norm computation $\sum_j g_j^2$ fails when g_j represents a negative number: $(p-5)^2 \approx p^2 \gg \tau^2$.

Sound approach: sign-magnitude decomposition. We decompose each gradient component into non-negative parts:

$$g[j] = g_{\text{pos}}[j] - g_{\text{neg}}[j]$$

with the constraint that at most one is non-zero:

$$\forall j: \quad g_{\text{pos}}[j] \cdot g_{\text{neg}}[j] = 0$$

Algorithm 2 Training Integrity Circuit

```
Require: Weights $\mathbf{w}$, batch $(\mathbf{x}, y)$, gradient decomposition
Ensure: Valid proof of correct gradient computation with norm bound
 1: // Step 1: Weight commitment
 2: assert $\texttt{root\_W} = \text{VectorHash}(\mathbf{w})$
 3: // Step 2: Batch Merkle membership
 4: for each sample $i$ in batch do
 5:     $leaf \leftarrow \text{VectorHash}(\mathbf{x}[i] \| y[i])$
 6:     assert $\text{MerkleVerify}(leaf) = \texttt{root\_D}$
 7: end for
 8: // Step 3: Gradient clipping (sign-magnitude)
 9: $normSq \leftarrow 0$
10: for $j = 0$ to $DIM - 1$ do
11:     assert $g_{\text{pos}}[j] \cdot g_{\text{neg}}[j] = 0$
12:     $normSq \leftarrow normSq + g_{\text{pos}}[j]^2 + g_{\text{neg}}[j]^2$
13: end for
14: assert $normSq \leq \tau^2$
15: // Step 4: Gradient correctness
16: for each sample $i$ do
17:     $pred[i] \leftarrow \langle \mathbf{w}, \mathbf{x}[i] \rangle$
18:     $error[i] \leftarrow pred[i] - y[i] \cdot \text{PRECISION}$
19:     $sampleGrad[i] \leftarrow error[i] \cdot \mathbf{x}[i]$
20: end for
21: $sumGrad \leftarrow \sum_i sampleGrad[i]$
22: // Verify division
23: assert $sumGrad = \mathbf{g} \cdot \text{DIVISOR} + remainder$
24: // Step 5: Gradient commitment
25: assert $\texttt{root\_G} = \text{Poseidon}(\texttt{id}, \texttt{round}, \text{VectorHash}(\mathbf{g}))$
```

The squared norm is then computed correctly as:

$$\|\mathbf{g}\|^2 = \sum_j \left(g_{\text{pos}}[j]^2 + g_{\text{neg}}[j]^2\right)$$

Algorithm 2 presents the complete constraint logic.

3.6 Component C: Secure Aggregation Circuit

Component C implements verifiable secure aggregation using pairwise masking. Each pair (i,j) derives identical masks from shared keys:

$$r_{ij}[k] = \text{Poseidon}(K_{ij}, \texttt{round}, \min(i,j), \max(i,j), k)$$

The masked update is:

$$m_i = g_i + \sum_{j \neq i} \sigma_{ij} \cdot r_{ij}$$

where $\sigma_{ij} = +1$ if $i < j$ and -1 otherwise. Upon aggregation, masks cancel: $\sum_i m_i = \sum_i g_i$.

Algorithm 3 presents the constraint logic.

Algorithm 3 Secure Aggregation Circuit

```
Require: Gradient $\mathbf{g}$, shared keys $K$, peer IDs
Ensure: Valid proof that $\mathbf{m}$ is correctly masked
// Step 1: Gradient commitment
$gradHash \leftarrow \text{VectorHash}(\mathbf{g})$
assert $\texttt{root\_G} = \text{Poseidon}(\texttt{id}, \texttt{round}, gradHash)$
// Step 2: Key material commitment
assert $\texttt{root\_K} = \text{Poseidon}(masterKey, K[0], \ldots)$
// Step 3: Gradient norm bound
assert $\|\mathbf{g}\|^2 \leq \tau^2$
// Step 4: Derive pairwise masks
$accumulated \leftarrow \mathbf{g}$
for each peer $j$ do
    $minId \leftarrow \min(\texttt{id}, peerId[j])$
    $maxId \leftarrow \max(\texttt{id}, peerId[j])$
    for $k = 0$ to $DIM - 1$ do
        $r[j][k] \leftarrow \text{Poseidon}(K[j], \texttt{round}, minId, maxId, k)$
    end for
    $\sigma \leftarrow (\texttt{id} < peerId[j])\ ?\ +1 : -1$
    $accumulated \leftarrow accumulated + \sigma \cdot r[j]$
end for
// Step 5: Verify masked update
assert $\mathbf{m} = accumulated$
```

