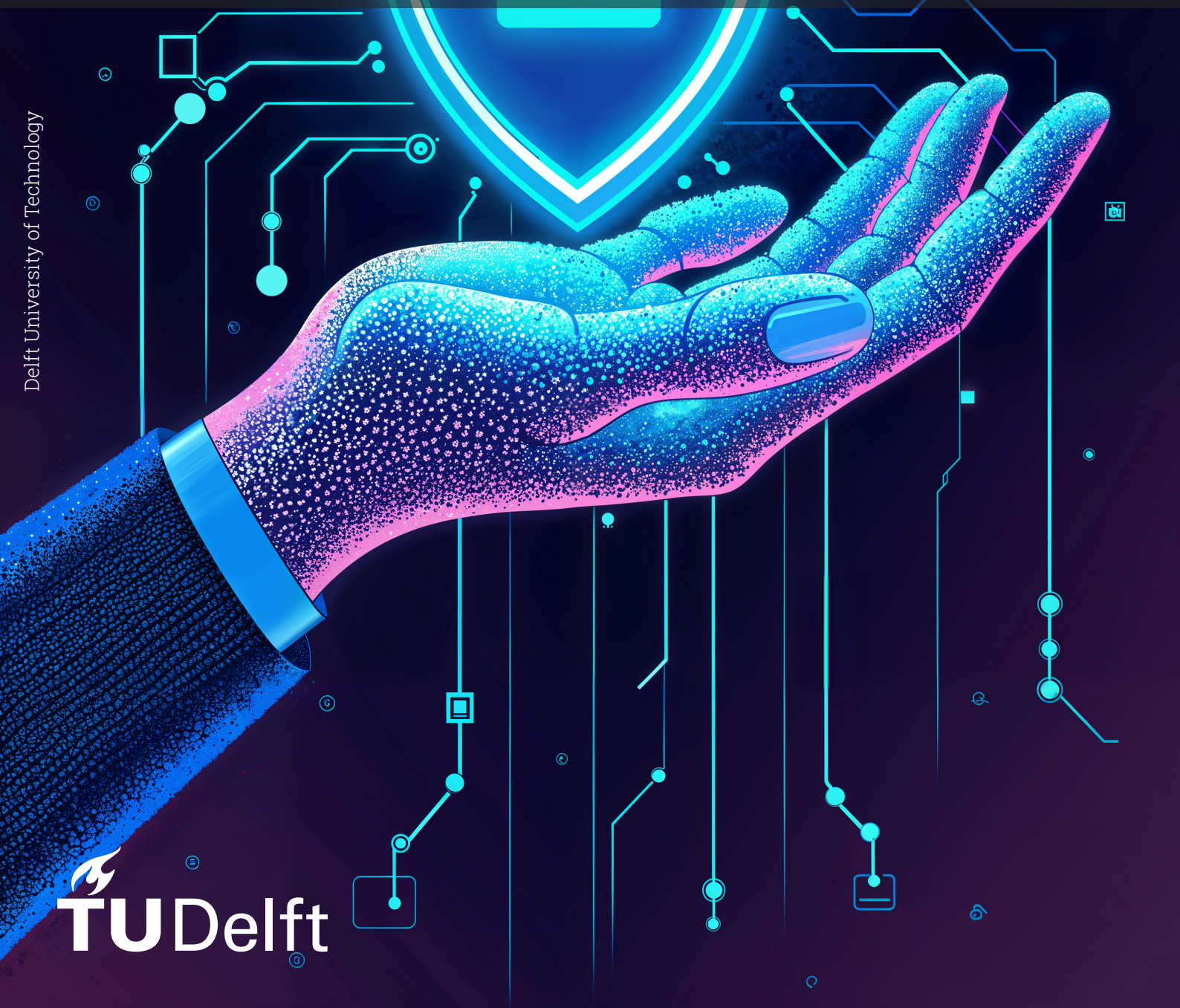
Verifiable Data Provenance in Machine Learning Training

Siddharth Sharma



Delft University of Technology

TU Delft

Verifiable Data Provenance in Machine Learning Training

by

Siddharth Sharma

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Friday, July 3, 2026, at 12:45 PM.

Student number:	6244548
Project duration:	November 10, 2025 – July 3, 2026
Thesis committee:	Dr. Zekeriya Erkin, TU Delft, supervisor
	Dr. Kubilay Atasu, TU Delft

Cover: Content by Atria Solutions (Adapted with Gemini)

An electronic version of this thesis is available at `http://repository.tudelft.nl/`.

TU Delft

Preface

This thesis marks the end of my two years at Delft and the end of a fruitful collaboration with Zeki and our research group. I first reached out to Zeki after the Cryptography exam, stating that I like Cryptography and Mathematics, and would like to pursue my thesis with him. Before the start of my thesis, Zeki told me to look into “Data Economy" and “Zero-Knowledge Proofs". In the past eight months, going from these two themes to scoping towards a problem and then working towards its solution has been enriching. I am truly grateful to Zeki for placing faith in me to pursue a challenging problem. The beauty of our discussions was that I rarely got an answer, but perhaps I could better weigh my alternatives. I hope all these learnings help me in my next chapter. I am grateful to Prof. Atasu for serving on my thesis committee.

I am also thankful for being surrounded by a wonderful group of Master’s students. Jay, Mitchell, Maik, Miloš, Erin, Roderick, Marco, and Alisea, thank you very much for your constructive criticism during the bi-weekly presentations and for being a calming influence when I was stressed.

I would not have made it this far without the constant support from my family and friends. They have supported me through my highs and lows, and I am fortunate to have them in my life.

I would also like to thank my table fan for keeping my head cool while writing the proofs, and to my trusty hard drive for backing me up all these years!

Siddharth Sharma
Delft, July 2026

AI Usage Statement

During the preparation of this thesis, I made limited use of generative artificial intelligence tools as research and writing aids.

For the implementation work, I primarily used Claude Sonnet 4.6 for refining paragraph-flow, creating tables and tikz diagrams. Suggestions produced by these tools were always reviewed, evaluated, and adapted before being incorporated into the report.

For literature review, I primarily used Google NotebookLM to obtain insights on survey papers, compare security and performance guarantees. In all such cases, the output provided was verified by me.

The AI tools were not used to generate original research contributions and security proofs. I take full responsibility for the content of the thesis report.

Abstract

Machine learning models trained on sensitive financial records can incorporate private data without the data owner's knowledge or consent. A bank that suspects a fintech company has trained a machine learning model on stolen customer records faces a structural dilemma. To prove misappropriation, the Bank has to disclose the dataset, which may itself constitute a further privacy violation under Article 5 of the General Data Protection Regulation. Existing verification techniques each fall short of resolving this dilemma. Membership inference attacks produce probabilistic scores rather than deterministic proofs, and their false-positive rates cannot be bounded without access to a reference model trained on excluded data. Dataset watermarks must be embedded proactively and can be stripped by an adversary before deployment. Zero-knowledge proofs of training certify that a model's weights are the correct output of gradient descent over a committed dataset, but place no constraint on whether that committed dataset overlaps with the data owner's private records.

In this thesis, we propose a three-party cryptographic protocol that simultaneously certifies the correctness of the training and characterizes the intersection between the trainer's dataset and the data owner's private dataset, without revealing any raw features or labels to the auditor. Before the training, the Bank posts a cryptographic commitment to its authorized records to a public ledger. The trainer posts a commitment to the training dataset and submits a proof after the training. The auditor verifies two properties from the proof and the two public commitments alone. For the first property, we use a sumcheck with multilinear KZG oracles, adapted from deep neural networks to linear and logistic regression, to verify that each gradient-descent step was computed honestly over the committed data. For the second property, for each training sample, we check whether it is in the Bank's authorized set using the zero-knowledge sets construction of Micali, Rabin, and Kilian. The two components connect without an additional circuit. The per-sample commitment used as a lookup key is already a public element of the training commitment, so the auditor derives it locally. The zero-knowledge property of the combined protocol is formally established via a simulator-based composition, thereby providing a reusable proof structure for pairing verifiable computation schemes with cryptographic set-membership proofs. We find and fix a soundness issue in the linear regression verifier, where error-label consistency was not checked, and bound the residual error probability to be negligible relative to the size of the field.

The protocol is benchmarked on the UCI Default of Credit Card Clients dataset across batch sizes up to 256 and up to 23 features. Proof size grows logarithmically with batch size and linearly with feature count, reaching approximately 80 KB per gradient step at the maximum configuration. Prover time scales linearly with dataset size and feature count, reaching approximately 3 seconds per step; the zero-knowledge set's membership check contributes a constant 20 KB and 4.5 ms per sample, independent of dataset size. Verifier time grows linearly with feature count only, reaching approximately 500 ms per step. The computational burden rests on the trainer, not auditor. End-to-end audits across all data-mixing ratios confirm perfect completeness. Every sample drawn from the Bank's private dataset is correctly identified, and no independently sourced sample produces a false membership certificate.

The remaining open problem is the Consistent DataCommitment (CDC) assumption, which states that the column-wise multilinear KZG commitments used for training correctness and the row-wise univariate KZG commitments used for data authorization bind orthogonal projections. The current protocol does not cryptographically enforce that they encode identical data. A cross-commitment scheme would close this gap and is the highest-priority direction for future work.

Contents

Notation

Parties

$\mathcal{B}$	Bank: owns private authorised dataset D
$\mathcal{T}$	Model Trainer: trains on dataset D'
$\mathcal{A}$	Auditor: verifies proofs using only public commitments

Datasets and Training

D	Bank's private authorised dataset
D'	Trainer's training dataset
$I_\cap = D \cap D'$	Certified intersection of D and D'
$\lvert D' \rvert$	Total number of training samples in the trainer's dataset
$n = \lvert B_t \rvert$	Gradient-descent batch size: number of samples selected by sel at step t
d	Number of features per sample
T	Number of gradient-descent steps
$v = \log_2 \lvert D' \rvert$	Number of sumcheck variables; sumchecks run over all of D', not just the batch
X	Feature matrix ($n \times d$)
X_j	j-th feature column of X
y	Label vector
$w^{(t)}$	Weight vector at gradient-descent step t
g_j	Gradient for feature j
g_{bias}	Bias gradient
ε	Error vector; $\varepsilon(i) = \text{pred}(i) - y_i$
pred	Prediction vector
sel	Batch-membership indicator vector for step t
B_t	Mini-batch of sample indices at step t

Polynomial Commitments (KZG / MKZG)

pp	Public parameters (structured reference string)
kzg_params	Univariate KZG SRS
mkzg_params	Multilinear KZG SRS
Com	Generic commitment function
$\tilde{f}$	Multilinear extension (MLE) of function f
$\tilde{X}_j$	MLE of the j-th feature column
column_commits[j]	MKZG commitment to feature column j; oracle for Step A sumchecks
sample_poly_commits[i]	Univariate KZG commitment to sample i; ZK Sets lookup key for Step B
DataCommitment	Combined public commitment = (column_commits, sample_poly_commits)

r, r'_j, r_c	Fiat-Shamir challenge points (forward, per-feature backward, error-consistency)
R_j	Zero-sum masking polynomial for feature j
S_t	Claimed forward-sumcheck sum at step t

ZK Sets (Pedersen-Hash Trie)

PK_D	Bank's public key: root commitment of the Pedersen-hash trie
k	ZK Sets tree depth (security parameter; $k = 128$ in experiments)
$H(a, b)$	Pedersen hash: $g^a \cdot h^b \bmod p$
c_v	Pedersen commitment at tree node v: $g^{m_v} \cdot h_v^{r_v}$
m_v	Message stored at node v (hash of children, or leaf value)
h_v	Node-specific generator at node v
r_v	Blinding factor at node v
e_v	Generator exponent at node v (trapdoor for frontier nodes)

Protocol Artifacts

π_t	Proof for gradient-descent step t
π_{PoT}	Full proof-of-training transcript $\{\pi_t\}_{t=0}^{T-1}$
π_{ZKS}	ZK Sets membership/non-membership proof component
fwd	Forward-pass sub-proof (selection and prediction sumcheck)
bwd	Backward-pass sub-proof (gradient and error-consistency sumchecks)

Cryptographic Primitives and Relations

$\mathbb{G}, \mathbb{G}_1, \mathbb{G}_2$	Elliptic-curve groups (BLS12-381)
$\mathbb{G}_T$	Target group for the pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$
$\mathbb{F}$	Scalar field of BLS12-381 ($\lvert\mathbb{F}\rvert \approx 2^{255}$)
p, q	Prime group orders
g, h	Public group generators ($\log_g h$ unknown)
λ	Security parameter (bit-security level; $\lambda = 128$)
$\mathsf{negl}(\lambda)$	A negligible function in λ
$\approx_c$	Computational indistinguishability
$\equiv$	Identical distributions
$\mathsf{View}[\cdot]$	Transcript view of a protocol execution
Adv	Advantage of a PPT adversary

Abbreviations

CDC	Consistent DataCommitment assumption
DL	Discrete Logarithm
KZG	Kate–Zaverucha–Goldberg polynomial commitment scheme
MIA	Membership Inference Attack
MLE	Multilinear Extension
MKZG	Multilinear KZG commitment scheme
MRK	Micali–Rabin–Kilian ZK Sets construction
PPT	Probabilistic Polynomial Time

q-SDH	q-Strong Diffie–Hellman assumption
SRS	Structured Reference String
ZK	Zero-Knowledge
zkMLProv	Zero-Knowledge Machine Learning Provenance (this protocol)
zkPoGD	Zero-Knowledge Proof of Gradient Descent (Algorithm 4)
zkPoT	Zero-Knowledge Proof of Training

1 Introduction

Consider a bank that licenses its customer data to a fintech company for credit-scoring. After a data breach, the bank suspects the fintech company has trained a model that incorporates records drawn from its private dataset without authorization. To convince an auditor of this misuse, the bank faces an immediate dilemma: proving misappropriation requires disclosing the very dataset it is trying to protect, potentially compounding the harm. European data protection law sharpens this tension further: Article 5 of the General Data Protection Regulation [15] requires that personal data be processed only for the purposes for which it was collected, meaning that disclosure to an auditor may itself constitute a further violation. No existing mechanism allows the bank to prove that a specific model was trained on records drawn from its private dataset without revealing that data. This thesis proposes a cryptographic framework that enables the bank to convince a skeptical third-party auditor that a given machine learning model was trained on records drawn from the bank's private dataset, certifying the intersection of the bank's data and the trainer's training set, while revealing nothing beyond the correctness of that claim.

1.1. Motivation

The financial sector has experienced a series of large-scale data breaches in which customer records were exposed to adversarial actors. In May 2019, a software vulnerability at First American Financial Corp. exposed approximately 885 million mortgage title insurance records dating back to 2003, including bank account numbers, bank statements, wire transfer receipts, and Social Security numbers [26]; the records were accessible without authentication to any user who incremented a single digit in a document URL. In July 2019, CapitalOne reported that a cloud misconfiguration exposed the credit card applications of 100 million customers in the United States and 6 million in Canada [24], including income figures, credit limits, and repayment histories that are precisely the inputs to a credit-scoring model. In 2017, Equifax disclosed that a web application vulnerability had exposed the names, Social Security numbers, dates of birth, and credit information of 148 million consumers [14]; the case was settled for up to $700 million, including a $100 million civil penalty to the Consumer Financial Protection Bureau. Most recently, in January 2024, a ransomware attack on LoanDepot exposed the names, dates of birth, Social Security numbers, and financial account numbers of approximately 16.6 million customers [43]. In each case, the perpetrators were identified and prosecuted. Prosecution does not, however, answer whether the stolen data was subsequently used to train a commercial machine learning model. To the best of our knowledge, there is currently no mechanism that settles this claim while preserving the security of the bank's dataset.

European data protection law creates a structural obstacle to answering this question through conventional means. Article 5 of the GDPR [15] simultaneously imposes two obligations that pull in opposite directions: a data owner must be able to demonstrate that its data was misused (accountability, Article 5(2)), yet may not disclose personal data beyond the purposes for which it was originally collected (purpose limitation, Article 5(1)(b)). Any verification approach that requires the bank to hand over its dataset to a court, an auditor, or a forensic expert fails to meet one or both obligations. Every data-breach plaintiff operating under European law faces this conflict, and it recurs across trade secret litigation and regulatory investigations more broadly [3].

Existing methods to verify data usage in model training, namely membership inference attacks, digital watermarks, and zero-knowledge proofs of training, each fall short, for reasons examined in Section 1.2. Before discussing why, it is useful to fix the protocol setting precisely.

This thesis considers three parties: the *data owner* (Bank, $\mathcal{B}$) who holds a private dataset D; the *model trainer* ($\mathcal{T}$) who trains on a dataset D', which may contain records drawn from D; and the *auditor* ($\mathcal{A}$) who must verify the trainer's claims without receiving any raw data. Before training, the bank publishes a cryptographic commitment PK_D to D on a public ledger. After training, the trainer publishes a proof π. Given only (π, PK_D), the auditor must answer two questions, illustrated in Figure 1.1.

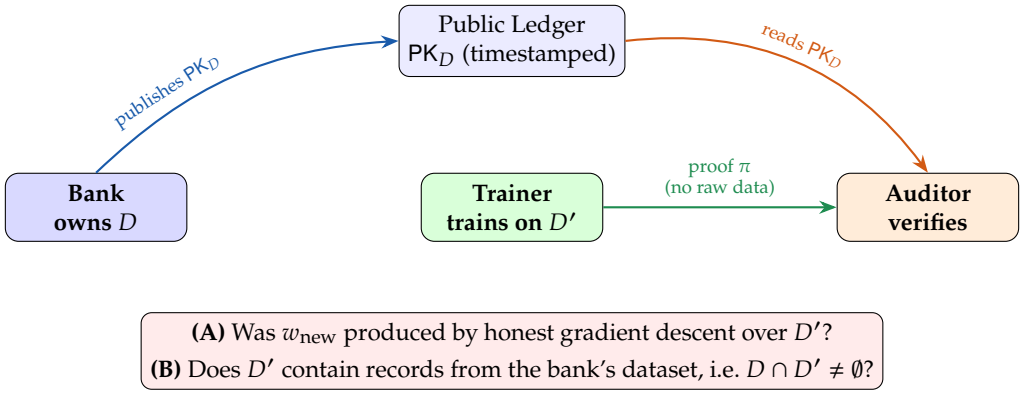


Figure 1.1: The three-party data-provenance protocol. The bank publishes a cryptographic commitment PK_D to its private dataset D on a public ledger. The trainer publishes a proof π attesting to the training computation. The auditor verifies two properties from π and PK_D alone, without receiving any raw training data.

In the setup described in Figure 1.1, w_{new} corresponds to the final weights of the Trainer and is public. The purpose of our problem is to address claims (A) and (B) while preserving the privacy of the dataset D and D'.

1.2. Why Existing Approaches Fail

Three families of techniques have been proposed to verify whether a machine learning model was trained on a particular dataset: membership inference attacks, dataset watermarks, and zero-knowledge proofs of training. Chapter 3 also examines private set intersection, secure multi-party computation, and statistical privacy frameworks; the following subsections focus on the three most directly relevant to the thesis contribution. A detailed treatment with empirical comparisons appears in Chapter 3.

Membership Inference : A membership inference attack (MIA) takes a trained model and a candidate data record as input. It outputs a binary prediction of whether that record was present in the training dataset [38]. Variants target individual samples [36, 33, 46] as well as entire datasets [28]. If an auditor could apply an MIA to the trainer's model using known bank records, a positive result would appear to answer the provenance question.

The fundamental limitation is that MIAs are probabilistic: they produce a score, not a proof. For a membership determination to constitute valid legal evidence, the false positive rate must be bounded and independently verifiable. Zhang et al. [47] prove formally that this bound cannot be established without access to a reference model trained on data that excludes the target record, which is infeasible when the full training set is unknown to the auditor. Dataset-level aggregation of MIA scores, proposed by Maini et al. [28], inherits the same limitation and additionally fails under distribution shift between members and non-members. What the data provenance problem requires is a deterministic, cryptographic guarantee that holds regardless of distributional assumptions; MIAs cannot provide such a guarantee.

Dataset Watermarks and Canaries: Dataset watermarking embeds subtle, verifiable patterns into training data before it is released [40]. If a model trained on the watermarked data reproduces those patterns when queried in a specific way, the presence of the watermark is taken as evidence that the model was trained on that data. The approach is appealing in principle because the data owner can inject watermarks unilaterally, without the trainer's cooperation.

The approach is, however, entirely proactive. The watermarks must be present in the data before it is released or stolen. In each of the breach scenarios described in Section 1.1, the adversary obtained raw, unwatermarked records directly from a compromised system. No watermarking scheme can be applied retrospectively to data that has already been exfiltrated. A motivated adversary who trains a model on stolen data can additionally detect and strip low-entropy watermarks through standard model pruning or fine-tuning before deployment [47]. Furthermore, watermarks have no non-membership analog. They cannot certify that a specific record was not used in training, which is precisely the guarantee required when a data subject exercises the right to erasure under GDPR Article 17 [15]. What is needed is a retrospective proof of provenance that operates on committed, unmodified training data.

Zero-Knowledge Proofs of Training: A zero-knowledge proof of training (ZkPoT) lets a trainer prove that a model's weights are the correct output of a gradient-descent algorithm applied to a committed dataset, without revealing the dataset or the intermediate computation [2]. Abbaszadeh et al. [2] demonstrate this for deep neural networks using a GKR-based sumcheck with multilinear KZG oracles, achieving proof sizes of $O(d \log N)$, where N denotes the size of the dataset. Earlier work by Garg et al. [18] constructs a similar guarantee for logistic regression using MPC-in-the-head techniques. ZkPoTs appear to be the right tool: they are already zero-knowledge and already reason about training data.

The gap is that a ZkPoT binds the proof to the dataset the trainer committed to, but imposes no constraint on the dataset's origin. A trainer who has obtained the bank's data through a breach can commit to that stolen dataset and produce a valid ZkPoT over it. The proof establishes correct training, but says nothing about whether the training data overlaps with the data owner's private dataset. Neither Abbaszadeh et al. nor Garg et al. address whether the committed dataset shares records with the data owner's private dataset. What is missing is a mechanism that simultaneously proves correct training and certifies which training samples are drawn from the data owner's private data, characterizing the intersection $D \cap D'$. Constructing such a mechanism is the contribution of this thesis.

1.3. Why a ZKP-Based Protocol?

Legal and regulatory disputes frequently require verification of the properties of information that a party is unwilling to disclose. Bamberger et al. [3] term this the *verification dilemma*: producing evidence of misuse requires exposing the very asset whose misuse is being alleged. The setting closest to ours is trade secret litigation, in which a data owner must prove that proprietary data was misappropriated without disclosing it to the court. Zero-knowledge proofs resolve this dilemma by allowing one party to convince another that a statement is true without revealing any information beyond the statement's truth.

Zero-knowledge proofs are well-suited to this problem for a structural reason: Goldreich, Micali, and Wigderson [20] proved that every NP language has a zero-knowledge proof system. Set intersection ($D \cap D' \neq \emptyset$) and correct gradient descent are both NP statements, so ZKPs are not merely a convenient tool here but a provably applicable one. The three properties that a good verification scheme must satisfy map directly onto the three defining properties of zero-knowledge proofs: an honest trainer must always be able to produce an accepting proof (completeness); a dishonest trainer must not be able to produce one (soundness); and the auditor must learn nothing beyond the yes-or-no verdict (zero-knowledge).

Secure multi-party computation (MPC) offers an alternative framing where the bank and trainer jointly evaluate a function that identifies which elements the trainer's dataset shares with the bank's private dataset, without either party learning the other's private input. MPC appears to solve the problem, but still requires all parties to be online simultaneously during the computation and produces no persistent, publicly verifiable record. A regulator conducting an investigation months after training occurred cannot verify an MPC execution retroactively, because the protocol produces no artifact that an absent third party can check. The Fiat-Shamir transform converts an interactive zero-knowledge proof into a non-interactive proof that any party can verify at any time, without the prover being present. This property of asynchronous public verifiability is essential when the auditor is a court or regulatory body rather than a protocol participant. For the regression models studied in this thesis, the KZG-oracle variant of the protocol produces proof sizes of approximately 32–80 KB per gradient-descent step at $d = 23$ features (scaling as $O(d \log |D'|)$ with training dataset size D') and achieves verifier times of approximately 224–497 ms on a standard laptop; the ZK Sets data-intersection step contributes a constant $\approx$20 KB and approximately 4.5 ms per sample regardless of dataset size, confirming that the approach is practical within the financial audit setting that motivates it.

1.4. Research Questions

This thesis studies how zero-knowledge proofs can simultaneously certify that a gradient-descent model was trained correctly *and* identify which training samples are drawn from the bank's private dataset, characterizing $D \cap D'$, without revealing the raw data to any verifying party. Using the three-party protocol of Figure 1.1, we investigate the following questions for linear and logistic regression models.

RQ1 (Training correctness). Given only a proof π and a public DataCommitment, can an auditor verify that a gradient-descent model was produced by an honest execution of the training algorithm on the committed dataset, without receiving any raw features or labels?

RQ2 (Data intersection). Can the same proof additionally certify which training samples are drawn from the bank's private dataset, characterizing the intersection $D \cap D'$, without revealing the raw content of those samples to the auditor?

RQ3 (Practicality). What are the concrete proof sizes, prover times, and verifier times of the KZG-oracle protocol on a real financial dataset, and how do they scale with batch size and feature count?

RQ1 is addressed in the Protocol description (Chapter 4, Section 4.7) with corresponding security analysis in Section 5.1.5. RQ2 is addressed in Sections 4.4 and 4.7. RQ3 is addressed in Chapter 6.

1.5. Contributions

This thesis makes the following contributions.

1. **Adaptation of a proof of training to linear and logistic regression.** Abbaszadeh et al. [2] present a zero-knowledge proof of training for deep neural networks using a GKR-style sumcheck with multilinear KZG oracles; Garg et al. [18] provide a zero-knowledge proof of training for logistic regression, but without succinctness. We instantiate the GKR-sumcheck framework for the shallower arithmetic circuits of linear and logistic regression, producing a proof of a single gradient-descent step with size $O(d \log |D'|)$ in the number of features d and the number of training samples $|D'|$.
2. **A three-party protocol for verifiable data provenance.** We design a protocol in which the auditor holds only a proof π and the bank's public commitment PK_D, and verifies two properties simultaneously: that gradient descent was performed honestly on the committed dataset (Step A, via zkPoGD), and which training samples are drawn from the bank's private dataset, certifying $D \cap D'$ (Step B, via the zero-knowledge sets scheme of Micali, Rabin, and Kilian [29]). The two components are connected by deriving each ZK Set's lookup key from the trainer's already-public per-sample KZG commitment. Because the key's input is public, the auditor recomputes it locally, and no additional circuit is required between the two components.
3. **Empirical evaluation on a real financial dataset.** We benchmark the ZkPoGD protocol (used in Algorithms 4 and 5) on a credit-card default dataset across dataset sizes $|D'| \in \{8, \ldots, 256\}$ and feature counts $d \in \{2, \ldots, 23\}$, characterising the concrete proof sizes, prover times, and verifier times, and characterise the security model including the Consistent DataCommitment assumption and its documented open gap.

1.6. Overview of Document

Chapter 3 surveys the existing literature across seven families and identifies for each family why it fails to satisfy the combination of post-hoc verifiability, per-sample granularity, and absence of a trusted setup that we require. Chapter 4 presents the three-party protocol: the sumcheck with multilinear KZG oracles for linear and logistic regression (Step A), the ZK Sets data-authorization mechanism (Step B), and the argument for combining the two components. The security analysis establishes the formal soundness properties of the combined protocol and characterizes the Consistent DataCommitment assumption and the bivariate polynomial commitment that would close it. The empirical evaluation characterizes the proof sizes, prover times, and verifier times of the zkPoGD protocol across the parameter ranges studied. The concluding chapter identifies directions for future work, including the bivariate polynomial commitment needed to close the CDC assumption and the lookup argument required to handle the logistic regression sigmoid without per-batch score leakage.

2

Background

The chapter is primarily designed to understand the primitives needed to implement the protocol in Chapter 4. Some of these results are also important in understanding the security analysis presented in Section 5.1. For every definition or result stated in this section, we also state whether it is used to implement a component in the Protocol or applied to the proof.

Zero-Knowledge Proofs are a powerful cryptographic primitive that help convince others of the validity of a statement without disclosing any details about it. To formalize this definition, we first define the class of interactive proofs ($\mathcal{IP}$).

Definition 2.0.1 (Definition 1: Interactive Proof Systems)**.** *A pair of interactive machines (P, V) is called an interactive proof system for a language L if machine V is polynomial-time and the following two conditions hold:*

- *Completeness: For every $x \in L$,*

$$Pr[< P, V > (x) = 1] \geq \frac{2}{3}$$

- *Soundness: For every $x \notin L$ and every prover B,*

$$Pr[< B, V > (x)] \leq \frac{1}{3}$$

Here $< P, V > (x)$ is used to represent the output of the interaction between P and V when provided with the common input x. The class $\mathcal{IP}$ consists of all languages having interactive proof systems. Now we formalize what is meant by "details regarding a statement".

