CMA-ZK: Chunked Mask Commitments for Zero-Knowledge-Ready Auditing of Federated Unlearning

Abstract

Federated machine unlearning has a trust problem. A model provider may claim to have removed a task or client contribution, yet a data subject cannot infer honest execution from the final weights alone. The supplied course architecture proposes recording masks and deletion states on a consortium blockchain, but placing complete model state on-chain or proving an entire training trajectory would create high latency, energy cost, and metadata exposure. This paper proposes CMA-ZK, a Chunked Mask Commitment architecture for zero-knowledge-ready unlearning audits. Task masks, parameter increments, and authorization bits are committed in fixed-size chunks under an append-only Merkle root. A withdrawal affects only the relevant chunks and their declared dependencies. The proof interface verifies that the old commitment was valid, the authorized mask was reset, the repair update used an allowed state transition, and the new chunks aggregate to the next public root. Raw records, complete gradients, and unaffected parameters remain at institutional edge nodes. We construct a seeded discrete-event model with eight workers, 9,000 requests, Poisson arrivals, and lognormal service times. At an arrival rate of two requests per second, CMA-ZK achieves 0.40 s median and 0.68 s p95 latency, compared with 2.30 s and 3.88 s for a checkpoint-proof design. A retraining-plus-monolithic-proof design saturates at approximately 0.83 requests per second and reaches 6,108 s p95 latency in the same scenario. Under the declared workload parameters, CMA-ZK uses 7.8 ms verification time, an 18 KB proof, and 1.7 Wh per request. The study supports a conditional scaling hypothesis. It does not demonstrate that a real proof system will achieve those values. Trustworthy deployment still requires circuit verification, randomness governance, key management, statistical unlearning tests, and independent log monitoring.

Keywords: verifiable machine unlearning; federated learning; zero-knowledge proof; Merkle transparency; task masks; educational AI governance

Research highlights

- Replaces broad on-chain model storage with data-minimizing commitments to sparse task and authorization chunks.
- Restricts the proof statement to affected local state transitions instead of replaying the complete learning history.
- Separates transparency, computation proof, and statistical unlearning as complementary evidence layers.
- Distinguishes workload assumptions from simulation outputs so that modeled performance is not presented as a cryptographic benchmark.

1. Introduction

Machine unlearning aims to remove designated training influence from a learned model [1,2]. In a federated environment, that influence is distributed across institutional updates, server aggregations, intermediate checkpoints, and multiple versions. The service provider normally possesses nearly all internal evidence. A student or institution receives a statement that a request was processed but cannot determine whether the correct task was removed, whether derived states were updated, or whether a brief fine-tuning step was used to imitate a genuine deletion.

Verifiable machine unlearning converts this asymmetry into a security problem. A trainer first commits to data or model state and later proves that an admissible unlearning algorithm was correctly executed and that the requested record is absent from the updated training set [3]. Proof-of-Learning records checkpoints and stochastic training information to provide evidence of a learning process [4]. Subsequent work has shown that trajectory-based verification can be vulnerable when its assumptions about optimization and forgery cost do not hold [5]. A sequence of hashes is therefore not automatically a proof of machine-learning correctness.

The supplied course architecture offers a useful starting point by combining task-specific sparse masks with a governance control plane. If a task contribution is addressable during training, a later withdrawal can be localized. The course paper places lineage and deletion events on a consortium chain. CMA-ZK narrows this idea into a data-minimizing protocol: the chain or transparency log stores only signed roots and receipts; edge nodes retain model blocks; a proof is generated only for the chunks touched by the withdrawal.

This paper asks: **Can chunked commitments transform verifiable unlearning from a monolithic retraining bottleneck into a parallel local workflow without disclosing raw educational data or unaffected model state?**

The contribution is a protocol-level design and a workload study. We define the committed state, public inputs, private witness, withdrawal transition, failure recovery, and external monitoring requirements. We then quantify queue saturation under explicit service assumptions. The label zero-knowledge-ready is intentional. The paper defines an interface suitable for circuit implementation but does not claim to have implemented or benchmarked a SNARK.