3.7 Implementation Details

We implement our framework using Circom 2.0.0 and snarkjs 0.7.0. The `VectorHash` function handles vectors by chunking into groups of 16 elements (Poseidon's maximum input size) and hashing recursively. Merkle verification uses the standard path-walking approach with binary selection based on path indices. Fixed-point arithmetic uses PRECISION$= 1000$, with division verified through multiplication constraints.

4 Evaluation

4.1 Experimental Setup

Hardware and software platform. All experiments were conducted on a consumer-grade laptop with Intel Core i7-10750H processor (6 cores, 2.6 GHz base), 16GB DDR4 RAM, running Windows 11. We used Node.js v18.17.0, Circom 2.0.0, and snarkjs 0.7.0 with the Groth16 proving system.

Benchmark configuration. We evaluate using a synthetic binary classification dataset with parameters shown in Table 1. The small scale enables tractable proof generation while demonstrating all security properties.

Dataset generation. Each client's dataset is synthetically generated with controlled class balance. Features are random integers in $[0, 1000)$, and labels are binary. Client 1 has a 4: 4 class split, Client 2 has 3:5, and Client 3 has 5:3, demonstrating varied but verified balance properties.

Table 1: Experimental configuration parameters.

Parameter	Description	Value
N	Samples per client	8
MODEL_DIM	Feature dimension	4
DEPTH	Merkle tree depth	3
BATCH_SIZE	Training batch size	8
NUM_CLIENTS	Federation size	3
τ^2	Clipping threshold	10^8
PRECISION	Fixed-point scale	1000
Field	Prime field	BN254

Table 2: Circuit constraint counts by component.

Component	Constraints	Wires	R1CS Size
A (Balance)	$\sim$12,500	$\sim$13,200	1. 2 MB
B (Training)	$\sim$18,700	$\sim$19,800	1.8 MB
C (SecureAgg)	$\sim$8,200	$\sim$8,900	0.9 MB
Total	$\sim$39,400	$\sim$41,900	3.9 MB

4.2 Circuit Complexity

Table 2 presents the constraint counts for each circuit component.

Component B (Training Integrity) has the highest constraint count due to gradient correctness verification requiring matrix operations and per-sample Merkle proofs ($8 \times 3 \times 153 \approx$ 3,672 constraints from Poseidon alone).

4.3 Performance Results

Table 3 presents timing measurements averaged over 10 runs.

Key observations. *Proof generation dominates latency* at $\sim$14 seconds per client. This aligns with known Groth16 characteristics where the prover performs $O(m \log m)$ group exponentiations. *Verification is extremely fast* at $\sim$25ms total per client, adding negligible server overhead. *Proof size is constant* at 192 bytes per component regardless of circuit complexity.

Table **??** shows the end-to-end timing for a complete round with 3 clients.

4.4 Scalability Analysis

Table 5 shows how performance scales with dataset size.

Constraint counts scale linearly with N, and proof generation time scales approximately linearly with logarithmic factors from FFT operations. For production deployments

Table 3: Performance metrics (mean ± std over 10 runs).

Comp.	**Witness (ms)**	**Prove (s)**	**Verify (ms)**	**Proof (bytes)**
A	45 ± 8	4.2 ± 0.3	8.1 ± 0.5	192
B	78 ± 12	6.8 ± 0.5	9.2 ± 0.6	192
C	32 ± 5	3.1 ± 0.2	7.8 ± 0.4	192
Total	155 ± 25	14.1 ± 1.0	25.1 ± 1.5	576

Table 4: End-to-end round timing (3 clients).

