Verifiable and Provably Secure Machine Unlearning

Thorsten Eisenhofer*, Doreen Riepel†, Varun Chandrasekaran‡,
Esha Ghosh§, Olga Ohrimenko¶, Nicolas Papernot‖

BIFOLD & TU Berlin*, CISPA Helmholtz Center for Information Security†,
University of Illinois Urbana-Champaign‡, Microsoft Research§,
The University of Melbourne¶, University of Toronto & Vector Institute‖

***Abstract*—Machine unlearning aims to remove points from the training dataset of a machine learning model after training: e.g., when a user requests their data to be deleted. While many unlearning methods have been proposed, none of them enable users to audit the procedure. Furthermore, recent work shows a user is unable to verify whether their data was unlearnt from an inspection of the model parameter alone. Rather than reasoning about parameters, we propose to view *verifiable unlearning* as a security problem. To this end, we present the first cryptographic definition of verifiable unlearning to formally capture the guarantees of an unlearning system. In this framework, the server first computes a proof that the model was trained on a dataset D. Given a user's data point d requested to be deleted, the server updates the model using an unlearning algorithm. It then provides a proof of the correct execution of unlearning *and* that $d \notin D'$, where D' is the new training dataset (i.e., d has been removed). Our framework is generally applicable to different unlearning techniques that we abstract as *admissible functions*. We instantiate a protocol in the framework, based on cryptographic assumptions, using SNARKs and hash chains. Finally, we implement the protocol for three different unlearning techniques and validate its feasibility for linear regression, logistic regression, and neural networks.**

I. Introduction

The right to be forgotten entitles individuals to self-determine the possession of their private data and compel its deletion. In practice, this is now mandated by regulations like the GDPR [1], CCPA [2], or PIPEDA [3]. Consider the case where a company or service provider collects data from its users. These regulations allow users to request a deletion of their data and legally compels the company to fulfil the request. However, this is challenging when the data is used for downstream analyses, *e.g.,* training machine learning (ML) models, where the relationship between model parameters and the data used to obtain them is complex [41]. In particular, ML models are known to memorize information from their training set [18], [13], resulting in attacks on the privacy of training data [15], [43].

Thus, techniques have been introduced for *unlearning*: a trained model is updated to remove the influence a training point had on the model's parameters and predictions [14]. Regardless of the particular approach, existing techniques [67], [12], [34], [27], [30], [8], [54] suffer from one critical limitation: they are unable to *provide the user with a proof that their data was indeed unlearnt.* This is problematic because dishonest service providers may falsify unlearning to avoid paying the large computational costs or to maintain model utility [55], [26].

Additionally, verifying that a point is unlearnt is non-trivial *from the user's perspective*. A primary reason is that users (or third-party auditors) cannot determine whether a data point is unlearnt (or not) by comparing the model's predictions or parameters before and after the claimed unlearning. The complex relationship between training data, models' parameters, and their predictions make it difficult to isolate the effects of any training point. In fact, prior work [58], [61] demonstrates that a model's parameters can be identical when trained with or without a data point. To address these concerns, we propose *a cryptographic approach to verify unlearning*. Rather than trying to verify unlearning by examining changes in the model, we ask the service provider (*i.e.,* the server) to present a cryptographic proof that an *agreed-upon* unlearning process was executed. This leads us to view unlearning as a security problem that we aim to solve with formal guarantees.

In this paper, we propose *the first formal framework* of verifiable machine unlearning. The framework describes the interface of an unlearning protocol in an algorithmic manner and also defines a game-based security notion which allows to prove security of protocol instantiations based on cryptographic assumptions. In order to capture the desired security goals, we find that the definition needs to ensure consistency of data during training and unlearning, *and* across model updates and evolving datasets as it requires a user to be able to verify that their data was not re-added at later stages. Therefore, we formalize unlearning in our framework as an iteration-based protocol; this requires the server to prove that it has honestly updated the model and dataset in each iteration, either due to training with new data or unlearning previously used data. Only then does the user have sufficient guarantees about deletion of their data. Under this definition, we can instantiate protocols using any unlearning technique and any cryptographic primitives that have appropriate security guarantees. We capture the relationship between models and datasets via initialization, training, and unlearning functions with an abstraction that we call *admissible functions*.

In our framework, we identify the following guarantees that need to be satisfied: (a) the model was trained *conceptually* from some dataset (more details in § IV), and (b) the user's

data point is not present in this dataset. Thus, the framework has two major components. First, the server computes a *proof of training* whenever data points are added to the model's training data: this establishes that it trained the model on a particular dataset. Second, when a user submits a request to unlearn a specific data point, the server computes a *proof of unlearning* that proves that the model and its conceptual training set was updated addressing the request. Additionally, the server provides the user with a *proof* that their data point is not part of the updated training set. By linking the executions of those components across iterations, these proofs also ensure that no data point can be added back to the training set *after* it was unlearnt.

We present a fully instantiated protocol in our framework. This instantiation uses SNARK-based verifiable computation as a generic approach for the proof of model updates induced by training or unlearning, and hash chains for the proof of non-membership in a model's training set. Under our security definition, we formally prove the correctness and security of this instantiation generically for any training and unlearning algorithms covered by the abstraction of admissible functions.

Finally, we provide the first implementation of verifiable unlearning based on cryptographic primitives. In particular, we instantiate the SNARK with the Spartan [56] proof system for verifiable training and unlearning. We consider three unlearning techniques: retraining-based unlearning, amnesiac unlearning [30] and optimization-based unlearning [37], [64]. We demonstrate the versatility and scalability of our construction on a variety of binary classification tasks from the PMLB benchmark suite [52], using linear regression, logistic regression, and simple neural networks.

Contributions. We make the following contributions:

- *Formal framework.* We introduce a framework to construct protocols for verifiable machine unlearning. Our framework is designed to be general enough to capture different unlearning algorithms and secure primitives. It models verifiable unlearning as a 2-party protocol (executed between a service provider and its users).
- *Security definition of verifiable machine unlearning.* We then propose a formal security definition of a verifiable machine unlearning scheme. This game-based definition allows to prove the security of unlearning protocols within our framework.
- *Instantiation.* We present a fully instantiated protocol. This construction is based on a generic interface of admissible functions for training and unlearning and thus applicable to any training and unlearning algorithm (as captured by the abstraction).
- *Practical implementation.* Finally, we provide the first implementation of verifiable unlearning based on cryptographic primitives. We study its applicability to three unlearning techniques, different classes of ML models, and benchmark datasets.

II. Background

In this section, we discuss the preliminaries needed to understand the contributions of our work.

Notation. Throughout the paper, let λ denote the security parameter. We call a function negligible in λ—denoted by $\mathsf{negl}(\lambda)$—if it is smaller than the inverse of any polynomial for all large enough values of λ. $[m : n]$ denotes the set $\{m, m+1, ..., n\}$ for integers $m < n$. For $m = 1$, we simply write $[n]$. $y \leftarrow \mathsf{M}(x_1, x_2, \ldots)$ denotes that on input $x_1, x_2, \ldots$, the probabilistic algorithm M returns y. An adversary $\mathcal{A}$ is a probabilistic algorithm, and is *efficient* or Probabilistic Polynomial-Time (PPT) if its run-time is bounded by some polynomial in the length of its input. We will use code-based games, where $\Pr[\mathsf{G} \Rightarrow 1]$ denotes the probability that the final output of game G is 1.

A. Machine Learning Preliminaries

We start with the required background on machine learning and introduce several techniques for unlearning.

Supervised Machine Learning. Supervised machine learning (ML) is the process of learning a parameterized function f_θ (often called a *model*) that is able to predict an output (from the space of outputs $\mathcal{Y}$) given an input (from the space of inputs $\mathcal{X}$), *i.e.,* $f_\theta : \mathcal{X} \to \mathcal{Y}$. Commonly learnt functions include linear regression, logistic regression, and neural networks.

The parameters of this function are typically optimized using methods such as stochastic gradient descent (SGD). Let $\theta_{initial}$ be randomly initialized parameters and $D = \{d_1, \ldots, d_n\}$ a set of training data points, where each $d = (d_x, d_y) \in \mathcal{X} \times \mathcal{Y}$. During training we iteratively update parameters as $\theta' \coloneqq \theta - \eta \nabla_\theta \mathcal{L}(f_\theta(d_x), d_y)$ for points $d \in D$ where $\mathcal{L}$ is a suitably chosen loss function (*e.g.,* cross-entropy loss) and η the *learning rate*.

In SGD, the update is calculated for a randomly chosen input $d \in D$ in each step. In practice, this is often extended to *batches* of data points in order to reduce the variance of each update. Often, multiple passes (called *epochs*) are repeated through the dataset. We can describe the full process of *training* a model m (interchangeably used with θ_m) by

$$\theta_m \coloneqq \theta_{initial} + \sum_{e \in [E]} \sum_{d \in D} \Delta_{e,d}\,,$$

with E being the number of epochs and $\Delta_{e,d}$ the update on the model's parameter from data point d in epoch e.

Machine Unlearning. In machine unlearning the goal is to design algorithms that enable an ML model (specifically, its parameters) to forget the contribution of a (subset of) data point(s). The canonical approach for this is to naively retrain the model from scratch. Hence, removing a data point d^* from a model m with *retraining-based* unlearning can be described as

$$\theta_{m'} \coloneqq \theta_{initial} + \sum_{e \in [E]} \sum_{d \in D \setminus \{d^*\}} \Delta_{e,d}\,.$$

As the resulting model $\theta_{m'}$ is completely devoid of data point d^* (by construction), this is an example for *exact*

unlearning [12], [14], [68], [46], which is desirable but often prohibitively expensive.

More practical unlearning techniques where the contribution of a data point cannot be *completely* removed and the guarantees tolerate some error [34], [16], [60], [8], [54], [27], [30] are commonly referred to as *approximate unlearning*. An example for this is *amnesiac unlearning* [30]. Given a model m, removing a data point d^* with amnesiac unlearning means that we compute

$$\theta_{m'} \coloneqq \theta_m - \sum_{e \in [E]} \Delta_{e,d^*} \,.$$

In other words, to unlearn data point d^*, we remove the updates to the model's parameters that were *directly* computed on that data point for all training epochs E. Yet, amnesiac unlearning only provides approximate guarantees since updates from unlearnt data points indirectly influence updates from later points during the iterative nature of the training [60].