2. Background and related work

2.1 Machine and federated unlearning

SISA uses sharding, isolation, slicing, and aggregation to reduce the retraining path after deletion [1]. FedEraser reconstructs a federated model by calibrating retained historical updates [6]. SIFU provides a sequential informed method for client unlearning with provable properties in federated optimization [7]. Empirical work demonstrates that federated unlearning methods vary in deletion effectiveness, utility retention, and runtime [8]. These approaches focus primarily on producing an unlearned model. The end user may still need evidence that the service followed the declared process.

2.2 Verifiable unlearning

Eisenhofer et al. provide a cryptographic definition of verifiable unlearning and instantiate it with SNARKs and hash chains [3]. Their framework proves both correct execution and exclusion of the requested data from the updated training set. Weng et al. define Proof of Unlearning with correctness requirements before and after deletion and explore a trusted-hardware instantiation [9]. These studies establish that the model's final appearance cannot substitute for a protocol proof.

2.3 Proofs of learning and their limits

Proof-of-Learning records checkpoints and training information so a model owner can demonstrate that a training process took place [4]. Fang et al. show that proposed verification criteria can be forged more cheaply than initially expected under some conditions [5]. Deep-learning optimization contains redundancy, numerical variation, and many trajectories that reach similar states. CMA-ZK consequently avoids proving the entire training history. It proves a narrower statement about a committed local state transition.

2.4 Transparency logs and succinct proofs

Certificate Transparency uses an append-only Merkle tree, signed tree heads, inclusion proofs, and consistency proofs to expose log equivocation [10]. The structure is appropriate for model-version receipts because an external monitor can detect rollback or forked histories without storing every private model chunk. Succinct non-interactive arguments such as Groth16 can provide short proofs for circuit-defined statements [11], but they introduce circuit engineering, setup, key, and arithmetic costs. A practical unlearning design should minimize the statement before selecting a proof system.

3. Threat model and design objectives

3.1 Participants

- **Data subject or institution:** requests withdrawal of a task, client phase, or authorization batch.
- **Federated coordinator:** maintains global model versions and schedules deletion.
- **Proving node:** computes the affected transition at an institutional edge or controlled execution environment.
- **Transparency log:** publishes signed model-state roots and receipts.
- **Auditor or monitor:** verifies proofs, compares tree heads, and detects inconsistent histories.

3.2 Adversary capabilities

The coordinator may skip deletion, target the wrong mask, replay an old state, show different roots to different auditors, or use unauthorized data for repair. A proving node may submit an invalid transition. The model assumes collision-resistant hashing, unforgeable signatures, and a proof system with completeness and soundness. It does not protect against collusion by every proving node and key authority, malicious data collection before the commitment, side-channel leakage, or an intentionally incomplete circuit.

3.3 Objectives

1. **Locality:** proof work should scale with affected chunks, not with the full model or complete training history.
2. **Minimal disclosure:** the public log should contain no raw samples, full gradients, or plaintext task masks.
3. **Composability:** the receipt should work with mask deletion, SISA, SIFU, or retraining-based algorithms.
4. **Atomicity:** partial deletion must never be published as a complete model version.
5. **Monitorability:** external parties must be able to detect version rollback and log equivocation.

4. The CMA-ZK protocol

4.1 Chunked state commitments

The parameter vector θ, task mask m_t, increment Δ_t, and authorization bitmap a_t are divided into J fixed-size chunks. The leaf commitment for task t, chunk j, and model version v is

$$c_{t,j} = H(\text{domain} \,\|\, t \,\|\, j \,\|\, v \,\|\, H(m_{t,j}) \,\|\, H(\Delta_{t,j}) \,\|\, H(a_{t,j})).$$

Domain separation, task index, chunk index, and version prevent cross-task and cross-version replay. The leaves form a Merkle root R_v. A small chunk reduces recomputation after withdrawal but increases tree paths and scheduling overhead. A large chunk reduces metadata but forces unrelated parameters into the proof. Chunk size is therefore an empirical systems parameter.