Definition 2.0.2 (Simulator-based definition Zero Knowledge)**.** *For this, we define an interactive machine M^* which is called the* simulator *for the interaction of a verifier V^* with a prover P. We say that an interactive proof system $< P, V >$ is perfect zero knowledge if for every probabilistic polynomial-time interactive machine V^* there exists an (ordinary) probabilistic polynomial-time algorithm M^* such that for every $x \in L$ the random variables $< P, V^* > (x)$ and $M^*(x)$ are identically distributed [19].*

Relevance

We use the existence of valid simulators to construct simulators for a composition to prove the zero-knowledge property in Section 5.1.5.

Definition 2.0.3 (Honest-Verifier Zero-Knowledge [39])**.** *An interactive proof system $\langle P, V \rangle$ for a language L with witness relation $\mathcal{R}$ is* honest-verifier zero-knowledge *(HVZK) if there exists a probabilistic polynomial-time simulator S such that for every $(x, w) \in \mathcal{R}$,*

$$S(x) \approx_c \text{View}_V \left[\langle P(x, w),\ V(x) \rangle\right],$$

where View_V denotes the complete message transcript seen by the honest verifier V during its interaction with P, and $\approx_c$ denotes computational indistinguishability. If the two distributions are identical, the protocol is perfectly *HVZK.*

When the Fiat-Shamir transform is applied to an HVZK protocol in the random oracle model (Section 2.7), the resulting non-interactive proof is zero-knowledge against *all* verifiers (including malicious ones), since no adaptive verifier challenge can be issued in a non-interactive proof.

Interactive proof systems can be used for tasks such as verifiable computation. This is a crucial building block for zero-knowledge proofs of training [2]. In this case, we consider the prover all-powerful and capable of performing complex computations. The verifier, on the other hand, should be able to make use of the prover to perform such computations and only accept them if they are correct. We now describe the SumCheck protocol, which is the core building block used in the proof of training.

2.1. Cryptographic Commitment Schemes

A cryptographic commitment scheme is the digital analog of a sealed envelope. A sender *commits* to a value m by placing it in the envelope and handing it to the receiver. The receiver cannot see the value (hiding), but the sender cannot later swap in a different value (binding). Formally, a commitment scheme consists of three algorithms: Setup which produces public parameters pp; Commit, which on input pp, message m, and randomness r outputs a commitment $c = \mathsf{Com}(m; r)$; and Open, which reveals (m, r) so the receiver can recompute and check $c = \mathsf{Com}(m; r)$.

Definition 2.1.1 (Hiding and Binding)**.** *A commitment scheme is* perfectly hiding *if for any two messages m_0, m_1, the distributions $\mathsf{Com}(m_0; r)$ and $\mathsf{Com}(m_1; r)$ (over uniformly random r) are identical. It is* computationally binding *if no polynomial-time adversary can produce (m_0, r_0) and (m_1, r_1) with $m_0 \neq m_1$ such that $\mathsf{Com}(m_0; r_0) = \mathsf{Com}(m_1; r_1)$, except with negligible probability.*

Pedersen Commitments. The Pedersen commitment scheme [39] is the building block for the ZK Sets construction used in this thesis. Let $\mathbb{G}$ be a cyclic group of prime order p with generators g and h, where the discrete logarithm of h with respect to g is unknown (i.e., $h = g^x$ for unknown $x \in \mathbb{Z}_p$). The commitment to a message $m \in \mathbb{Z}_p$ with randomness $r \in \mathbb{Z}_p$ is:

$$\mathsf{Com}(m; r) = g^m \cdot h^r = g^m \cdot g^{xr} = g^{m+xr}.$$

Perfect hiding holds because for any fixed m, the commitment g^{m+xr} is uniformly distributed over $\mathbb{G}$ as r ranges over $\mathbb{Z}_p$. *Computational binding* holds because any adversary that opens c to two different messages (m_0, r_0) and (m_1, r_1) with $m_0 \neq m_1$ must satisfy $m_0 + xr_0 = m_1 + xr_1 \pmod p$, from which $x = (m_0 - m_1)(r_1 - r_0)^{-1} \pmod p$, which recovers the discrete logarithm of h, contradicting the hardness assumption.

Pedersen Hash. For vectors, the Pedersen hash of two group elements $(A, B) \in \mathbb{G}^2$ is defined as $H(A, B) = A^u \cdot B^v$ for fixed public exponents u, v. A collision in H requires solving a system of discrete logarithm equations, which is hard under DL.

Relevance

This hash is used in the ZK Sets Merkle tree to commit to pairs of child-node hashes at each internal node (Section 2.12).

2.2. Polynomial Commitments

A polynomial commitment scheme [2] enables a prover $\mathcal{P}$ to commit to a polynomial and later open the commitment at any evaluation point chosen by the verifier, producing a short proof that the evaluation is correct. Formally, a polynomial commitment scheme consists of four procedures:

- KeyGen: On input the security parameter λ, number of variables ℓ, and degree bound d, returns a public parameter pp.
- Commit: On input a polynomial f, randomness ρ, and public parameters pp, returns a commitment σ to f.
- Open: On input a point z, polynomial f, randomness ρ, and pp, returns the evaluation $y = f(z)$ and an evaluation proof π.
- Verify: On input a commitment σ, point z, claimed evaluation y, and proof π, returns Accept or Reject.

A polynomial commitment scheme is *hiding* if the commitment σ reveals no information about f. It satisfies *evaluation binding* if no prover can open a commitment σ to two distinct evaluations $y_0 \neq y_1$ at the same point z. *Knowledge soundness* ensures that a prover who produces a valid proof must know the underlying polynomial. A *zero-knowledge* polynomial commitment additionally ensures that the opening proof π reveals nothing about f beyond the value $f(z)$ itself.

Relevance

Polynomial commitments play the role of the oracle in the sumcheck protocol: rather than the verifier querying f directly (which would require reading all 2^v evaluations), the prover commits to f and later opens the single evaluation $f(r)$ required at the end of each sumcheck round. The concrete instantiation we use is the KZG scheme, described after the necessary pairing-group prerequisites.

2.3. Schwartz-Zippel Lemma

In many of the soundness bounds involved in interactive proof protocols. The following basic property of polynomials is exploited.

Lemma 2.3.1 (Schwartz-Zippel)**.** *Let $\mathbb{F}$ be any field and let $g : \mathbb{F}^m \to \mathbb{F}$ be a nonzero m-variate polynomial of total degree at most d. Then for a finite set $B \subseteq \mathbb{F}$, we have*

$$Pr_{x \leftarrow S^m}[g(x) = 0] \leq \frac{d}{|S|}$$

In other words, if x is chosen uniformly at random from S^m, then the probability that $g(x) = 0$ is at most $d/|S|$. This means that any two distinct polynomials of total degree at most d can agree on at most a $d/|S|$ fraction of points in S^m.

Proof. An elegant proof is provided in [31]. □

Relevance

The Schwartz-Zippel lemma is used to prove soundness in a composition of zero-knowledge proofs in Section 5.1.4.

2.4. Multilinear Extensions

In designing interactive proofs, multivariate polynomials with very low degrees in each variable ensure low communication and fast verification.

Definition 2.4.1 (Multilinearity)**.** *A multivariate polynomial g is multilinear if the degree of the polynomial in each variable is at most one.*

Theorem 2.4.2 (Multilinear Extensions of a polynomial)**.** *Any function $f : \{0,1\}^v \to \mathbb{F}$ has a unique multilinear extension (MLE) over $\mathbb{F}$, and we denote this extension of f with $\tilde{f}$.*

Proof. This theorem states that for every function f, $\tilde{f}$ is the unique multilinear polynomial over $\mathbb{F}$ such that $\tilde{f}(x) = f(x) \forall x \in \{0,1\}^v$. To construct this proof, we first establish the existence of a multilinear polynomial extending f. Then we show that if $\tilde{f}_1$ and $\tilde{f}_2$ are two multilinear polynomials extending the function f. Then $\tilde{f}_1(x) = \tilde{f}_2(x) \forall x \in \{0,1\}^v$. For now, we talk about the existence of such a polynomial. □

Relevance

In zkMLProv, every feature column $\mathbf{x}_j$ and auxiliary vector (sel, err, mask) is encoded as its MLE over $\{0,1\}^{\log |D'|}$ and committed under multilinear KZG (Section 2.11).

Lemma 2.4.3 (Lagrange Interpolation of multilinear polynomials). *Let $f : \{0,1\}^v \to \mathbb{F}$ be any function. Then the following multilinear polynomial $\tilde{f}$ extends f:*

$$\tilde{f}(x_1, \cdots, x_v) = \sum_{w \in \{0,1\}^v} f(w) \cdot \chi_w(x_1, \cdots, x_v),$$

where for any $w = (w_1, \cdots, w_v)$,

$$\chi_w(x_1, \cdots, x_v) := \prod_{i=1}^{v} (x_i w_i + (1 - x_i)(1 - w_i))$$

The set $\{\chi_w : w \in \{0,1\}^v\}$ is the set of multilinear Lagrange basis polynomials with interpolating set $\{0,1\}^v$.

Proof. For proof, please refer to [39]. □

2.5. SumCheck Protocol

The SumCheck Protocol [27] enables a verifier to obtain the following sum for a v-variate polynomial defined over a finite field $\mathbb{F}$.

$$H := \sum_{b_1 \in \{0,1\}} \sum_{b_2 \in \{0,1\}} \cdots \sum_{b_v \in \{0,1\}} g(b_1, \cdots, b_v) \tag{2.1}$$

Many natural problems can be solved using this interactive proof system. The main advantage of a verifier to make use of this protocol is to reduce the computational complexity for obtaining the value of H in equation (2.1) by computing the value of the polynomial g over 2^v inputs to $O(v + [\text{the cost to evaluate } g \text{ over a single point}])$. The pseudocode for the sumcheck protocol is described in Table 1.

Algorithm 1 Sumcheck Protocol

Require: $\mathcal{P}$ knows polynomial $g : \mathbb{F}^v \to \mathbb{F}$ with degree d in each variable
Require: Claimed sum: $H = \sum_{x \in \{0,1\}^\ell} g(x)$
Ensure: Verifier $\mathcal{V}$ accepts iff claim is correct (with high probability)

1: **Round 1:**
2: $\mathcal{P}$: Define univariate polynomial $g_1(X_1) = \sum_{(x_2, \cdots, x_v) \in \{0,1\}^{v-1}} g(X_1, x_2, \cdots, x_v)$.
3: $\mathcal{V}$: Check that $C_1 = g_1(0) + g_1(1)$, and that g_1 is a univariate polynomial of degree at most $deg_1(g)$. If this does not hold, the verifier rejects. $\mathcal{V}$ sends $r_1 \in_R \mathbb{F}$ to $\mathcal{P}$.
4: **Round** $j (1 < j < v)$:
5: $\mathcal{P}$: Sends a univariate polynomial $g_j(X_j) = \sum_{(x_{j+1}, \cdots, x_v) \in \{0,1\}^{v-j}} g(r_1, \cdots, r_{j-1}, X_j, x_{j+1}, \cdots, x_v)$.
6: $\mathcal{V}$: Checks that g_j is a univariate polynomial of degree at most $deg_j(g)$ and $g_{j-1}(r_{j-1}) = g_j(0) + g_j(1)$. If either of these does not hold, $\mathcal{V}$ rejects.
7: **Round** v:
8: $\mathcal{P}$: Sends a univariate polynomial $g_v(X_v)$ claimed to equal $g(r_1, \cdots, r_{v-1}, X_v)$.
9: $\mathcal{V}$: Checks that g_v is a univariate polynomial of degree at most $deg_v(g)$ and whether $g_{v-1}(r_{v-1}) = g_v(0) + g_v(1)$. If either of these does not hold, $\mathcal{V}$ rejects.
10: $\mathcal{V}$: Chooses $r_v \in_R \mathbb{F}$ and evaluates $g(r_1, \cdots, r_v)$ with a single oracle query to g. Checks if $g_v(r_v) = g(r_1, \cdots, r_v)$. If this does not hold, $\mathcal{V}$ rejects.
11: If $\mathcal{V}$ has not yet rejected, $\mathcal{V}$ accepts the proof.

Each variable of the polynomial g is involved in one round of the sumcheck protocol. This gives us the communication complexity of the sumcheck protocol to be

$$\mathcal{P} - \mathcal{V} \text{ communication complexity} = O\left(\sum_{i=1}^{v} deg_i(g)\right).$$

The verifier time complexity over the entire protocol is the total communication complexity plus the cost of a single oracle query to compute $g(r_1, \cdots, r_v)$. To compute the complexity of $\mathcal{P}$'s runtime, we observe that $\mathcal{P}$ can specify g_j by sending

$$g_j(i) = \sum_{(x_{j+1}, \cdots, x_v) \in \{0,1\}^{v-j}} g(r_1, \cdots, r_{j-1}, i, x_{j+1}, \cdots x_v)$$

for every $i \in \{0, \cdots, deg_j(g)\}$. This gives us $\mathcal{P}$'s complexity to be $O(\sum_{i=1}^{v} deg_i(g) \cdot 2^{v-i} \cdot T)$. It is important to note that the provability complexity can be significantly reduced by using multilinear extensions. In particular, we can compute $H = \sum_{x \in \{0,1\}^v} g(x)$ for some function $g : \{0,1\}^v \to \mathbb{F}$ by applying the sum-check protocol on a low-degree extension f of g. This follows from the below lemma

Lemma 2.5.1. *Let $p_1, p_2, \cdots, p_k$ be $l-$variate multilinear polynomials. Suppose that for each p_i there is an algorithm that evaluates p_i at all inputs in $\{0,1\}^l$ in $O(2^l)$ time. Let $f = p_1 \cdot p_2 \cdots p_k$ be the product of these multilinear polynomials. Then the sum-check protocol applied to polynomial f can be implemented by an honest prover in $O(k \cdot 2^l)$ time.*

Proof. For details regarding the proof. Please refer to [12]. □

2.6. Product Sumcheck

A frequent special case of the sumcheck protocol arises when the polynomial g is the pointwise product of two or more multilinear polynomials. The Product Sumcheck handles the two-factor case:

$$H = \sum_{x \in \{0,1\}^v} p_1(x) \cdot p_2(x), \tag{2.2}$$

where $p_1, p_2 : \{0,1\}^v \to \mathbb{F}$ are multilinear. Because each p_i has degree at most 1 in each variable, their product has degree at most 2 in each variable. Consequently, in each round k the prover sends a univariate polynomial of degree 2, specified by three evaluations at $X = 0, 1, 2$.

Evaluation at $X = 2$ by linear extrapolation. Because p_i is multilinear, its restriction to $X_k = X$ with all preceding variables fixed at $(r_1, \ldots, r_{k-1})$ is a degree-1 polynomial in X. Linear extrapolation, therefore gives:

$$p_i(X{=}2) = 2 \cdot p_i(X{=}1) - p_i(X{=}0).$$

The product evaluation at $X = 2$ follows as $p_1(X{=}2) \cdot p_2(X{=}2)$. This avoids any division or polynomial reconstruction; all three evaluations are obtained through simple arithmetic on the current array values.

Folding. After the verifier sends challenge r_k, both arrays are folded in place: for $i < |a|/2$,

$$a[i] \leftarrow (1 - r_k) \cdot a[i] + r_k \cdot a[i + |a|/2],$$

and similarly for b. Folding halves the array size each round. After v rounds the oracle value is simply $a[0] \cdot b[0]$.

Complexity. By Lemma 4.5 of [39], an honest prover runs the Product Sumcheck in $O(k \cdot 2^v)$ time for k factors. For $k = 2$ (degree-2 case, used in the forward pass of our protocol) and $k = 3$ (degree-3 Triple Product Sumcheck, used in the backward pass), this is $O(2^v)$ and $O(3 \cdot 2^v)$ respectively. The Triple Product Sumcheck requires four evaluations per round ($X = 0, 1, 2, 3$), with extrapolation at $X = 3$ given by $3 \cdot p_i(1) - 2 \cdot p_i(0)$. The transcript of the degree-2 product sumcheck over $N = 2^v$ inputs consists of $3 \log N$ field elements (three evaluations per round over v rounds); the degree-3 triple product sumcheck transcript consists of $4 \log N$ field elements.

Relevance

The degree-2 Product Sumcheck verifies the forward-pass gradient sum $S_t^{\mathsf{fwd}} = \sum_{i \in 0,1^v} \widetilde{\mathsf{sel}}(i) \cdot \widetilde{\mathsf{pred}}(i)$, and the degree-3 Triple Product Sumcheck verifies each of the d backward-pass gradient sums $\sum_i \widetilde{\mathsf{sel}}(i) \cdot \widetilde{\mathsf{err}}(i) \cdot \tilde{X}_j(i)$, in Algorithms 4 and 5. At the end of each sumcheck, the oracle query is answered by a multilinear KZG opening on the committed column polynomials (Section 2.11).

2.7. Fiat-Shamir Transform

A *public-coin* interactive proof is one in which every message sent by $\mathcal{V}$ is a uniformly random challenge, revealed to $\mathcal{P}$ as soon as it is chosen [39]. The Fiat-Shamir transformation [39] converts any such ℓ-round protocol $\mathcal{I} = (\mathcal{P}, \mathcal{V})$ into a non-interactive, publicly verifiable argument $Q = (\mathcal{P}_{\mathsf{FS}}, \mathcal{V}_{\mathsf{FS}})$ in the *random oracle model* (ROM). In the ROM, a cryptographic hash function R is modeled as an ideal random function: its output on any fresh input is independent and uniformly distributed, and it is consistent on repeated queries.

In round i of $\mathcal{I}$, the prover sends a message $g_i \in \mathbb{F}$, and the verifier responds with a uniformly random challenge $r_i \in \mathbb{F}$.

Prove. On input (x, w) with $(x, w) \in \mathcal{R}$, the prover $\mathcal{P}_{\mathsf{FS}}$ simulates all ℓ rounds without interaction:

1. Compute the first prover message g_1.
2. For $i = 1, \ldots, \ell$: derive the verifier challenge

$$r_i \leftarrow R(x,\ g_1,\ \ldots,\ g_i),$$

then compute the next prover message g_{i+1} using r_i.

Output the proof $\pi = (g_1, \ldots, g_\ell)$.

Verify. On input (x, π), the verifier $\mathcal{V}_{\mathsf{FS}}$:

1. Parse $\pi = (g_1, \ldots, g_\ell)$.
2. For $i = 1, \ldots, \ell$: recompute $r_i \leftarrow R(x,\ g_1, \ldots, g_i)$.
3. Run the interactive verifier $\mathcal{V}$ on the transcript $(g_1, r_1, \ldots, g_\ell, r_\ell)$; accept iff $\mathcal{V}$ accepts.

The statement x must appear in every hash query. Omitting it allows a cheating prover to fix a transcript first and then search for an x that fits, breaking adaptive soundness [39]. In the ROM, the Fiat-Shamir transformation preserves the completeness and soundness of $\mathcal{I}$. Moreover, if $\mathcal{I}$ is honest-verifier zero-knowledge (Definition 2.0.3), then Q is zero-knowledge against all verifiers. Once the proof string is fixed, no verifier can issue adaptive challenges.

Relevance

In our protocol, SHA-256 instantiates R, which is modeled as a random oracle. All sumcheck challenges and polynomial-commitment evaluation points are derived via the Fiat-Shamir transform.

2.8. Bilinear Mappings

The KZG polynomial commitment scheme (Section 2.10) relies on *bilinear pairings*, which are algebraic structures over elliptic-curve groups that enable a specific type of two-input multiplication in a target group. This section introduces the necessary group-theoretic background.

Groups and Generators Let $(\mathbb{G}, +)$ be a cyclic group of prime order p under point addition. Every element $A \in \mathbb{G}$ can be written as $A = a \cdot g$ for a unique scalar $a \in \mathbb{Z}_p$ and a fixed generator g. The scalar a is called the *discrete logarithm* of A with respect to g, written $a = \log_g A$. The Discrete Logarithm (DL) problem asks to recover a from the pair (g, A); it is believed to be computationally hard in appropriately chosen groups.

Let $g_1 \in \mathbb{G}_1$ and $g_2 \in \mathbb{G}_2$ denote the respective generators. We write $[a]_1 := a \cdot g_1$ and $[a]_2 := a \cdot g_2$ to denote scalar multiples of generators. This bracket notation suppresses the explicit generator and lets us write the group law as addition: $[a]_1 + [b]_1 = [a + b]_1$.

Bilinear Pairings

Definition 2.8.1 (Bilinear Pairing [6])**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$, and $\mathbb{G}_T$ be cyclic groups of the same prime order p. A* bilinear pairing *is an efficiently computable map*

$$e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$$

satisfying the following properties:

- ***Bilinearity**: For all $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$, and $a, b \in \mathbb{Z}_p$,*

$$e(a \cdot A,\ b \cdot B) = e(A, B)^{ab}.$$

- ***Non-degeneracy**: $e(g_1, g_2) \neq 1_{\mathbb{G}_T}$, where $g_1 \in \mathbb{G}_1$, $g_2 \in \mathbb{G}_2$ are the respective generators.*
- ***Computability**: $e(A, B)$ can be computed in polynomial time for any $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$.*

The most important consequence of bilinearity is:

$$e([a]_1,\ [b]_2) = e(g_1, g_2)^{ab} = e([b]_1,\ [a]_2). \tag{2.3}$$

This allows multiplications "in the exponent" to be checked without knowing a or b explicitly. Concretely, given $[a]_1$ and $[b]_2$, one can verify that two ciphertexts encode the same scalar by checking a pairing equation, even though the discrete logarithms are hidden.

Type-3 Pairings and BLS12-381 Pairings come in three types based on the relationship between $\mathbb{G}_1$ and $\mathbb{G}_2$. In a *Type-3* (asymmetric) pairing, $\mathbb{G}_1 \neq \mathbb{G}_2$ and there is no known efficient isomorphism from $\mathbb{G}_2$ to $\mathbb{G}_1$. Type-3 pairings offer the best-known security-to-performance trade-off and are the basis for all modern pairing-based cryptography.

This thesis uses the **BLS12-381** elliptic curve, a Type-3 pairing-friendly curve standardized for use in cryptographic protocols [2]. Its key parameters are:

- A 381-bit prime field $\mathbb{F}_p$, giving elements of $\mathbb{G}_1$ a 48-byte compressed representation.
- An embedding degree of 12, meaning $\mathbb{G}_T \subset \mathbb{F}^*_{p^{12}}$; this gives the pairing 128 bits of security against current best attacks.
- A 255-bit scalar field $\mathbb{F}_r$ (the order of $\mathbb{G}_1$ and $\mathbb{G}_2$), making scalar arithmetic efficient in a 64-bit word.
- Efficient pairing computation via the optimal Ate pairing, with pairing times around 1–2 ms on modern hardware.

2.9. The Strong Diffie-Hellman Assumption

The security of KZG polynomial commitments rests on a specific computational hardness assumption, stronger than the plain Discrete Logarithm problem.

Computational Diffie-Hellman The *Computational Diffie-Hellman* (CDH) problem asks: given $(G, [a]_1, [b]_1)$ for unknown $a, b \in \mathbb{Z}_p$, compute $[ab]_1$. CDH is at least as hard as DL (since computing a from $[a]_1$ directly solves CDH) and is believed to be strictly harder in pairing groups. CDH is the foundation for Diffie-Hellman key exchange and many public-key schemes.

The q-Strong Diffie-Hellman Assumption The q-SDH assumption is a structured version of CDH that allows the adversary to see powers of a secret scalar.

Definition 2.9.1 (q-SDH [2])**.** *Let $\mathbb{G}_1$, $\mathbb{G}_2$ be groups of prime order p with generators G_1, G_2 and a bilinear pairing e. The q-*Strong Diffie-Hellman *(q-SDH) problem is: given the structured reference string*

$$\left([1]_1,\ [\tau]_1,\ [\tau^2]_1,\ \ldots,\ [\tau^q]_1,\ [1]_2,\ [\tau]_2\right)$$

for a uniformly random secret $\tau \leftarrow \mathbb{Z}_p$, find a pair $(c,\ \pi) \in \mathbb{Z}_p \times \mathbb{G}_1$ such that

$$e(\pi,\ [\tau]_2 - [c]_2) = e([1]_1,\ [1]_2), \quad \textit{i.e., } \pi = \left[\frac{1}{\tau - c}\right]_1, \quad c \neq \tau.$$

The q-SDH assumption states that no polynomial-time adversary can solve this problem with non-negligible probability.

Intuitively, q-SDH says that even after seeing q powers of the secret τ in the exponent, it is hard to compute $1/(\tau - c) \cdot G_1$ for any chosen c. This is precisely the computation that would be required to forge a KZG opening proof, as we show in the next section.

The best known algorithms for q-SDH are generic discrete-log algorithms running in time $O(\sqrt{p})$, which is exponential in the bit-length of p, so 128-bit security is achieved with a 255-bit scalar field (as in BLS12-381).

2.10. KZG Polynomial Commitments

The Kate-Zaverucha-Goldberg (KZG) scheme [2] is the polynomial commitment underlying the column oracles in our zkPoT. It achieves an $O(1)$- size commitment and an $O(1)$- size proof (each a single group element), with verification requiring a constant number of pairing operations independent of the polynomial degree.

Trusted Setup The scheme requires a *structured reference string* (SRS) generated from a secret τ that must be discarded after setup. On security parameter λ and degree bound d:

$$\text{srs} = \Big(\underbrace{[1]_1,\ [\tau]_1,\ [\tau^2]_1,\ \ldots,\ [\tau^d]_1}_{\in\,\mathbb{G}_1^{d+1}},\ \underbrace{[1]_2,\ [\tau]_2}_{\in\,\mathbb{G}_2^2}\Big).$$

The prover can compute $[f(\tau)]_1 = \sum_{i=0}^{d} f_i \cdot [\tau^i]_1$ for any polynomial $f(X) = \sum_i f_i X^i$ of degree $\le d$, without knowing τ directly. This is the key property of the SRS: it encodes powers of τ in the exponent, enabling polynomial evaluation at the secret point using only group operations.

Commitment Given srs and a polynomial $f \in \mathbb{F}_p[X]$ of degree $\le d$, the commitment is:

$$C_f := [f(\tau)]_1 = \sum_{i=0}^{d} f_i \cdot [\tau^i]_1. \tag{2.4}$$

This is a single element of $\mathbb{G}_1$ (48 bytes on BLS12-381) regardless of the degree d.

Evaluation and Proof To prove that $f(z) = y$ for a point $z \in \mathbb{F}_p$:

1. Compute the remainder $y = f(z)$ by evaluating f at z using the coefficient vector.
2. By the Polynomial Remainder Theorem, $(X - z)$ divides $f(X) - y$ exactly, so the *quotient polynomial*

   $$q(X) := \frac{f(X) - y}{X - z}$$

   is a polynomial of degree $d - 1$ with no remainder.
3. Compute the proof:

   $$\pi := [q(\tau)]_1 = \sum_{i=0}^{d-1} q_i \cdot [\tau^i]_1. \tag{2.5}$$

The prover sends (y, π) to the verifier. The proof π is a single $\mathbb{G}_1$ element.

Verification The verifier checks the following pairing equation:

$$e\big(C_f - [y]_1,\ [1]_2\big) \ = \ e\big(\pi,\ [\tau]_2 - [z]_2\big). \tag{2.6}$$

Correctness. Expanding both sides using bilinearity (Equation 2.3):

$$\text{LHS} = e\big([f(\tau) - y]_1,\ [1]_2\big) = e(G_1, G_2)^{f(\tau)-y},$$
$$\text{RHS} = e\big([q(\tau)]_1,\ [\tau - z]_2\big) = e(G_1, G_2)^{q(\tau)(\tau-z)}.$$

These are equal if and only if $f(\tau) - y = q(\tau)(\tau - z)$, which holds if and only if $f(z) = y$ (since $q(X)(X - z) = f(X) - y$ as polynomials, so evaluating at τ gives the equality).

Soundness under q-SDH. Suppose a cheating prover can open C_f to two distinct values $y_0 \neq y_1$ at the same point z, with proofs π_0 and π_1. Then:

$$e(\pi_0 - \pi_1,\ [\tau - z]_2) = e([y_1 - y_0]_1,\ [1]_2).$$

Rearranging, the cheating prover has computed $[1/(\tau - z)]_1$ (scaled by the known value $y_1 - y_0$), which is exactly the q-SDH challenge with $c = z$. Any cheating prover therefore breaks the $(d + 1)$-SDH assumption, so KZG is evaluation-binding under q-SDH.

Hiding via Blinding The commitment $C_f = [f(\tau)]_1$ is not hiding by default: if the verifier can query f at enough points, they can reconstruct f by Lagrange interpolation. To achieve hiding, the prover uses a *blinding polynomial*: replace f with $\hat{f}(X) = f(X) + \delta \cdot (X - z)$ for a uniformly random $\delta \leftarrow \mathbb{F}_p$ before committing. Since $\hat{f}(z) = f(z)$, the evaluation is unchanged, but the commitment now hides f information-theoretically against a verifier who sees only the evaluation at z. In our implementation, the blinding polynomial is added per query.