Other approaches for approximate unlearning formulate unlearning as an optimization problem [37], [64] (similar to training). In every step, instead of reducing the loss of a data point, we increase it (*i.e., unlearn* this data point). We refer to this as *optimization-based* unlearning. Formally, we iteratively compute an update Δ_{e,d^*} for the current model that is *subtracted* from its parameters:

$$\theta_{m'} \coloneqq \theta_m - \sum_{e \in [\hat{E}]} \Delta_{e,d^*} \,,$$

where $\hat{E}$ denotes the number of *unlearning epochs* and Δ_{e,d^*} the update from data point d^* in epoch e. For model parameters θ (in epoch e), we define $\Delta_{e,d^*} \coloneqq -\hat{\eta} \nabla_\theta \mathcal{L}(f_\theta(d_x^*), d_y^*)$ with *unlearning rate* $\hat{\eta}$ and loss function $\mathcal{L}$.

B. Cryptographic Preliminaries

We want to provide the user with a cryptographic proof that their data were indeed deleted. Therefore, we require several cryptographic primitives that we introduce next.

Collision-Resistant Hash Functions. A family of hash function $\mathcal{H} : \{0,1\}^\lambda \times \{0,1\}^n \to \{0,1\}^\kappa$ is collision-resistant if it is length-compressing, *i.e.,* $\kappa < n$ and it is hard to find collisions, *i.e.,* for all PPT adversaries $\mathcal{A}$, and for all security parameters λ,

$$\Pr\left[\begin{matrix} k \xleftarrow{\$} \{0,1\}^\lambda; (x_0, x_1) \leftarrow \mathcal{A}(1^\lambda, \mathcal{H}_k) : \\ x_0 \neq x_1 \wedge \mathcal{H}_k(x_0) = \mathcal{H}_k(x_1) \end{matrix}\right] \leq \mathsf{negl}(\lambda) \,.$$

In this work, we will denote Hash as a randomly chosen function from $\mathcal{H}$.

Proof Systems. An interactive proof system describes a protocol between a *prover* and a *verifier*, where the prover wants to convince the verifier that some statement ϕ for a given polynomial time decidable relation R is true. The prover holds a witness ω for the statement. In this work we are concerned with *non-interactive* proof systems. A Succinct Non-Interactive Argument of Knowledge (SNARK) allows the prover to non-interactively prove the statement with a short (or succinct) cryptographic proof which can be verified in time sublinear in the size of the statement. We denote a SNARK by Π and define it by the three algorithms $(\Pi.\mathsf{Setup}, \Pi.\mathsf{Prove}, \Pi.\mathsf{Vrfy})$:

- $\mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R)$: The setup algorithm outputs public parameters pp for a polynomial-time decidable relation R.
- $\pi \leftarrow \Pi.\mathsf{Prove}(R, \mathsf{pp}, \phi, \omega)$: The prover algorithm takes as input pp and $(\phi, \omega) \in R$ and returns an argument π, where ϕ is termed the statement and ω the witness.
- $b \leftarrow \Pi.\mathsf{Vrfy}(R, \mathsf{pp}, \phi, \pi)$: The verification algorithm takes pp, a statement ϕ and an argument π and returns a bit b, where $b = 1$ indicates success and $b = 0$ indicates failure.

Perfect Completeness. Given any true statement, an honest prover should be able to convince an honest verifier. More formally, let $\mathcal{R}$ be a sequence of families of efficiently decidable relations R. For all $R \in \mathcal{R}$ and $(\phi, \omega) \in R$

$$\Pr\left[\Pi.\mathsf{Vrfy}(R, \mathsf{pp}, \phi, \pi) \,\middle|\, \begin{matrix} \mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R); \\ \pi \leftarrow \Pi.\mathsf{Prove}(R, \mathsf{pp}, \phi, \omega) \end{matrix}\right] = 1 \,.$$

Computational Soundness. We say that Π is sound if it is not possible to prove a false statement. Let L_R be the language consisting of statements for which there exists corresponding witnesses in R. For a relation $R \sim \mathcal{R}$, we require that for all non-uniform PPT adversaries $\mathcal{A}$

$$\Pr\left[\begin{matrix} \phi \notin L_R \textbf{ and} \\ \Pi.\mathsf{Vrfy}(R, \mathsf{pp}, \phi, \pi) \end{matrix} \,\middle|\, \begin{matrix} \mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R); \\ (\phi, \pi) \leftarrow \mathcal{A}(R, \mathsf{pp}) \end{matrix}\right] \leq \mathsf{negl}(\lambda) \,.$$

We further define the notion of witness extractability or knowledge soundness.

Computational Knowledge Soundness. Π satisfies computational knowledge soundness if there exists an extractor that can compute a witness whenever the adversary produces a valid argument. Formally, for a relation $R \sim \mathcal{R}$, we require that for all non-uniform PPT adversaries $\mathcal{A}$ there exists a non-uniform PPT extractor $\mathcal{E}$ such that

$$\Pr\left[\begin{matrix} (\phi, \omega) \notin R \textbf{ and} \\ \Pi.\mathsf{Vrfy}(R, \mathsf{pp}, \phi, \pi) \end{matrix} \,\middle|\, \begin{matrix} \mathsf{pp} \leftarrow \Pi.\mathsf{Setup}(1^\lambda, R); \\ ((\phi, \pi); \omega) \leftarrow (\mathcal{A} \| \mathcal{E})(R, \mathsf{pp}) \end{matrix}\right] \leq \mathsf{negl}(\lambda) \,.$$

A SNARK Π is secure if it satisfies perfect completeness, computational soundness and knowledge soundness.

III. Verifiable Machine Unlearning

We consider the following ecosystem: there are many *users* $\mathcal{U}$, each of whom has access to a set of data points (or dataset). They share their data with a *server* S which uses it to learn an ML model. Users can send requests to either delete or add new data. We assume the server is malicious and may not faithfully execute unlearning requests. This may be because the server is unwilling to tolerate model performance degradation after data deletion [55], [26], or pay the computational penalty associated with model updating [12],

[30]. Our goal is to develop a method to verify whether the server is adhering to users' requests.

Before presenting our framework, we will first review conceptually simpler constructions. Specifically, we will discuss three naive approaches **A1-A3**, and explain why they are insufficient. From this, we will derive necessary criteria **D1-D3** that form the basis for our framework.

A1: Proof via Model Parameters. As a first step towards verifiable unlearning, assume the server holds a dataset on which it trained an ML model. To prove that it has unlearnt a specific data point, the server may provide the user with the trained model parameters (including random seeds) and the entire dataset. The user could then locally retrain the model and compare the resulting parameters with the server's, or apply influence techniques [41] to assess whether their data point contributes to the model's parameters. However, both of these methods suffer from a fundamental problem: it is possible to arrive at the same model parameters even if data was deleted. For example, recent work by Shumailov et al. [58] and Thudi et al. [61] describe how a user's contribution (towards model parameters) can be approximated from other entries in a dataset, rendering such approaches insufficient: the server can claim to have obtained the exact same model parameters from a number of different datasets.

A2: One-Shot Verified Unlearning. To account for this, we could extend **A1** as follows: Instead of providing the user the model parameters and dataset only *after* their data is (claimed to be) unlearnt, we require the server to *prove* that it has executed a pre-specified unlearning algorithm. At this point, we do not want to delve into the specifics of what such a proof would entail, but one could envision verified computation techniques such as cryptographic methods or trusted-execution environments. Instead, we want to point out some assumptions that are implicitly made; namely, we have not specified how the initial model (to which the unlearning algorithm is applied) was trained. For instance, we might assume that the model was truthfully trained which is commonly assumed when evaluating the efficacy of unlearning algorithms [12], [30], [64]. However, since we are considering a scenario where the server may behave maliciously during the unlearning process, we can not automatically assume that the training was conducted honestly either.

A3: A Naive Iteration-based Protocol. Therefore, we also need to require the server not only to prove the execution of the unlearning algorithm but also the training process itself. As a result, we aim to capture the *evolution* of an ML model across multiple iterations of training and unlearning. For instance, after one user asked to unlearn their data point, another user might share new data with the server. However, this approach remains insufficient: a server could prove in one iteration that it has deleted the first user's data, but then, in the next training iteration, it might simply reintroduce the same data point. This needs to be captured and a protocol needs additional mechanisms to avoid such behavior.

Desiderata. From the discussion thus far, we unearth three main requirements to achieve our goal of verifiable unlearning.

D1. The user must be able to verify that the unlearning algorithm was correctly executed and that the requested data point has been removed.

D2. The user must be able to verify the model's evolution, including the execution of training algorithms.

D3. The system must ensure that the server cannot reintroduce previously unlearned data (unless explicitly requested to do so).

IV. Our Framework

We now present our formal framework for verifiable machine unlearning encompassing the requirements outlined in the previous section. This framework defines verifiable unlearning as an interactive protocol summarized in Figure 1.

Dataset. Let $\mathcal{D}$ be the distribution of data points. Each user $u \in \mathcal{U}$ possesses a set of data points $\widehat{D}_u \sim \mathcal{D}$. At the server side, there is an initially empty "server's dataset" $D_0 = \emptyset$ (which is later populated for training an ML model). During the execution of the protocol, users can request to add or delete data points which the server adds respectively deletes from their dataset. Different versions of the resulting server's dataset (as it develops during the execution of the protocol) are denoted by their corresponding index (*i.e.,* $D_0, D_1, \dots$). To attribute data points to users, we assume the server prepends a unique identifier to each data point. Therefore, entries in D_i are tuples of the form $(u, d) \in \mathcal{U} \times \widehat{D}_u$. We refer to such a unique representation as a *data record*.

Admissible Functions. Our approach relies on proving the execution of a *pre-specified unlearning algorithm*. To capture formally that we cannot consider training and unlearning in isolation, we propose a generic interface that we call *admissible functions*. One instance is described by a triplet of functions $f = (f_I, f_T, f_U)$, where f_I is an initialization function, f_T is a training function, and f_U is an associated unlearning function. The set of all admissible functions is denoted by $\mathcal{F}$. We let pp_f denote public hyperparameter which are used for initialization. W.l.o.g., we assume the functions to be deterministic. Randomness required, *e.g.,* for the training, can be obtained deterministically with a pseudo-random number generator and a suitable seed contained in the hyperparameter and stored in the state. More explicitly:

- $(\mathsf{st}_f, m) \coloneqq f_I(\mathsf{pp}_f)$: The initialization function f_I takes as input hyperparameter pp_f and outputs an initial state st_f and model m (*e.g.,* its initial weights).
- $(\mathsf{st}_f, m) \coloneqq f_T(\mathsf{st}_f, D^+)$: The training function f_T takes as input the current state st_f and the set D^+ of data records to be added. It outputs an updated state and a new model.
- $(\mathsf{st}_f, m) \coloneqq f_U(\mathsf{st}_f, U^+)$: The unlearning function f_U takes as input the current state st_f and the set U^+ of data records to be deleted. It outputs an updated state and a new model.