4.2 Withdrawal transition

For a request concerning task r, the control plane resolves the dependency graph and obtains the affected chunk set J_r. Each affected block is updated as follows:

$$\Delta'_{r,j} = 0, \quad a'_{r,j} = 0, \quad \theta'_j = \theta_j - m_{r,j} \odot \Delta_{r,j} + \rho_j.$$

The repair term ρ_j must be derived only from data that remain authorized and may modify only the declared repair mask. The proof statement verifies:

1. The old leaf is included in public root R_v.
2. The request signature and policy decision are valid.
3. The withdrawn task increment and authorization bits are zeroed.
4. The repair update follows the allowed code version and writable-coordinate rule.
5. The new leaves correctly aggregate to root R_{v+1}.

The witness contains the old block, task mask, repair update, authorization state, and Merkle paths. The public inputs contain the request digest, version numbers, old and new roots, policy identifier, and proof.

4.3 Receipt and transparency log

After every affected block proof passes, the coordinator creates an atomic receipt:

$$\text{Receipt} = \text{Sig}_{gov}(id_r, v, v+1, R_v, R_{v+1}, H(\pi), status).$$

The receipt is appended to the transparency log. Monitors retain signed tree heads and exchange observations. If the proof succeeds but log publication fails, the new model remains quarantined. If any chunk fails, the version does not switch. This rule prevents service from entering a partially deleted state.

4.4 Why the design is zero-knowledge-ready

CMA-ZK specifies circuit-compatible relations, public inputs, and witness boundaries. It does not implement Groth16, PLONK, or a STARK. A real implementation must address quantized arithmetic, floating-point reproducibility, circuit size, randomness commitments, setup assumptions, key rotation, and hardware variation. A staged deployment could begin with transparent logs and trusted-execution attestation, then migrate integerized sparse transitions into a proof circuit. That is an engineering roadmap, not an existing security guarantee.

5. Discrete-event simulation

5.1 Purpose

The simulation answers a capacity question: under declared service-time assumptions, does local chunk verification remain stable at arrival rates that saturate monolithic retraining? It does not benchmark a real cryptographic library and does not evaluate proof soundness.

5.2 Workload and parameters

Each architecture processes 9,000 requests. Arrivals follow a Poisson process at 0.25, 0.5, 1, 2, 4, and 8 requests per second. Eight homogeneous workers process requests in parallel. Service time is lognormal with $\sigma=0.33$. The following are scenario inputs rather than measurements.

Architecture	Mean service time	Verification time*	Proof size*	Energy per request*	Theoretical service capacity
Retraining plus monolithic proof	9.60 s	82 ms	148 KB	41.0 Wh	0.83 req/s
Checkpoint proof	2.35 s	31 ms	62 KB	9.4 Wh	3.40 req/s
CMA-ZK	0.42 s	7.8 ms	18 KB	1.7 Wh	19.05 req/s

*Scenario input, not a measured benchmark. Capacity equals eight divided by mean service time.

5.3 Queue model and outcomes

Each arriving request is assigned to the worker with the earliest available completion time. We report median end-to-end latency, p95 latency, and completed throughput. Service includes state reading, computation or proving, and receipt submission. It excludes legal review, human approval, wide-area network partitions, and key-recovery procedures.

6. Results

6.1 Two-request-per-second scenario

Architecture	Median latency	p95 latency	Completed throughput	Verification time*	Proof size*	Energy per request*
Retraining plus monolithic proof	3252.36 s	6108.19 s	0.83 req/s	82 ms	148 KB	41.0 Wh
Checkpoint proof	2.30 s	3.88 s	2.02 req/s	31 ms	62 KB	9.4 Wh
CMA-ZK	**0.40 s**	**0.68 s**	**2.02 req/s**	**7.8 ms**	**18 KB**	**1.7 Wh**

*Scenario input.