Complexity. Computing a commitment C_f requires $O(d)$ scalar multiplications in $\mathbb{G}_1$, one per SRS element. Generating an opening proof π requires $O(d)$ group operations to evaluate the quotient polynomial at τ. Verification (Equation 2.6) requires exactly two pairing evaluations, independent of d [39].

2.11. Multilinear KZG Commitments

The sumcheck protocol (Section 2.5) produces random evaluation points in $\mathbb{F}^v$ for v-variate polynomials, not scalar points in $\mathbb{F}$. The univariate KZG scheme of the previous section handles only single-variable polynomials. To commit to multilinear polynomials over $\{0,1\}^v$, as required for the feature-column oracles in our zkPoT, we use the multilinear KZG scheme of Papamanthou, Shi, and Tamassia (PST13) [2].

Multilinear Polynomials and the SRS Let $f : \mathbb{F}^v \to \mathbb{F}$ be a multilinear polynomial:

$$f(X_1, \ldots, X_v) = \sum_{S \subseteq [v]} f_S \prod_{i \in S} X_i,$$

with one coefficient $f_S \in \mathbb{F}$ per subset $S \subseteq \{1, \ldots, v\}$ (so 2^v coefficients total). The PST13 SRS encodes all products of the secrets $(\tau_1, \ldots, \tau_v)$ in the exponent:

$$\mathsf{srs}_{\mathrm{ML}} = \left\{ \left[\prod_{i \in S} \tau_i \right]_1 : S \subseteq [v] \right\} \cup \left\{ [1]_2,\ [\tau_1]_2,\ \ldots,\ [\tau_v]_2 \right\}.$$

The $\mathbb{G}_1$ part has 2^v elements and the $\mathbb{G}_2$ part has $v + 1$ elements.

Commitment, Opening, and Verification

Commitment. Given f and $\mathsf{srs}_{\mathrm{ML}}$:

$$C_f := [f(\tau_1, \ldots, \tau_v)]_1 = \sum_{S \subseteq [v]} f_S \cdot \left[\prod_{i \in S} \tau_i \right]_1.$$

This is again a single $\mathbb{G}_1$ element, regardless of v.

Opening at $\mathbf{r} = (r_1, \ldots, r_v)$. By the multilinear quotient identity, any multilinear f satisfies:

$$f(X_1, \ldots, X_v) - f(r_1, \ldots, r_v) = \sum_{j=1}^{v} (X_j - r_j) \cdot q_j(X_1, \ldots, X_v),$$

where each q_j is itself a multilinear polynomial of degree $v - 1$ (one lower than f). The opening proof consists of v commitments, one per quotient:

$$\pi_j := [q_j(\tau_1, \ldots, \tau_v)]_1, \quad j = 1, \ldots, v.$$

Verification. The verifier checks:

$$\sum_{j=1}^{v} e\big(\pi_j,\ [\tau_j - r_j]_2\big) = e\big(C_f - [y]_1,\ [1]_2\big), \tag{2.7}$$

where y is the claimed evaluation $f(r_1, \ldots, r_v)$. Verification requires v pairings on the left and one on the right.

Connection to the sumcheck. After v rounds of the sumcheck protocol, the verifier holds a random challenge point $\mathbf{r} = (r_1, \ldots, r_v) \in \mathbb{F}^v$ and needs to check that the oracle value $f(\mathbf{r})$ is correct. The multilinear KZG opening proof (Equation 2.7) supplies this check in v pairings, replacing what would otherwise be a query to all 2^v evaluations of f. In our zkPoT, each column_commits[j] is a PST13 commitment to the j-th feature column, and the sumcheck verifier checks the column evaluations at the Fiat-Shamir challenge point $\mathbf{r}$ using this verification equation.

Complexity. For a multilinear polynomial with $N = 2^\ell$ evaluations, commitment requires $O(N)$ scalar multiplications in $\mathbb{G}_1$. Generating an opening proof at $\mathbf{r} \in \mathbb{F}^\ell$ requires $O(N)$ group operations to compute the ℓ quotient polynomials; the proof itself consists of $\ell = \log N$ group elements. Verification (Equation 2.7) requires $\ell + 1 = O(\log N)$ pairing evaluations [39].

2.12. Zero-Knowledge Sets

This section describes the ZK Sets mechanism used in Step B of our protocol. The goal is to allow the bank to commit to its authorized dataset D before training, and later allow the auditor to verify, for any specific sample, whether it is authorized, without learning anything about the rest of D.

Encrypted Databases A *zero-knowledge encrypted database* (ZK EDB) [29] is a generalization of a set: it is a partial function $D : \mathcal{K} \rightharpoonup \mathcal{V}$ mapping keys to values. The prover commits to D with a short commitment PK_D, and can later produce a proof for any query k:

- **Membership:** If $k \in \mathrm{dom}(D)$, prove $D(k) = v$ for the correct value v.
- **Non-membership:** If $k \notin \mathrm{dom}(D)$, prove $D(k) = \perp$.

The proofs must be zero-knowledge: a verifier who sees PK_D and the proof for key k learns only the answer to the query about k, and nothing about any other key or value in D. A ZK Set is the special case where all values are a fixed constant (e.g., " authorized ").

Pedersen-Merkle-Tree Construction We describe the construction of Micali, Rabin, and Kilian [29]. Fix a security parameter k. The scheme uses a complete binary tree T_k of depth k with 2^k leaves, and a collision-free Pedersen hash $H(a, b) = g^a \cdot h^b \bmod p$ over a prime-order group $G \subset \mathbb{Z}_p^*$ of order q, where $\log_g h$ is unknown to the committer.

Full and frontier nodes. Each node v stores a Pedersen commitment $c_v = g^{m_v} \cdot h_v^{r_v} \bmod p$, where h_v is a node-specific generator and r_v is drawn via a pseudorandom function keyed on a secret seed. Nodes fall into two categories:

- **Full nodes** are ancestors of actual database entries. Their generator is $h_v = h^{e_v}$, where e_v is pseudorandom and $\log_g h_v$ is unknown. The commitment is genuinely binding.
- **Frontier nodes** are boundary nodes just outside the subtree covering the database entries. Their generator is $h_v = g^{e_v}$, where the committer knows $e_v = \log_g h_v$. The commitment is a trapdoor that can be opened to any value.

Each key x maps to a leaf position $H(x) \in \{0,1\}^k$. For a full leaf, the stored message is $m_v = H(D(x))$; for a frontier leaf, $m_v = 0$, which is not a valid hash output and unambiguously signals non-membership. Internal nodes compute $m_v = H(c_{v_0}, h_{v_0}, c_{v_1}, h_{v_1})$ (a collision-resistant encoding of their two children). The root commitment $(c_\varepsilon, h_\varepsilon)$ is published as PK_D.

An implementation must store five values per node: the commitment c_v, the message m_v, the node-specific generator h_v, the blinding factor r_v, and the generator exponent e_v (the trapdoor for frontier nodes, or the pseudorandom index for full nodes). At 256 bits each, a tree of depth k over a dataset of size $|D|$ occupies $O(5 \cdot 32 \cdot k \cdot |D|)$ bytes in memory.

Membership Proofs To prove $D(x) = y$ for $x \in \mathrm{dom}(D)$, the prover reveals the full node values along the path from the leaf $H(x)$ to the root. For each node v on that path, the prover provides (m_v, h_v, r_v) and the sibling commitment c_u. The verifier checks:

1. $c_v = g^{m_v} \cdot h_v^{r_v}$ (Pedersen opening).
2. At each internal node, $m_v = H(c_{v_0}, h_{v_0}, c_{v_1}, h_{v_1})$ (parent–child consistency).
3. At the leaf, $m_{H(x)} = H(y)$ (the stored message is the hash of the value).
4. The root commitment matches PK_D.

Because internal nodes are full nodes ($h_v = h^{e_v}$ with DL unknown), the Pedersen hash is binding,g, and the prover cannot forge a path. Proof size is $O(k)$ group elements.

Non-Membership via Frontier Nodes To prove $D(x) = \perp$ for $x \notin \mathsf{dom}(D)$, the prover exploits the frontier-node trapdoor. The path from the root to the leaf $H(x)$ follows full nodes until it reaches the first *frontier* ancestor, say, at depth d'. All nodes from depth d' down to the leaf $H(x)$ are frontier nodes whose generators $h_v = g^{e_v}$ the prover knows. The prover:

1. Provides the path from the root to depth d' exactly as in the membership case (full nodes, binding).
2. Extends the path from depth d' to leaf $H(x)$ using fresh frontier commitments, setting $m_{H(x)} = 0$ at the leaf (0 is not a valid hash output, unambiguously signaling absence).
3. Recomputes all internal messages along the extension using the known trapdoors and shows consistency with the last full-node commitment at depth d'.

The verifier checks the full-node segment against PK_D, verifies the frontier extension, and accepts if $m_{H(x)} = 0$. Because the prover cannot alter the full-node segment (binding), the frontier extension can only be consistent if the leaf was genuinely absent from the committed set. Non-membership proof size is $O(k)$ group elements.

Security
Theorem 2.12.1 (ZK Sets Security, informal [29])**.** *Under the Discrete Logarithm assumption in $\mathbb{G}$:*

- ***Soundness**: No polynomial-time prover can produce a valid membership proof for a key $k \notin \mathsf{dom}(D)$, or a valid non-membership proof for a key $k \in \mathsf{dom}(D)$, except with negligible probability.*
- ***Zero-knowledge**: There exists a simulator that, given only PK_D and the query key k and its answer, produces a view indistinguishable from the real proof, without knowing any other entries of D.*

Soundness reduces to the binding of the Pedersen hash: a cheating prover who forges a membership proof finds a collision $H(A, B) = H(A', B')$ with $(A, B) \neq (A', B')$, which contradicts DL hardness. Zero-knowledge follows from the perfect hiding of Pedersen commitments: nodes not on the queried path can be replaced by random group elements without changing the verifier's view.

Complexity. Committing to a set of n elements requires $O(kn)$ hash evaluations and modular exponentiations in total, where k is the security parameter (tree depth). Membership proofs are generated in $O(1)$ time by a table lookup of pre-stored path values, and verified with $O(k)$ Pedersen hash checks. Non-membership proofs require $O(k)$ hashings and modular exponentiations to generate, and $O(k)$ checks to verify [29].

Relevance

In our protocol, the database D maps each authorised sample's *per-sample KZG commitment* to the constant value "authorised":

$$D\big(\mathsf{CommitmentToZkSetsKey}(\mathsf{sample\_poly\_commits}[i])\big) = \texttt{authorized}.$$

The bank builds D before training and publishes PK_D. During auditing (Step B), for each sample index i used in any training batch, the auditor derives the ZK Sets key from the already-public sample_poly_commits$[i]$ and verifies a membership proof against PK_D. Because the key derivation is a public deterministic function, the auditor computes it locally without requiring any additional circuit or proof. The connection between sample_poly_commits$[i]$ and the column oracles column_commits$[j]$ used in Step A is discussed in Section 5.1.1.

2.13. Masking Polynomials

An interactive proof is zero-knowledge if the verifier learns nothing beyond the truth of the claim. In the standard sumcheck, the round polynomial $g_k(X)$ is a partial sum of the underlying function g: seeing enough rounds can reveal structure about g itself. Two approaches exist to make sumcheck zero-knowledge [39].

The first adds a uniformly random polynomial to the transcript in each round, ensuring that every message is uniformly random. Masking achieves perfect zero-knowledge but roughly doubles the communication.

The second, more efficient approach is the *masking polynomial* technique [39], which is the method used in this thesis. The prover samples a random function $R : \{0,1\}^v \to \mathbb{F}$ satisfying the zero-sum constraint:

$$\sum_{x \in \{0,1\}^v} R(x) = 0.$$

It then runs the sumcheck on $g'(x) = g(x) + R(x)$ in place of $g(x)$. Because R sums to zero over the hypercube, the claimed total is unchanged: $\sum_x g'(x) = H$. However, since R is drawn uniformly subject only to the zero-sum constraint, the joint distribution of all round messages for g' is independent of g, making the transcript perfectly hiding.

Commitment requirement. The prover must commit to R *before* any challenge is drawn. If R could be chosen after seeing the Fiat-Shamir challenges, the prover could adapt R to cancel out information in a targeted way, thereby breaking soundness. In our protocol, SampleZeroSumMask samples R and the commitment mask_commit $= \mathrm{mkzg}(R)$ is published before the sumcheck transcript begins. At the final oracle query point r, the verifier opens mask_commit at r and checks

$$g'(r) = g(r) + R(r),$$

where $g(r)$ is reconstructed from the already-published KZG column commitments. This single check closes the Oracle query while maintaining zero-knowledge.

2.14. Polynomial Sigmoid Approximation

Logistic regression maps a linear score $s_i = \mathbf{w}^\top \mathbf{x}_i + b$ to a class probability using the sigmoid function $\sigma(x) = 1/(1 + e^{-x})$. The sigmoid is not a polynomial, which poses a fundamental problem for ZKP-based verification that uses the sumcheck protocol and multilinear KZG commitments. Any non-polynomial operation in the computation must be replaced by a polynomial before a proof can be constructed.

This thesis follows [2] and replaces σ with its degree-3 Taylor approximation at $x = 0$:

$$P_3(x) = \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3.$$

The x^2 coefficient is absent because $\sigma''(0) = 0$. For $x \in [-5, 5]$, the maximum pointwise error is less than 0.03. Since features are min-max normalized to $[-1, +1]$ and the training regime keeps weights small, the linear score s_i typically falls within this range.

Replacing σ with P_3 makes every operation in the forward and backward passes polynomial, so the sumcheck reduction applies without modification. The degree-3 term does introduce one complication: the prover must certify the score, the square of the score, and the cube of the score without revealing any per-sample value. The protocol addresses this by publishing MKZG commitments to these vectors and running four Fiat-Shamir consistency checks before the backward-pass sumchecks.

Relevance

The polynomial sigmoid approximation P_3 is applied in the logistic regression branch of Trainer Prove (Section 4.6) and Auditor Verify (Section 4.7). The additional commitment overhead it introduces is quantified in Section 6.

3

Related Work

Deciding whether a specific record was used to train a machine learning model, and proving that decision to a sceptical third party, draws on two separate bodies of literature. The first concerns membership signals, examining what information about training data is recoverable from a trained model and how reliably such inference can be made. The second concerns verifiable computation, examining how to certify that a learning algorithm was executed correctly on a committed dataset without revealing that dataset to the verifier. This chapter examines both bodies of work and shows that existing approaches address each question in isolation, leaving a gap that the thesis protocol is designed to close.

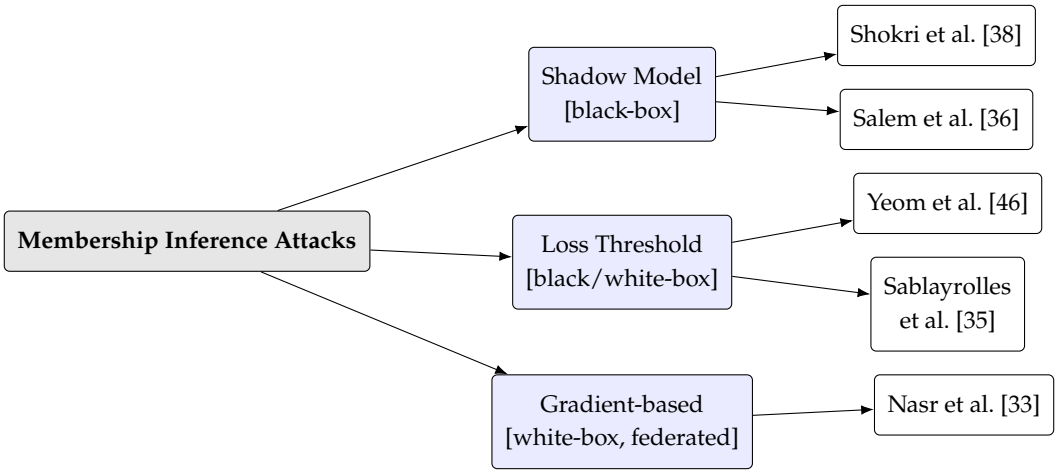


Figure 3.1: Taxonomy of membership inference attack techniques covered in this section.

3.1. Privacy Challenges in Machine Learning

In privacy-related attacks, an adversary attempts to recover information about the training data or the model that was not intended to be disclosed. Privacy attacks span several families, including reconstruction, property inference, model extraction, and membership inference. Of these, only membership inference attacks (MIAs) directly address the question of whether a specific record was present in the training set, making them the most natural candidate for a data provenance proof. Figure 3.1 organises the MIA literature by the mechanism used to mount each attack. The following paragraphs examine each mechanism in turn and establish why none provides a sound provenance proof.

Shokri et al. [38] introduce the first systematic MIA. Machine learning models trained with gradient descent memorise training records and consequently exhibit higher softmax confidence on members that on unseen data. In their proposed attack, an adversary with black-box access to the model's output confidence vector trains a collection of shadow models on auxiliary data to mimic the target model's behaviour. A meta-classifier

trained on the shadow models' outputs learns to distinguish member confidence patterns from non-member ones, achieving a median attack precision of 94% against Google Prediction API and 74% against Amazon ML on retail transaction datasets, and over 70% on a hospital discharge dataset. Salem et al. [36] relax three key assumptions of this attack: a single shadow model suffices, the shadow data need not match the target distribution, and knowledge of the target model architecture is not required. The relaxations significantly lower the barrier to mounting an attack, though the adversary still requires repeated query access to the target model.

A different family of attacks achieves membership inference without shadow model training entirely. Yeom et al. [46] formalise the connection between overfitting and membership leakage. They show that a training member's per-sample loss is systematically lower than a non-member's precisely when the model has memorised training data. A simple likelihood-ratio threshold on the loss achieves a membership advantage bounded by the generalisation gap. Sablayrolles et al. [35] prove that the optimal membership inference strategy depends only on the per-sample loss, implying that white-box access to model parameters provides no additional advantage over a black-box loss-based attack. Together, these results show that the loss-threshold attack space is well-understood and saturated.

Nasr et al. [33] demonstrate that white-box access to model gradients substantially strengthens membership inference beyond what black-box confidence vectors allow. In the passive variant, gradient vectors serve directly as membership signals without modifying the training protocol. In the active variant, the adversary manipulates the aggregated gradient update to amplify membership signals for targeted records, reaching 76.7% accuracy for an adversarial participant and 82.1% for a central server on CIFAR-100, compared to 72.2% and 79.2% in the passive setting. Although their evaluation targets the federated setting, the same gradient signal is available in centralised training where the server has full parameter visibility. The attack nonetheless produces only a probabilistic score and leaves no artifact that a third-party auditor can verify after training concludes.

Maini et al. [28] challenge the reliability of per-record MIAs, showing that apparent successes are confounded by temporal distribution shifts rather than genuine membership signals. Methods such as MIN-K% PROB achieve high AUC when members and non-members are drawn from different time periods but perform no better than random guessing on IID splits of the same dataset. They propose dataset inference as a more robust alternative, aggregating 52 MIA variants into a single statistical test at the dataset level, which correctly distinguishes training and validation splits of the Pile at significance level 0.1 with no false positives. The method requires both that the suspect and validation sets be IID and that the validation set remain entirely private to the victim, conditions that cannot be assumed to hold following an unauthorised data breach.

Zhang et al. [47] formalise the soundness problem common to MIA-based proofs. Estimating the false positive rate requires a reference model trained without the target record, which is infeasible when the auditor does not know the full training procedure. They show that common heuristics for approximating the FPR are statistically unsound, and propose membership tests on injected canaries, detection of watermarked training data, and verbatim extraction attacks as alternatives that do admit sound inference. Each requires anticipatory intervention before data is stolen, a limitation shared by the watermarking techniques discussed in Section 3.2 and equally inapplicable to the breach scenarios described in Section 1.1.

3.2. Dataset Watermarks and Canaries

Uchida et al. [40] introduce the first general framework for embedding a watermark in model parameters. They achieve this by modifying the training objective to include a secondary term that encodes a hidden bit-string into model parameters.The secondary term penalises the distance between a secret linear projection of the model's weight vector and the target bit pattern, steering selected linear combinations of weights toward predetermined sign values without disrupting gradient descent on the primary objective. Any party holding the secret key can then extract the watermark from a suspect model and verify that the model was trained under the owner's supervision. The watermark does not affect the model's utility. The classification accuracy on CIFAR-10 is virtually unchanged and the mark persists after aggressive compression. Sheybani et al. [37] extend model watermarking to a zero-knowledge setting, proving that a watermark can be extracted from a deployed model without revealing the private trigger key. The zero-knowledge proof establishes that the model's parameters satisfy the embedding constraint for some valid secret key, without the verifier learning anything about the projection matrix or the trigger set used during training. The proof mechanism does not, however, change the fact that the watermark must be injected before the training begins. In the breach scenarios considered in this

thesis, the adversary trains independently on stolen data without the data owner's intervention; no watermark is embedded, and verification fails.

Rouhant et al. [34] shift the embedding target from model weights to activation statistics using their proposed framework DeepSigns. This framework records the owner's signature in the probability density function of activation maps across chosen network layers, tying the mark to both the input data and the model architecture. Experiments show that the embedded mark survives pruning, fine-tuning, and deliberate overwriting. Despite these robustness properties, the approach offers no remedy for data provenance. If records were taken without authorisation before any embedding took place, no trace of owner supervision exists in the stolen data, and no post-hoc stripping argument is relevant.

A structural limitation cuts across all watermarking families. Every scheme certifies presence of the owner's mark, but none certifies its absence. If a data subject withdraws consent under GDPR Article 17 [15], the bank must demonstrate that the record plays no part in any retraining. This is a claim no watermark can support. The approach required for data provenance must instead operate retrospectively on committed data and supply cryptographic certificates for both record inclusion and exclusion without advance preparation by the data owner. The sections that follow examine the cryptographic primitives that the thesis protocol draws on to close this gap, beginning with the tools that at first appear applicable but ultimately fall short.

3.3. Private Set Intersection

Private set intersection (PSI) is a two-party cryptographic protocol in which a client and a server, each holding a private set, jointly compute their set intersection while learning nothing beyond the result. For a protocol requiring a bank to certify record inclusion to a trainer without disclosing its full dataset, PSI appears at first to be the natural primitive, since it answers exactly this kind of membership question between two private inputs. Formalised by Freedman et al. in 2004 [17] the area has since matured through successively efficient instantiations drawing on oblivious transfer extensions, oblivious polynomial evaluation, homomorphic encryption, and oblivious key-value stores [30, 41]. State-of-the-art two-party PSI can intersect sets of one million elements in under six seconds over a 100 Mbit/s link [41].

Kamara et al. [23] offload the computational burden of PSI to an untrusted server without compromising the privacy of either input party. Each party contributes only a linear number of block-cipher operations while the server performs the intersection computation, scaling to one billion elements over wide-area networks with performance improvements of five orders of magnitude over general-purpose secure computation. The protocol achieves security against semi-honest, covert, and malicious adversaries, and supports hiding the cardinality of set intersection. The server offloading reduces cost for the input parties but introduces no verifiable record of correct computation. An auditor has no artifact to inspect after the protocol concludes, and both input parties must still be simultaneously online to receive the result.

Falzon et al. [16] enrich the PSI model with an authorisation requirement, introducing Authorised PSI (APSI). A trusted judge certifies the client's input set in a preliminary Authorise phase before any intersection is computed. The judge's digital signature then serves as a credential during the single-round Intersect phase, eliminating the need for further judge participation. A partial variant reveals only a fraction of the client's elements to the judge, trading full authorisation coverage for a guarantee that the judge sees only a sampled fraction of the client's input set and cannot reconstruct the client's complete dataset from what it observes. Malicious security against a corrupt client is achieved under the Decisional Bilinear Diffie-Hellman assumption, with linear communication complexity.

Despite these advances, every PSI variant shares a structural limitation that rules it out for data provenance certification. PSI produces a plaintext intersection. Once the protocol concludes, no persistent cryptographic artifact remains that a third party can independently verify. Both input parties must be simultaneously online during execution, and an auditor absent at that time has no means of checking the output later. APSI certifies inputs before the protocol runs, but the intersection itself is still revealed in plaintext and no post-execution certificate is produced. The primitive required here is fundamentally different: a compact, self-contained proof that a specific element belongs to a committed set, verifiable by any party at any future time, without revealing the set itself or requiring any party to be online.

3.4. Zero-Knowledge Sets

Cryptographic accumulators, introduced by Benaloh and de Mare [4], allow a party to compress an arbitrary set into a single short value from which any included element can later be proved present through a constant-sized membership witness. The construction uses modular exponentiation over a rigid RSA modulus.Quasi-commutativity ensures the accumulated value is order-independent, and witness verification costs exactly one exponentiation. Camenisch and Lysyanska [9] make accumulators dynamic by introducing a trapdoor that permits efficient insertion and deletion, and show that membership witnesses can be updated at unit cost without the trapdoor. They additionally construct zero-knowledge proofs of membership, making the scheme applicable to anonymous credential revocation. Both constructions rely on the strong RSA assumption and require a trusted setup to generate the modulus, as knowledge of the prime factors enables forging witness. Neither construction supports non-membership proofs in zero knowledge.

Zero-knowledge sets, introduced by Micali, Rabin, and Kilian [29], address both limitations of accumulators in a single construction. A prover commits to a finite set and can later produce a self-contained certificate answering membership or non-membership for any queried string, revealing nothing about the set beyond the answer to that query. The scheme requires no trusted setup and its security reduces to the hardness of the discrete logarithm problem. The full construction and its integration into the thesis protocol are described in Section 2.12.

Benarroch et al. [5] reformulate ZK set membership as a commit-and-prove problem. The prover commits to the queried element with a Pedersen commitment and proves membership or non-membership relative to that commitment, without revealing u to the verifier. The membership gadget is modular and composes with any compatible proof system such as Bulletproofs or Groth16 as a plug-and-play component, with non-membership as a first-class operation. Proof size is logarithmic in the number of set elements. Table 3.1 summarises these constructions and their predecessors.

Table 3.1: Comparison of cryptographic set membership constructions (per-element asymptotic complexity).

Construction	Proof size	Verify	Non-mem.	ZK witness	Setup
Benaloh & de Mare [4]	$O(1)$	$O(1)$	×	×	Trusted
Camenisch & Lysyanskaya [9]	$O(1)$	$O(1)$	×	✓	Trusted
MRK [29]	$O(k)^*$	$O(k)$	✓	✓	Transparent
Benarroch et al. [5]	$O(\log N)$	$O(\log N)$	✓	✓	Optional
Curve Trees [11]	$O(\log N)$	$O(N^{1/D})$	×	✓	Transparent
Curve Forests [10]	$O(\log N)^\dagger$	$O(N^{1/D})^\dagger$	×	✓	Transparent

$^*O(k)$ is constant in N: tree depth is fixed by the security parameter k, not the set size. †Amortised per element in a batch; D is the depth parameter of the curve-tree construction (small constant).

Curve Trees [11] replace the Pedersen trie with a tree of inner-product arguments over cycles of elliptic curves, eliminating the need for a trusted setup entirely. For a set of 2^{40} elements the proof is 2.9,KB and verification takes 40,ms on the Pasta curve instantiation. Curve Forests [10] extend this to the batching setting; a batch of elements enjoys approximately $2\times$ proving speedup, $3\times$ verification speedup, and 60% proof-size reduction relative to the same number of independent Curve Trees proofs. Neither construction provides non-membership proofs. Thus rendering them futile in Right To Erasure claims.

3.5. Zero-Knowledge Proofs of Training

A zero-knowledge proof of training (zkPoT) is a cryptographic primitive that certifies the output of a learning algorithm without revealing its inputs. Garg et al. [18] introduce the first formal definition, capturing correctness, soundness, and zero-knowledge as three independent properties of the scheme. Definition 3.5.1 states these formally; Table 3.2 compares the constructions that instantiate them.

Definition 3.5.1 (Zero-Knowledge Proof of Training [18])**.** *A* zero-knowledge proof of training *(zkPoT) for $\mathcal{L}$ is a triple of PPT algorithms* (Commit, Prove, Verify)*:*

- $c_{\mathcal{D}} \leftarrow \mathsf{Commit}(\mathcal{D};\, \rho)$*: commits to the dataset using randomness ρ.*

- $(M,\, c_M,\, \pi) \leftarrow \mathsf{Prove}(\mathcal{D};\, \rho)$*: runs* $\mathcal{L}$ *on* $\mathcal{D}$*, outputs the trained model* M*, a commitment* c_M *to* M*, and a proof* π*.*
- $b \leftarrow \mathsf{Verify}(c_{\mathcal{D}},\, c_M,\, \pi)$*: outputs* $b \in \{0, 1\}$*.*

The scheme must satisfy the following three properties.