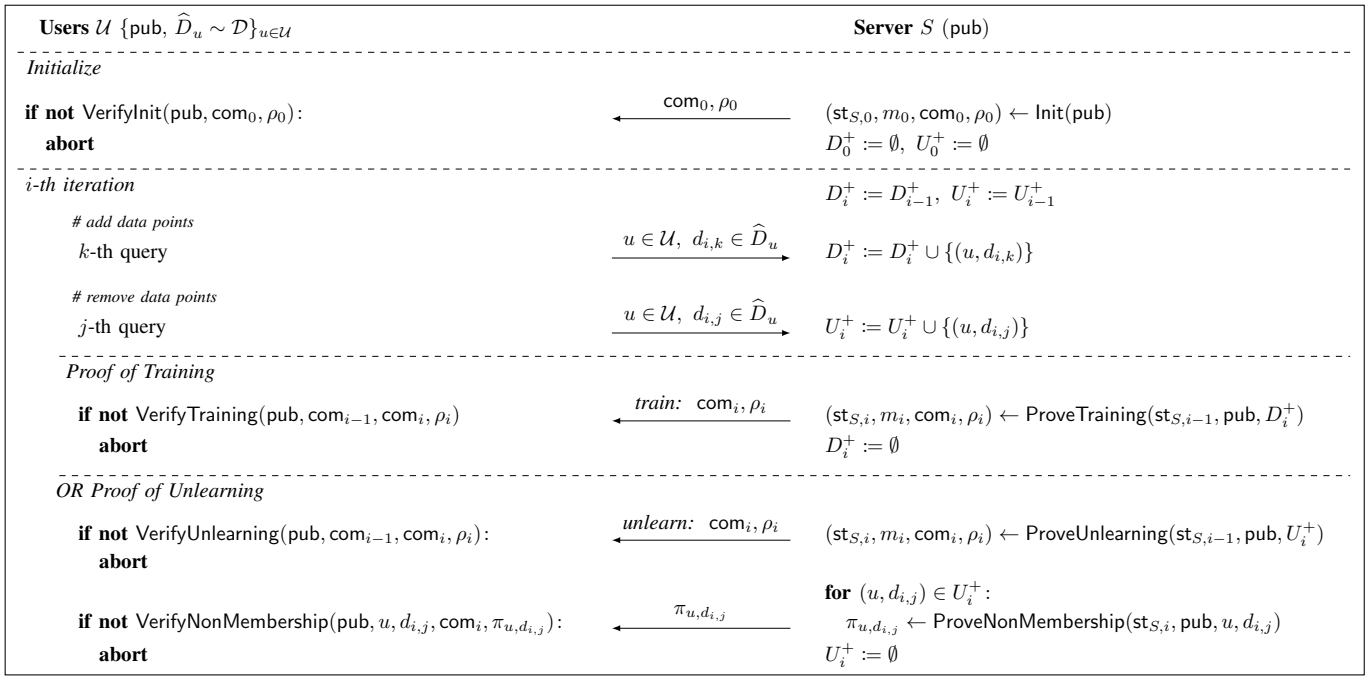


Fig. 1: *Unlearning Framework.* We describe protocols in this framework based on a set of admissible functions f. After initialization, execution proceeds in iterations. In the beginning of each iteration i, users $\mathcal{U}$ can issue requests for data to be added or deleted. After this phase, the server S either performs a *proof of training* by adding the requested data records in D_i^+ to the model or a *proof of unlearning* by removing the requested data records in U_i^+. It computes a commitment com_i on the updated model m_i and updated training dataset. Furthermore, the server computes a proof ρ_i that m_i was obtained from this dataset. The users verify this proof and the commitment. In each iteration of unlearning the server additionally creates a *proof of non-membership* for every unlearnt data point conforming to a user that it has complied with a data deletion request. This proof can be verified by the user against com_i.

These functions allow to establish an abstraction to track the relation between a model and its underlying dataset; we refer to this as the *conceptual* dataset. If D is the current conceptual dataset, removing data records from U^+ with f_U updates the dataset as $D := D \setminus U^+$. Before executing the protocol, the server and users must agree on f, similar to the agreement process in the TLS handshake protocol. This agreement must ensure that f is semantically meaningful and relevant to the context of the application. Just as in the TLS protocol, where the security of the entire protocol depends on selecting a secure cipher suite, the selection of f is crucial for ensuring the integrity of the process.

Protocol Execution. We denote a protocol in our framework by Φ_f for a triplet of functions $f = (f_I, f_T, f_U) \in \mathcal{F}$. The execution of Φ_f can then be described with two phases (executed in an iterative manner):

- **P1. Data Addition/Deletion:** At the beginning of each iteration, user can issue addition/deletion requests to the server. The server batches multiple addition/deletion requests by storing all requests in intermediate datasets D_i^+ (addition) and U_i^+ (deletion).
- **P2. Proof of Training (resp. Unlearning):** At the end of an iteration i, the server updates its dataset by adding (resp. deleting) the data records stored in D_i^+ (resp. U_i^+). For training (resp. unlearning), the server updates the model using function f_T (resp. f_U) on all records requested to be added (resp. deleted). It then computes a proof of training (resp. unlearning) to be verified by the users. These proofs establishes the state of the evolving dataset across iterations and model updates *and* that unlearnt data records can not be re-added. Based on the current state, the server then provides each user who requested a point to be deleted with an individual proof that their data record is not part of the dataset (*i.e.,* a proof of non-membership). Finally, at the end of an iteration, the dataset D_i^+ (resp. U_i^+) is reset.

A. Protocol Syntax

An unlearning protocol Φ_f w.r.t. admissible functions f specifies nine algorithms that are executed by the server and users as shown in Figure 1.

1. Setup and Initialization. A global setup procedure generates public parameters pub, *i.e.,* $\mathsf{pub} \leftarrow \mathsf{Setup}(1^\lambda)$, where λ is the security parameter. We assume that pub additionally include functions f with hyperparameters pp_f. Depending on the application, this procedure can be executed either by the server or some external entity. Subsequently, pub is given to all actors.

During initialization, the ML model and protocol state are initialized using algorithms Init and $\mathsf{VerifyInit}$. Intuitively, the state captures all information required by the protocol (*e.g.,* information to keep track of the evolving dataset) as well as information required by f for training and unlearning. For example, for retraining-based unlearning the state would contain the training dataset D, while for amnesiac unlearning, the set of deltas Δ_{e,d^*} (cf. Section II-A) as well as current model m would be included. Formally, the following two algorithms are run:

Server: $(\mathsf{st}_{S,0}, m_0, \mathsf{com}_0, \rho_0) \leftarrow \mathsf{Init}(\mathsf{pub})$

The server initializes m_0 using initialization function f_I and hyperparameter pp_f contained in pub. It stores the resulting state st_f in $\mathsf{st}_{S,0}$. The set of training records is initialized empty, *i.e.,* $D_0 \coloneqq \emptyset$. It then commits to m_0 and D_0 with $\mathsf{com}_0 \coloneqq (\mathsf{com}_0^m \| \mathsf{com}_0^D)$. We assume that the commitment to the initial dataset (and all its updates) is computed deterministically from pub using a function Commit, *i.e.,* $\mathsf{com}_0^D \coloneqq \mathsf{Commit}(\mathsf{pub}, D_0)$. We do not make additional assumptions about the commitment at this point, but we will later see that it needs to be *binding*. Finally, proof ρ_0 attests the initialization of m_0.

User: $0/1 \leftarrow \mathsf{VerifyInit}(\mathsf{pub}, \mathsf{com}_0, \rho_0)$

Users verify two things: (a) the commitment com_0 with $D_0 = \emptyset$, and (b) the model initialization m_0 against proof ρ_0. If verification is successful, the algorithm outputs 1. On failure, it outputs 0.

2A. Proof of Training. In each iteration i, where the server performs a proof of training, two algorithms are run:

Server: $(\mathsf{st}_{S,i}, m_i, \mathsf{com}_i, \rho_i) \leftarrow \mathsf{ProveTraining}(\mathsf{st}_{S,i-1}, \mathsf{pub}, D_i^+)$

The server computes the updated model m_i by executing training function f_T on state $\mathsf{st}_{S,i-1}$ and all newly added data records D_i^+. The training set of m_i is defined as the union of the previous dataset and new data records, *i.e.,* $D_i \coloneqq D_{i-1} \cup D_i^+$. The server commits to both the model and training data with $\mathsf{com}_i \coloneqq (\mathsf{com}_i^m \| \mathsf{com}_i^D)$ and computes the proof ρ_i that (a) model m_i was updated by applying f_T, and (b) training data D_i does not contain any unlearnt record, *i.e.,* $D_i \cap U_i = \emptyset$, where $U_i \coloneqq \bigcup_{k \in [i]} U_k^+$ is the set of all unlearnt data records so far. The proof also attests that (c) the set of unlearnt data records has not changed, *i.e.,* $U_{i-1} = U_i$.

User: $0/1 \leftarrow \mathsf{VerifyTraining}(\mathsf{pub}, \mathsf{com}_{i-1}, \mathsf{com}_i, \rho_i)$

Users validate properties (a)-(c) (as described above in $\mathsf{ProveTraining}$) and the update on commitment com_i by verifying the proof ρ_i against the previous commitment com_{i-1} and the new commitment com_i.

2B. Proof of Unlearning. In each iteration i, where the server performs a proof of unlearning, four algorithms are run:

Server: $(\mathsf{st}_{S,i}, m_i, \mathsf{com}_i, \rho_i) \leftarrow \mathsf{ProveUnlearning}(\mathsf{st}_{S,i-1}, \mathsf{pub}, U_i^+)$

The server unlearns all records collected in U_i^+ and computes the updated m_i by executing function f_U on state $\mathsf{st}_{S,i-1}$ and U_i^+. Thus, conceptually, the new training set of m_i is defined as $D_i \coloneqq D_{i-1} \setminus U_i^+$. Similar to $\mathsf{ProveTraining}$, the server commits to both the model and training data with com_i and computes the proof ρ_i that (a) model m_i was updated by applying f_U, and (b) training data D_i does not contain any unlearnt records, *i.e.,* $D_i \cap U_i = \emptyset$, where $U_i \coloneqq U_{i-1} \cup U_i^+$. The proof also attests that (c) the previous set of unlearnt data records is a subset of the updated set $U_{i-1} \subset U_i$. This ensures that the set of unlearnt data records is append-only and records can never be removed.