Phase	**Time (s)**
Dataset generation	0.3
Merkle tree construction	0.4
Balance proofs (3×)	12.6
Training proofs (3×)	20.4
SecureAgg proofs (3×)	9.3
All verifications (9×)	0.08
Aggregation	0.02
Total round time	∼43

with $N = 1000$ samples, proof generation would require approximately 25–30 minutes per client.

4.5 Security Validation

We validated security properties through targeted attack tests summarized in Table 6.

All attack attempts were successfully detected and rejected. For example, in the dataset substitution attack, we attempted to generate a valid training proof using samples not in the committed Merkle tree; the Merkle verification constraint failed with "Assert Failed" on root equality.

4.6 Aggregation Correctness

We verified that $\sum_i m_i = \sum_i g_i$ by computing both sums across all test runs. Perfect match was observed in all cases, confirming mask cancellation. Sample outputs from our test suite:

- Masked aggregate: $[0.098, -0.301, -0.002, -0.109]$
- Unmasked aggregate: $[0.071, -0.030, -0.063, -0.005]$
- All individual gradients correctly hidden from server

Table 5: Scaling with dataset size N.

N	Depth	Balance	Training	Prove (s)
8	3	∼12.5K	∼18.7K	14.1
16	4	∼24.8K	∼36.2K	28.3
32	5	∼49.4K	∼71.5K	56.7
64	6	∼98.5K	∼142K	114.2
128	7	∼197K	∼283K	231.5

Table 6: Security test results.

Attack Vector	**Detection Point**	**Result**
Dataset substitution	Merkle constraint	✓Blocked
Gradient inflation	Norm bound check	✓Blocked
Wrong model weights	Weight commitment	✓Blocked
Mask manipulation	Update equality	✓Blocked
Cross-component mismatch	Server binding check	✓Blocked
Gradient fabrication	Correctness verify	✓Blocked

5 Discussion

5.1 Limitations

Trusted setup requirement. Groth16 requires a trusted setup ceremony for each circuit. If the "toxic waste" is not properly destroyed, an adversary could forge proofs. Universal SNARKs (e.g., PLONK [11]) could eliminate this requirement at the cost of larger proofs.

Model architecture constraints. Our gradient verification circuit is specialized for linear regression. Extending to neural networks requires implementing activation functions in arithmetic circuits and handling the quadratic constraint explosion for deep networks.

Fixed circuit parameters. Changing dataset size, model dimension, or client count requires recompilation and new trusted setup, limiting dynamic federation membership.

No differential privacy. Our framework ensures computational privacy but does not provide differential privacy guarantees. A curious server analyzing aggregates across many rounds might still infer information about individual clients' data distributions.

Dropout handling. Our prototype does not implement secret sharing for mask recovery when clients drop out, which would be necessary for production deployments.

5.2 Practical Deployment Considerations

Proof computation offloading. Clients with limited resources could offload proof generation to untrusted cloud servers. The zero-knowledge property ensures the cloud learns nothing about the witness.

Hardware acceleration. FPGA and GPU implementations of Groth16 have demonstrated 10–100× speedups [12]. With such acceleration, our framework could become practical for production deployments.

5.3 Future Directions

Universal zkSNARKs. Transitioning to PLONK or Halo 2 would eliminate per-circuit trusted setup while maintaining similar proof sizes.

zkML integration. Emerging frameworks like EZKL automatically compile neural networks to ZK circuits. Integrating these with our federated learning protocol would enable verifiable training for deep learning models.

Recursive proofs. Using recursive SNARKs, each client could generate a single proof that compresses all three component proofs, reducing communication and simplifying verification.

Byzantine fault tolerance. Combining ZK-based integrity with Byzantine-tolerant aggregation rules (e.g., coordinate-wise median [4]) would provide robustness even when multiple malicious clients collude.

6 Related Work

Zero-knowledge machine learning. zkCNN [10] demonstrated practical ZK proofs for convolutional neural network inference by exploiting convolution structure. EZKL compiles ONNX models to Halo 2 circuits for inference verification. These works focus on inference rather than training verification, which is our contribution.