- ***Correctness.*** *An honestly generated proof is always accepted: for any* $c_{\mathcal{D}} = \mathsf{Commit}(\mathcal{D}; \rho)$*,*

$$\Pr\Big[\, b = 1 \;\Big|\; (M,\, c_M,\, \pi) \leftarrow \mathsf{Prove}(\mathcal{D}; \rho), \quad b \leftarrow \mathsf{Verify}(c_{\mathcal{D}},\, c_M,\, \pi) \,\Big] = 1.$$

- ***Soundness.*** *No PPT adversary* $\mathcal{A}$ *can convince the verifier to accept an incorrect model: for any* $c_{\mathcal{D}} = \mathsf{Commit}(\mathcal{D}; \rho)$*,*

$$\Pr\left[\, b = 1 \;\wedge\; M^* \neq M \;\middle|\; \begin{array}{l} (M,\, c_M,\, \pi) \leftarrow \mathsf{Prove}(\mathcal{D}; \rho) \\ (M^*,\, c^*_M,\, \pi^*) \leftarrow \mathcal{A}(\mathcal{D}, \rho) \\ b \leftarrow \mathsf{Verify}(c_{\mathcal{D}},\, c^*_M,\, \pi^*) \end{array} \right] = \mathsf{negl}(\lambda).$$

- ***Zero-knowledge.*** *There exists a PPT simulator* Sim *such that for all* $\mathcal{D}$ *and* ρ*, the output of* $\mathsf{Sim}(1^\lambda)$ *is computationally indistinguishable from the honest transcript* $(c_{\mathcal{D}},\, c_M,\, \pi)$*.*

Table 3.2: Zero-Knowledge Proofs of Training: comparison

Paper	ML Model	Succinct?	ZK?	Data authorisation?
[18] (Garg et al.)	Logistic regression	No	Yes	None
[2] (Kaizen)	Deep neural networks	Yes	Yes	None

The two constructions that satisfy Definition 3.5.1 differ significantly in how they achieve these guarantees and at what computational cost.

Garg et al. [18] instantiate zkPoT for logistic regression by combining MPC-in-the-head paradigm [22] with zk-SNARKs. In this paradigm, the prover mentally simulates a virtual multi-party execution of training algorithm, commits to each virtual party's view, and uses zk-SNARK sub-proofs to certify the consistency of the shared randomness and data shares. The resulting protocol divides into three phases: a *data-independent offline phase* that pre-computes correlated randomness, a *data-dependent phase* that verifies consistency of the packed training-data shares, and an *online phase* that depends on both the committed data and the trained model weights. On a $4\,\mathrm{GB}$ dataset with 1024 features and $262{,}144$ records, total proof size is under $350\,\mathrm{MB}$ and online prover and verifier times are under 10 minutes and 30 seconds respectively; however, the data-dependent prover phase takes approximately one hour. For a logistic regression model trained over serveral hundred epochs this means total proof generation would require days of compute, which renders the protocol impractical for training-scale deployment beyond proof-of-concept runs. Proof size grows proportionally to the circuit representing one gradient-descent step, making the protocol *non-succinct*: each additional training iteration adds a proportional contribution to the proof.

Circuit-based zkPoT protocols such as Garg et al. produce proofs that grow with the number of training steps, limiting their applicability to long training runs. Kaizen [2] resolves this by constructing each gradient-descent step as a GKR-style sumcheck argument known as a proof of gradient descent (PoGD). These individual PoGDs are composed together across all iterations via an incrementally verifiable computation (IVC) framework. The IVC framework uses an aggregation scheme for multivariate polynomial commitments to keep recursion overhead at linear prover time and logarithmic verifier overhead, independent of iteration count. Column-oriented multilinear KZG commitments are used to access the training data. When evaluated on VGG-11 (10 million parameters, CIFAR-10, batch size 16), Kaizen produces a $1.63\,\mathrm{MB}$ proof in 15 minutes per iteration, with $130\,\mathrm{ms}$ verification. Both of these measurements are independent of the number of training iterations and the dataset size. The protocol does not certify data provenance: the committed columns establish correctness

of the gradient computation, but not that the underlying records were authorised for training. Both Garg et al. and Kaizen therefore satisfy Definition 3.5.1 while leaving the authorisation question entirely open.

A related but distinct line of work shifts the provenance question from training time to inference time. Given a deployed model, the protocol produces a proof that a specific query response originated from a model whose weights are cryptographically bound to an authorised dataset collection; it does not verify that individual gradient-descent steps were executed correctly. Training is represented through per-layer weight differences that capture the transformation induced by fine-tuning, and HyperNova folding compresses the per-layer verification into a single succinct proof. Datasets are committed via Reckle Tree roots. The verifier confirms that the fine-tuning corpus as a whole derives from an authorised collection, but cannot certify whether a specific record was present or absent from the training data. This dataset-level granularity is sufficient for response-provenance verification but does not support per-sample audit queries. The end-to-end overhead is under $3.3\,\mathrm{s}$ for models up to $8\mathrm{B}$ parameters, with proof generation and verification each completing in under $1.8\ \mathrm{s}$ [32]. Both constructions prove that gradient descent was executed correctly on some committed dataset, but neither asks whether the parties contributing that data were authorised to do so, nor whether multiple data owners could train jointly without exposing their records to the trainer. A separate line of work addresses the latter question through secure multi-party computation.

3.6. Secure Multi-Party Computation for Machine Learning

Secure multi-party computation (MPC) allows multiple parties to jointly evaluate a function over their private inputs without any party learning the others parties' inputs, beyond what the function output reveals. Applied to machine learning, MPC enables collaborative training: several data holders can jointly train a model on the union of their datasets without revealing individual records. Wagh et al. [42] construct three-party protocols for the core operations such as matrix multiplication, convolution, ReLU, and Maxpool in neural network training. Their proposed protocols avoid garbled circuits entirely while achieving $79\times$ lower communication than prior two-server work. Knott et al. [25] make MPC accessible to practitioners through a PyTorch-compatible API backed by a two-server secret-sharing protocol, supporting gradient-descent training under a semi-honest threat model. Zheng et al. [48] add *compute policies* that restrict which parties learn which outputs, together with *cryptographic auditing* that enforces these policies without exposing the private data to the system operator.

MPC-based approaches produce no persistent, publicly verifiable artifact. All parties must be simultaneously online during the joint computation, and the guarantee that the protocol was executed correctly is meaningful only to the participants present at the time. A third-party auditor absent at training time has no mechanism to retroactively verify that gradient-descent steps were performed honestly on authorised data. Revealing raw inputs to enable such verification undoes the privacy guarantee, while withholding them leaves the auditor with no recourse. Bamberger et al. [3] term this the *verification dilemma*. The dilemma applies equally when MPC is augmented with differential privacy. Das et al. [13] combine secret sharing with discrete Gaussian noise to achieve both training-time secrecy and post-training inference resistance, yet the resulting protocol still produces no artifact a third-party auditor can inspect after training has concluded.

3.7. Statistical Privacy Frameworks

Differential privacy (DP) provides a formal, composable guarantee that the output of a computation reveals approximately the same information regardless of whether any single individual's data was included. A randomised mechanism is (ε, δ)-differentially private if any two adjacent datasets produce output distributions that are exponentially close, with the privacy budget tracked through the parameters ε and δ. Abadi et al. [1] instantiate DP for neural network training via DP-SGD: per-sample gradients are clipped to a fixed norm bound to control sensitivity, calibrated Gaussian noise is added to the clipped aggregate before each parameter update, and a moments accountant tracks the cumulative privacy budget (ε, δ) across all training iterations. The resulting guarantee bounds the advantage any adversary gains from observing the trained model about any individual training record.

PAC Privacy [44] generalises DP by shifting from a worst-case criterion to an *instance-based reconstruction hardness* criterion. DP bounds the maximal f-divergence between output distributions for any two adjacent inputs, a measure that is input-independent and therefore can be loose for well-generalising models. PAC Privacy instead quantifies the posterior advantage an adversary gains from observing $\mathcal{M}(X)$ towards recovering X, measured against the prior distribution on X via f-divergence. A mechanism satisfies $(\delta, \rho, \mathrm{D})$-PAC Privacy

if no adversary can produce a reconstruction $\hat{X}$ satisfying the measure function satisfying the reconstruction criterion with probability exceeding $1 - \delta$. Security parameters are generated automatically via Monte-Carlo simulation for any black-box data-processing oracle, requiring no closed-form sensitivity analysis. For tasks where the model already generalises well, the required perturbation need not scale with dimensionality and can be $O(1)$, whereas the worst-case input-independent DP regime imposes a lower bound scaling with the square root of the feature dimension.

Both DP and PAC Privacy answer the question of how much does the trained model reveal about any given training sample. This is logically independent of the question of which records were authorised for inclusion in training. A model trained with DP-SGD on unauthorized data satisfies the indistinguishability guarantee while violating the provenance requirement. Conversely, a model trained using standard gradient descent on a correctly authorised dataset satisfies the provenance requirement without any DP guarantee, so the model could still leak information about individual records. Augmenting MPC with differential privacy, as in Das et al. [13] addresses both training-time secrecy and post-training inference resistance, yet still produces no artifact a third-party auditor can use to verify which records were authorised for training. Statistical privacy frameworks bound leakage but cannot certify provenance.

Table 3.3: Comparison of related work families across key properties.

Approach	**Training-time privacy**	**Post-hoc verifiable**	**Per-sample granularity**	**No trusted setup**
Membership inference [38, 36, 46, 33]	–	–	Statistical only	–
Dataset watermarks [40, 37, 34]	–	×	×	–
MPC-based training [42, 25, 48]	✓	×	×	✓
Differential privacy / PAC Privacy [1, 44, 13]	✓	Partial	×	✓
ZKPoT [18, 2, 32]	✓	✓	×	×

Table 3.3 summarises the findings across all six literature families against the four properties a sound provenance protocol must jointly satisfy. Membership inference and statistical privacy frameworks produce probabilistic signals rather than verifiable certificates. Watermarking and MPC require either anticipatory intervention or simultaneous online presence from all parties, leaving no artifact a third-party auditor can inspect after training concludes. Zero-knowledge proofs of training come closest, certifying that gradient descent was executed on a committed dataset, but leave the authorisation question entirely open. The thesis bridges this remaining gap by coupling ZK set membership with a zkPoT construction. The ZK set provides per-sample inclusion and exclusion certificates for each authorised record, and the zkPoT proof certifies that gradient descent was executed on those records and no others.

4

Protocol for Data Provenance

The protocol must simultaneously answer the two audit questions posed in Chapter 1 without revealing any raw features or labels to the auditor. Concretely, given only the bank's public commitment PK_D and the trainer's proof π, the auditor must verify:

A **Training correctness**: The weight vector $w^{(T)}$ is the output of T honest gradient-descent steps initialised at $w^{(0)}$, executed on the dataset encoded in the DataCommitment.

B **Data intersection**: For each sample encoded in the DataCommitment, the auditor determines whether it belongs to the bank's private dataset D, thereby characterising $D \cap D'$. A non-empty intersection indicates that the trainer has included records from D without authorisation.

These two goals impose different cryptographic requirements on how the training data is committed. Step A reduces to an evaluating an arithmetic circuit over the features: the sumcheck protocol accesses the feature matrix column-by-column, opening the multilinear extension $\tilde{X}_j$ of each feature column j at a Fiat-Shamir challenge point via a multilinear KZG (MKZG) commtiment. Step B reduces to proving set membership: the ZK Sets scheme requires a compact, per-sample key derived from a commitment to the sample as a whole. No single polynomial commitment serves both roles efficiently, so the DataCommitment exposes two objects over the same feature matrix X:

- column_commits[j]: One MKZG commitment per feature column, serving as the sumcheck oracle for Step A.
- sample_poly_commits[i]: One univariate KZG commitment per training sample, from which the ZK Sets lookup key is derived for Step B.

The two objects are computed from the same matrix but are not cryptographically linked. For this we introduce the Consistent DataCommitment (CDC) assumption. This assumption states that a prover publishes the column_commits and sample_poly_commits from the same training dataset. Section 5.1 characterizes this assumption precisely. The CDC gap would only affect soundness. A cross-commitment opening proof that closes CDC is identified there as a direction for future work [7].

4.1. Protocol Overview

We call this combined construction zkMLProv (**zk**ero-knowledge **M**achine **L**earning **Prov**enance): a non-interactive, publicly verifiable three-party protocol composed of Algorithms 2–5, in which all raw training data remains hidden from the auditor. The zkMLProv flow is shown in Figure 4.1. The Bank first computes PK_D using ZK Sets according to Algorithm 2 and publishes PK_D. Figure 4.1 shows the zkMLProv flow; Algorithms 2–5 give the pseudocode for each phase. Sections 4.4 to 4.7 expand each phase with full cryptographic detail. Section 4.2 describes the subprotocols that appear as subroutines within these algorithms.

Before training begins, the bank runs Algorithm 2 to build a Pedersen-hash trie over its authorised records and publish the resulting public key PK_D to the public ledger. The trainer then runs Algorithm 3 to compute the DataCommitment and publish it to the same ledger, before a single gradient step is taken. DataCommitment

binds the feature matrix to two KZG objects: one column-wise for the sumcheck oracle and one row-wise for the ZK Sets keys. This pre-training publication rules out adaptive data selection, because the commitment is fixed before any challenge is drawn. After T gradient steps, Algorithm 4 produces proof π, which carries sumcheck transcripts and polynomial commitment openings (Section 2.10) but no raw data. Algorithm 5 then takes only π, the DataCommitment, and PK_D as inputs. Step A checks each PoGD transcript against the published column commitments. Step B checks, for each training sample's commitment key, whether it is present in the bank's trie; a membership result indicates the sample belongs to the bank's private dataset D. Together, Step A and Step B allow the auditor to certify training correctness and characterise $D \cap D'$ without ever seeing the underlying features or labels.

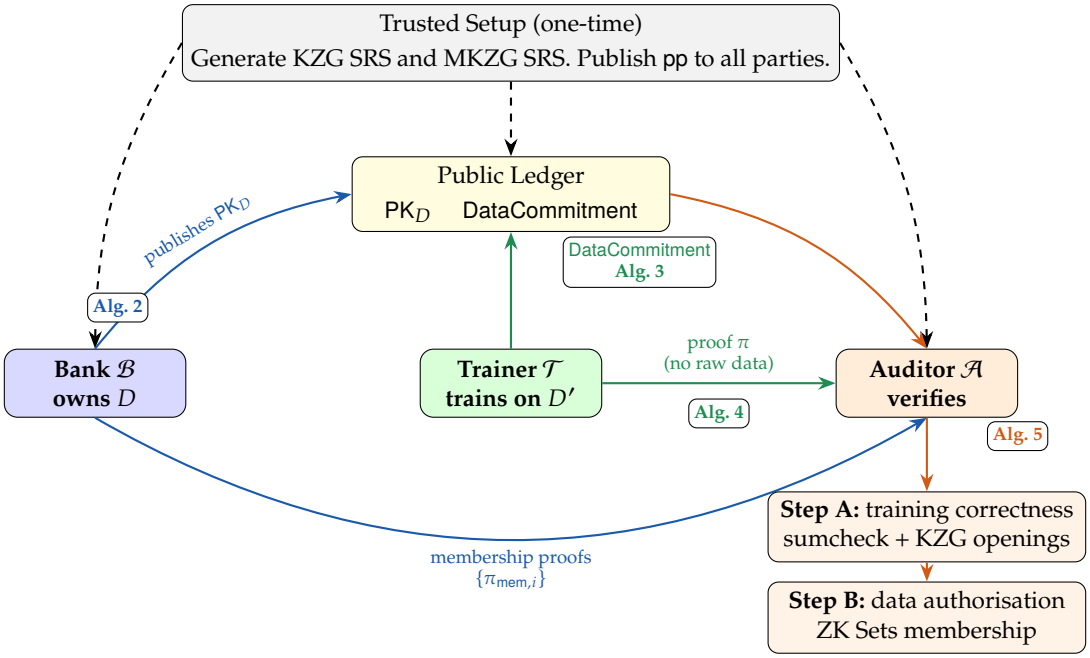


Figure 4.1: zkMLProv flow with algorithm cross-references. The trusted setup generates the KZG and MKZG structured reference strings and distributes them to all parties before zkMLProv begins. The bank publishes PK_D before training (Algorithm 2); the trainer publishes a DataCommitment (Algorithm 3), then after T gradient-descent steps sends proof π without any raw data (Algorithm 4). The bank reads the public DataCommitment, derives the ZK Sets key for each training sample, generates membership or non-membership proofs $\{\pi_{\mathsf{mem},i}\}$, and sends them to the auditor for Step B verification. The auditor checks Step A (training correctness via sumchecks and KZG openings) and Step B (data authorisation via ZK Sets membership) from π and the two public commitments alone (Algorithm 5).

4.2. Important subprotocols

At the core of zkMLProv we make use of five sub-protocols. The ZkProductSumcheck and ZkTripleProduct-Sumcheck protocols build on the sumcheck protocol and masking techniques described in Sections 2.6 and 2.13. The remaining three are protocol-specific constructions. We provide a brief description regarding their usage and purpose below:

1. **CommitmentToZkSetsKey**: Used in Algorithm 2 (Bank Setup) and Algorithm 5 (Auditor Verify, Step B) to derive the 48-byte ZK Sets lookup key from a KZG commitment.
2. **CommitSampleFP**: Used in Algorithm 2 (Bank Setup) and Algorithm 3 (Trainer Setup) to produce a per-sample univariate KZG commitment.
3. **ProvePower**: Used in Algorithm 4 (Trainer Prove), logistic regression branch only, to certify that pow2 and pow3 are the correct pointwise square and cube of score.
4. **ZkProductSumcheck**: Comprises a ZkProductSumcheck.Prove call in the forward pass of Algorithm 4 and a ZkProductSumcheck.Verify call in Algorithm 5 (forward verification, line 7). Used to prove sums of the form

$$H = \sum_{i=0}^{N-1} f_1(i) \cdot f_2(i)$$

in zero-knowledge.

5. **ZkTripleProductSumcheck**: Comprises a ZkTripleProductSumcheck.Prove call in the backward pass of Algorithm 4 (once per feature), also called internally by ProvePower (Algorithm 11), and a ZkTripleProductSumcheck.Verify call in Algorithm 5 (backward verification). Used to prove sums of the form
$$H = \sum_{i=0}^{N-1} f_1(i) \cdot f_2(i) \cdot f_3(i)$$
in zero-knowledge.

4.3. Trusted Setup

Before any party can run the protocol, a one-time setup must be performed to generate the public parameters pp shared by all three parties. This setup produces two structured reference strings: a univariate KZG SRS for per-sample commitments (Section 2.10) and a multilinear KZG SRS for the column commitments used as the sumcheck oracle (Section 2.11). Both SRS objects must be generated before the Bank or Trainer compute any commitment, since all commitments are bound to the same public parameters. In a production deployment, the SRS would be produced by a multi-party computation ceremony so that no single participant learns the trapdoor value τ [8]. The security implications of the current implementation, which generates both SRS objects deterministically from fixed seed strings, are discussed in Chapter 7.

4.4. Algorithm 2: Bank Setup

Algorithm 2 describes how the bank builds and publishes PK_D before training begins.

Algorithm 2 Bank Setup: build and publish PK_D

Require: Approved dataset $D = \{(\mathbf{x}_s, y_s)\}$, univariate KZG parameters kzg_params, Pedersen parameters ped_params
Ensure: PK_D = (pk_c, pk_h) published on public ledger ▷ commitment root and hash root of the Pedersen-Merkle trie
1: edb ← ∅
2: **for** each approved sample $(\mathbf{x}_s, y_s) \in D$ **do**
3: $C_s \leftarrow$ CommitSampleFP($\mathbf{x}_s$, y_s, kzg_params) ▷ KZG commitment to the degree-d polynomial
4: $\mathsf{key}_s \leftarrow$ CommitmentToZkSetsKey(C_s) ▷ Compressed $\mathbb{G}_1$ point, 48 bytes;
5: edb[key_s] ← `"authorized"`
6: **end for**
7: (pk_c, pk_h) ← ZKSets.Commit(edb, ped_params)
8: **Publish** PK_D on public ledger

Before the auditor $\mathcal{A}$ considers the proof of training sent by the trainer $\mathcal{T}$, the Bank commits to its private dataset using zero-knowledge sets and publishes the resulting key PK_D to the public ledger. For each approved sample $(\mathbf{x}_s, y_s) \in D$ (Algorithm 2, line 2), it computes a univariate KZG commitment using CommitSampleFP over the feature-and-label values to produce $C_s = [f_s(\tau)]_1$ (Algorithm 2, line 3). Here, f_s is the unique degree-d polynomial whose value at integer j is $[\mathbf{x}_s; y_s]_j$. This commitment is deterministic as it only depends on the feature values, the label and the KZG parameters. This allows the trainer to produce the same C_s for the same record during Algorithm 3 without any further communication between the two parties. Absence of determinism would lead to each party (the Bank $\mathcal{B}$ and trainer $\mathcal{T}$) would use a different masking polynomial, yielding a different C_s value. Different C_s values would lead to different 48-byte keys. Thus the trainer's key would not appear in PK_D and Step B would fail for a sample used by the trainer that indeed belongs to the Bank's dataset.

The key inserted into the EDB (Algorithm 2, line 4) is the compressed $\mathbb{G}_1$ encoding of C_s, a 48-byte string produced by CommitmentToZkSetsKey. It is infeasible to key into the trie by hashing the raw feature bytes. This is because it would require the auditor to prove in zero knowledge that those secret keys hash to the claimed key, adding an expensive circuit. We thus make use of C_s directly because it already appears as sample_poly_commits[s] in the DataCommitment published by the trainer (Algorithm 3, line 6). The auditor can therefore recompute the key from that public value with no additional circuit (see Algorithm 5, line 31).

Finally, the EDB is committed via the unmodified Micali-Rabin-Kilian ZK Sets construction [29] (Algorithm 2, line 6), which builds a Pedersen-hash trie over the key space. A plain Merkle tree would support membership proofs but would reveal the leaf value to the verifier, disclosing which records the bank has authorised. The MRK construction avoids this because the the verifier learns only whether a specific key is present or absent, nothing about any other entry in the trie. Non-membership certificates additionally allow the bank to certify that a withdraw record is absent from the authorised set under GDPR Article 17 [15], without disclosing what remains.

4.5. Algorithm 3: Trainer Setup

Algorithm 3 describes how the trainer builds the two-part DataCommitment. Both halves must be fixed and published before the first gradient step (line 6), so the trainer cannot adaptively choose the committed dataset after observing the model's behaviour.

Algorithm 3 Trainer Setup: build and publish DataCommitment

Require: Training dataset $D' = \{(\mathbf{x}_i, y_i)\}_{i=0}^{N-1}$, parameters mkzg_params, kzg_params
Ensure: DataCommitment = (column_commits, sample_poly_commits) published
1: **for** $j = 0$ **to** $d - 1$ **do** ▷ One multilinear KZG commit per feature column
2: column_commits$[j] \leftarrow$ mkzg_commit(mkzg_params, $[x_{0,j}, x_{1,j}, \ldots, x_{N-1,j}]$)
3: **end for**
4: column_commits$[d] \leftarrow$ mkzg_commit(mkzg_params, $[y_0, y_1, \ldots, y_{N-1}]$) ▷ Label column
5: **for** $i = 0$ **to** $N - 1$ **do** ▷ One univariate KZG commit per sample row
6: sample_poly_commits$[i] \leftarrow$ CommitSampleFP($\mathbf{x}_i$, y_i, kzg_params)
7: **end for**
8: **Publish** DataCommitment on public ledger ▷ Fixed before training; trainer cannot adaptively change it

The first loop (lines 1-2) commits each feature column j under the multilinear KZG (PST13, [2]). Here, column_commits$[j]$ is a single $\mathbb{G}_1$ point that binds all N entries of column j and acts as the sumcheck oracle for Step A across all T gradient steps. Multilinear KZG is necessary because at the end of each sumcheck round the verifier accumulates a challenge point $\mathbf{r} \in \mathbb{F}^{\log N}$ and queries $\tilde{X}_j(\mathbf{r})$. A univariate KZG commitment opens at a single field element and cannot answer a multi-variate challenge, so it cannot serve as a sumcheck oracle.

The label column is committed under the same scheme (line 3) because the backward pass must verify that the error vector is consistent with both the committed predictions and the committed labels. For linear regression the verifier checks

$$\tilde{\mathrm{err}}(r_c) = \sum_{j=0}^{d-1} w_j \cdot \tilde{X}_j(r_c) - \tilde{y}(r_c) \tag{4.1}$$

at a shared challenge point r_c. Each term on the right-hand side is an MKZG opening of an already-published entry in column_commits$[0 \ldots d]$, so no additional oracle is needed.

The second loop (lines 5-7) uses univariate KZG rather than MKZG because Step B requires a per-sample commitment, not a per-column one. The column-wise MKZG commits to N values simultaneously and cannot be opened to certify a single row in isolation. CommitSampleFP instead commits to the degree-d Lagrange polynomial whose value at integer j equals $x_{i,j}$ for $j < d$ and y_i at $j = d$, producing the same $\mathbb{G}_1$ point that Algorithm 2 computed for each record in the bank's private dataset. The bank and trainer therefore arrive at matching ZK Sets keys independently, without any additional exchange. Univariate KZG is preferred over a hash function because it shares the algebraic structure of the column commitments, which any future cross-commitment consistency proof would require to close the CDC gap.

Finally, the two loops write to the same matrix in orthogonal directions. column_commits encodes it column-by-column under MKZG, while sample_poly_commits encodes it row-by-row under the univariate KZG. Nothing in the current protocol cryptographically enforces that both bind the same underlying matrix. This is the Consistent DataCommitment introduced in Section 4.1, and its effect on soundness is addressed in Chapter 5.1.

4.6. Algorithm 4: Trainer Prove

A single gradient step on batch $B_t \subseteq \{0, \ldots, N-1\}$ computes

$$w_j^{(t)} = w_j^{(t-1)} - \frac{\alpha}{|B_t|}(\text{pred}(i) - y_i) \cdot x_{i,j}, \tag{4.2}$$

where $\text{pred}(i) = \sum_{j'=0}^{d-1} w_{j'}^{(t-1)} x_{i,j'}$ for linear regression. Introducing the selection indicator $\text{sel}(i) = \mathbb{1}[i \in B_t]$ lifts both sums from the batch to the full dataset (trainer's dataset D') of size N.

The batch prediction sum becomes

$$S_t = \sum_{i=0}^{N-1} \text{sel}(i) \cdot \text{pred}(i), \tag{4.3}$$

a degree-2 product over N terms. The j-th gradient component becomes

$$g_j = \sum_{i=0}^{N-1} \text{sel}(i) \cdot \text{err}(i) \cdot x_{i,j}, \tag{4.4}$$

where $\text{err}(i) = \text{pred}(i) - y_i$, making the summand degree-3. The sumcheck protocol handles both, reducing each claimed sum to a single multilinear evaluation query at a Fiat-Shamir challenge, answered by MKZG openings of the column commitments in DataCommitment. For logistic regression $\text{err}(i) = P_3(\text{pred}(i)) - y_i$, where P_3 is the degree-3 sigmoid approximation, and the structure of Equation (4.4) is unchanged.

The forward pass proves S_t against the committed dataset without revealing which indices form batch B_t. Running the sumcheck directly would expose batch membership, because the round polynomials are determined by both sel and pred, giving the verifier enough information to reconstruct B_t from the transcript. The ZK product sumcheck hides the batch selector by adding a zero-sum mask. The trainer samples R with $\sum_{i=0}^{N-1} R(i) = 0$ and commits it before drawing any Fiat-Shamir challenge. The zero-sum constraint leaves the claimed total S_t unchanged, so the verifier still accepts the correct value. The round messages, however, are indistinguishable from a sumcheck over a uniformly random polynomial, so sel and pred leak nothing to the transcript.

No separate commitment to pred is needed. By MLE linearity, $\widetilde{\text{pred}}(r) = \sum_{j=0}^{d-1} w_j \tilde{X}_j(r)$, and the same holds at the commitment level by $\mathbb{G}_1$-linearity of MKZG. Define the masked forward function $f_t(i) = \text{sel}(i) \cdot \text{pred}(i) + R(i)$; this is the function whose sum the forward sumcheck reduces to a single oracle query. At the final challenge point $r \in \mathbb{F}^{\log N}$, the prover opens sel_commit, mask_commit, and column_commits$[j]$ to let the verifier confirm that the MLE of f_t at r equals

$$\tilde{f}_t(r) = \widetilde{\text{sel}}(r) \cdot \widetilde{\text{pred}}(r) + \tilde{R}(r). \tag{4.5}$$

The verifier derives $\widetilde{\text{pred}}(r)$ from the public weights and the already-opened column commitments. The forward pass therefore certifies S_t while hiding the batch selector.

Certifying the gradient requires a separate sumcheck for each of the d feature columns. Before running any of the d backward sumchecks, the trainer commits the error vector as err_commit. A fresh zero-sum mask R_j is drawn independently for each feature j. Mask independence across features is a correctness requirement. Reusing the same R for all d sumchecks would let the verifier subtract consecutive oracle values and recover information about $\widetilde{\text{err}}$. For each j, the ZK triple product sumcheck proves Equation (4.4) and opens sel_commit, err_commit, column_commits$[j]$, and mask_commit$_j$ at challenge point r'_j.