User: $0/1 \leftarrow \mathsf{VerifyUnlearning}(\mathsf{pub}, \mathsf{com}_{i-1}, \mathsf{com}_i, \rho_i)$

Users validate properties (a)-(c) (as described above in $\mathsf{ProveUnlearning}$) and the update on commitment com_i by verifying the proof ρ_i against the previous commitment com_{i-1} and the new commitment com_i.

Server: $\pi_{u,d_{i,j}} \leftarrow \mathsf{ProveNonMembership}(\mathsf{st}_{S,i}, \mathsf{pub}, u, d_{i,j})$

For each record $(u, d_{i,j}) \in U_i^+$, the server computes a proof $\pi_{u,d_{i,j}}$ using information from $\mathsf{st}_{S,i}$ that this record is not part of the training set m_i, *i.e.,* $(u, d_{i,j}) \notin D_i$.

User: $0/1 \leftarrow \mathsf{VerifyNonMembership}(\mathsf{pub}, u, d_{i,j}, \mathsf{com}_i, \pi_{u,d_{i,j}})$

The user verifies with both $\pi_{u,d_{i,j}}$ and com_i that $(u, d_{i,j})$ was not part of the training data D_i of m_i.

Practical considerations. The framework ensures that once data records are deleted, they cannot be re-added later. Therefore, it suffices if a majority of honest users verify the updates. Even if a user stops participating after confirming their data has been deleted, the verification of updates by the honest majority ensures correct server behavior. In practice, an additional mechanism will be needed for users to report invalid proofs and trigger penalties for the server.

If users trust a third party (such as an auditor), the verification algorithms $\mathsf{VerifyTraining}$ and $\mathsf{VerifyUnlearning}$, can be executed by this entity to minimize redundant computations. The results can then be shared with all users (refer to Section VII-A for further discussion).

Furthermore, it is important to ensure that the server uses the most up-to-date model for inference. This can be achieved with techniques for verifiable inference [42], [44], [65], [19], [38], which is related to, but independent of the problem of unlearning that we consider in this work.

B. Completeness and Security

Within the proposed framework, we can formally describe completeness and security requirements of protocols for verifiable unlearning.

Completeness. For completeness, we require that an honest execution of the protocol yields the expected outputs: for an honest server, the users successfully verify the initialization of the model and the proofs for all updates (training and unlearning) performed by the server. A proof of non-membership

generated for an unlearnt data record is also successfully verified by the corresponding user. We formally capture this with the following definition.

Definition 1 (Completeness)**.** *Let λ be the security parameter. A protocol Φ_f is complete if for all* $\mathsf{pub} \leftarrow \mathsf{Setup}(1^\lambda)$*, the following properties are satisfied:*

1) *Let* $(\mathsf{st}_{S,0}, m_0, \mathsf{com}_0, \rho_0) \leftarrow \mathsf{Init}(\mathsf{pub})$*. Then*

$$\Pr[\,\mathsf{VerifyInit}(\mathsf{pub}, \mathsf{com}_0, \rho_0) = 0\,] \leq \mathsf{negl}(\lambda)\ .$$

2) *Let $mode_i \in \{train, unlearn\}$ indicate whether proof of training or proof of unlearning has been performed in iteration i. Let $\mathcal{A}$ be a PPT adversary that outputs a valid sequence of datasets either to be added $\{train\colon\ D_i^+\}$ or to be deleted $\{unlearn\colon\ U_i^+\}$ for all $i \in [\ell]$.*

   For all $i \in [\ell]$, if $mode_i = train$, let $(\mathsf{st}_{S,i}, m_i, \mathsf{com}_i, \rho_i) \leftarrow \mathsf{ProveTraining}(\mathsf{st}_{S,i-1}, \mathsf{pub}, D_i^+)$ *and if $mode_i = unlearn$, let* $(\mathsf{st}_{S,i}, m_i, \mathsf{com}_i, \rho_i) \leftarrow \mathsf{ProveUnlearning}(\mathsf{st}_{S,i-1}, \mathsf{pub}, U_i^+)$.

   Then for all $mode_i = train$:

$$\Pr[\mathsf{VerifyTraining}(\mathsf{pub}, \mathsf{com}_{i-1}, \mathsf{com}_i, \rho_i) = 0] \leq \mathsf{negl}(\lambda)\ ,$$

   and for all $mode_i = unlearn$:

$$\Pr[\mathsf{VerifyUnlearning}(\mathsf{pub}, \mathsf{com}_{i-1}, \mathsf{com}_i, \rho_i) = 0] \leq \mathsf{negl}(\lambda)\ ,$$

   where validity is defined via the following conditions: $\forall i, j$ s.t. $i \neq j\colon D_i^+ \cap D_j^+ = \emptyset$ and $\forall i, j$ s.t. $j < i\colon D_i^+ \cap U_j^+ = \emptyset$.

3) *For all $i \in [\ell]$ s.t. $mode_i = unlearn$: for all $(u, d) \in U_i^+$, let* $\pi_{u,d} \leftarrow \mathsf{ProveNonMembership}(\mathsf{st}_{S,i}, \mathsf{pub}, u, d)$*, then*

$$\Pr[\mathsf{VerifyNonMembership}(\mathsf{pub}, u, d, \mathsf{com}_i, \pi_{u,d}) = 0] \leq \mathsf{negl}(\lambda)\,.$$

In this definition, we only require *computational* completeness to allow for a wide range of instantiations. For example, an instantiation that works on hash values of data records cannot achieve perfect completeness because of hash collisions. By allowing for computational completeness, however, we only require that it should be hard for a PPT adversary to find such collisions (*i.e.,* with a negligible probability).

Security. The security of an unlearning protocol can be modelled in terms of a security game. In this game, the adversary, described by a probabilistic algorithm $\mathcal{A}$, takes the role of the server. Intuitively, the definition need to capture that a malicious server cannot add (and train on) a data record that a user requested to delete in a previous iteration. Formally this is modeled as the winning condition in the security game in Figure 2. To this end, we define an *extractability-based* security model; this allows to cover realistic attackers that only need to output valid transcripts of the interactions, while the extractability property ensures that the server must know some underlying dataset for these transcripts. Note that similar security definition are commonly used in the context of hash functions or SNARKs [10], [20], [31].

The adversary in our game has to provide the protocol outputs (*i.e.,* the commitments and proofs), whereas the extractor outputs the corresponding inputs that the adversary used (*i.e.,* the underlying datasets). We give a formal description of game $\mathsf{GameUnlearn}$ in Figure 2, which is divided into two stages:

S1. Simulation. The game draws the public parameters pub using Setup and runs the adversary $\mathcal{A}$ on input pub. The extractor $\mathcal{E}$ is run on the same input and random coins. Additionally, we provide auxiliary input aux which captures any extra information the adversary may have whenever the protocol is used in combination with other cryptographic schemes and allows for a wide range of possible instantiations. As commonly done, we restrict this input to only *benign* inputs [11], [9], e.g., we do not allow the auxiliary input to encode an arbitrary (possibly obfuscated) circuit. At some point, $\mathcal{A}$ will terminate and output a sequence of tuples $(k, (u, d), \pi_{u,d}, \{mode_i\colon \mathsf{com}_i, \rho_i\}_{i\in[0,\ell]})$ for some $\ell \in \mathbb{N}$, where (u, d) is a data record that was proved to be deleted in the k-th iteration, and $mode_i \in \{train, unlearn\}$. At the same time, the extractor outputs a sequence of datasets $(D_0, \ldots, D_\ell)$.

S2. Finalize. After the adversary has terminated, the game uses the extractor's output to compute the set of data records unlearnt in the k-th iteration based on the datasets D_k and D_{k-1}. Recall that the commitment in the framework consists of two parts com_i^m and com_i^D, where we need the second part for verification. The game checks for the following conditions: (a) com_i^D was obtained from D_i, (b) the initial proof ρ_0 verifies for the initial commitment com_0, (c) each proof of training ρ_i verifies for commitments com_{i-1} and com_i, (d) each proof of unlearning ρ_i verifies for commitments com_{i-1} and com_i, (e) the proof of non-membership $\pi_{u,d}$ verifies for (u, d) and com_k, (f) $k < \ell$ and (u, d) was unlearnt in iteration k and re-added in iteration ℓ. If all these properties are satisfied, then the game outputs 1 and $\mathcal{A}$ wins.

We summarize this with the following definition.

Definition 2 (Unlearning)**.** *Let λ be the security parameter and consider game* $\mathsf{GameUnlearn}$ *in Figure 2. Protocol Φ_f for data distribution $\mathcal{D}$ is unlearning-secure if for all PPT adversaries $\mathcal{A}$ there exists an extractor $\mathcal{E}$ such that for all benign auxiliary inputs* aux*:*

$$\Pr[\mathsf{GameUnlearn}_{\mathcal{A},\mathcal{E},\Phi_f,\mathcal{D}}(1^\lambda) \Rightarrow 1] \leq \mathsf{negl}(\lambda)\,.$$

V. Instantiation

Our framework defines a generic interface for constructing protocols for verifiable unlearning. To instantiate a protocol within this framework, we need to address two main challenges. First, we need to be able to verify the correct execution of the training and unlearning algorithms to validate any changes to the dataset. Second, we require a mechanism to keep track of the dataset across iterations, enabling queries to verify (non-)membership of specific data records.

In this section, we introduce the core components to account for these challenges and present a practical instantiation of

$\mathsf{GameUnlearn}_{\mathcal{A},\mathcal{E},\Phi_f,\mathcal{D}}(1^\lambda)$

$\mathsf{pub} \leftarrow \mathsf{Setup}(1^\lambda)$

$(k,(u,d),\pi_{u,d},\{mode_i\colon \mathsf{com}_i,\rho_i\}_{i\in[0:\ell]};$
$\{D_i\}_{i\in[0:\ell]}) \leftarrow (\mathcal{A}\|\mathcal{E})(\mathsf{pub},\mathsf{aux})$

 # Pre-processing

$U_k^+ \coloneqq D_{k-1} \setminus D_k$

Parse com_i as $(\mathsf{com}_i^m \| \mathsf{com}_i^D)$ $\forall i \in [0:\ell]$

 # Evaluate winning condition

if $\mathsf{Commit}(\mathsf{pub}, D_i) = \mathsf{com}_i^D$ $\forall i \in [0:\ell]$ *# Datasets*

and $\mathsf{VerifyInit}(\mathsf{pub}, \mathsf{com}_0, \rho_0)$ *# Initialization*

and $\mathsf{VerifyTraining}(\mathsf{pub}, \mathsf{com}_{i-1}, \mathsf{com}_i, \rho_i)$
$\forall i : mode_i = train$ *# Training*

and $\mathsf{VerifyUnlearning}(\mathsf{pub}, \mathsf{com}_{i-1}, \mathsf{com}_i, \rho_i)$
$\forall i : mode_i = unlearn$ *# Unlearning*

and $\mathsf{VerifyNonMembership}(\mathsf{pub}, u, d, \mathsf{com}_k, \pi_{u,d})$ *# Non-Membership*

and $(u,d) \in U_k^+$ *# Point unlearnt*

and $(u,d) \in D_\ell$ **and** $k < \ell$: *# Point re-added later*

return 1

return 0

Fig. 2: *Security Game.* We define the security of an protocol Φ_f in terms of game $\mathsf{GameUnlearn}$. The notation $(\mathcal{A}\|\mathcal{E})$ denotes that both algorithms are run on the same input and random coins and assigning their results to variables before resp. after the semicolon. Input aux refers to auxiliary input.

an unlearning protocol based on SNARKs and hash functions. Our instantiation is generic and we prove its completeness and security universally for any triplet (f_I, f_T, f_U) of admissible functions. An overview of the full protocol is depicted in Appendix A.