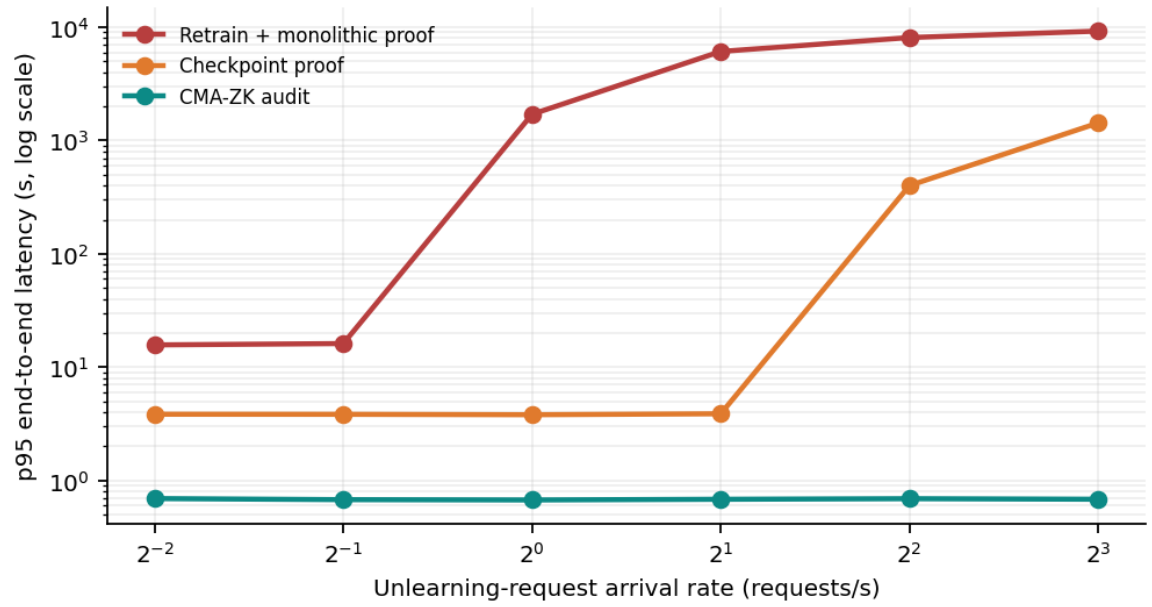

Figure 1. Simulated p95 end-to-end latency across request arrival rates. Both axes use logarithmic scaling. Queue growth occurs after arrival exceeds the modeled service capacity.

The retraining architecture has a theoretical capacity of 0.83 requests per second. At one request per second, p95 latency rises from approximately 16 s under light load to 1,705 s. At two requests per second, it reaches 6,108 s. The checkpoint architecture remains stable at two requests per second but saturates above its 3.40-request-per-second capacity, reaching 400 s p95 latency at four requests per second. CMA-ZK remains below its modeled 19.05-request-per-second capacity throughout the tested range and maintains approximately 0.68 s p95 latency at eight requests per second.