It remains to verify that err_commit encodes $\text{pred}(i) - y_i$ rather than an arbitrary vector. At a shared Fiat-Shamir challenge r_c, the verifier checks

$$\widetilde{\text{err}}(r_c) = \sum_{j=0}^{d-1} w_j \cdot \tilde{X}_j(r_c) - \tilde{y}(r_c). \tag{4.6}$$

Each term on the right is an MKZG opening of an already-published commitment. The identity holds by MLE linearity, so no additional sumcheck is required. The backward pass therefore certifies all d gradient components against the committed column and error vectors.

Algorithm 4 Trainer Prove: produce π_t for one gradient-descent iteration

Require: column_commits, dataset D', current weights $\mathbf{w}^{(t-1)}$, batch $B_t \subseteq \{0, \ldots, |D'|-1\}$, learning rate α
Ensure: Step proof π_t = (fwd, bwd)

// Forward pass (degree-2 product sumcheck)
for $i = 0$ **to** $|D'| - 1$ **do**
  $\mathsf{sel}(i) \leftarrow \mathbb{1}[i \in B_t]$
end for
sel_commit $\leftarrow$ mkzg_commit(mkzg_params, sel) ▷ Committed before any Fiat-Shamir challenge
for $i = 0$ **to** $|D'| - 1$ **do**
  $\mathsf{pred}(i) \leftarrow \sum_{j=0}^{d-1} w_j^{(t-1)} \cdot x_{i,j}$
end for
$S_t \leftarrow \sum_i \mathsf{sel}(i) \cdot \mathsf{pred}(i)$
$R \leftarrow$ ZeroSumMask($|D'|$); mask_commit $\leftarrow$ mkzg_commit(mkzg_params, R) ▷ Committed before any Fiat-Shamir challenge
fwd $\leftarrow$ ZkProductSumcheck.prove(sel, pred, S_t, R)
$r \leftarrow$ fwd.challenges ▷ Fiat-Shamir challenge point; $r \in \mathbb{F}^v$
for $j = 0$ **to** $d - 1$ **do**
  fwd.feature_opens[j] $\leftarrow$ mkzg_open(column_commits[j], r) ▷ d opens so verifier can compute $\widetilde{\mathsf{pred}}(r)$
end for
fwd.sel_open $\leftarrow$ mkzg_open(sel_commit, r)
fwd.mask_open $\leftarrow$ mkzg_open(mask_commit, r)
// Backward pass (degree-3 triple-product sumcheck)
for $i = 0$ **to** $|D'| - 1$ **do**
  $\mathsf{err}(i) \leftarrow \mathsf{pred}(i) - y_i$ ▷ Linear regression; for logistic: $P_3(\mathsf{pred}(i)) - y_i$, where $P_3(x) = \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3$
end for
err_commit $\leftarrow$ mkzg_commit(mkzg_params, err)
for $j = 0$ **to** $d - 1$ **do**
  $g_j \leftarrow \sum_i \mathsf{sel}(i) \cdot \mathsf{err}(i) \cdot x_{i,j}$
  $R_j \leftarrow$ ZeroSumMask($|D'|$)
  mask_commit$_j$ $\leftarrow$ mkzg_commit(mkzg_params, R_j) ▷ Fresh mask per feature; committed before any FS challenge
  bwd[j] $\leftarrow$ ZkTripleProductSumcheck.prove(sel, err, $\mathbf{f}_j$, g_j, R_j)
  $r'_j \leftarrow$ bwd[j].challenges ▷ Per-feature Fiat-Shamir challenge; $r' \in \mathbb{F}^v$
  bwd[j].col_open $\leftarrow$ mkzg_open(column_commits[j], r'_j)
  bwd[j].err_open $\leftarrow$ mkzg_open(err_commit, r'_j)
  bwd[j].sel_open $\leftarrow$ mkzg_open(sel_commit, r'_j)
  bwd[j].mask_open $\leftarrow$ mkzg_open(mask_commit$_j$, r'_j)
end for
$g_{\text{bias}} \leftarrow \sum_i \mathsf{sel}(i) \cdot \mathsf{err}(i)$ ▷ Bias gradient; feature column is constant $\mathbf{1}$, no oracle check needed
$R_{\text{bias}} \leftarrow$ ZeroSumMask($|D'|$); bias_mask_commit $\leftarrow$ mkzg_commit(mkzg_params, R_{bias})
bwd.bias $\leftarrow$ ZkProductSumcheck.prove(sel, err, g_{bias}, R_{bias}) ▷ Degree-2, not triple product
$r_{\text{bias}} \leftarrow$ bwd.bias.challenges
bwd.bias_sel_open $\leftarrow$ mkzg_open(sel_commit, r_{bias}); bwd.bias_err_open $\leftarrow$ mkzg_open(err_commit, r_{bias})
$r_c \leftarrow$ FiatShamir(fwd.sel_commit, fwd.mask_commit, bwd.err_commit, column_commits) ▷ Shared challenge derived after all sumcheck transcripts are fixed
bwd.err_open_rc $\leftarrow$ mkzg_open(err_commit, r_c) ▷ $d + 2$ opens; verifier checks $\widetilde{\mathsf{err}}(r_c) = \sum_j w_j \tilde{X}_j(r_c) - \tilde{y}(r_c)$
bwd.label_open $\leftarrow$ mkzg_open(column_commits[d], r_c)
for $j = 0$ **to** $d - 1$ **do**
  bwd.feat_opens_rc[j] $\leftarrow$ mkzg_open(column_commits[j], r_c)
end for
$\mathbf{w}^{(t)} \leftarrow \mathbf{w}^{(t-1)} - \frac{\alpha}{|B_t|}\mathbf{g}$ ▷ Local state update; not included in π_t
$\pi_t \leftarrow$ (fwd, bwd)
return π_t

Logistic regression requires four additional consistency checks because the error vector depends on the per-sample prediction through the degree-3 approximation P_3. The sigmoid is not polynomial, so the committed column vectors alone cannot determine err. Before drawing any Fiat-Shamir challenge, the trainer publishes MKZG commitments to the score, squared-score, and cubic-score vectors.

Checks A and D are verified by MKZG openings alone. Check A confirms at r_{sc} that

$$\widetilde{\text{score}}(r_{\text{sc}}) = \sum_{j=0}^{d-1} w_j \cdot \tilde{X}_j(r_{\text{sc}}), \tag{4.7}$$

establishing that the separately committed score vector is consistent with the column commitments. Check D then verifies at r_e that

$$\widetilde{\text{err}}(r_e) = C_0 + C_1 \cdot \widetilde{\text{score}}(r_e) \cdot \Lambda + C_3 \cdot \widetilde{\text{pow3}}(r_e) \cdot \Lambda^3 - \tilde{y}(r_e), \tag{4.8}$$

where $C_0 = \frac{1}{2}$, $C_1 = \frac{1}{4}$, $C_3 = -\frac{1}{48}$, and $\Lambda = (2^{16})^{-1} \bmod p$ is the fixed-point scaling factor. Every term on the right is an MKZG opening of a published commitment.

Checks B and C each require a ZK triple product sumcheck via ProvePower. Check B proves at r_2 that

$$\sum_{i=0}^{N-1} \text{score}(i)^2 \cdot \chi_i(r_2) = \widetilde{\text{pow2}}(r_2), \tag{4.9}$$

where $\chi_i(r)$ is the i-th multilinear Lagrange basis polynomial (Section 2.4), so the left-hand side equals $\widetilde{\text{score}^2}(r_2)$. Check C proves at r_3 that

$$\sum_{i=0}^{N-1} \text{pow2}(i) \cdot \text{score}(i) \cdot \chi_i(r_3) = \widetilde{\text{pow3}}(r_3). \tag{4.10}$$

No individual score value reaches the verifier. Once all four checks pass, the per-feature backward sumchecks proceed identically to the linear regression case.

4.7. Algorithm 5: Auditor Verify

Algorithm 5 has two phases with different output semantics. Step A produces a binary outcome: any failing cryptographic check causes immediate **Reject**, meaning the training proof is invalid and no further certification is possible. Step B always produces **Accept**($I_\cap$): the auditor returns the certified intersection regardless of its size. $I_\cap = \emptyset$ is a valid clean-audit result, certifying that the trainer used none of the bank's private data. $I_\cap \neq \emptyset$ flags unauthorised data use, identifying exactly which training samples were drawn from the bank's dataset D. Step A replays forward and backward verification for each of the T gradient steps, and Step B certifies the data intersection using ZK Sets. The forward check reconstructs the batch prediction oracle from public information alone. Both mask_commit and sel_commit were committed before any Fiat-Shamir challenge was drawn, so the verifier opens them at the challenge point r that the sumcheck transcript determines. No separate commitment to pred is opened. Because $\text{pred}(i) = \sum_j w_j x_{i,j}$ and the multilinear extension is linear in its input, the verifier reconstructs

$$\widetilde{\text{pred}}(r) = \sum_{j=0}^{d-1} w_j^{(t-1)} \cdot \tilde{X}_j(r) \tag{4.11}$$

from the d column openings and the public weights. The forward sumcheck passes if and only if the transcript is consistent with the oracle value $\widetilde{\text{sel}}(r) \cdot \widetilde{\text{pred}}(r) + \tilde{R}(r)$.

Published before training and fixed thereafter, column_commits is load-bearing for the whole protocol. A trainer who could substitute different column commitments at verification time could produce a valid sumcheck transcript for a gradient computed on different data. The commitment scheme's binding property under the q-SDH assumption (Section 2.9) prevents this.

For each feature j, the verifier opens mask_commit_j, sel_commit, err_commit, and column_commits[j] at the triple-product sumcheck challenge r'_j, then checks

$$\pi_t.\text{bwd}[j].\text{oracle} = \widetilde{\text{sel}}(r'_j) \cdot \widetilde{\text{err}}(r'_j) \cdot \tilde{X}_j(r'_j) + \tilde{R}_j(r'_j). \tag{4.12}$$

Algorithm 5 Auditor Verify: Step A (zkPoT) then Step B (ZK Sets membership)

Require: Proof π = (DataCommitment, $\{\pi_t\}_{t=1}^{T}$), bank public key PK_D, bank membership proofs $\{\pi_{\mathsf{mem},i}\}_{i=0}^{|D'|-1}$, initial weights $\mathbf{w}^{(0)}$, learning rate α, batch size b
Ensure: **Accept** or **Reject**

// Step A: Verify proof of training
for $t = 1$ **to** T **do**
  // Forward verification (mirrors forward pass of Algorithm 4)
  $r \leftarrow \pi_t$.fwd.challenges ▷ Re-derived from sumcheck transcript
  mkzg_verify(π_t.fwd.mask_commit, π_t.fwd.mask_open) $\rightarrow \tilde{R}(r)$
  mkzg_verify(π_t.fwd.sel_commit, π_t.fwd.sel_open) $\rightarrow \widetilde{\mathsf{sel}}(r)$
  for $j = 0$ **to** $d-1$ **do**
    mkzg_verify(column_commits[j], π_t.fwd.feature_opens[j]) $\rightarrow \tilde{X}_j(r)$
  end for
  $\widetilde{\mathsf{pred}}(r) \leftarrow \sum_{j=0}^{d-1} w_j^{(t-1)} \cdot \tilde{X}_j(r)$ ▷ KZG linearity; no separate pred commitment needed
  Check π_t.fwd.T_oracle = $\widetilde{\mathsf{sel}}(r) \cdot \widetilde{\mathsf{pred}}(r) + \tilde{R}(r)$
  ProductSumcheck.verify(π_t.fwd, π_t.fwd.claimed_S, π_t.fwd.T_oracle)
  // Backward verification (mirrors backward pass of Algorithm 4)
  for $j = 0$ **to** $d-1$ **do**
    $r'_j \leftarrow \pi_t$.bwd[j].challenges ▷ Per-feature challenge from triple sumcheck transcript
    mkzg_verify(π_t.bwd[j].mask_commit, π_t.bwd[j].mask_open) $\rightarrow \tilde{R}_j(r'_j)$
    mkzg_verify(π_t.bwd[j].sel_commit, π_t.bwd[j].sel_open) $\rightarrow \widetilde{\mathsf{sel}}(r'_j)$ ▷ sel_commit lives in fwd
    mkzg_verify(π_t.bwd.err_commit, π_t.bwd[j].err_open) $\rightarrow \widetilde{\mathsf{err}}(r'_j)$
    mkzg_verify(column_commits[j], π_t.bwd[j].col_open) $\rightarrow \tilde{X}_j(r'_j)$
    Check π_t.bwd[j].T_oracle = $\widetilde{\mathsf{sel}}(r'_j) \cdot \widetilde{\mathsf{err}}(r'_j) \cdot \tilde{X}_j(r'_j) + \tilde{R}_j(r'_j)$
    TripleProductSumcheck.verify(π_t.bwd[j], π_t.bwd[j].claimed_sum, π_t.bwd[j].T_oracle) ▷ Linear; for logistic: also verify Checks A–D (see prose)
  end for
  // Err-consistency at shared challenge r_c (linear regression)
  $r_c \leftarrow$ FiatShamir(π_t.fwd.sel_commit, π_t.fwd.mask_commit, π_t.bwd.err_commit, column_commits) ▷ Shared challenge derived after all sumcheck transcripts are fixed
  mkzg_verify(π_t.bwd.err_commit, π_t.bwd.err_open_rc) $\rightarrow \widetilde{\mathsf{err}}(r_c)$
  mkzg_verify(column_commits[d], π_t.bwd.label_open) $\rightarrow \tilde{y}(r_c)$
  for $j = 0$ **to** $d-1$ **do**
    mkzg_verify(column_commits[j], π_t.bwd.feat_opens_rc[j]) $\rightarrow \tilde{X}_j(r_c)$
  end for
  Check $\widetilde{\mathsf{err}}(r_c) = \sum_{j=0}^{d-1} w_j^{(t-1)} \cdot \tilde{X}_j(r_c) - \tilde{y}(r_c)$
  Check $\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \frac{\alpha}{b}\,\mathbf{g}_t$ ▷ b is a public training parameter; not read from π_t
  if any check fails **then return Reject**
  end if
end for
// Step B: Certify authorised training samples (no raw data seen)
$I \leftarrow \{0, \ldots, |D'|-1\}$ ▷ All samples in DataCommitment; batch indices B_t are not disclosed in π
$I_\cap \leftarrow \emptyset$
for each $i \in I$ **do**
  $\mathsf{key}_i \leftarrow$ CommitmentToZkSetsKey(DataCommitment.sample_poly_commits[i]) ▷ Public input; no circuit needed
  if ZKSets.Verify(key_i, $\pi_{\mathsf{mem},i}$, PK_D) = true **then** $I_\cap \leftarrow I_\cap \cup \{i\}$
  end if
end for
return Accept($I_\cap$) ▷ $I_\cap = \emptyset$: clean audit (no bank samples used); $I_\cap \neq \emptyset$: unauthorised use flagged

Consistency of the oracle with the transcript certifies that gradient component g_j was computed honestly over the committed data. What the sumcheck cannot verify on its own is the content of err_commit. At a shared challenge r_c, the verifier opens err_commit, column_commits$[d]$, and all d feature columns, then checks

$$\widetilde{\text{err}}(r_c) = \sum_{j=0}^{d-1} w_j^{(t-1)} \cdot \tilde{X}_j(r_c) - \tilde{y}(r_c). \tag{4.13}$$

By MLE linearity, this equality holds if and only if err encodes the honest residual $\text{pred}(i) - y_i$ at every index. The weight update $\mathbf{w}^{(t)} = \mathbf{w}^{(t-1)} - \frac{\alpha}{b}\mathbf{g}_t$ is checked last, using the publicly declared batch size b. Any failing check causes immediate rejection.

For logistic regression, Checks A–D run before the per-feature backward sumchecks. Check A verifies score-column consistency at r_{sc}. Checks B and C call $\mathrm{VerifyPower}$ to confirm pow2_commit and pow3_commit encode the correct pointwise squares and cubes. Check D replaces Equation (4.13) with the field-arithmetic version of P_3, verifying

$$\widetilde{\text{err}}(r_e) + \tilde{y}(r_e) = C_0 + C_1 \cdot \widetilde{\text{score}}(r_e) \cdot \Lambda + C_3 \cdot \widetilde{\text{pow3}}(r_e) \cdot \Lambda^3, \tag{4.14}$$

where all openings at r_e are from already-published commitments. Once the four checks pass, the per-feature backward sumchecks proceed identically to the linear regression case.

Step B certifies which training samples are drawn from the bank's private dataset. No additional zero-knowledge circuit is needed because sample_poly_commits$[i]$ is already a public element of DataCommitment, published before training began. The ZK Sets key for sample i is derived deterministically from those public bytes by $\mathrm{CommitmentToZkSetsKey}$. Since DataCommitment is public, the bank independently derives the same keys and generates membership or non-membership proofs for all $|D'|$ samples using the unmodified MRK prover [29]; the auditor receives these proofs and calls only $\mathrm{ZKSets.Verify}$.

Step B operates over the full index set $I = \{0, \ldots, |D'| - 1\}$. The bank generates a membership or non-membership proof for each key; the auditor verifies each against PK_D. Those that pass accumulate in $I_\cap$. The auditor always returns **Accept**$(I_\cap)$: an empty $I_\cap$ certifies a clean audit (the trainer provably used no bank data); a non-empty $I_\cap$ is the certified intersection $D \cap D'$, flagging unauthorised use.

5 Analysis

In this chapter we perform a theoretical analysis regarding the security and complexity of zkMLProv (Chapter 4). Prior to proving the completeness, soundness, and zero-knowledge property of zkMLProv, we state our assumptions and consider their viability. To prove completeness and soundness we look at how the completeness and soundness characteristics of the individual zero-knowledge proofs behave when considered together. To prove the zero-knowledge property, we consider the simulator-based definition (See section 2.0.2) and show how a simulator for zkMLProv can be constructed from each of the individual simulators.

5.1. Security Analysis

In order to understand the benfits of the security guarantees, we need to understand the security model and the associated assumptions. We first understand the security assumptions to then understand the capabilities of the three participants, namely, the Bank ($\mathcal{B}$), the Model Trainer ($\mathcal{T}$) and the Auditor ($\mathcal{A}$).

5.1.1. Assumptions

The functionality of the protocol described in Chapter 4 is based on two assumptions. Firstly, we assume that a trusted ceremony is feasible which gives the SRS string responsible for the generation of kzg_params and mkzg_params. This corresponds to Phase 0 of the protocol. Once these parameters are generated, they are shared with all the participants of the protocol. This is a valid assumption to have as such trusted ceremonies exist in literature [8].

Secondly, we consider the Consistent DataCommitment (CDC) assumption. We formally state this as

Assumption 5.1.1. *CDC Assumption While constructing* DataCommitment*, the model trainer ($\mathcal{T}$) constructs* sample_poly_commits *and* column_commits *from the same dataset D' using Algorithm 3.*

Assumption 5.1.1 is a strong assumption. However, it can be readily solved by making use of cross-commitment SNARKs. The development of such a SNARK is beyond the scope of this protocol [7].

5.1.2. Security Model

We consider a malicious model for the trainer ($\mathcal{T}$) to publish consistent sample_poly_commits and column_commits associated with the training dataset D', the security guarantees hold in the malicious model.

We now analyze each of the three properties of completeness, soundness, and zero-knowledge independently, and state the impact of Assumption 5.1.1 on their validity.

5.1.3. Completeness

Completeness says that a valid proof is more likely to be accepted by the verifier (See Section 2.0.1). In our setup, this implies that an honest trainer should always convince the auditor. In Algorithm 4.6, we make use of zero knowledge proofs to show the correctness of various steps. Each of the involved sub-protocols such as the sumcheck variants, MKZG evaluations, and ZK Sets satisfy perfect completeness, so the composed protocol also satisfies completeness. We show this formally below

Lemma 5.1.2 (One-step proof is complete)**.** *If the trainer runs Algorithm 4 honestly for step t, the auditor accepts π_t with probability* 1*.*

Proof. Each component of π_t verifies by direct inspection:

- **Forward sumcheck**: An honest prover always produces an accepting sumcheck transcript [39]. The MKZG openings of sel_commit, mask_commit, and column_commits$[j]$ at r are correct by KZG correctness [2].
- **Backward sumchecks**: Each ZkTripleProductSumcheck.prove is an extension of ProductSumcheck and is also complete [39]. The four MKZG openings per feature at r'_j verify correctly.
- **Bias sumcheck**: ZkProductSumcheck.prove(sel, err, g_{bias}, R_{bias}) is complete [39] and its two openings verify correctly.
- **Error-consistency**: By construction $\varepsilon(i) = \text{pred}(i) - y_i$ for all i (Algorithm 4, line 20). By MLE linearity, $\widetilde{\varepsilon}(r_c) = \sum_j w_j \tilde{X}_j(r_c) - \tilde{y}(r_c)$, so Equation (4.6) holds. All $d+2$ MKZG openings at r_c verify by KZG correctness.

This implies that an honest prover who generates a transcript π_t will pass all of the checks. Thus, proving the completeness. □

Theorem 5.1.3 (zkMLProv is perfectly complete)**.** *If the trainer runs Algorithms 3–4 honestly on dataset D' for T iterations of gradient descent, the auditor correctly certifies the intersection $I_\cap = D \cap D'$ with probability* 1*.*

Proof. Each π_t is accepted by Lemma 5.1.2. For Step B (line B), the bank has inserted every record of D into the MRK tree during Algorithm 2. For each index $i \in \{0, |D'| - 1\}$, if $D'[i] \in D$, its commitment key is present in PK_D and a membership proof verifies by the perfect completeness of the MRK scheme [29], adding i to $I_\cap$. If $D'[i] \notin D$, its key is absent and a non-membership proof verifies by the same guarantee, excluding i from $I_\cap$. Hence $I_\cap = D \cap D'$. □

5.1.4. Soundness

Under Assumption 5.1.1, soundness says that a cheating trainer cannot cause the auditor to output a falsely certified intersection $I_\cap \neq D \cap D'$. Step A ensures the gradient computation was performed correctly on the committed dataset D'. Step B ensures $I_\cap$ faithfully reflects $D \cap D'$. A trainer cannot forge a membership proof for $D'[i] \notin D$ (falsely claiming a sample came from D), nor forge a non-membership proof for $D'[i] \in D$ (concealing that a sample came from D). The CDC assumption closes the gap between the two steps by binding column_commits and sample_poly_commits to the same underlying D'.

Lemma 5.1.4 (Step A is sound)**.** *Under the q-SDH assumption, no PPT cheating trainer can produce a valid Step A transcript for an incorrectly computed gradient, except with negligible probability.*

Proof. Each sumcheck transcript is sound by the Schwartz-Zippel lemma (Section 2.3), which states that a false polynomial identity holds at a uniformly random challenge with probability at most $deg(g)/|\mathbb{F}|$, which is negligible for BLS12-381's scalar field. The MKZG openings are bound by KZG evaluation binding under q-SDH (Section 2.9), so a PPT prover cannot open column_commits$[j]$ or err_commit to a value other than the committed one. Finally, the error-consistency check (Equation (4.6)) ties err_commit to the already-bound column and label commitments at r_c. Any mismatch is caught with overwhelming probability by Schwartz-Zippel applied to the consistency polynomial. □

Lemma 5.1.5 (Step B is sound)**.** *Under the DL assumption, no PPT cheating trainer can falsify the certified intersection $I_\cap$. This means they cannot produce a valid membership proof for a sample not in PK_D, nor a valid non-membership proof for a sample that is in PK_D, except with negligible probability.*

Proof. Both directions follow from MRK soundness [29]. The intuition is that producing a valid membership proof for key $k \notin \text{dom}(D)$, or a valid non-membership proof for key $k \in \text{dom}(D)$, requires finding a collision in the Pedersen hash H, which contradicts DL hardness (Section 2.12). □

Theorem 5.1.6 (zkMLProv soundness)**.** *Under Assumption 5.1.1, the q-SDH assumption, the DL assumption, and the CDC assumption, no PPT cheating trainer can cause the auditor to output an incorrect certified intersection. The auditor's output satisfies $I_\cap = D \cap D'$ with overwhelming probability.*

Proof. Suppose the auditor outputs a certified intersection $I_\cap$. By Lemma 5.1.4, the gradient updates across all T steps are consistent with column_commits. By Lemma 5.1.5, $I_\cap$ is faithful. Hence, every sample appearing in $I_\cap$ truly belongs to PK_D, and no sample in D has been concealed via a forged non-membership proof.

The CDC assumption closes the remaining gap. Without it, column_commits (multilinear, used in Step A sumchecks) and sample_poly_commits (univariate per-sample KZG, used as ZK Sets keys in Step B) are two separate objects. A cheating trainer could commit to dataset D_1 column-wise for Step A and present ZK Sets keys from a different dataset D_2 for Step B, passing both checks independently. CDC asserts that both commit to the same underlying training matrix, ruling this out. Under CDC, the data certified as belonging to D in Step B is exactly the data whose gradient computation was verified in Step A. Together, the three conditions imply the trainer ran gradient descent correctly on D', and the certified intersection $I_\cap = D \cap D'$ faithfully identifies which training samples belong to the bank's private dataset D. □

5.1.5. Zero-Knowledge

We structure the argument in three lemmas. First, we analyze the zero-knowledge property for a single gradient-descent step. Then we use this to analyze the zero-knowledgene property for all T steps. Finally, we consider the ZK Sets component, before combining them in Theorem 5.1.10.

Lemma 5.1.7 (One-step of proof of training in Algorithm 4.6 is Computationally Zero-Knowledge)**.** *The proof* $(\mathsf{fwd}, \mathsf{bwd})_t$ *produced by Algorithm 4 for a single gradient-descent step t is computationally zero-knowledge in the random oracle model. Specifically, there exists a PPT simulator S_t such that*

$$S_t(x_t) \approx_c \mathsf{View}\big[(\mathsf{fwd}, \mathsf{bwd})_t\big],$$

*where x_t denotes the public inputs for step t (*DataCommitment*, claimed weight update, step index t).*

Proof. Each proof $(\mathsf{fwd}, \mathsf{bwd})_t$ comprises five sub-proofs. The masking-polynomial sumcheck is perfectly honest-verifier zero-knowledge by Thaler [39], meaning the simulator's output is identically distributed to the real transcript. Verifier challenges are replaced by SHA-256 hash outputs via the Fiat-Shamir transform (Section 2.7), making every sub-proof non-interactive. The Fiat-Shamir transform is secure only in the random oracle model, which assumes SHA-256 is indistinguishable from a truly random function against computationally bounded adversaries. This is why the resulting proof is computationally rather than perfectly zero-knowledge. Since the proof is a static string, no verifier can adaptively choose challenges, and the ZK guarantee extends to all verifiers.

The five sub-proofs are:

1. **Forward ZkProductSumcheck**: Proves $\sum_{i \in B_t} \widetilde{\mathsf{sel}}(i) \cdot \widetilde{\mathsf{pred}}(i) = S_t$.
2. **Error commitment**: err_commit = MKZG(ε), fixed before any Fiat-Shamir challenge is drawn.
3. d **backward ZkTripleProductSumchecks**: One per feature j, each proving $g_j = \sum_i \widetilde{\mathsf{sel}}(i) \cdot \varepsilon(i) \cdot \tilde{X}_j(i)$, masked by an independent zero-sum polynomial R_j.
4. **Bias ZkProductSumcheck**: Proves $g_{\mathsf{bias}} = \sum_i \widetilde{\mathsf{sel}}(i) \cdot \varepsilon(i)$.
5. **Error-consistency check**: MKZG openings of err_commit, the label column, and all d feature columns at a shared Fiat-Shamir point r_c, verifying $\varepsilon = \mathsf{pred} - y$ point-wise.

Since sub-proof (3) has d instances, there are $4 + d$ sub-proofs in total. Let $\ell \in \{1, \ldots, 4 + d\}$ index them. Sub-proofs (1), (3), and (4) are masked sumchecks. By Thaler [39], each admits an efficient simulator. Sub-proof (2) is a commitment fixed before any Fiat-Shamir challenge is drawn, so it is trivially simulatable. Sub-proof (5) admits a simulator by the hiding property of the PST13 commitment scheme [2]. Let $S^{(\ell)}$ denote the simulator for sub-proof ℓ.