Data Representation. To represent the dataset, we split all data records as those belonging to either training data D or unlearnt data U. For our instantiation, the server stores two ordered sets $\mathcal{H}_D$ and $\mathcal{H}_U$ of hashed training data records and unlearnt data records. From both sets, we additionally compute a hash value in the form of a hash chain (cf. $\mathsf{HashData}$ in Appendix A). This representation allows for efficient caching of intermediate hashes and, for $\mathcal{H}_U$, enables us to easily prove that entries are append-only (i.e., prevent records from being removed from the chain) as well as fast membership verification for unlearnt data records. To account for the partition of training and unlearnt data as well as the user admissible function, we instantiate the commitment com as a tuple of four elements: hash of (a) the state h_{st_f} (defined by f), (b) the model h_m, (c) the training data h_D, and (d) the unlearnt data h_U. Looking ahead, a collision-resistant hash function is sufficient for the binding property; it ensures that the adversary cannot come up with a second input that has the same hash value.

Proof System. To verify the correct execution of f_I, f_T, and f_U, we use proof systems. More specifically, SNARKs, which allow (broadly speaking) to prove statements of the form that an output y is the result of applying a function f on an input x, i.e., $y \coloneqq f(x)$. Therefore, we define the verification of the initialization, training updates and unlearning updates in terms of polynomial decidable binary relations R_I, R_T and R_U over circuits C_I, C_T and C_U (resp.) as introduced in Section II-B. These circuits describe the required computations—based on f_I, f_T, and f_U. We exemplary outline C_U in Figure 3; for circuit C_I, C_T refer to Appendix B. Note that for verification, only public parameters are required. Furthermore, by using SNARKs, we can keep the instantiation generic and universally prove its completeness and security for any triplet (f_I, f_T, f_U).

$C_U(\textbf{public}\ h_{\mathsf{st}_{f,i}}, h_{\mathsf{st}_{f,i-1}}, h_{m_i}, h_{D_i}, h_{D_{i-1}}, h_{U_i}, h_{U_{i-1}},$
$\textbf{private}\ \mathsf{st}_{f,i-1}, \mathcal{H}_{D_{i-1}}, U_i^+)$

 # Check input set of hashed training data records

if $h_{D_{i-1}} \neq \mathsf{HashData}(\mathcal{H}_{D_{i-1}})$:

  return false

 # Update and check set of hashed unlearnt data records and training data records

$\mathcal{H}_{U_i^+} \coloneqq \{\mathsf{HashDataRecord}(u,d)\}_{(u,d)\in U_i^+}$

$\mathcal{H}_{D_i} \coloneqq \mathcal{H}_{D_{i-1}} \setminus \mathcal{H}_{U_i^+}$

if $h_{U_i} \neq \mathsf{AppendHashData}(h_{U_{i-1}}, \mathcal{H}_{U_i^+})$ **or** $h_{D_i} \neq \mathsf{HashData}(\mathcal{H}_{D_i})$:

  return false

 # Check input state, perform unlearning and check outputs

if $h_{\mathsf{st}_{f,i-1}} \neq \mathsf{HashState}(\mathsf{st}_{f,i-1})$:

  return false

$(\mathsf{st}_{f,i}, m_i) \coloneqq f_U(\mathsf{st}_{f,i-1}, U_i^+)$

if $h_{\mathsf{st}_{f,i}} \neq \mathsf{HashState}(\mathsf{st}_{f,i})$ **or** $h_{m_i} \neq \mathsf{HashModel}(m_i)$:

  return false

return true

Fig. 3: *Circuits* C_U. Based on the circuit, we prove correct execution of admissible functions for the proof of unlearning.

1. Initialization. During the protocol's initialization, function f_I is run to obtain the initial state $\mathsf{st}_{f,0}$ and initial model m_0. Also, the sets of hashed training data and unlearnt data records are initialized, *i.e.,* $\mathcal{H}_{D_0} = \emptyset$ and $\mathcal{H}_{U_0} = \emptyset$. The commitment consists of hashes to these four values, *i.e.,* $\mathsf{com}_0 = (h_{\mathsf{st}_{f,0}}, h_{m_0}, h_{D_0}, h_{U_0})$. Correct initialization is proved using the SNARK for relation R_I captured by circuit C_I. The proof of training ρ_0 consists of the statement ϕ_0 and resulting SNARK proof π_0, which can be verified by the user using com_0.

2A. Proof of Training. The server starts by executing $\mathsf{ProveTraining}$. In the i-th iteration, it first performs the model update by running function f_T on the previous state $\mathsf{st}_{f,i-1}$ and new data records D_i^+, the result being an updated state $\mathsf{st}_{f,i}$ and a new model m_i. Then the server updates the set of hashed training data records $\mathcal{H}_{D_i}$ with D_i^+ and computes the new commitment $\mathsf{com}_i = (h_{\mathsf{st}_{f,i}}, h_{m_i}, h_{D_i}, h_{U_{i-1}})$, where the commitment to the unlearnt data records is the same as in the previous iteration since no data was deleted.

The proof ρ_i is computed using the SNARK for relation R_T captured by circuit C_T. The corresponding statement ϕ_i and proof π_i attest that (a) the model and state were updated correctly with D_i^+, (b) the set of hashed unlearnt data was not changed, and (c) no data record that was previously unlearnt is added. The server sends (ρ_i, com_i) to the users. Subsequently, the users execute $\mathsf{VerifyTraining}$ and verify ρ_i using com_i and the previous commitment com_{i-1}.

2B. Proof of Unlearning. The proof of unlearning consists of two parts: the model update for deleting data records and the proof of non-membership. The server first runs $\mathsf{ProveUnlearning}$. In the i-th iteration, it performs the

model update by running function f_U on the previous state $\mathsf{st}_{f,i-1}$ and the set U_i^+ of data records to be deleted. The result is the updated state $\mathsf{st}_{f,i}$ and model m_i. The set $\mathcal{H}_{U_i}$ is computed by appending hashed records of U_i^+ to $\mathcal{H}_{U_{i-1}}$. At the same time, $\mathcal{H}_{D_i}$ is computed from $\mathcal{H}_{D_{i-1}}$ by removing those entries. The commitment com_i consists of the hash values $(h_{\mathsf{st}_{f,i}}, h_{m_i}, h_{D_i}, h_{U_i})$. The whole procedure is proved using circuit C_U for relation R_U, producing a SNARK proof π_i for the corresponding statement ϕ_i, which can be verified by the user using com_i and com_{i-1}.

The second part of the proof of unlearning is to provide a proof on non-membership to all users that requested a data record $(u, d) \in U_i^+$ to be deleted. We prove this by proving its membership in $\mathcal{H}_{U_i}$. If $\mathcal{H}_{U_i} \cap \mathcal{H}_{D_i} = \emptyset$, it follows that $(u, d) \notin D_i$ (which we show to hold when proving completeness). Specifically, we use the hash chain for $\mathcal{H}_{U_i}$: for a data record, we compute a membership proof as a path in this chain; this path can be verified by recomputing the chain and comparing the final result with the hash in the commitment (*i.e.,* hash value h_{U_i}).

Thus, the server performs ProveNonMembership by computing the chain path to a data record $(u, d) \in U_i^+$. It outputs this as proof $\pi_{u,d}$ which is sent to the user. The user uses the hash h_{U_i} from the commitment to verify membership. If the path leads to that hash, the user accepts, and aborts otherwise.

A. Completeness and Security

We now want to prove the completeness and security of the instantiated protocol. We start to show that our instantiation is complete according to Definition 1. We give a sketch in the following; refer to Appendix C for the full proof.

Theorem 1. *Let* Π *be a complete SNARK and* Hash *a collision-resistant hash function. Then the instantiated protocol satisfies completeness.*

Proof (Sketch). Completeness of the initialization (first property) is easy to observe since the two hashed datasets are initialized as empty and the execution of function f_I is proven with the SNARK for relation R_I. By completeness of the SNARK, the users can successfully verify the proof, additionally using the commitments to state, model and datasets. The second property follows from the completeness of the SNARKs for relations R_T and R_U and collision-resistance of the hash function. However, note that if a hash collision occurs, it is not possible to provide the proof of training. Thus, only computational completeness can be achieved. Given that the proofs of training and unlearning are successful, completeness of the proof of non-membership (third property) follows from the construction and correctness of the hash chain. □

We continue to prove that the instantiation is a secure unlearning protocol according to Definition 2. We again give a sketch below; refer to Appendix D for the full proof.

Theorem 2. *Let* Hash *be a collision-resistant hash function and* Π *be a secure SNARK. Then the instantiated protocol satisfies unlearning security.*

Proof (Sketch). Let $\mathcal{A}$ be an adversary in the unlearning security game (cf. Figure 2). By knowledge soundness of the SNARK, there exists an extractor which outputs the witness and thus the datasets D_i corresponding to the outputs of the adversary. We then use the soundness of the SNARK. That is, $\mathcal{A}$ must have computed the proof using a witness, *i.e.,* the state $\mathsf{st}_{f,i}$ and the dataset D_i^+ (in the proof of training) or dataset U_i^+ (in the proof of unlearning), which also determine the model m_i and must correspond to the hash values in the commitment. By collision-resistance of the hash function, the adversary cannot find another state, model or dataset for the same commitment. Thus, applying function f_T (or f_U) to the previous state and datasets results in same state and model as used by $\mathcal{A}$.