Secure aggregation. Bonawitz et al. [8] introduced the pairwise masking protocol we build upon, using threshold secret sharing for dropout tolerance. Their protocol ensures privacy but does not verify training integrity. Bell et al. improved efficiency using graph-based key agreement, reducing communication complexity.

Verifiable computation. Pinocchio [9] was an early practical system using pairing-based SNARKs for verifiable computation. Groth16 [6] optimized proof size and verification time to the current optimal. We build on these foundations using Circom [7] for circuit development.

Byzantine-tolerant federated learning. Blanchard et al. [4] introduced Krum, a Byzantine-tolerant aggregation rule. Subsequent work proposed coordinate-wise median, trimmed mean, and other robust aggregators. These defenses are complementary to our approach: we ensure gradients are correctly computed, while robust aggregation handles valid-but-adversarial gradients.

Privacy attacks on federated learning. Gradient inversion attacks [3] demonstrated that training examples can be reconstructed from gradient updates, motivating our secure aggregation component. Membership and property inference attacks extract information from model updates; while our framework protects individual gradients, analyzing aggregated models may still leak information.

7 Conclusion

This paper presented a comprehensive cryptographic framework for verifiable federated learning that addresses three fundamental challenges: proving dataset properties without revealing data, ensuring gradient computation integrity, and enabling privacy-preserving aggregation. Our framework integrates zero-knowledge proofs with Merkle tree commitments and secure aggregation, providing formal security guarantees previously unavailable in federated learning systems.

Key technical contributions include: (1) a unified commitment scheme using Poseidon-based Merkle trees that cryptographically binds all proof components, preventing composition attacks; (2) a sound gradient norm verification approach using sign-magnitude decomposition that correctly handles signed arithmetic within finite field constraints; and (3) a complete implementation with comprehensive security validation.

Our evaluation reveals both promise and challenges. Verification is efficient ($\sim$25ms per client) and proofs are compact (576 bytes per client). However, proof generation requires $\sim$14 seconds per client, representing significant overhead that will grow for production-scale models. From our development experience, we offer insights for future research: the constraint-generation bottleneck is real but addressable through hardware acceleration; finite field arithmetic requires careful handling of signed values; and modular design is essential for composability.

We believe the convergence of zero-knowledge proofs and machine learning will be transformative for privacy-preserving AI. Our framework provides a foundation demonstrating that strong cryptographic guarantees for federated learning are achievable with current technology, even if not yet at production scale.

References

[1] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, “Communication-efficient learning of deep networks from decentralized data,” in *Proc. AISTATS*, 2017.

[2] T. Yang, G. Andrew, H. Eichner, H. Sun, W. Li, N. Kong, D. Ramage, and F. Beaufays, "Applied federated learning: Improving Google keyboard query suggestions," *arXiv: 1812.02903*, 2018.

[3] L. Zhu, Z. Liu, and S. Han, "Deep leakage from gradients," in *Proc. NeurIPS*, 2019.

[4] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proc. NeurIPS*, 2017.

[5] 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*, 2021.

[6] J. Groth, "On the size of pairing-based non-interactive arguments," in *Proc. EUROCRYPT*, 2016.

[7] iden3, "Circom: A circuit compiler for zero-knowledge proofs," https://docs.circom.io/, 2023.

[8] K. Bonawitz et al., "Practical secure aggregation for privacy-preserving machine learning," in *Proc. ACM CCS*, 2017.

[9] B. Parno, J. Howell, C. Gentry, and M. Raykova, "Pinocchio: Nearly practical verifiable computation," in *Proc. IEEE S&P*, 2013.

[10] T. Liu, X. Xie, and Y. Zhang, "zkCNN: Zero knowledge proofs for convolutional neural network predictions and accuracy," in *Proc. ACM CCS*, 2021.

[11] A. Gabizon, Z. J. Williamson, and O. Ciobotaru, "PLONK: Permutations over Lagrange-bases for oecumenical noninteractive arguments of knowledge," *IACR ePrint 2019/953*, 2019.

[12] Y. Zhang et al., "PipeZK: Accelerating zero-knowledge proof with a pipelined architecture," in *Proc. ISCA*, 2021.