The joint simulator $S_t = (S^{(1)}, \ldots, S^{(4+d)})$ concatenates the simulated transcripts. Independence across sub-proofs holds because each sub-proof's Fiat-Shamir hash is prefixed with the sub-proof index, so their random oracle queries are disjoint. Hence $S_t(x_t) \approx_c \mathsf{View}[(\mathsf{fwd}, \mathsf{bwd})_t]$. □

Lemma 5.1.8 (T-steps of gradient descent in Algorithm 4 is Computationally Zero-Knowledge)**.** *The full proof-of-training transcript $\pi_{\mathsf{PoT}} = \{(\mathsf{fwd}, \mathsf{bwd})_t\}_{t=0}^{T-1}$ is computationally zero-knowledge in the random oracle model. There exists a PPT simulator $S_{0:T}$ such that*

$$S_{0:T}(x) \approx_c \mathsf{View}[\pi_{\mathsf{PoT}}].$$

Proof. We apply a T-step hybrid argument. Let us define a hybrid H_k as the transcript in which steps $\{0, \ldots, k-1\}$ are replaced by the simulated outputs of $S_0, \ldots, S_{k-1}$ from Lemma 5.1.7, while steps $\{k, \ldots, T-1\}$ remain real. H_0 is the real distribution and H_T is fully simulated.

Each transition $H_k \to H_{k+1}$ replaces a single real step-k proof with $S_k(x_k)$. So, if there exists a PPT distinguisher that separates H_k from H_{k+1}, it would simultaneously distinguish a single real step-k proof from $S_k(x_k)$, contradicting Lemma 5.1.7.

Independence across steps holds for two reasons. Firstly, the weights w_t at each step are publicly derivable from the DataCommitment and the weight-update claims are already present in the proof transcript, so no secret witness is shared between steps. Secondly, each step independently samples a masking polynomial R_t, so the commitment $\mathsf{mask\_commit}_t$ differs between steps with high probability, ensuring that each step's Fiat-Shamir hash draws from a distinct region of the random oracle.

By a union bound over T hybrids, we have

$$\mathsf{Adv}[H_0, H_T] \leq T \cdot \mathsf{negl}(\lambda) \tag{5.1}$$

$$= \mathsf{negl}(\lambda), \tag{5.2}$$

so $S_{0:T}(x) = (S_0(x_0), \ldots, S_{T-1}(x_{T-1}))$ satisfies the claim. Here T is the number of training iterations and is independent of the security parameter $\lambda = 128$ which is the bit-security level of BLS12-381. □

Lemma 5.1.9 (ZK Sets are perfectly Zero Knowledge)**.** *The ZK Sets membership and non-membership proofs (Section 2.12) are perfectly zero-knowledge. There exists a simulator S_{zkSets} such that for any query key k and answer v (or $\perp$),*

$$S_{zkSets}(\mathsf{PK}_D,\ k,\ v) \equiv \mathsf{View}[\mathsf{ZKSets.Prove}(D,\ k)],$$

where $\equiv$ denotes identical distributions.

Proof. This result follows from Micali, Rabin, and Kilian [29], Theorem 2. The simulator S_{zkSets} exploits the perfect hiding of Pedersen commitments at each tree node. Since the commitment $c_v = g^{m_v} \cdot h_v^{r_v}$ is uniformly distributed over $\mathbb{G}$ for any fixed message m_v (as the blinding factor r_v is drawn uniformly), the simulator can replace every off-path node with a fresh uniformly random group element. On-path nodes are reconstructed consistently from the query answer v using the frontier trapdoors. The resulting simulated path is identically distributed to a real proof path, giving perfect zero-knowledge. Please note that perfectly zero-knowledge is a stronger notion than computational zero-knowledge. □

Theorem 5.1.10 (zkMLProv is ZK under CDC)**.** *Under Assumption 5.1.1, zkMLProv (Algorithms 3–5) is computationally zero-knowledge in the random oracle model. There exists a PPT simulator S_{protocol} such that*

$$S_{\mathsf{protocol}}(x) \approx_c \mathsf{View}\big[(\pi_{\mathsf{PoT}},\ \pi_{\mathsf{ZKS}})\big],$$

where $x = (\mathsf{DataCommitment},\ \mathsf{PK}_D,\ \mathsf{claimed\_weights})$ is the full public input.

Proof. By Lemmas 5.1.8 and 5.1.9 we have simulators $S_{0:T}$ for the zkPoT component and S_{zkSets} for the ZK Sets component. Define the combined simulator as

$$S_{\mathsf{protocol}}(x) = \big(S_{0:T}(x_{\mathsf{PoT}}),\ S_{zkSets}(x_{\mathsf{ZKS}})\big).$$

Indistinguishability from the real proof $(\pi_{\mathsf{PoT}}^{\mathsf{real}},\ \pi_{\mathsf{ZKS}}^{\mathsf{real}})$ follows by a two-step hybrid:

$$\underbrace{(\pi_{\mathsf{PoT}}^{\mathsf{real}},\ \pi_{\mathsf{ZKS}}^{\mathsf{real}})}_{\text{real}} \approx_c \underbrace{(S_{0:T}(x_{\mathsf{PoT}}),\ \pi_{\mathsf{ZKS}}^{\mathsf{real}})}_{\text{hybrid}} \equiv \underbrace{(S_{0:T}(x_{\mathsf{PoT}}),\ S_{zkSets}(x_{\mathsf{ZKS}}))}_{\text{simulated}}.$$

Step 1 ($\pi^{\text{real}}_{\text{PoT}} \to S_{0:T}$): By Lemma 5.1.8, no PPT distinguisher separates $\pi^{\text{real}}_{\text{PoT}}$ from $S_{0:T}(x_{\text{PoT}})$, even given $\pi^{\text{real}}_{\text{ZKS}}$ as side information. This holds because the ZK Sets proofs use only Pedersen hash evaluations, which are disjoint from the Fiat-Shamir random oracle queries used by the zkPoT component.

Step 2 ($\pi^{\text{real}}_{\text{ZKS}} \to S_{zkSets}$): By Lemma 5.1.9, $\pi^{\text{real}}_{\text{ZKS}} \equiv S_{zkSets}(x_{\text{ZKS}})$ (identically distributed), so this transition is exact and holds even given $S_{0:T}(x_{\text{PoT}})$ alongside it.

Assumption 5.1.1 governs soundness, not zero-knowledge. It ensures that the sample_poly_commits[i] used as ZK Sets keys are consistent with the column_commits[j] used in Step A sumchecks, preventing a dishonest trainer from committing to inconsistent data. The zero-knowledge simulator argument above holds independently of whether CDC is satisfied. □

5.2. Complexity Analysis

We analyze Algorithms 2 to 5 using the following notation: $|D'|$ is the number of training samples with $v = \log_2 |D'|$ sumcheck variables; d is the number of features; T is the number of gradient-descent steps; $|D|$ is the bank's private dataset size; and k is the ZK Sets security parameter (tree depth, fixed at $k = 384$ for BLS12-381). We report time in group operations (scalar multiplications in $\mathbb{G}_1$, the dominant cryptographic cost) and pairing evaluations ($e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$). Output size is measured in group elements. Table 5.1 summarises all results.

5.2.1. Bank Complexities

The Bank participates in two phases: the one-time setup (Algorithm 2) and an audit-time proving step in which it generates ZK Sets proofs for the public DataCommitment. The setup algorithm is described in Section 4.4. Here, the CommitSampleFP is called on each of the samples in the Bank's dataset. CommitSampleFP computes a univariate KZG commitment over the d features and the label, costing $O(d)$ group operations (Section 2.10). CommitmentToZkSetsKey compresses CommitSampleFP's output, costing $O(1)$. The tree construction algorithm is based on the ZK Sets paper by [29]. The complexity of ZKSETS.Commit follows from the MRK construction which requires $O(k)$ hash evaluations and modular exponentiations per element when building the depth-k Pedersen-hash tree (Section 2.12). We consider that the ZkSets tree can hold all the elements in the Bank's dataset D. Hence, we have $k = O(\log |D|)$. From Algorithm 2, we can compute the setup time complexity as follows

$$\begin{aligned} O(\text{Bank Setup}) &= |D|\,(O(\text{CommitSampleFP}) + O(\text{CommitmentToZkSetsKey})) + O(\text{ZKSets.Commit}) \\ &= |D|\,(O(d+1) + O(1)) + O(|D| \cdot k) \\ &= O(|D| \cdot d) + O(|D| \log |D|) \\ &= O(|D| \cdot (d + \log |D|)). \end{aligned} \tag{5.3}$$

The published PK_D is the tree root, i.e. $O(1)$ group elements.

During the audit, the Bank reads the public DataCommitment and generates a membership or non-membership proof for each of the $|D'|$ training samples. Each proof requires traversing the depth-k Pedersen-hash tree, costing $O(k)$ hash evaluations and modular exponentiations (Section 2.12). With $k = O(\log |D|)$, the total audit-time cost for the Bank is:

$$O(\text{Bank Step B}) = |D'| \cdot O(k) = O(|D'| \cdot \log |D|). \tag{5.4}$$

The proofs sent to the auditor have total size $O(|D'| \cdot k) = O(|D'| \log |D|)$ group elements.

5.2.2. Trainer Complexities

The trainer participates in Algorithm 3 and 4. In Algorithm 3, the entire operation consists of $d + 1$ calls to the MKZG commitment for each sample in the trainer's dataset D'. This corresponds to each of the d features of the sample and the label. This operation creates the column_commits component of DataCommitment. For one feature, the MKZG commitment costs $O(|D'|)$ group operations. Then, we create sample_poly_commits by making calls to CommitSampleFP on each of the samples in the trainer's dataset D'. As computed in Section 5.2.1, one operation of CommitSampleFP costs $O(d)$ group operations. We can compute the computational

complexity for trainer's setup as

$$\begin{aligned} \mathcal{O}(\text{Trainer Setup}) &= d\mathcal{O}(\text{MKZG Commit}) + \mathcal{O}(\text{MKZG Commit}) + |D'|\mathcal{O}(\text{CommitSampleFP}) \\ &= d \cdot \mathcal{O}(|D'|) + \mathcal{O}(|D'|) + |D'| \cdot \mathcal{O}(d+1) \\ &= \mathcal{O}(d \cdot |D'|) \end{aligned} \tag{5.5}$$

The published DataCommitment contains $d+1$ MKZG commits and $|D'|$ KZG commits: $O(|D'|+d)$ group elements in total.

For Algorithm 4, the trainer provides a proof for each of the T Gradient Descent steps. For every gradient descent step, we need to generate a proof for the correctness of forward pass (fwd) and a proof for the correctness of backward pass (bwd). We analyze the complexities for each of these below.

Forward pass: This corresponds to the lines 2-17 in Algorithm 4. Here, the trainer must first build the sel vector which represents which data samples in the trainer's dataset D' belong to the batch involved in the gradient descent step. This requires $\mathcal{O}(|D'|)$ group operations. The MKZG commitment to sel, requires $\mathcal{O}(|D'|)$ group operations. The trainer must then compute the pred vector, where for each of the $|D'|$ samples the dot product is computed for the d features. This yields a cost of $\mathcal{O}(d \cdot |D'|)$. The computation of S_t vector involves $\mathcal{O}(|D'|)$ mutiplications. The cost to generate a proof using ProductSumCheck is $\mathcal{O}(|D'|)$ from Section 2.6. Finally, for the Fiat Shamir challenge corresponding to the forward pass, we need $d+2$ MKZG openings corresponding to d features for the challenge in column_commits, opening for batch membership in sel_commit and opening for the mask in mask_commit. Each opening has a cost of $\mathcal{O}(|D'|)$ (Refer Section 2.11), giving a total cost of $\mathcal{O}((d+2) \cdot |D'|)$. Thus for the forward pass, we have the following time complexity

$$\begin{aligned} \mathcal{O}(\text{Forward Pass for one iteration}) &= \mathcal{O}(|D'|) + \mathcal{O}(|D'|) \\ &\quad + \mathcal{O}(d \cdot |D'|) + \mathcal{O}(|D'|) + \mathcal{O}(|D'|) + \mathcal{O}((d+2) \cdot |D'|) \\ &= \mathcal{O}((2d+6) \cdot |D'|) \\ &= \mathcal{O}(d \cdot |D'|) \end{aligned} \tag{5.6}$$

The forward pass generates a sumcheck transcript corresponding to ZkProductSumcheck (line 11) in the order of $\mathcal{O}(3 \log |D'|)$ since this is a degree-2 Product Sumcheck (See 2.6). In order to check the evaluations for the Fiat-Shamir challenge point r, the commitments for the sel vector and the mask vector (See line 11 of Algorithm 4.6) along with their openings (See lines 16-17 in Algorithm 4.6) are sent in the proof for the forward pass. The commitments for sel and mask vectors have a size of the order $\mathcal{O}(1)$ each. Similarly the openings for sel_commit and mask_commit at the Fiat Shamir challenge point is of the order of $\mathcal{O}(\log |D'|)$ each. Additionally, since pred is never committed directly, the verifier recomputes $\widetilde{\text{pred}}(r) = \sum_{j=0}^{d-1} w_j \cdot \tilde{X}_j(r)$ at the Fiat-Shamir challenge point r using openings of the d feature columns in column$_c$ommits (See lines 13–14 in Algorithm 4). Each opening is of the order $\mathcal{O}(\log |D'|)$, giving a total cost of $\mathcal{O}(d \log |D'|)$ for all feature openings. This gives us the proof size of the forward pass as

$$\begin{aligned} \mathcal{O}(\text{Forward Pass Proof Size for one iteration}) &= \mathcal{O}(3 \log |D'|) + 2 \times \mathcal{O}(1) + (d+2) \times \mathcal{O}(\log |D'|) \\ &= \mathcal{O}(d \log |D'|) \end{aligned} \tag{5.7}$$

Backward pass: This corresponds to lines 18-44 in Algorithm 4. Here, the trainer first computes the error vector err (line 20, Algorithm 4) for each of the samples in the trainer's dataset $|D'|$. This requires $\mathcal{O}(|D'|)$ operations. The MKZG commitment to err requires $\mathcal{O}(|D'|)$ group operations. For each of the d features, the trainer computes the gradient g_j (line 24, Algorithm 4), costing $\mathcal{O}(|D'|)$ per feature, giving $\mathcal{O}(d \cdot |D'|)$ in total. A fresh zero-sum mask R_j is committed per feature, each costing $\mathcal{O}(|D'|)$ group operations, for a total of $\mathcal{O}(d \cdot |D'|)$. The cost to generate a proof using ZkTripleProductSumcheck is $\mathcal{O}(|D'|)$ per feature (Section 2.6), giving $\mathcal{O}(d \cdot |D'|)$ for all d features. The bias gradient $g_{\text{bias}} = \sum_i \text{sel}(i) \cdot \text{err}(i)$ is a special case: the bias feature column is the constant vector $\mathbf{1}$ and requires no column oracle check, so its proof uses a degree-2 ZkProductSumcheck rather than a triple product sumcheck, costing $\mathcal{O}(|D'|)$ and adding 2 MKZG openings — both absorbed by the $\mathcal{O}(d \cdot |D'|)$ bound. Finally, the error consistency check requires $d+2$ MKZG openings at

r_c, each costing $O(|D'|)$ (Section 2.11). Thus for the backward pass, we have the following time complexity:

$$\begin{aligned} O(\text{Backward Pass for one iteration}) &= O(|D'|) + O(|D'|) + O(d \cdot |D'|) + O(d \cdot |D'|) \\ &\quad + O(d \cdot |D'|) + O((d+2) \cdot |D'|) \\ &= O(d \cdot |D'|) \end{aligned} \tag{5.8}$$

The backward pass generates d sumcheck transcripts corresponding to d calls to ZkTripleProductSumcheck, each of the order $O(4 \log |D'|)$ field elements since this is a degree-3 Triple Product Sumcheck (See Section 2.6). Before any Fiat-Shamir challenge is drawn, the prover commits to the error vector err and d per-feature masks R_j, giving $d+1$ commitments each of size $O(1)$.

For each of the d features, the verifier must check the evaluation of $\widetilde{\text{sel}}(r')$, $\widetilde{\text{err}}(r')$, $\tilde{X}_j(r')$, and $\tilde{R}_j(r')$ at the per-feature Fiat-Shamir challenge point r'_j. This requires 4 MKZG openings per feature (See lines 29-32 in Algorithm 4), each of size $O(\log |D'|)$, giving $O(4d \log |D'|)$ in total.

Additionally, to verify that err_commit encodes pred$(i) - y_i$ and not an arbitrary vector, the prover sends $d+2$ openings at a shared Fiat-Shamir challenge point r_c: one for err_commit, one for the label column column_commits$[d]$, and d for the feature columns column_commits$[j]$ (See lines 35-38 in Algorithm 4). Each opening is $O(\log |D'|)$.

Thus the backward pass proof size for one iteration is:

$$\begin{aligned} O(\text{Backward Pass Proof Size for one iteration}) &= (d+1) \times O(1) + O(4d \log |D'|) + O(4d \log |D'|) \\ &\quad + O((d+2) \log |D'|) \\ &= O(d \log |D'|) \end{aligned} \tag{5.9}$$

Hence, using Equations 5.6 and 5.8, for T steps of gradient descent we have

$$O(\text{Trainer Prove for } T \text{ steps of Gradient Descent}) = O(T \cdot d \cdot |D'|). \tag{5.10}$$

Similarly, using Equations 5.7 and 5.9, the proof size for T steps of gradient descent is

$$O(\text{Trainer Prove Proof size for } T \text{ steps of Gradient Descent}) = O(T \cdot d \cdot \log |D'|). \tag{5.11}$$

5.2.3. Auditor Complexities

The Auditor participates in Algorithm 5. Unlike the Trainer, the Auditor performs no commitment or sumcheck proving operations. It only verifies MKZG opening proofs and checks sumcheck transcripts sent by the Trainer. We consider the complexities associated with the two verification steps A and B.

Step A: This corresponds to lines 2-34 in Algorithm 5. For the forward verification, the Auditor verifies $d+2$ MKZG openings (lines 5-6 in Algorithm 5), corresponding to the d feature columns, sel_commit, and mask_commit at the Fiat-Shamir challenge point r. Each MKZG verification costs $O(\log |D'|)$ pairing evaluations (Section 2.11), giving $O((d+2) \log |D'|)$ pairings. The Auditor then verifies the forward sumcheck transcript using ZkProductSumcheck.verify, costing $O(\log |D'|)$ field operations (Section 2.6).

For the backward verification, for each of the d features, the Auditor verifies 4 MKZG openings (See lines 16-19 in Algorithm 5) corresponding to sel_commit, err_commit, column j, and mask_commit$_j$ at the per-feature challenge point r'_j. This gives $4d$ MKZG verifications, each costing $O(\log |D'|)$ pairings, for a total of $O(4d \log |D'|)$. The Auditor also verifies d triple product sumcheck transcripts using ZkTripleProductSumcheck.verify costing $O(\log |D'|)$ field operations per feature (Section 2.6), giving a total of $O(d \cdot \log |D'|)$.

Finally, for the error consistency check, the Auditor verifies $d+2$ MKZG openings at the shared challenge point r_c (See line 30 in Algorithm 5), each costing $O(\log |D'|)$ pairings, giving $O((d+2) \log |D'|)$. Thus for Step A, we have the following time complexity for one iteration:

$$\begin{aligned} O(\text{Step A, one iteration}) &= O((d+2) \log |D'|) + O(4d \log |D'|) + O((d+2) \log |D'|) \\ &= O(d \log |D'|) \end{aligned} \tag{5.12}$$

Over T iterations we have $O(T \cdot d \cdot \log |D'|)$ pairing evaluations.

Step B: This corresponds to lines 35-42 in Algorithm 5. Since DataCommitment is public, the Bank reads all sample_poly_commits[i] and independently generates a membership or non-membership proof for each of the $|D'|$ sample keys, at cost $O(k)$ per proof (Section 2.12). For each of the $|I| \leq |D'|$ unique sample indices, the Auditor derives the ZK Sets key via CommitmentToZkSetsKey, costing $O(1)$, then verifies the bank-supplied proof at cost $O(k)$ Pedersen hash evaluations (Section 2.12). Since $k = O(\log |D|)$ as established in Section 5.2.1, the auditor's Step B cost is:

$$\begin{aligned} O(\text{Step B, Auditor}) &= |I| \cdot (O(1) + O(k)) \\ &= O(|D'| \cdot \log |D|) \end{aligned} \tag{5.13}$$

The bank's cost for generating all $|D'|$ proofs is:

$$O(\text{Step B, Bank}) = |D'| \cdot O(k) = O(|D'| \cdot \log |D|). \tag{5.14}$$

We summarize all of the computation and communication complexity analysis in Table 5.1.

Table 5.1: Complexity summary. Group ops are scalar multiplications in $\mathbb{G}_1$; pairings are $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$ evaluations.

Participant	Phase	Time	Output size
Bank $\mathcal{B}$	Setup (once)	$O\big(\|D\|(d + \log \|D\|)\big)$ group ops	$O(1)$ group elements
Bank $\mathcal{B}$	Step B (prove)	$O(\|D'\| \log \|D\|)$ hash ops	$O(\|D'\| \log \|D\|)$ group elements
Trainer $\mathcal{T}$	Setup (once)	$O(\|D'\|d)$ group ops	$O(\|D'\| + d)$ group elements
Trainer $\mathcal{T}$	Prove (T iters)	$O(T\|D'\|d)$ group ops	$O(Td \log \|D'\|)$ elements
Auditor $\mathcal{A}$	Step A (T iters)	$O(Td \log \|D'\|)$ pairings	—
Auditor $\mathcal{A}$	Step B	$O(\|D'\|k)$ operations	—

6

Evaluation

In this chapter, we present benchmark results for the two components of zkMLProv, namely the zero-knowledge proof of gradient descent (zkPoGD) and the ZK Sets membership proof. To allow comparison with the theoretical analysis in Section 5.2, we organize the evaluations by participant role. All experiments run single-threaded on BLS12-381, using a 61-bit Mersenne prime field. We test committed dataset sizes $|D'| \in \{8, 16, 32, 64, 128, 256\}$ and feature counts $d \in \{2, 4, 8, 16, 23\}$, where $d = 23$ matches the feature count of the credit-scoring dataset used throughout. Note that sel, mask, and all sumcheck operations are defined over the full committed dataset D', so proving complexity scales with $|D'|$, not with the gradient-descent batch $|B_t|$ (See Appendix A). Our research questions (Section 1.4) concern the correctness of training and membership proofs, not model accuracy or dataset type, so we do not vary the dataset across experiments. We begin by describing the dataset, then show how zkMLProv identifies trainers with unauthorized access, and finally present results for each participant.

6.1. Dataset

The benchmarks use the *Default of Credit Card Clients* dataset from the UCI Machine Learning Repository [45], collected from a Taiwanese bank in 2005. The dataset has 30,000 samples and 23 features. The binary label records whether a client defaulted on payment the following month, with 0 indicating no default and 1 indicating default. All features are min-max normalized to $[-1, +1]$ before encoding as 16-bit fixed-point field elements, as required by the field arithmetic (Section 4.5).

The 23 features fall into four groups as listed in Table 6.1. The demographic block captures static client attributes. The six repayment-status features record monthly payment behavior over a six-month window from September to April. The ordinal scale runs from -2 (no consumption) through -1 (paid in full) up to positive integers representing the number of months of payment delay. The bill-amount and payment-amount blocks each contain six continuous monetary variables covering the same window. The target variable is binary, it represents whether the candidate has defaulted next month.

6.2. Identifying Trainers with Unauthorised Access

To evaluate the data authorisation component of zkMLProv, we simulate a trainer whose batch mixes records drawn from the bank's private dataset D with independently sourced samples. The total training dataset has $|D'| = 4{,}096$ samples, and the bank's private dataset D also contains $4{,}096$ samples. The fraction of samples drawn from D ranges from 5% to 100%. For each sample, the bank generates a ZK Sets proof. If the sample's KZG commitment key is present in the bank's Merkle tree, the bank generates a membership proof. Otherwise, it generates a non-membership proof. The bank then sends this proof to the auditor, who verifies it against PK_D.

Figures 6.1 and 6.2 show that the auditor correctly identifies every sample drawn from the bank's private dataset and correctly confirms every clean sample, across all mixing ratios. Every zkPoGD training proof also verifies successfully, confirming that gradient descent was executed correctly on the committed data. Note that we do not generate fake proofs in this evaluation, so this outcome is expected (Due to Theorem 5.1.3). Since

Table 6.1: Features of the Default of Credit Card Clients dataset [45]. TWD = New Taiwan Dollar.

Name	Description	Type
Demographic		
LIMIT_BAL	Credit limit (TWD)	Continuous
SEX	1 = male, 2 = female	Categorical
EDUCATION	1 = graduate, 2 = university, 3 = high school, 4 = other	Ordinal
MARRIAGE	1 = married, 2 = single, 3 = other	Categorical
AGE	Age in years	Continuous
Repayment status (6 features)		
PAY_0–PAY_6	Monthly repayment status, Sep–Apr; −2 = no consumption, −1 = paid in full, ≥ 1 = months delayed	Ordinal
Bill amount (6 features)		
BILL_AMT1–6	Monthly statement balance, Sep–Apr (TWD)	Continuous
Payment amount (6 features)		
PAY_AMT1–6	Monthly payment made, Sep–Apr (TWD)	Continuous
Target		
DEFAULT	Default payment next month; 0 = no default, 1 = default	Binary

ZK Sets is a complete proof system (Section 5.1.3), a sample absent from the bank's Merkle tree will produce a passing non-membership proof. Taken together, these results are consistent with the perfect completeness of zkMLProv.

The right panel of each figure breaks down the per-step audit cost by component. ZK Sets membership verification dominates at high mixing ratios, since the auditor must process a membership proof for every bank sample in D'. Non-membership verification is more expensive per sample ($\approx$11.5 ms vs. $\approx$4.5 ms) because the bank must additionally traverse sibling nodes to construct the absence certificate, so its share of the total cost grows as the fraction of non-bank samples increases. The zkPoGD verification cost (purple) remains constant across mixing ratios at $\approx$475 ms for linear regression and $\approx$497 ms for logistic regression, since it depends only on $|D'|$ and d, not on which samples are bank-authorised.

6.3. Evaluations for the Bank

The bank runs BankSetup once before training begins. In this step, it builds a Pedersen Merkle ZK Sets tree over the authorized dataset D and publishes the root PK_D. During the audit, the bank generates one membership or non-membership proof for each training sample belonging to the model trainer and sends it to the auditor.

ZK Sets membership proving. Figure 6.3 shows per-sample proving and verification time as the dataset size varies. Membership proving time is roughly constant at under 1.5 ms across $|D| \in \{512, 1024, 2048, 4096\}$, consistent with the $O(k)$ operation count: each proof requires $2k + 1$ hash-map lookups on the pre-built tree (Section 2.12), and the working set of accessed nodes fits within the CPU cache across the tested dataset sizes. Verification is also constant at about 7.9 ms regardless of $|D|$, since it checks a fixed-length k-element proof path without any tree traversal, consistent with the $O(k)$ bound. The per-sample proving cost does not depend on the zkPoGD batch size, since ZK Sets operates on individual sample keys.

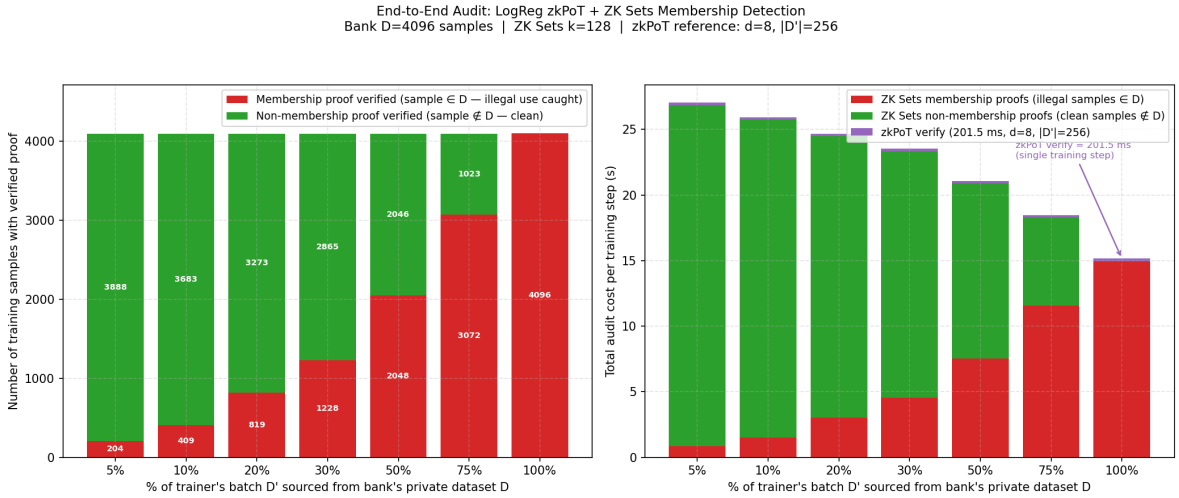


Figure 6.1: End-to-end audit for logistic regression ($d = 8$, $|D'| = 4{,}096$, $D = 4{,}096$, $k = 128$). Left: samples with verified membership proof (red, $\in D$) and non-membership proof (green, $\notin D$) per mixing ratio. Right: total audit cost per training step, broken down by ZK Sets membership, non-membership, and zkPoGD verification. The auditor achieves 100% detection at every mixing ratio and every zkPoGD training proof verifies successfully, consistent with the perfect completeness of zkMLProv. All proofs of membership and non-membership are accepted, consistent with completeness (Section 5.1).