Since all proofs as well as the proof of non-membership of data record (u, d) must verify successfully, the hash of (u, d) must be contained in the set $\mathcal{H}_{U_k}$ which was used to create the proof. Here, k is the iteration where (u, d) was unlearnt; the observation holds by assuming soundness of the SNARK and collision-resistance of the hash function. We can further infer that (u, d) must also be part of future sets $\mathcal{H}_{U_i}$, $k < i \leq \ell$ and by collision-resistance (u, d) must also be part of the underlying datasets U_i. Finally, we use the fact that the proof attests that the intersection of $\mathcal{H}_{U_\ell}$ and $\mathcal{H}_{D_\ell}$ is empty. This yields a contradiction and shows that (u, d) cannot be present in the last dataset D_ℓ. □

VI. Implementation

Next, we implement and compare the main building blocks of the instantiated protocol. For our implementation, we consider different triplets of functions $f = (f_I, f_T, f_U)$ based on techniques from the machine unlearning literature; namely, retraining-based unlearning, amnesiac unlearning and optimization-based unlearning (cf. Section II-A). Additionally, we study the applicability to different ML models and datasets.

To practical implement the protocol, we first need to instantiate SNARK Π and hash function Hash, and then define circuits C_I, C_T and C_U for all three sets of functions f. Our code is available at http://github.com/verifiable-unlearning/artifacts. Experiments are performed on a server running Ubuntu 22.04 with 256 GB RAM and two Intel Xeon Gold 5320 CPUs.

Proof System. Our instantiation is generic and can be implemented with any secure SNARK that satisfies completeness, soundness, and knowledge soundness (cf. Section II-B). In this work, we use Spartan [56] as it is efficient and, more importantly, transparent, *i.e.,* it does not require a trusted setup. Spartan comes in two variants, as a succinct non-interactive zero-knowledge (NIZK) proof system and as a SNARK. Similar to the work of Angel *et al.* [7], we use the NIZK variant, where verification time is linear in the size of the R1CS instance (see below). By using the SNARK variant, some verification cost can be offset to the server and a one-time pre-processing step for the user. Depending on the application, it may be beneficial to use a different proof system. One alternative is Groth16 [31], which, for example,

TABLE I: *Run-Time of Protocol Functions.* We compare the running time between the protocols subtasks. We consider retraining-based unlearning, amnesiac unlearning, and optimization-based unlearning. We report the relative difference with retraining in gray.

	Retraining		Amnesiac		Optimization	
Proof of Training						
R1CS	8,056,887	×1.00	8,130,535	×1.01	7,980,878	×0.99
Π.Prove w/ R_T	4m 32s	×1.00	4m 32s	×1.00	4m 31s	×0.99
Π.Vrfy w/ R_T	1m 36s	×1.00	1m 37s	×1.01	1m 35s	×0.99
Proof of Unlearning						
R1CS	8,102,288	×1.00	616,005	×0.08	919,456	×0.11
Π.Prove w/ R_U	4m 58s	×1.00	2m 18s	×0.46	0m 53s	×0.18
Π.Vrfy w/ R_U	1m 48s	×1.00	0m 49s	×0.45	0m 20s	×0.19
Proof of Non-Membership						
ComputeChainPath	$<$ 1s	×1.00	$<$ 1s	×1.00	$<$ 1s	×1.00
VerifyChainPath	$<$ 1s	×1.00	$<$ 1s	×1.00	$<$ 1s	×1.00

R1CS: #constraints

requires a trusted setup, but has the advantage of constant verification time and proof size.

Circuits. Spartan is implemented on the `ristretto255` elliptic curve, a prime-order group abstraction atop `curve25519`. Following prior work on verifiable computation [7], [49], [67], we convert the computation of our circuits into *Rank-1 Constraint Systems* (R1CS) instances; *i.e.,* the statements in R_I, R_T and R_U (cf. Figure 3 and Appendix B) are represented as a constraint system over a *finite field*. More specifically, an R1CS instance is described by a tuple $(\mathbb{F}, A, B, C, io, n)$, where $\mathbb{F}$ is the finite field, $A, B, C \in \mathbb{F}^{n \times n}$ are matrices of size $n \geq |io| + 1$ and io is the public input and output of the instance. R1CS is a generalization of arithmetic circuit satisfiability. We say an R1CS instance is satisfiable if there exists a witness $\omega \in \mathbb{F}^{n-|io|-1}$ such that $(A \cdot z) \circ (B \cdot z) = (C \cdot z)$ for $z = (io, 1, \omega)$, where $\cdot$ is the matrix-vector product and $\circ$ the Hadamard product. Since A, B, C are generally sparse matrices, a parameter n is sometimes specified, denoting the maximum number of non-zero entries in each matrix.

We implement the algorithms for training and unlearning as arithmetic circuits using the ZoKrates programming language [17] and use CirC [48] for compilation into R1CS instances. To represent data and other parameters in a finite field, we convert them into fixed precision real numbers.

Hash Function. We only require collision-resistance for the hash function. Although our instantiation is generic and can work with any hash function, it is beneficial to use an algebraic hash function where most operations can be directly done in the finite field of the SNARK. Bit-wise hash functions such as the SHA family of hash functions are much slower in that regard. For our construction, we use Poseidon [29] as it is particularly designed for zero-knowledge proof systems. Other good options include Pedersen Hash [36, p.76] or MIMC [6]. To be used with Spartan, we implement a version of Poseidon for the `ristretto255` curve.

A. Protocol Functions

We start by comparing the subtasks of proof of training, proof of unlearning, and proof of non-membership between the different sets of unlearning algorithms. Therefore, we implement the high-level functions of the instantiated protocol for retraining-based unlearning, amnesiac unlearning [30], and optimization-based unlearning [37], [64]. The main focus is on the comparison as well as to show feasibility and versatility of our approach. We discuss possible improvements, *e.g.,* regarding scalability, in Section VII-B.

First, we want to compare and understand the overheads of each subtask between techniques. To this end, we consider a linear regression model and train this model for 3 epochs with SGD as a general purpose approach. We use a synthetic dataset D and set the batch size to 1. We compute a proof of training with the addition of 100 data points with 10 features each. We set $|D_0| = 0$, $|D_1^+| = 100$, and $|U_0| = 0$ accordingly. Subsequently, we compute the proof of unlearning and simulate the deletion of 10 data points and set $|D_1| = 100$, $|U_1| = 0$, and $|U_2^+| = 10$. For optimization-based unlearning, we unlearn for 3 epochs. The results are presented in Table I. The complexity of a statement for Spartan can be measured as the numbers of R1CS constraints. Across all techniques, compilation time of R1CS instances ranges between 17s (optimization-based) and 48m 45s (retraining-based).

Proof of Training. We observe that the complexity of the training is comparable between unlearning approaches. The underlying R1CS instances have between $7,980,878$—$8,130,535$ constraints with proving time between 4m 31s—4m 32s. Recall that in amnesiac unlearning, we also need to collect model updates that are later used for unlearning, which introduces negligible overhead compared to the training costs.

Proof of Unlearning. Runtime of generating and verifying the proof of unlearning shows more variance. Amnesiac unlearning is over $2\times$ faster and optimization-based unlearning over $5\times$ faster than retraining-based unlearning. This is despite the R1CS instance of optimization-based unlearning being almost 50% larger compared to the amnesiac instance (919,456 vs. 616,005 constraints) but it is still more efficient to compute as it is 63% more sparse (*i.e.,* 7,660,455 vs. 12,248,390 entries are non-zero). The main difference is that amnesiac unlearning requires to maintain and verify a state from training (*i.e.,* the model updates) while optimization-based unlearning does not require a state.

Proof System. In general, we observe that verification is $2\times$–$3\times$ faster than proof generation. This is dependent on the choice of the proof system. For example, by using the SNARK variant of Spartan, we can offload some of the verification costs to the server and an additional pre-processing for the user. In this case, proving time increases to 33m 39s—34m 12s for the proof of training across all techniques and verification time reduces to $<$ 1s, but the user needs to run a one-time pre-processing step which takes between 8m 6s—8m 12s.

TABLE II: *Proving Time vs. Model Capacity.* We compare the proving time of proof of training for different classes of models with increasing capacity.

Classifier	R1CS	Π.Prove	Π.Vrfy
Linear Regression	8,056,887	4m 33s	1m 36s
Logistic Regression	9,048,909	5m 8s	1m 45s
Neural Network ($N = 2$)	21,867,010	9m 50s	3m 34s
Neural Network ($N = 4$)	42,030,731	24m 16s	6m 42s

R1CS: #constraints

Proof of Non-Membership. Finally, proof of non-membership is highly efficient. The implementation is independent of the unlearning scheme, and both proof generation and verification take less than one second.

B. Model and Dataset Complexity

The dominant component of the protocols' run-time is the complexity of the circuit used to generate proofs of training and unlearning. This complexity depends mainly on (a) the unlearning technique, (b) the complexity of the model, and (c) the size of the dataset. In the following, we first consider model complexity and study different classes of models. Next, we look on the complexity of the dataset. In both cases, we focus on retraining-based unlearning as the baseline from Table I and, more specifically, on the training circuit C_T.

Models. To understand the effects of the choice of ML model, we follow related work [69], and consider linear regression, logistic regression and neural networks for classification. For the neural networks, we focus on models with one hidden layer and varying numbers of (hidden) neurons $N \in \{2, 4\}$. For activation, we use the sigmoid function and approximate it with a third-order polynomial as done in [39], [40]. Again, we train each model with SGD for 3 epochs on a synthetic dataset consisting of 100 training points with 10 features each.

The results are summarized in Table II. We observe that the R1CS instance increases together with the complexity of the model. For example, the number of constraints increases by $1.12\times$ to $9,048,909$ constraints when going from linear to logistic regression. This is intuitive: in logistic regression, we additionally need to evaluate the sigmoid activation which induces this overhead. In a similar vein, moving from logistic regression to neural networks increases the circuit further to $21,867,010$ ($N = 2$) and $42,030,731$ ($N = 4$) constraints.

Benchmark Datasets. To understand the impact of the dataset, we choose several datasets from the PMLB benchmark suite [52] (as considered in related work [7] on verifiable computation of numerical optimization problems) and train a linear regression model for all datasets. To make results comparable, we train all models for 3 epochs with a learning rate of 0.1. As commonly done, we split the data into 80:20 train test split. Models achieve a test accuracy between 73 % and 92 %.

Results are presented in Appendix E. For all models, we observe a linear dependence between run-time and dataset size. Generating a proof for the smallest dataset with 600 total features (*i.e.,* total points $\times$ features) requires 2m 22s and for the largest dataset with 3,324 total features requires 13m 36s.

VII. Discussion

Next, we discuss alternative ways to instantiate protocols within our framework and improvements to our construction.