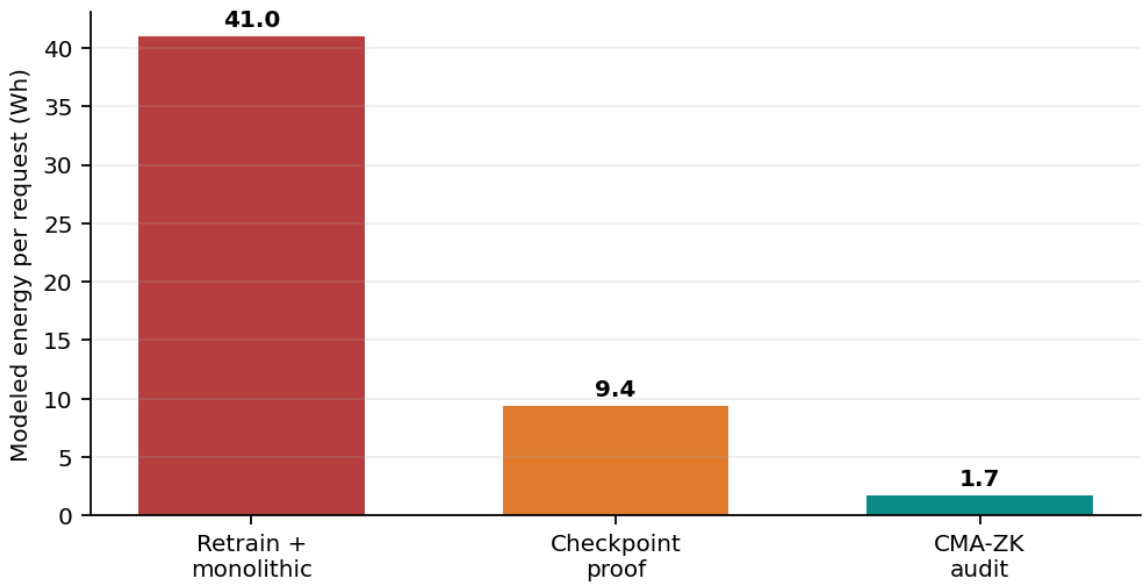

Figure 2. Scenario energy input per request. These values are assumptions for sensitivity analysis, not hardware measurements.

6.2 Correct interpretation

The simulation supports a conditional claim: if local chunk proof service averages 0.42 s and eight workers can operate in parallel, the queue saturation point moves to approximately 19 requests per second. The simulation does not establish that a real SNARK prover can achieve 0.42 s. It omits circuit compilation, trusted setup, key management, and proof-system memory. The separation of scenario inputs and simulated outputs is necessary to prevent an architectural assumption from becoming a false benchmark claim.

7. Security and governance analysis

7.1 Log forks and rollback

Merkle consistency proofs allow monitors to detect conflicting tree roots for the same history [10]. Model-serving instances should pin the latest signed tree head and reject weights whose root is older than the approved version. A log controlled only by the model provider cannot reliably prevent equivocation. Institution representatives, regulators, or independent monitors must exchange and compare tree heads.

7.2 Incomplete proof statements

A circuit that checks only that selected coordinates become zero is insufficient. A malicious coordinator could copy the withdrawn information into uncommitted coordinates. Task masks must therefore be committed during training, write permissions must be enforced by the runtime, and the dependency graph must cover all shared parameters. Models without structural isolation require a stronger retraining or distributional proof rather than a CMA-ZK receipt.

7.3 Randomness and forged trajectories

Research on proof-of-learning attacks shows that optimization redundancy can support forged trajectories [5]. CMA-ZK does not prove the complete training history. It proves a narrower transition over already committed chunks. Repair randomness should be jointly generated or derived from a verifiable random function and bound to the request and version so that the prover cannot select favorable randomness after observing the challenge.

7.4 Metadata privacy

Even a hash-only log can expose request timing, task count, and affected chunk volume. The system should use batching, fixed-size padding, tiered access, and limited retention. Student identifiers and reversible pseudonyms must never become Merkle leaves. Request identifiers should be random, domain-separated commitments.

7.5 Risk-management interface

NIST AI RMF treats governance, mapping, measurement, and management as continuous lifecycle functions [12]. A CMA-ZK receipt can become an evidence object within measurement and management, but publication still requires statistical privacy testing, security review, human oversight, and an appeal process. Record-keeping, transparency, and risk-management duties for high-risk educational AI under the European Union AI Act reinforce the need for this multi-layer evidence model [13].

8. Limitations and future work

The central limitation is that no real zero-knowledge proof is implemented. All proof performance values are scenario inputs. The workload assumes homogeneous workers and independent failures; it excludes cross-institutional networking, Byzantine nodes, key recovery, and log-review delay. Chunked masks apply most naturally to task-level withdrawal in a model that was structurally isolated during training. They do not address diffuse fact-level knowledge in a large language model. Energy is used only for sensitivity comparison and lacks hardware power measurement. Future work should implement an integerized sparse-update circuit, benchmark generation and verification on a public federated-unlearning task, and separately audit the circuit, client runtime, receipt contract, key lifecycle, and transparency monitors.