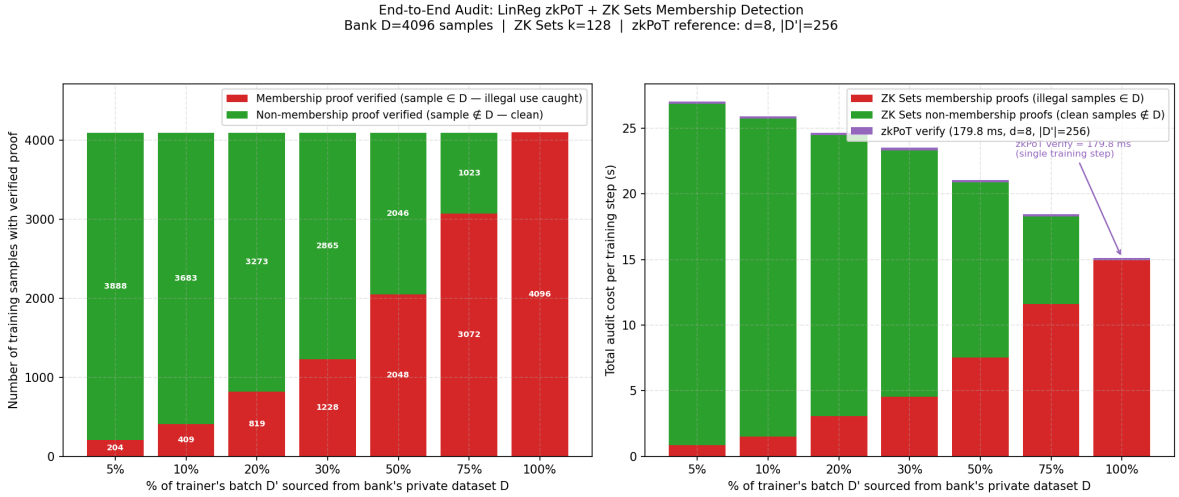


Figure 6.2: End-to-end audit for linear regression ($d = 8$, $|D'| = 4{,}096$, $D = 4{,}096$, $k = 128$). Same layout as Figure 6.1. Results are consistent with perfect completeness and soundness of the full protocol. The zkPoGD verification cost (purple) is lower for linear regression in this configuration due to the absence of the sigmoid function.

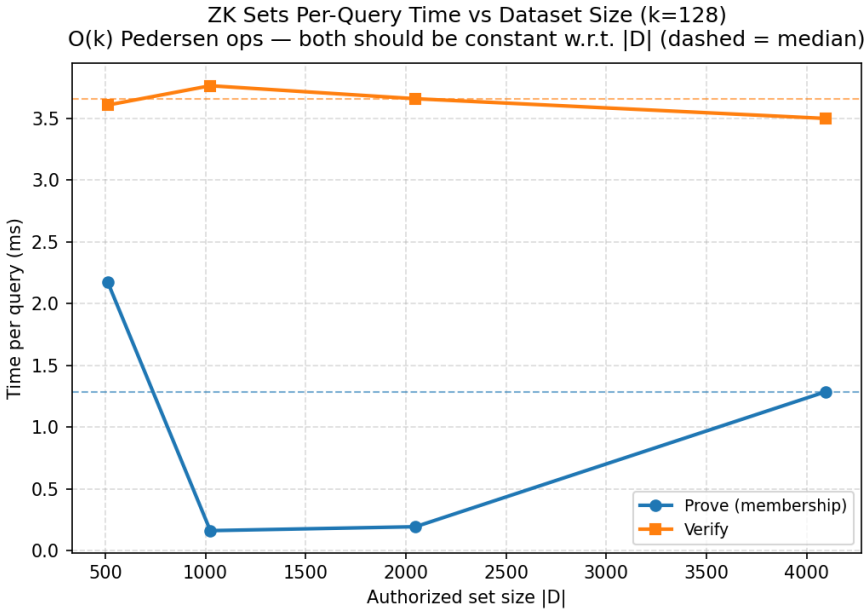


Figure 6.3: ZK Sets per-sample membership prove and verify time vs. dataset size ($k = 128$). Verification is constant at ≈3.7 ms ($O(k)$ path check).

6.4. Evaluations for the Trainer

The trainer's main cost is due to TrainerProve. For each gradient-descent step, this component runs the forward and backward sumchecks and generates the accompanying KZG opening proofs. Figure 6.4 shows the raw cost of each KZG operation on BLS12-381 as a function of SRS degree. Setup, commit, and open all scale linearly with degree, as each is dominated by a multi-scalar multiplication. Verify is constant at about 1 ms regardless of degree, since it requires only a single pairing check.

Prove time. Figure 6.5 shows prover execution time per gradient-descent step. Proving time grows as $O(|D'| \cdot d)$. Each of the d backward sumchecks iterates over all $|D'|$ samples, and each sumcheck is paired with a KZG opening whose cost is dominated by the multi-scalar multiplication shown in Figure 6.4. At the maximum configuration of $d = 23$ and $|D'| = 256$, logistic regression takes about 3.1 s per step and linear regression about 2.9 s. The small difference comes from the two additional ProvePower triple-product sumchecks that the polynomial sigmoid approximation requires.

Proof size. From Figure 6.6a we can see that the proof size for zkPoGD grows logarithmically with the training dataset size as compared to the proof size for ZK Sets. Figure 6.6b shows proof size per gradient-descent step grows linearly with respect to the number of features. From Figure 6.7, we can see that the zkPoGD proof grows logarithmically with committed dataset size, from about 32 KB at $|D'| = 8$ to 80 KB at $|D'| = 256$ (For d=23). This is because the sumcheck transcript adds one round of field elements for each factor of two in $|D'|$. The ZK Sets membership proof stays constant at about 20 KB regardless of $|D'|$, since its size is $O(k)$ and the tree depth $k = 128$ is fixed. Each additional feature adds one backward-pass sumcheck and one KZG opening, which contributes a fixed number of $\mathbb{G}_1$ elements to the transcript.

Figure 6.7 compares proof size for logistic and linear regression at $d = 23$. Logistic regression produces proofs that are roughly 2 KB to 5 KB larger, and the gap grows with $|D'|$. This overhead comes from the degree-3 polynomial sigmoid approximation $P_3(x) = \frac{1}{2} + \frac{1}{4}x - \frac{1}{48}x^3$. The prover must additionally commit to the score, squared-score, and cubic-score vectors, then run four Fiat-Shamir consistency checks (Checks A–D, Section 4.6), whose sumcheck transcripts grow logarithmically with $|D'|$. Linear regression avoids this overhead entirely. Because the residual $\varepsilon_i = \text{pred}_i - y_i$ is linear in the committed columns, error consistency follows directly from MLE linearity without any extra commitments.

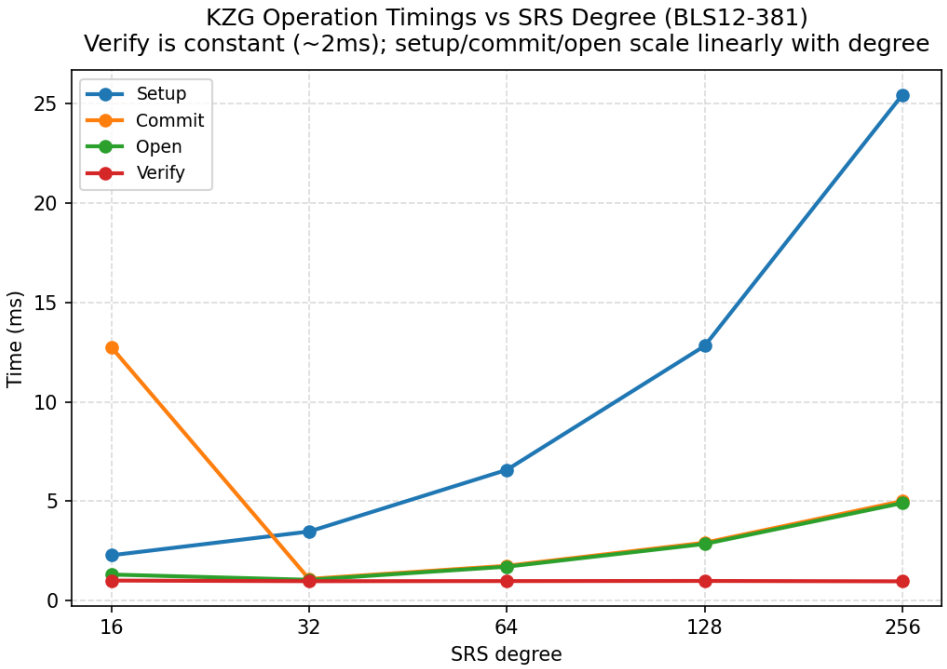


Figure 6.4: KZG operation timings vs. SRS degree on BLS12-381. Setup, commit, and open are linear in degree (multi-scalar multiplication cost); verify is constant at ~1 ms (one pairing check).

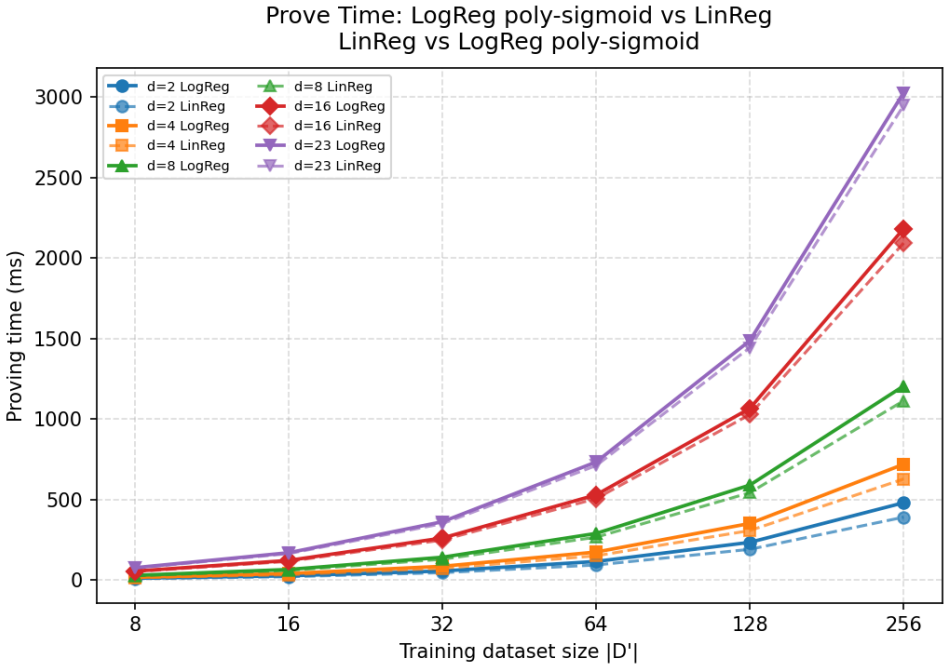
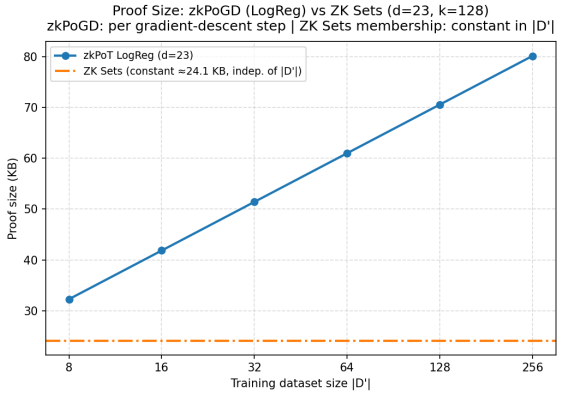
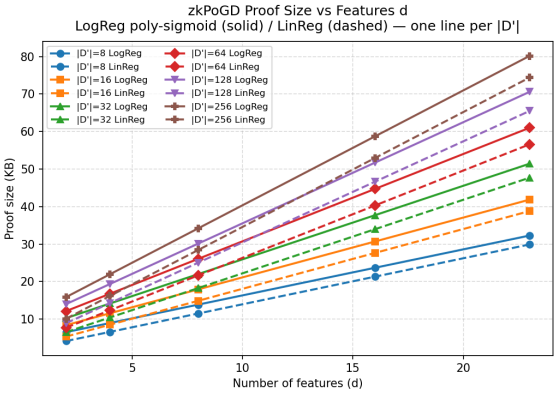


Figure 6.5: Prover time per gradient-descent step vs. committed dataset size $|D'|$, one line per feature count d. Proving time is $O(|D'| \cdot d)$. Solid lines: logistic regression; dashed lines: linear regression.



(a) Proof size vs. committed dataset size $|D'|$ at $d = 23$: logistic regression zkPoGD grows logarithmically with $|D'|$; ZK Sets is constant at ≈20 KB because tree depth $k = 128$ is fixed.



(b) zkPoGD proof size vs. number of features d, one line per committed dataset size $|D'|$. Proof size is $O(d)$.

Figure 6.6: zkPoGD and ZK Sets proof sizes (KZG oracle, one gradient-descent step).

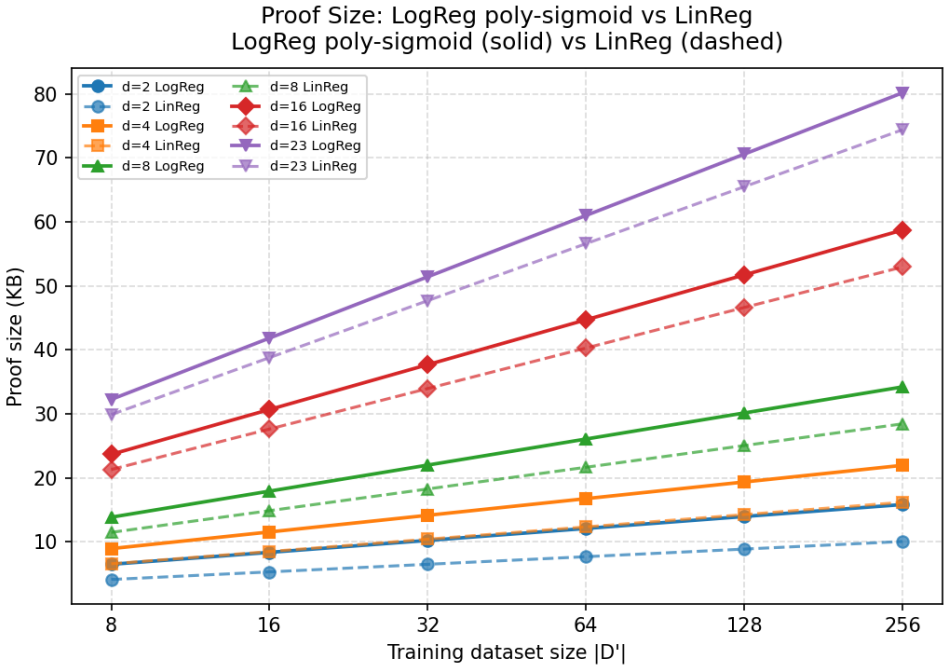


Figure 6.7: Proof size for logistic regression (solid) vs. linear regression (dashed). The proof size grows logarithmically with the training dataset size $|D'|$ for both linear and logistic regression. Logistic regression has a higher proof size due to the additional sumcheck transcripts involved in the polynomial sigmoid approximation.

6.5. Evaluations for the Auditor

The auditor verifies two independent properties per training run. The first is training correctness, checked via zkPoGD. The second is data authorisation, checked via ZK Sets membership proofs. Both verifications use only the public proof transcript and the published commitments, without access to any raw training data.

zkPoGD verifier time. Figure 6.8 shows verifier execution time per gradient-descent step. Verification time grows logarithmically with committed dataset size. The dominant cost is $O(d)$ KZG pairing checks, each taking about 1 ms as shown in Figure 6.4. The mild dependence on $|D'|$ comes from the $O(\log |D'|)$ sumcheck evaluation rounds the verifier must process per backward pass. At $d = 23$, logistic regression verification takes about 234 ms at $|D'| = 8$ and 497 ms at $|D'| = 256$. Linear regression is slightly faster at 224 ms and 475 ms for the same dataset sizes, since logistic regression requires additional pairing checks for the polynomial sigmoid openings.

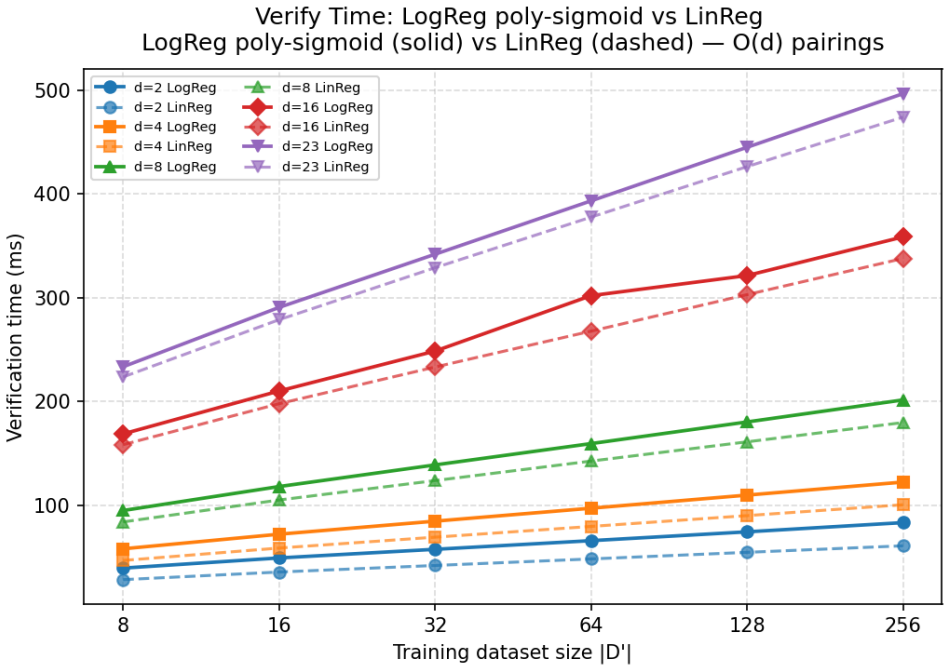


Figure 6.8: Verifier time per gradient-descent step vs. committed dataset size $|D'|$. Lines grow logarithmically with $|D'|$. Logistic regression has higher verification times due to checks on the polynomial approximation.

Per-sample audit cost. Figure 6.9 breaks down the per-sample cost of each ZK Sets proof type across mixing ratios. Membership proving and verification together cost about 3.7 ms per sample. Non-membership proving and verification cost about 6.5 ms per sample, because the bank must additionally traverse sibling nodes in the Merkle tree to construct the absence certificate. Both costs remain constant across all mixing ratios, consistent with the $O(k)$ bound for fixed tree depth $k = 128$. The per-sample verification time shown in Figure 6.3 is also constant in dataset size, confirming that the auditor's per-sample workload does not increase as the authorised set grows.

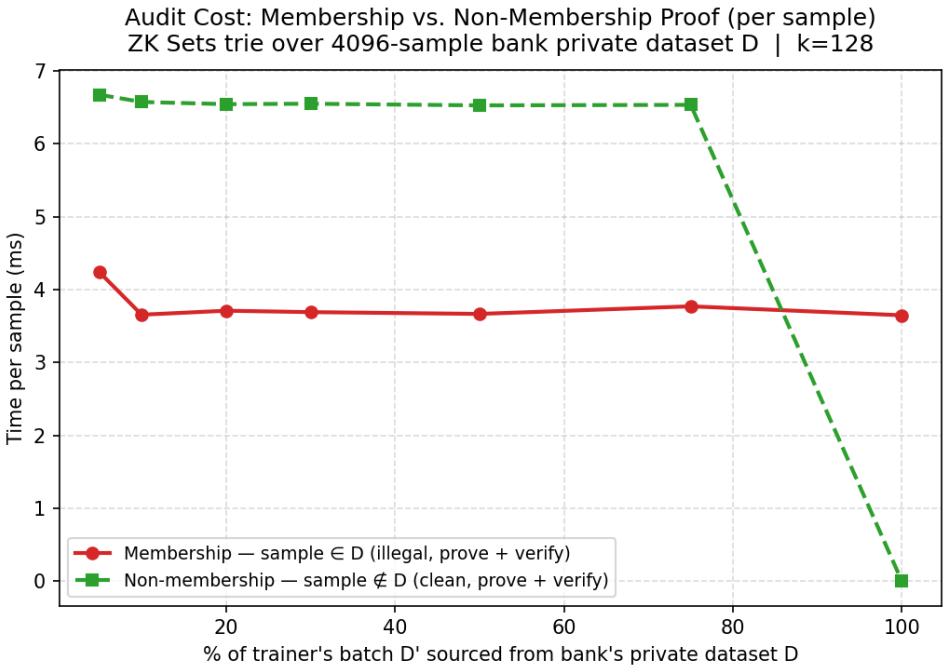


Figure 6.9: Per-sample audit cost (prove + verify) for membership and non-membership proofs across mixing ratios ($k = 128$, $D = 4{,}096$). Both are constant in the mixing ratio; non-membership is costlier because the bank constructs a sibling-path absence certificate. At 100% mixing all samples belong to D, so no non-membership proofs are generated.

7

Discussion

This chapter discusses zkMLProv as proposed in Chapter 4 and evaluated in Chapter 6. Section 7.1 places this work in the context of the two closest prior approaches. Section 7.2 covers the experimental constraints that shaped the evaluation. Section 7.3 describes the remaining limitations of the protocol design. Section 7.4 extrapolates performance to production scale. Section 7.5 outlines the most promising extensions.

7.1. Comparison with Existing Approaches

Table 7.1 places this work alongside the two closest prior zero-knowledge proofs of training. Garg et al. [18] certify logistic regression using MPC-in-the-head, producing a non-succinct proof with verifier cost that scales linearly in dataset size. Kaizen [2] uses a GKR sumcheck with multilinear KZG commitments to verify deep neural networks, achieving $O(d)$ verifier cost per gradient step. This work matches Kaizen's asymptotic complexity for proof size and verifier time on a per-step basis, and adds data provenance. ZK Sets membership and non-membership proofs tie each training sample to the bank's authorized dataset without revealing raw features or labels to the auditor. Neither prior work addresses data provenance.

	Garg et al.	**Kaizen**	**This work**
Model	Logistic regression	Deep NN	Lin. + log. regression
Proof size	Non-succinct	$O(d \log \lvert D' \rvert)$/step	$O(d \log \lvert D' \rvert)$/step
Verifier time	$O(\lvert D' \rvert \cdot d)$/step	$O(d)$	$O(d \log \lvert D' \rvert)$/step
Data provenance	No	No	Yes

Table 7.1: Algorithmic complexity of zero-knowledge proof-of-training protocols. $|D'|$ is dataset size, d is feature dimension. Kaizen reports $O(d)$ verifier cost treating $\log |D'|$ as a constant; this work's $O(d \log |D'|)$ verifier matches that in practice for $|D'| \leq 256$ ($\log |D'| \leq 8$) and adds data provenance via ZK Sets.

Kaizen [2] uses Incremental Verifiable Computations, which enables a prover to fold the proofs associated with multiple gradient descent steps into one. This is the reason why Kaizen's proof size is independent of the number of training iterations.

7.2. Research Limitations

Three constraints shaped the evaluation but did not affect the protocol design itself.

We capped the benchmark scale using a trusted setup generated with 256 group elements, which limits training samples to at most $|D'| = 256$. The binding constraint is the MKZG SRS used for column_commits (Algorithm 3): committing a feature column of $|D'|$ values requires one SRS element per multilinear monomial in $v = \log_2 |D'|$ variables, totalling $2^v = |D'|$ elements. The univariate KZG SRS used for sample_poly_commits depends only on the feature count $d = 23$, not on $|D'|$, so it does not impose this cap. Generating a larger MKZG SRS requires a trusted setup ceremony whose coordination overhead was out of scope for this prototype;

in practice, a production deployment would reuse an existing public SRS [8]. The ZK Sets tree was deployed with depth $k = 128$, supporting datasets of up to 2^{128} elements. Both are infrastructure choices rather than fundamental limitations.

All benchmarks run single-threaded. The d backward-pass sumchecks are independent and could be parallelized across d threads, and GPU-accelerated multi-scalar multiplication for KZG openings would bring prover time down by an order of magnitude. The reported prover times are therefore conservative upper bounds on what a parallel implementation could achieve.

The evaluation uses a single dataset at under one percent of its full scale. The empirical $O(|D'| \cdot d)$ prover time scaling (See Figure 6.5) holds across the range we tested, but the absolute numbers are not representative of production. The UCI credit-card dataset is a realistic proxy for a financial ML setting, not an actual bank deployment.

7.3. zkmlprov Limitations

Three limitations remain in the protocol design regardless of implementation scale.

The most significant is the Consistent DataCommitment (CDC) assumption. The column commits used for training correctness (MKZG, column-wise) and the sample polynomial commits used for data authorization (univariate KZG, row-wise) bind orthogonal projections of the same matrix. Nothing in the current protocol cryptographically enforces that both sets of commitments actually encode the same underlying data. A cheating trainer could use one dataset for training correctness and a different bank-authorized dataset for data authorization, passing both checks while training on data that was never audited. Assumption 5.1.1 in Chapter 5 gives the direction that closes this gap. A bivariate opening at a shared random entry (i^*, j^*) would force both commit families to agree on every matrix entry. Implementing that proof is the highest-priority item for future work.

The logistic regression variant provides honest-verifier zero-knowledge rather than full zero-knowledge. The protocol uses the cubic approximation $P_3(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48}$ instead of the exact sigmoid. A Fiat-Shamir challenge during the sigmoid consistency check reveals the score MLE at one random field point. By the Schwartz-Zippel lemma, this leaks a single field element of negligible information, so it is not a practical concern. The linear regression variant has no such disclosure and is fully zero-knowledge.

Standard KZG commitments are binding but not hiding. The DataCommitment is published before training begins, so a determined adversary with access to the structured reference string can try to verify guesses about the raw training features. For continuous high-entropy values, this is computationally infeasible, but for low-entropy or binary feature spaces, it is a real concern. Blinded KZG commitments would close this gap and are a natural next step.

7.4. Hypothetical Real-World Performance

The benchmarks in Chapter 6 cover $|D'| \leq 256$ and $d \leq 23$. Extrapolating to the full credit-card dataset at $|D'| = 30{,}000$ and $d = 23$ using the confirmed $O(|D'| \cdot d)$ (See Figure 6.5 and Table 5.1) scaling gives a prover time of about 6 minutes per gradient step. A 100-step training run would take approximately 10 hours. In a forensic audit setting where the proof is generated once after training and handed to the auditor $\mathcal{A}$, that is expensive but plausible. A legal audit of similar scope can take weeks.

The verifier cost grows slowly as $O(d \log |D'|)$. At the evaluated scale ($|D'| = 256$, $d = 23$), the auditor $\mathcal{A}$ takes approximately 475 ms per step for linear regression and 497 ms for logistic regression. Extrapolating to $N = 30{,}000$ using the $\log$ factor yields approximately 1 second per step, which is still practical for a forensic audit, where verification runs once offline.

Proof size grows as $O(d \log |D'|)$ per gradient step. At $D' = 30{,}000$ and $d = 23$, one step produces about 16 KB. Over $T = 100$, the full audit package is roughly 16 MB, small enough to send as a file attachment. ZK Sets adds about 20 KB per unique training sample checked, a one-time cost that does not grow with the number of training steps.

7.5. Future Work

We identify some directions for extending the protocol, ordered by expected impact.

First, implement the CDC cross-commitment proof [7]. Opening a bivariate polynomial at a single shared random entry (i^*, j^*) would force the MKZG column commits, and the univariate sample commits to encode the same data matrix, removing the Two-Step Attack and eliminating the Assumption 5.1.1.

Second, extend the proof of gradient descent to shallow neural networks. The GKR protocol [21] supports layered arithmetic circuits, so encoding one or two hidden layers with low-degree polynomial activations or with lookup arguments would cover a much wider class of models without any change to the ZK Sets component.

Replacing the polynomial sigmoid approximations with lookup arguments can reduce the proof size. Lookup arguments circumvent the need to verify the correctness of higher-degree computations.

Fourth, parallelize the d independent backward-pass sumchecks and use GPU-accelerated multi-scalar multiplication for KZG openings. This would bring the provability time down by an order of magnitude and make the 12-hour extrapolation much more tractable in practice.

Fifth, the simulator-based composition proof (Theorem 5.1.10) establishes a general template. Any two zero-knowledge protocols whose random-oracle queries are disjoint can be composed into a single proof that inherits zero-knowledge from each component independently. This opens the door to analogous constructions that pair other verifiable computation schemes with ZK set membership, without requiring a bespoke zero-knowledge argument for each combination.