A. Alternative Instantiations

External Trust. Our approach eliminates the need for a trusted third party by relying solely on cryptographic protocols to ensure security. To enhance efficiency and reduce the burden on users, a trusted auditor can be introduced to verify on their behalf (as we discuss towards the end of Section IV). This could be done using either a dedicated trusted third party, such as another cloud provider with no incentive to collude with the server, or by employing distributed auditors where trust derives from independent verifications. For instance, Meta recently announced that they will implement key transparency in WhatsApp to complement the manual scanning of QR codes, outsourcing verification to a third party [4].

Trusted Hardware. If TEEs (*e.g.,* Intel SGX [45]) are available, training and unlearning procedures can be performed within them. More specifically, we can replace the SNARK in our instantiation with a TEE such that the circuit is computed inside the TEE. In this case, the proof consists of a digest signed by the TEE provider, which can be verified by the user. However, when using a TEE, one needs to consider common concerns such as trusting a hardware vendor, availability of said vendor for signing the digests, limited memory [32], their applicability to ML-related tasks that involve GPU computation [62], and side-channels [47], [63]. Some of these issues have been addressed by Weng *et al.* [66] (cf. Section VIII).

Minimizing Redundancy. In our instantiation, users who request unlearning are required to verify future updates to ensure that their data has not been reintroduced. If we combine VC with an additional proof of secure data erasure, we can give similar guarantees while not requiring the user to verify all updates. However, secure erasure is a non-trivial problem in itself and was considered in *e.g.,* [51]. Formalizing deletion compliance from a server's perspective [24] can also be seen as complementary problem.

B. Scalability

Our experiments show that the run-time of the instantiated protocol is dominated by generating and verifying the proof of training and unlearning. We base our construction on *Verified Computation* (VC) and, as a result, inherit its scalability limitations. This can also be observed for other VC-based approaches in the ML setting [57], [35], [39], [40]. Any future advances in VC will lead to run-time improvements for our approach. Nevertheless, we want discuss how we can improve performance with the primitives available today.

SNARK-friendly Techniques. Certain computations are more amendable to efficient SNARK verification than others, for

example, the development of SNARK-friendly hash functions [29], [6], [36]. Similarly, there exist ML paradigms that are also more amenable to verification. For instance, inference using quantized models [38], [19] or lookup tables for expensive computations [38] reduce costs. Furthermore, when there exists a unique ML model (*i.e.,* a global optimum for the underlying optimization problem), proving and verification complexity can be improved even further [7]. In our experiments, we observe that online computation of model updates in optimization-based unlearning is faster than verifying model updates in amnesiac unlearning as the verification of input values involves expensive calculation of hash values. We envision future work to focus on developing SNARK-friendly unlearning techniques combining above observations.

Offloading Computation. Orthogonal to the employed unlearning technique and ML model, one can offload expensive proof generation steps to the user (*e.g.,* the evaluation of a non-linear activation function). We can split the proving and verification processes such that the server creates a proof for certain types of computations and shares partial results with the (honest) user who performs (and thus verifies) expensive computations themselves.

Application-specific Relaxations. Finally, depending on the application, it might be possible to avoid the expensive generation of the proof of training. Consider, for instance, an application where data is collected once and *only* be removed at a later point in time (*e.g.,* biomedical user studies or other human-involved data collection processes). In this case, proof of training only needs to be performed once and—if users further trust the initial training phase—it might be sufficient to only prove unlearning.

C. Privacy

Formalizing privacy for unlearning protocols is an interesting direction for future work and requires to establish an additional security definition. Although it is out-of-scope for our work, we want to highlight that our instantiation does not require the users to know the datasets or model. In fact, they only see hash commitments and the SNARK proofs. If the hash function satisfies pseudo-randomness or is modeled as a random oracle, and assuming that the input space is large enough, then hash values do not leak information about the underlying data points. Additionally, if the SNARK satisfies the zero-knowledge property [28] (which most SNARKs including Spartan do), then the proof also does not leak information about the witness. However, we require users to know whether training or unlearning happened because they need to know which verification procedure to run. Privacy in the context of model inference has been studied more extensively, *e.g.,* Gao *et al.* [21] define security notions for deletion hiding and reconstruction. An overview for different formalizations of inference privacy is also given by Salem *et al.* [53].

VIII. Related Work

Our approach to verifiable unlearning intersects with various areas of security and ML research. Below, we explore related concepts and methods.

Verifying Unlearning. Prior work [22], [59] aims at verifying unlearning by embedding backdoors [33] in models (using data whose unlearning is to be verified) and verifying backdoor removal on unlearning. However, such approaches are probabilistic with no theoretical guarantees of when they work, unlike our cryptography-informed approach which produces verifiable proofs. The work of Guo *et al.* [34] provides end-users with a certificate that the new model is influenced by the specific data in a quantifiably low manner. While this certificate conceptually bounds the influence of a data point from an algorithmic perspective, it provides no guarantee that the entity executing the algorithm (*i.e.,* server) did so correctly. In our work, we aim to capture exactly this and provide cryptographic guarantees. Weng *et al.* [66] propose an unlearning framework based on TEEs. Their protocol uses SISA unlearning [12] and can be captured by our framework as well. In contrast to our instantiation that is based only on cryptographic primitives, their approach relies on trusted hardware (*i.e.,* the correctness and integrity of the SGX enclave) as well as cryptographic assumptions (*i.e.,* EUF-CMA security of the signature scheme used by the enclave and collision-resistance of the hash function).

Verifiable Computation. We use verifiable computation for proof of training and proof of unlearning. There has been a series of works demonstrating a remarkable progress in making these schemes (and those related to verification of data used for computation) practical (*e.g.,* [56], [25], [50], [20]). To verify the computation of training an ML model, Zhao *et al.* [69] also propose verification using a SNARK. However, their primary objective is to design a scheme to ensure that the payments made to servers are correct. In our work, we design a scheme to verify the correctness of data deletion when training ML models. Otti [7] is a compiler that is aimed at designing efficient arithmetic circuits for problems that involve optimization (such as those commonly found in ML). DIZK [67] is a distributed system capable of distributing the compute required for proof creation. Garg *et al.* [23] describes an approach to verify ML training based on MPC-in-the-head, while the work of Abbaszadeh *et al.* [5] developes a sumcheck-based proof system for the gradient-descent algorithm and framework for recursive composition of proofs. Both approaches may be extended to a full-fledged (retraining-based) unlearning protocol using our framework.

Proving Model Inference. There exist various approaches to proving inference using SNARKs [42], [44], [65], [19], [38] which complements our protocol in that regard. Another approach would be to use trusted execution environments to do so as suggested in [66].

IX. Conclusion

The problem of unlearning has gained significant interest in terms of definitions and algorithms for updating model parameters. However, regardless of the definition or the algorithm the server uses to update the model, the user has no way to verify that the server indeed executed the unlearning procedure. In this paper, we define unlearning as a security problem and propose a framework to capture the guarantees verifiable unlearning needs to provide. We propose the first verifiable unlearning procedure based on cryptographic primitives instantiated using SNARKs and hash chains. Our implementation shows the feasibility of our approach on several benchmark datasets and machine learning models. Future work includes determining which unlearning techniques are most suitable for efficient verifiable computation, while at the same time devising methods specifically for verifying machine learning code.

Acknowledgements

This work was supported by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under the project ALISON (492020528) and Germany's Excellence Strategy - EXC 2092 CASA - 390781972. Thorsten Eisenhofer conducted this research while interning at the Vector Institute. Olga Ohrimenko has been supported in part by the joint CATCH MURI-AUSMURI. Nicolas Papernot would like to acknowledge his sponsors, who support his research with financial and in-kind contributions, including Apple, CIFAR through the Canada CIFAR AI Chair program, DARPA through the GARD program, Intel, NSERC through the Discovery grant, Meta, and Ontario through the Early Researcher Award program. Resources used in preparing this research were provided, in part, by the Province of Ontario, the Government of Canada through CIFAR, and companies sponsoring the Vector Institute. We would like to thank members of the CleverHans Lab for their feedback. We would also like to thank Sebastian Angel, Jess Woods, and Eleftherios Ioannidis for input related to the SNARK compilers.

References

[1] General Data Protection Regulation (GDPR). Official Legal Text, 2016.

[2] California Consumer Privacy Act (CCPA). Official Legal Text, 2018.

[3] Personal Information Protection and Electronic Documents Act (PIPEDA). Official Legal Text, 2019.

[4] Deploying Key Transparency at WhatsApp. Engineering at Meta, 2023.

[5] Kasra Abbaszadeh, Christodoulos Pappas, Jonathan Katz, and Dimitrios Papadopoulos. Zero-knowledge proofs of training for deep neural networks. In *ACM Conference on Computer and Communications Security (CCS)*, 2024.

[6] Martin R. Albrecht, Lorenzo Grassi, Christian Rechberger, Arnab Roy, and Tyge Tiessen. MiMC: Efficient Encryption and Cryptographic Hashing with Minimal Multiplicative Complexity. In *International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT)*, 2016.

[7] Sebastian Angel, Andrew J. Blumberg, Eleftherios Ioannidis, and Jess Woods. Efficient Representation of Numerical Optimization Problems for SNARKs. In *USENIX Security Symposium*, 2022.

[8] Thomas Baumhauer, Pascal Schöttle, and Matthias Zeppelzauer. Machine Unlearning: Linear Filtration for Logit-based Classifiers. *Machine Learning*, 2022.

[9] Nir Bitansky, Ran Canetti, Alessandro Chiesa, Shafi Goldwasser, Huijia Lin, Aviad Rubinstein, and Eran Tromer. The Hunting of the SNARK. *Journal of Cryptology*, 2017.

[10] Nir Bitansky, Ran Canetti, Alessandro Chiesa, and Eran Tromer. From Extractable Collision Resistance to Succinct Non-Interactive Arguments of Knowledge, and Back Again. In *Innovations in Theoretical Computer Science (ITCS)*, 2012.

[11] Nir Bitansky, Ran Canetti, Omer Paneth, and Alon Rosen. On the Existence of Extractable One-Way Functions. *SIAM Journal on Computing*, 2016.

[12] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. Machine Unlearning. In *IEEE Symposium on Security and Privacy (S&P)*, 2021.

[13] Gavin Brown, Mark Bun, Vitaly Feldman, Adam D. Smith, and Kunal Talwar. When is Memorization of Irrelevant Training Data Necessary for High-Accuracy Learning? In *ACM SIGACT Symposium on Theory of Computing (STOC)*, 2021.

[14] Yinzhi Cao and Junfeng Yang. Towards Making Systems Forget with Machine Unlearning. In *IEEE Symposium on Security and Privacy (S&P)*, 2015.

[15] Nicholas Carlini, Chang Liu, Úlfar Erlingsson, Jernej Kos, and Dawn Song. The Secret Sharer: Evaluating and Testing Unintended Memorization in Neural Networks. In *USENIX Security Symposium*, 2019.