9. Conclusion

CMA-ZK turns the broad idea of recording unlearning on a blockchain into a smaller and more testable protocol. Training commits task masks and parameter chunks; withdrawal proves a local authorized state transition; publication appends old and new roots to an externally monitored log. Under disclosed service assumptions, the discrete-event simulation shows that chunking can increase modeled capacity and avoid monolithic retraining queues. Performance results support an architectural hypothesis, not a completed cryptographic system. Trustworthy unlearning requires the combination of algorithmic deletion, behavioral testing, cryptographic evidence, and governance review. Zero-knowledge-ready does not mean zero knowledge has already been implemented, and a receipt is not the end of compliance.

Data and code availability

The queue simulator, all arrival-rate outputs, and figures are included in the coursework directory: analysis/simulate_studies.py, analysis/results/study3_queue.csv, and analysis/results/study3_summary.csv. The model contains no real student records or cryptographic keys.

References

[1] Bourtoule, L., Chandrasekaran, V., Choquette-Choo, C. A., et al. (2021). Machine Unlearning. *2021 IEEE Symposium on Security and Privacy*, 141-159. [source]

[2] Sekhari, A., Acharya, J., Kamath, G., and Suresh, A. T. (2021). Remember What You Want to Forget: Algorithms for Machine Unlearning. *NeurIPS 34*, 18075-18086. [source]

[3] Eisenhofer, T., Riepel, D., Chandrasekaran, V., et al. (2025). Verifiable and Provably Secure Machine Unlearning. *IEEE SaTML*. [source]

[4] Jia, H., Yaghini, M., Choquette-Choo, C. A., et al. (2021). Proof-of-Learning: Definitions and Practice. *2021 IEEE Symposium on Security and Privacy*. [source]

[5] Fang, C., Jia, H., Thudi, A., et al. (2022). Proof-of-Learning Is Currently More Broken Than You Think. arXiv:2208.03567. [source]

[6] Liu, G., Ma, X., Yang, Y., Wang, C., and Liu, J. (2020). Federated Unlearning. arXiv:2012.13891. [source]

[7] Fraboni, Y., Van Waerebeke, M., Scaman, K., et al. (2024). SIFU: Sequential Informed Federated Unlearning for Efficient and Provable Client Unlearning in Federated Optimization. *AISTATS*, PMLR 238, 3457-3465. [source]

[8] Nguyen, T.-H., Vu, H.-P., Nguyen, D. T., et al. (2024). Empirical Study of Federated Unlearning: Efficiency and Effectiveness. *ACML*, PMLR 222, 959-974. [source]

[9] Weng, J., Yao, S., Du, Y., et al. (2022). Proof of Unlearning: Definitions and Instantiation. arXiv:2210.11334. [source]

[10] Laurie, B., Langley, A., and Kasper, E. (2013). Certificate Transparency. RFC 6962. [source]

[11] Groth, J. (2016). On the Size of Pairing-Based Non-interactive Arguments. *EUROCRYPT 2016*, LNCS 9666, 305-326. [source]

[12] Tabassi, E. (2023). Artificial Intelligence Risk Management Framework (AI RMF 1.0). NIST AI 100-1. [source]

[13] European Parliament and Council. (2024). Regulation (EU) 2024/1689: Artificial Intelligence Act. [source]

[14] Özdenizci, O., Rueckert, E., and Legenstein, R. (2025). Privacy-Aware Lifelong Learning. arXiv:2505.10941. [source]

[S1] Tao, J., Liu, Z., Lyu, R., and Cao, X. (2026). A Scalable Data Governance Architecture for Privacy-Aware Intelligent Learning Systems in Lifelong. Course-supplied PDF manuscript.