8

Conclusion

This thesis was written to answer three research questions about verifiable data provenance in machine learning training, in a three-party setting where the bank $\mathcal{B}$ owns a private authorized dataset, the trainer $\mathcal{T}$ trains a model on its own dataset D', and the auditor $\mathcal{A}$ must certify both training correctness and data authorization without seeing any raw features or labels. Through the security analysis presented in Chapter 5 (Section 5.1) and the evaluations presented in Chapter 6 (See Figure 6.2), we have shown that the composition of the zero-knowledge proofs upholds completeness, soundness, and the property of zero-knowledge under the CDC assumption (Stated in Assumption 5.1.1). This answers the RQ1 and RQ2 while highlighting the importance of closing this gap. However, there are concerns about the practicality of using such a composition of zero-knowledge proofs to prove data provenance in machine learning.

From Chapter 5, we see that the proof size grows logarithmically with the size of the training dataset, and proving time grows linearly with the training dataset. Though these complexities are reasonable for smaller datasets and fewer training iterations, these costs can be extremely taxing when we train models that require more iterations or data points for convergence. This highlights the importance of developing IVC schemes, which would reduce the overall proof size and, due to succinctness, faster verification.

Another issue with developing such proofs of training is that the proving and verification steps are rigidly associated with the training steps. Hence, in machine learning, it is essential to understand which steps influence the weights and how to encode them in verification problems to then make use of the Sumcheck or GKR protocols. A limitation of Sumcheck protocols is that the evaluation should be expressed as polynomials. This means that any non-linearity needs to either be approximated using polynomials or substituted with lookup arguments. These operations can substantially increase the computational complexity depending on the degree of the polynomial. With regard to authorization, which involves generating membership and non-membership proofs, the Bank $\mathcal{B}$ must provide proofs for each element in the trainers dataset. Hence, if proving non-membership is not essential, integrating Curve Trees or Curve Forests from Section 3.4 can substantially decrease the Bank's ($\mathcal{B}$) computational costs.

zkMLProv shows that verifiable data provenance in machine learning training is achievable at a meaningful level of cryptographic rigor. The auditor $\mathcal{A}$ receives a single proof π and two public commitments (DataCommitment and PK_D) and can independently certify both that training was honest and that each training sample was bank-authorized. The burden of proof sits on the trainer $\mathcal{T}$, not on $\mathcal{A}$.

The main open problem is the Consistent DataCommitment (CDC) assumption. The MKZG column commitments and the univariate KZG sample commitments are computed from the same matrix but are not cryptographically linked. A dishonest trainer could satisfy both checks using two different datasets. Assumption 5.1.1 in Chapter 5 identifies the cross-commitment opening proof that closes this gap. Implementing it is the most important direction for future work. Until then, the CDC assumption is an explicit and stated trust anchor, not a hidden flaw. The verifier cost remains practical at the production scale, regardless, which is the property that matters most for adoption.

References

[1] Martín Abadi et al. "Deep Learning with Differential Privacy". In: *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, Vienna, Austria, October 24-28, 2016*. Ed. by Edgar R. Weippl et al. ACM, 2016, pp. 308–318. DOI: 10.1145/2976749.2978318. URL: https://doi.org/10.1145/2976749.2978318.

[2] Kasra Abbaszadeh et al. *Zero-Knowledge Proofs of Training for Deep Neural Networks*. Cryptology ePrint Archive, Paper 2024/162. 2024. URL: https://eprint.iacr.org/2024/162.

[3] Kenneth A. Bamberger et al. "Verification Dilemmas, Law, and the Promise of Zero-Knowledge Proofs". In: *Social Science Research Network* (2021). URL: https://api.semanticscholar.org/CorpusID:234905663.

[4] Josh Cohen Benaloh and Michael de Mare. "One-Way Accumulators: A Decentralized Alternative to Digital Sinatures (Extended Abstract)". In: *Advances in Cryptology - EUROCRYPT '93, Workshop on the Theory and Application of of Cryptographic Techniques, Lofthus, Norway, May 23-27, 1993, Proceedings*. Ed. by Tor Helleseth. Vol. 765. Lecture Notes in Computer Science. Springer, 1993, pp. 274–285. DOI: 10.1007/3-540-48285-7_24. URL: https://doi.org/10.1007/3-540-48285-7%5C_24.

[5] Daniel Benarroch et al. "Zero-knowledge proofs for set membership: efficient, succinct, modular". In: *Des. Codes Cryptogr.* 91.11 (2023), pp. 3457–3525. DOI: 10.1007/S10623-023-01245-1. URL: https://doi.org/10.1007/s10623-023-01245-1.

[6] Dan Boneh and Victor Shoup. *A Graduate Course in Applied Cryptography*. Version 0.6, https://crypto.stanford.edu/~dabo/cryptobook/. 2023.

[7] Jonathan Bootle et al. "Gemini: Elastic SNARKs for Diverse Environments". In: *Advances in Cryptology — EUROCRYPT 2022*. Vol. 13276. Lecture Notes in Computer Science. Springer, 2022, pp. 427–457.

[8] Sean Bowe, Ariel Gabizon, and Ian Miers. "Scalable Multi-party Computation for zk-SNARK Parameters in the Random Beacon Model". In: *IACR Cryptol. ePrint Arch.* 2017 (2017), p. 1050. URL: https://api.semanticscholar.org/CorpusID:3636212.

[9] Jan Camenisch and Anna Lysyanskaya. "Dynamic Accumulators and Application to Efficient Revocation of Anonymous Credentials". In: *Advances in Cryptology - CRYPTO 2002, 22nd Annual International Cryptology Conference, Santa Barbara, California, USA, August 18-22, 2002, Proceedings*. Ed. by Moti Yung. Vol. 2442. Lecture Notes in Computer Science. Springer, 2002, pp. 61–76. DOI: 10.1007/3-540-45708-9_5. URL: https://doi.org/10.1007/3-540-45708-9%5C_5.

[10] Matteo Campanelli, Mathias Hall-Andersen, and Simon Holmgaard Kamp. "Curve Forests: Transparent Zero-Knowledge Set Membership with Batching and Strong Security". In: *IACR Cryptol. ePrint Arch.* 2024 (2024), p. 1647. URL: https://eprint.iacr.org/2024/1647.

[11] Matteo Campanelli, Mathias Hall-Andersen, and Simon Holmgaard Kamp. "Curve Trees: Practical and Transparent Zero-Knowledge Accumulators". In: *32nd USENIX Security Symposium, USENIX Security 2023, Anaheim, CA, USA, August 9-11, 2023*. Ed. by Joseph A. Calandrino and Carmela Troncoso. USENIX Association, 2023, pp. 4391–4408. URL: https://www.usenix.org/conference/usenixsecurity23/presentation/campanelli.

[12] Graham Cormode, Justin Thaler, and Ke Yi. "Verifying Computations with Streaming Interactive Proofs". In: *Electronic Colloquium on Computational Complexity - ECCC* 17 (Sept. 2011). DOI: 10.14778/2047485.2047488.

[13] Sankha Das et al. "Communication Efficient Secure and Private Multi-Party Deep Learning". In: *Proc. Priv. Enhancing Technol.* 2025.1 (2025), pp. 169–183. DOI: 10.56553/POPETS-2025-0010. URL: https://doi.org/10.56553/popets-2025-0010.

[14] Electronic Privacy Information Center. *Equifax Data Breach*. EPIC Privacy Archive. Accessed: 2026-06-18. 2017. URL: https://archive.epic.org/privacy/data-breach/equifax/.

[15] European Union. *Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation)*. Accessed: 2025-02-10. 2016. URL: http://data.europa.eu/eli/reg/2016/679/oj.

[16] Francesca Falzon and Evangelia Anna Markatou. "Re-visiting Authorized Private Set Intersection: A New Privacy-Preserving Variant and Two Protocols". In: *Proc. Priv. Enhancing Technol.* 2025.1 (2025), pp. 792–807. DOI: 10.56553/POPETS-2025-0041. URL: https://doi.org/10.56553/popets-2025-0041.

[17] Michael J. Freedman, Kobbi Nissim, and Benny Pinkas. "Efficient Private Matching and Set Intersection". In: *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*. Ed. by Christian Cachin and Jan Camenisch. Vol. 3027. Lecture Notes in Computer Science. Springer, 2004, pp. 1–19. DOI: 10.1007/978-3-540-24676-3_1. URL: https://doi.org/10.1007/978-3-540-24676-3%5C_1.

[18] Sanjam Garg et al. *Experimenting with Zero-Knowledge Proofs of Training*. Cryptology ePrint Archive, Paper 2023/1576. 2023. URL: https://eprint.iacr.org/2023/1576.

[19] O. Goldreich. *Foundations of Cryptography: Volume 1*. Cambridge University Press, 2001.

[20] Oded Goldreich, Silvio Micali, and Avi Wigderson. "Proofs that Yield Nothing But their Validity and a Methodology of Cryptographic Protocol Design (Extended Abstract)". In: *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*. IEEE Computer Society, 1986, pp. 174–187. DOI: 10.1109/SFCS.1986.47. URL: https://doi.org/10.1109/SFCS.1986.47.

[21] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. "Delegating Computation: Interactive Proofs for Muggles". In: *J. ACM* 62.4 (Sept. 2015). ISSN: 0004-5411. DOI: 10.1145/2699436. URL: https://doi.org/10.1145/2699436.

[22] Yuval Ishai et al. "Zero-Knowledge Proofs from Secure Multiparty Computation". In: *SIAM J. Comput.* 39.3 (2009), pp. 1121–1152. DOI: 10.1137/080725398. URL: https://doi.org/10.1137/080725398.

[23] Seny Kamara et al. "Scaling Private Set Intersection to Billion-Element Sets". In: *Financial Cryptography and Data Security - 18th International Conference, FC 2014, Christ Church, Barbados, March 3-7, 2014, Revised Selected Papers*. Ed. by Nicolas Christin and Reihaneh Safavi-Naini. Vol. 8437. Lecture Notes in Computer Science. Springer, 2014, pp. 195–215. DOI: 10.1007/978-3-662-45472-5_13. URL: https://doi.org/10.1007/978-3-662-45472-5%5C_13.

[24] Shaharyar Khan et al. "A Systematic Analysis of the Capital One Data Breach: Critical Lessons Learned". In: *ACM Trans. Priv. Secur.* 26.1 (Nov. 2022). ISSN: 2471-2566. DOI: 10.1145/3546068. URL: https://doi.org/10.1145/3546068.

[25] Brian Knott et al. "CrypTen: Secure Multi-Party Computation Meets Machine Learning". In: *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*. Ed. by Marc'Aurelio Ranzato et al. 2021, pp. 4961–4973. URL: https://proceedings.neurips.cc/paper/2021/hash/2754518221cfbc8d25c13a06a4cb8421-Abstract.html.

[26] Brian Krebs. *First American Financial Corp. Leaked Hundreds of Millions of Title Insurance Records*. Krebs on Security. Accessed: 2026-06-17. May 2019. URL: https://krebsonsecurity.com/2019/05/first-american-financial-corp-leaked-hundreds-of-millions-of-title-insurance-records/.

[27] Carsten Lund et al. "Algebraic Methods for Interactive Proof Systems". In: *Journal of the ACM* 39 (Apr. 1999). DOI: 10.1145/146585.146605.

[28] Pratyush Maini et al. *LLM Dataset Inference: Did you train on my dataset?* 2024. arXiv: 2406.06443 [cs.LG]. URL: https://arxiv.org/abs/2406.06443.

[29] Silvio Micali, Michael O. Rabin, and Joe Kilian. "Zero-Knowledge Sets". In: *44th Annual IEEE Symposium on Foundations of Computer Science (FOCS 2003)*. IEEE Computer Society, 2003, pp. 80–91.

[30] Daniel Morales, Isaac Agudo, and Javier López. "Private set intersection: A systematic literature review". In: *Comput. Sci. Rev.* 49 (2023), p. 100567. DOI: 10.1016/J.COSREV.2023.100567. URL: https://doi.org/10.1016/j.cosrev.2023.100567.

[31] Dana Moshkovitz. "An Alternative Proof of The Schwartz-Zippel Lemma." In: *Electronic Colloquium on Computational Complexity (ECCC)* 17 (Jan. 2010), p. 96.

[32] Mina Namazi, Alexander Nemecek, and Erman Ayday. *ZKPROV: A Zero-Knowledge Approach to Dataset Provenance for Large Language Models*. arXiv:2506.20915. 2025. URL: https://arxiv.org/abs/2506.20915.

[33] Milad Nasr, Reza Shokri, and Amir Houmansadr. "Comprehensive Privacy Analysis of Deep Learning: Passive and Active White-box Inference Attacks against Centralized and Federated Learning". In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, May 2019, pp. 739–753. DOI: 10.1109/sp.2019.00065. URL: http://dx.doi.org/10.1109/SP.2019.00065.

[34] Bita Darvish Rouhani, Huili Chen, and Farinaz Koushanfar. "DeepSigns: An End-to-End Watermarking Framework for Ownership Protection of Deep Neural Networks". In: *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS 2019, Providence, RI, USA, April 13-17, 2019*. Ed. by Iris Bahar et al. ACM, 2019, pp. 485–497. DOI: 10.1145/3297858.3304051. URL: https://doi.org/10.1145/3297858.3304051.

[35] Alexandre Sablayrolles et al. *White-box vs Black-box: Bayes Optimal Strategies for Membership Inference*. 2019. arXiv: 1908.11229 [stat.ML]. URL: https://arxiv.org/abs/1908.11229.

[36] Ahmed Salem et al. "ML-Leaks: Model and Data Independent Membership Inference Attacks and Defenses on Machine Learning Models". In: *arXiv e-prints*, arXiv:1806.01246 (June 2018), arXiv:1806.01246. DOI: 10.48550/arXiv.1806.01246. arXiv: 1806.01246 [cs.CR].

[37] Nojan Sheybani et al. "ZKROWNN: Zero Knowledge Right of Ownership for Neural Networks". In: *60th ACM/IEEE Design Automation Conference, DAC 2023, San Francisco, CA, USA, July 9-13, 2023*. IEEE, 2023, pp. 1–6. DOI: 10.1109/DAC56929.2023.10247798. URL: https://doi.org/10.1109/DAC56929.2023.10247798.

[38] Reza Shokri et al. "Membership Inference Attacks Against Machine Learning Models". In: *2017 IEEE Symposium on Security and Privacy (SP)*. 2017, pp. 3–18. DOI: 10.1109/SP.2017.41.

[39] Justin Thaler. *Proofs, Arguments, and Zero-Knowledge*. 1st ed. Washington D.C.: Georgetown University, 2023.

[40] Yusuke Uchida et al. "Embedding Watermarks into Deep Neural Networks". In: *Proceedings of the 2017 ACM International Conference on Multimedia Retrieval*. 2017, pp. 269–277. DOI: 10.1145/3078971.3078974.

[41] Jelle Vos, Mauro Conti, and Zekeriya Erkin. "SoK: Collusion-resistant Multi-party Private Set Intersections in the Semi-honest Model". In: *IEEE Symposium on Security and Privacy, SP 2024, San Francisco, CA, USA, May 19-23, 2024*. IEEE, 2024, pp. 465–483. DOI: 10.1109/SP54263.2024.00079. URL: https://doi.org/10.1109/SP54263.2024.00079.

[42] Sameer Wagh, Divya Gupta, and Nishanth Chandran. "SecureNN: 3-Party Secure Computation for Neural Network Training". In: *Proc. Priv. Enhancing Technol.* 2019.3 (2019), pp. 26–49. DOI: 10.2478/POPETS-2019-0035. URL: https://doi.org/10.2478/popets-2019-0035.

[43] Zack Whittaker. *Almost 17 Million LoanDepot Customers Had Sensitive Personal Data Stolen in Ransomware Attack*. TechCrunch. Accessed: 2026-06-17. Feb. 2024. URL: https://techcrunch.com/2024/02/26/loandepot-millions-sensitive-personal-data-ransomware.

[44] Hanshen Xiao and Srinivas Devadas. *PAC Privacy: Automatic Privacy Measurement and Control of Data Processing*. arXiv:2210.03458. 2023. URL: https://arxiv.org/abs/2210.03458.

[45] I-Cheng Yeh and Che-hui Lien. "The comparisons of data mining techniques for the predictive accuracy of probability of default of credit card clients". In: *Expert Systems with Applications* 36.2 (2009), pp. 2473–2480.

[46] Samuel Yeom et al. “Privacy Risk in Machine Learning: Analyzing the Connection to Overfitting”. In: *2018 IEEE 31st Computer Security Foundations Symposium (CSF)*. 2018, pp. 268–282. DOI: 10.1109/CSF.2018.00027.

[47] Jie Zhang et al. *Membership Inference Attacks Cannot Prove that a Model Was Trained On Your Data*. 2025. arXiv: 2409.19798 [cs.LG]. URL: https://arxiv.org/abs/2409.19798.

[48] Wenting Zheng et al. “Cerebro: A Platform for Multi-Party Cryptographic Collaborative Learning”. In: *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*. Ed. by Michael D. Bailey and Rachel Greenstadt. USENIX Association, 2021, pp. 2723–2740. URL: https://www.usenix.org/conference/usenixsecurity21/presentation/zheng.

A

Subprotocol Algorithms

This appendix gives pseudocode for the six subprotocols called from the main algorithms in Chapter 4. Each algorithm states where it is called and the corresponding source file in the implementation.

A.1 SampleZeroSumMask

Used in: Algorithm 4 (TrainerProve), called before every forward-pass and backward-pass sumcheck to sample the ZK masking polynomial.

Algorithm 6 SampleZeroSumMask

Require: Batch size N (power of 2)
Ensure: $R \in \mathbb{F}^N$ with $\sum_{i=0}^{N-1} R[i] = 0$
1: **for** $i = 0$ **to** $N - 2$ **do**
2: $\quad R[i] \leftarrow \mathsf{Uniform}(\mathbb{F})$ ▷ sample uniformly from $\mathbb{F}_{2^{61}-1}$
3: **end for**
4: $R[N-1] \leftarrow -\sum_{i=0}^{N-2} R[i] \pmod{p}$ ▷ enforce $\sum R[i] = 0$
5: **return** R

A.2 ZkProductSumcheck.Prove

Used in: Algorithm 4 (TrainerProve), forward pass, to prove $S_t = \sum_i \mathsf{sel}(i) \cdot \mathsf{pred}(i)$ with zero-knowledge. The mask R is committed before this call; the resulting oracle_out is the masked MLE evaluation that the verifier checks against KZG openings.

Algorithm 7 ZkProductSumcheck.Prove

Require: $f_1, f_2, R \in \mathbb{F}^N$ with $\sum_i R[i] = 0$; Fiat-Shamir transcript prefix τ
Ensure: Proof π = (claimed_sum, $\{(g_0^{(k)}, g_1^{(k)}, g_2^{(k)})\}_k$, $\{r_k\}_k$); oracle value out

$v \leftarrow \log_2 N$
claimed_sum $\leftarrow \sum_{i=0}^{N-1} f_1[i] \cdot f_2[i]$
$\tau \leftarrow \tau \parallel$ claimed_sum
$a \leftarrow f_1;\ b \leftarrow f_2;\ r\_arr \leftarrow R$
for $k = 0$ **to** $v - 1$ **do**
    $half \leftarrow |a|/2$
    $g_0 \leftarrow g_1 \leftarrow g_2 \leftarrow 0$
    for $i = 0$ **to** $half - 1$ **do**
        g_0 += $a[i] \cdot b[i] + r\_arr[i]$
        g_1 += $a[half + i] \cdot b[half + i] + r\_arr[half + i]$
        g_2 += $(2a[half + i] - a[i]) \cdot (2b[half + i] - b[i]) + (2r\_arr[half + i] - r\_arr[i])$ ▷ evaluate at $X = 2$ by linear extrapolation
    end for
    Append (g_0, g_1, g_2) to π.rounds
    $\tau \leftarrow \tau \parallel g_0 \parallel g_1 \parallel g_2$
    $r_k \leftarrow H(\tau)$ ▷ Fiat-Shamir challenge
    Fold: $a[i] \leftarrow (1 - r_k) \cdot a[i] + r_k \cdot a[half + i]$ for $i < half$ (similarly b, r_arr)
    $a, b, r\_arr \leftarrow a[0..half - 1],\ b[0..half - 1],\ r\_arr[0..half - 1]$
end for
out $\leftarrow a[0] \cdot b[0] + r\_arr[0]$
return π, out

A.3 ZkTripleProductSumcheck.Prove

Used in: Algorithm 4 (TrainerProve), backward pass, once per feature j to prove $g_j = \sum_i \text{sel}(i) \cdot \text{err}(i) \cdot X_{i,j}$ with zero-knowledge. The degree-3 structure requires four evaluations per round ($X = 0, 1, 2, 3$) rather than three.

Algorithm 8 ZkTripleProductSumcheck.Prove

Require: $f_1, f_2, f_3, R \in \mathbb{F}^N$ with $\sum_i R[i] = 0$; transcript prefix τ
Ensure: Proof π = (claimed_sum, $\{(g_0^{(k)}, g_1^{(k)}, g_2^{(k)}, g_3^{(k)})\}_k$, $\{r_k\}_k$); oracle value out

$v \leftarrow \log_2 N$
claimed_sum $\leftarrow \sum_{i=0}^{N-1} f_1[i] \cdot f_2[i] \cdot f_3[i]$
$\tau \leftarrow \tau \parallel$ claimed_sum
$a \leftarrow f_1;\ b \leftarrow f_2;\ c \leftarrow f_3;\ r\_arr \leftarrow R$
for $k = 0$ **to** $v - 1$ **do**
    $half \leftarrow |a|/2;\quad g_0 \leftarrow g_1 \leftarrow g_2 \leftarrow g_3 \leftarrow 0$
    for $i = 0$ **to** $half - 1$ **do**
        g_0 += $a[i] \cdot b[i] \cdot c[i] + r\_arr[i]$
        g_1 += $a[half + i] \cdot b[half + i] \cdot c[half + i] + r\_arr[half + i]$
        Extrapolate a_2, b_2, c_2 at $X = 2$: $a_2 \leftarrow 2a[half + i] - a[i]$, etc.
        g_2 += $a_2 \cdot b_2 \cdot c_2 + (2r\_arr[half + i] - r\_arr[i])$
        Extrapolate at $X = 3$: $a_3 \leftarrow 3a[half + i] - 2a[i]$, etc.
        g_3 += $a_3 \cdot b_3 \cdot c_3 + (3r\_arr[half + i] - 2r\_arr[i])$
    end for
    Append (g_0, g_1, g_2, g_3) to π.rounds
    $\tau \leftarrow \tau \parallel g_0 \parallel g_1 \parallel g_2 \parallel g_3$
    $r_k \leftarrow H(\tau)$
    Fold: $a[i] \leftarrow (1 - r_k) \cdot a[i] + r_k \cdot a[half + i]$ for $i < half$ (similarly b, c, r_arr)
    $a, b, c, r\_arr \leftarrow$ first-half slices
end for
out $\leftarrow a[0] \cdot b[0] \cdot c[0] + r\_arr[0]$
return π, out

A.4 CommitSampleFP

Used in: The bank's DataCommitment publication step and, identically, the trainer's DataCommitment setup to produce sample_poly_commits$[i]$. The commitment is deterministic: identical inputs produce identical $\mathbb{G}_1$ points, which is the mechanism that allows the bank and trainer to agree on per-sample keys without any communication.

Algorithm 9 CommitSampleFP

Require: Feature vector $\mathbf{x} \in \mathbb{F}^d$, label $y \in \mathbb{F}$, KZG parameters params
Ensure: C_s = KZG.Commit(f_s) $\in \mathbb{G}_1$

values $\leftarrow [x_0, x_1, \ldots, x_{d-1}, y]$ ▷ $d + 1$ field elements
Compute Lagrange basis polynomials L_j over domain $\{0, 1, \ldots, d\}$
$f_s \leftarrow \sum_{j=0}^{d}$ values$[j] \cdot L_j$ ▷ unique degree-d polynomial with $f_s(j) =$ values$[j]$
$C_s \leftarrow$ KZG.Commit(params, f_s) ▷ $= [f_s(\tau)]_1$ via MSM over SRS
return C_s

A.5 CommitmentToZkSetsKey

Used in: Algorithm 5 (AuditorVerify), Step B, to derive the ZK Sets lookup key from the published sample_poly_commits$[i]$. The bank applies the same encoding when building its authorised EDB, so the keys match without any additional negotiation.

Algorithm 10 CommitmentToZkSetsKey

Require: KZG commitment $C_s \in \mathbb{G}_1$ (BLS12-381 affine point)
Ensure: key $\in \{0, 1\}^{384}$ (48-byte string)

key $\leftarrow$ CompressG1(C_s) ▷ 48-byte compressed BLS12-381 $\mathbb{G}_1$ encoding (big-endian)
return key

A.6 ProvePower

Used in: Algorithm 4 (TrainerProve), logistic regression branch, twice per gradient step: once to prove pow2$[i]$ = score$[i]^2$ (called with f_1 = f_2 = score_vec, tgt = pow2_vec) and once to prove pow3$[i]$ = pow2$[i]$ · score$[i]$ (called with f_1 = pow2_vec, f_2 = score_vec, tgt = pow3_vec). These power vectors are needed for the degree-3 sigmoid approximation $P_3(x) = \frac{1}{2} + \frac{x}{4} - \frac{x^3}{48}$.

Algorithm 11 ProvePower

Require: Vectors $f_1, f_2, \mathsf{tgt} \in \mathbb{F}^N$; commitments $C_{f_1}, C_{f_2}, C_{\mathsf{tgt}}$; evaluation point $\mathbf{r} \in \mathbb{F}^v$ ($v = \log_2 N$, Fiat-Shamir derived)
Ensure: PowerProof p

1: eq ← make_eq_vec($\mathbf{r}, N$) ▷ eq$[i] = \widetilde{\mathrm{eq}}(\mathbf{r}, \mathsf{bits}(i))$
2: $R \leftarrow$ SampleZeroSumMask(N) ▷ Algorithm 6
3: p.mask_commit ← MKZG.Commit(R)
4: $\tau \leftarrow$ power_sc_transcript($\mathbf{r}$, p.mask_commit)
5: ($p.\pi_{\mathrm{sc}}$, p.oracle) ← ZkTripleProductSumcheck.Prove(f_1, f_2, eq, R, τ) ▷ Alg. 8; proves $\sum_i f_1[i] \cdot f_2[i] \cdot \mathsf{eq}[i] = \widetilde{\mathsf{tgt}}(\mathbf{r})$
6: $\mathbf{r}' \leftarrow$ to_mkzg_point($p.\pi_{\mathrm{sc}}$.challenges) ▷ sumcheck challenges reinterpreted as MKZG eval point
7: $p.\pi_{f_1} \leftarrow$ MKZG.Open(f_1, $\mathbf{r}'$)
8: $p.\pi_{f_2} \leftarrow$ MKZG.Open(f_2, $\mathbf{r}'$)
9: $p.\pi_R \leftarrow$ MKZG.Open(R, $\mathbf{r}'$)
10: $p.\pi_{\mathsf{tgt}} \leftarrow$ MKZG.Open(tgt, $\mathbf{r}$) ▷ opened at original point $\mathbf{r}$, not $\mathbf{r}'$
11: **return** p

B Batch Independence on Prover Complexity

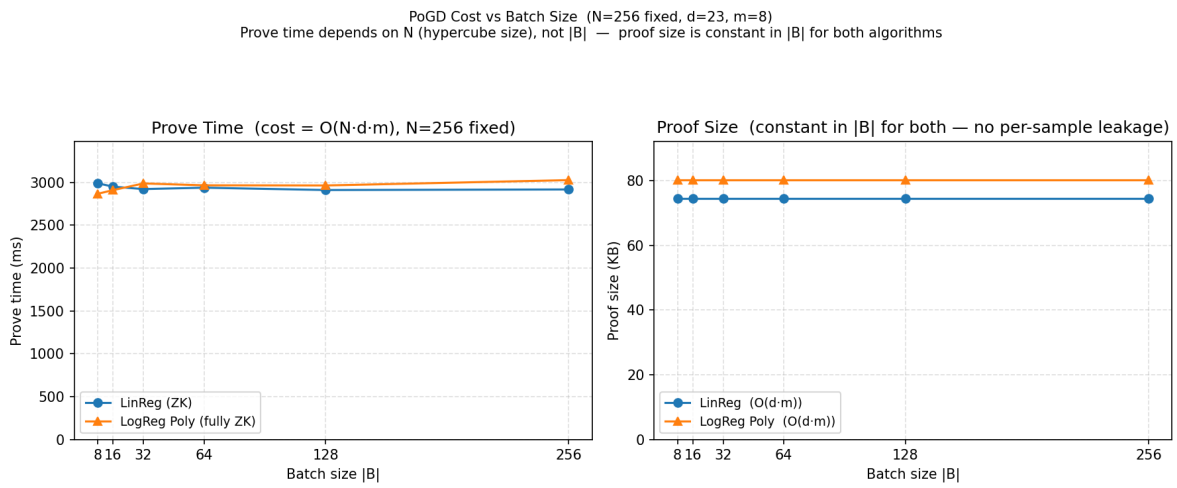


Figure A.1: Prover complexity with respect to Batch Size