[16] Min Chen, Zhikun Zhang, Tianhao Wang, Michael Backes, Mathias Humbert, and Yang Zhang. Graph Unlearning. In *ACM Conference on Computer and Communications Security (CCS)*, 2021.

[17] Jacob Eberhardt and Stefan Tai. ZoKrates - Scalable Privacy-Preserving Off-Chain Computations. In *IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, 2018.

[18] Vitaly Feldman. Does Learning Require Memorization? A Short Tale About a Long Tail. In *ACM SIGACT Symposium on Theory of Computing (STOC)*, 2020.

[19] Boyuan Feng, Lianke Qin, Zhenfei Zhang, Yufei Ding, and Shumo Chu. ZEN: Efficient Zero-Knowledge Proofs for Neural Networks. *Cryptology ePrint Archive*, 2021.

[20] Dario Fiore, Cédric Fournet, Esha Ghosh, Markulf Kohlweiss, Olga Ohrimenko, and Bryan Parno. Hash First, Argue Later: Adaptive Verifiable Computations on Outsourced Data. In *ACM Conference on Computer and Communications Security (CCS)*, 2016.

[21] Ji Gao, Sanjam Garg, Mohammad Mahmoody, and Prashant Nalini Vasudevan. Deletion Inference, Reconstruction, and Compliance in Machine (Un)learning. In *Privacy Enhancing Technologies Symposium (PETS)*, 2022.

[22] Xiangshan Gao, Xingjun Ma, Jingyi Wang, Youcheng Sun, Bo Li, Shouling Ji, Peng Cheng, and Jiming Chen. VeriFi: Towards Verifiable Federated Unlearning. *Computing Research Repository (CoRR)*, 2022.

[23] Sanjam Garg, Aarushi Goel, Somesh Jha, Saeed Mahloujifar, Mohammad Mahmoody, Guru-Vamsi Policharla, and Mingyuan Wang. Experimenting with zero-knowledge proofs of training. In *ACM Conference on Computer and Communications Security (CCS)*, 2023.

[24] Sanjam Garg, Shafi Goldwasser, and Prashant Nalini Vasudevan. Formalizing Data Deletion in the Context of the Right to Be Forgotten. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2020.

[25] Rosario Gennaro, Craig Gentry, and Bryan Parno. Non-interactive Verifiable Computing: Outsourcing Computation to Untrusted Workers. In *Annual International Cryptology Conference (CRYPTO)*, 2010.

[26] Amirata Ghorbani and James Zou. Data Shapley: Equitable Valuation of Data for Machine Learning. In *International Conference on Machine Learning (ICML)*, 2019.

[27] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2020.

[28] Shafi Goldwasser, Silvio Micali, and Charles Rackoff. The Knowledge Complexity of Interactive Proof Systems. *SIAM Journal on Computing (SICOMP)*, 1989.

[29] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. Poseidon: A New Hash Function for Zero-Knowledge Proof Systems. In *USENIX Security Symposium*, 2021.

[30] Laura Graves, Vineel Nagisetty, and Vijay Ganesh. Amnesiac Machine Learning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 2021.

[31] Jens Groth. On the Size of Pairing-Based Non-interactive Arguments. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT)*, 2016.

[32] Karan Grover, Shruti Tople, Shweta Shinde, Ranjita Bhagwan, and Ramachandran Ramjee. Privado: Practical and Secure DNN Inference with Enclaves. *Computing Research Repository (CoRR)*, 2018.

[33] Tianyu Gu, Brendan Dolan-Gavitt, and Siddharth Garg. BadNets: Identifying Vulnerabilities in the Machine Learning Model Supply Chain. *Computing Research Repository (CoRR)*, 2017.

[34] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. Certified Data Removal from Machine Learning Models. In *International Conference on Machine Learning (ICML)*, 2020.

[35] Inbar Helbitz and Shai Avidan. Reducing ReLU Count for Privacy-Preserving CNN Speedup. *Computing Research Repository (CoRR)*, 2021.

[36] Daira Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. Zcash. Protocol Specification (Version 2022.3.4), 2022.

[37] Joel Jang, Dongkeun Yoon, Sohee Yang, Sungmin Cha, Moontae Lee, Lajanugen Logeswaran, and Minjoon Seo. Knowledge Unlearning for Mitigating Privacy Risks in Language Models. *Computing Research Repository (CoRR)*, 2022.

[38] Daniel Kang, Tatsunori Hashimoto, Ion Stoica, and Yi Sun. Scaling up Trustless DNN Inference with Zero-Knowledge Proofs. *Computing Research Repository (CoRR)*, 2022.

[39] Andrey Kim, Yongsoo Song, Miran Kim, Keewoo Lee, and Jung Hee Cheon. Logistic Regression Model Training based on the Approximate Homomorphic Encryption. *Cryptology ePrint Archive*, 2018.

[40] Miran Kim, Yongsoo Song, Shuang Wang, Yuhou Xia, and Xiaoqian Jiang. Secure Logistic Regression Based on Homomorphic Encryption: Design and Evaluation. *JMIR Medical Informatics*, 2018.

[41] Pang Wei Koh and Percy Liang. Understanding Black-box Predictions via Influence Functions. In *International Conference on Machine Learning (ICML)*.

[42] Seunghwan Lee, Hankyung Ko, Jihye Kim, and Hyunok Oh. vCNN: Verifiable Convolutional Neural Network. *Cryptology ePrint Archive*, 2020.

[43] Klas Leino and Matt Fredrikson. Stolen Memories: Leveraging Model Memorization for Calibrated White-Box Membership Inference. In *USENIX Security Symposium*, 2020.

[44] Tianyi Liu, Xiang Xie, and Yupeng Zhang. ZkCNN: Zero Knowledge Proofs for Convolutional Neural Network Predictions and Accuracy. In *ACM Conference on Computer and Communications Security (CCS)*, 2021.

[45] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V. Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R. Savagaonkar. Innovative Instructions and Software Model for Isolated Execution. In *Workshop on Hardware and Architectural Support for Security and Privacy (HASP)*, 2013.

[46] Seth Neel, Aaron Roth, and Saeed Sharifi-Malvajerdi. Descent-to-Delete: Gradient-Based Methods for Machine Unlearning. In *Algorithmic Learning Theory (ALT)*, 2021.

[47] Olga Ohrimenko, Felix Schuster, Cedric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. Oblivious Multi-Party Machine Learning on Trusted Processors. In *USENIX Security Symposium*, 2016.

[48] Alex Ozdemir, Fraser Brown, and Riad S. Wahby. CirC: Compiler Infrastructure for Proof Systems, Software Verification, and more. In *IEEE Symposium on Security and Privacy (S&P)*, 2022.

[49] Alex Ozdemir, Riad S. Wahby, Barry Whitehat, and Dan Boneh. Scaling Verifiable Computation Using Efficient Set Accumulators. In *USENIX Security Symposium*, 2020.

[50] Bryan Parno, Jon Howell, Craig Gentry, and Mariana Raykova. Pinocchio: Nearly Practical Verifiable Computation. In *IEEE Symposium on Security and Privacy (S&P)*, 2013.

[51] Daniele Perito and Gene Tsudik. Secure Code Update for Embedded Devices via Proofs of Secure Erasure. In *European Symposium on Research in Computer Security (ESORICS)*, 2010.

[52] Joseph D. Romano, Trang T. Le, William G. La Cava, John T. Gregg, Daniel J. Goldberg, Praneel Chakraborty, Natasha L. Ray, Daniel S. Himmelstein, Weixuan Fu, and Jason H. Moore. PMLB v1.0: An Open-Source Dataset Collection for Benchmarking Machine Learning Methods. *Bioinformatics*, 2022.

[53] Ahmed Salem, Giovanni Cherubin, David Evans, Boris Köpf, Andrew Paverd, Anshuman Suri, Shruti Tople, and Santiago Zanella Béguelin. SoK: Let The Privacy Games Begin! A Unified Treatment of Data Inference Privacy in Machine Learning. *Computing Research Repository (CoRR)*, 2022.

[54] Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. Remember What You Want to Forget: Algorithms for Machine Unlearning. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[55] Ozan Sener and Silvio Savarese. Active Learning for Convolutional Neural Networks: A Core-Set Approach. In *International Conference on Learning Representations (ICLR)*, 2017.

[56] Srinath Setty. Spartan: Efficient and General-Purpose zkSNARKs Without Trusted Setup. In *Annual International Cryptology Conference (CRYPTO)*, 2020.

[57] Avital Shafran, Gil Segev, Shmuel Peleg, and Yedid Hoshen. Crypto-Oriented Neural Architecture Design. In *IEEE International Conference on Acoustics, Speech and Signal Processing, (ICASSP)*, 2021.

[58] Ilia Shumailov, Zakhar Shumaylov, Dmitry Kazhdan, Yiren Zhao, Nicolas Papernot, Murat A. Erdogdu, and Ross J. Anderson. Manipulating SGD with Data Ordering Attacks. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[59] David Marco Sommer, Liwei Song, Sameer Wagh, and Prateek Mittal. Towards Probabilistic Verification of Machine Unlearning. *Computing Research Repository (CoRR)*, 2020.

[60] Anvith Thudi, Gabriel Deza, Varun Chandrasekaran, and Nicolas Papernot. Unrolling SGD: Understanding Factors Influencing Machine Unlearning. In *IEEE European Symposium on Security and Privacy (EuroS&P)*, 2022.

[61] Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. On the Necessity of Auditable Algorithmic Definitions for Machine Unlearning. In *USENIX Security Symposium*, 2022.

[62] Stavros Volos, Kapil Vaswani, and Rodrigo Bruno. Graviton: Trusted Execution Environments on GPUs. In *Symposium on Operating Systems Design and Implementation, (OSDI)*, 2018.

[63] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. Leaky Cauldron on the Dark Land: Understanding Memory Side-Channel Hazards in SGX. In *ACM Conference on Computer and Communications Security (CCS)*, 2017.

[64] Alexander Warnecke, Lukas Pirch, Christian Wressnegger, and Konrad Rieck. Machine Unlearning of Features and Labels. In *Symposium on Network and Distributed System Security (NDSS)*, 2023.

[65] Chenkai Weng, Kang Yang, Xiang Xie, Jonathan Katz, and Xiao Wang. Mystique: Efficient Conversions for Zero-Knowledge Proofs with Applications to Machine Learning. In *USENIX Security Symposium*, 2021.

[66] Jia-Si Weng, Shenglong Yao, Yuefeng Du, Junjie Huang, Jian Weng, and Cong Wang. Proof of Unlearning: Definitions and Instantiation. *Computing Research Repository (CoRR)*, 2022.

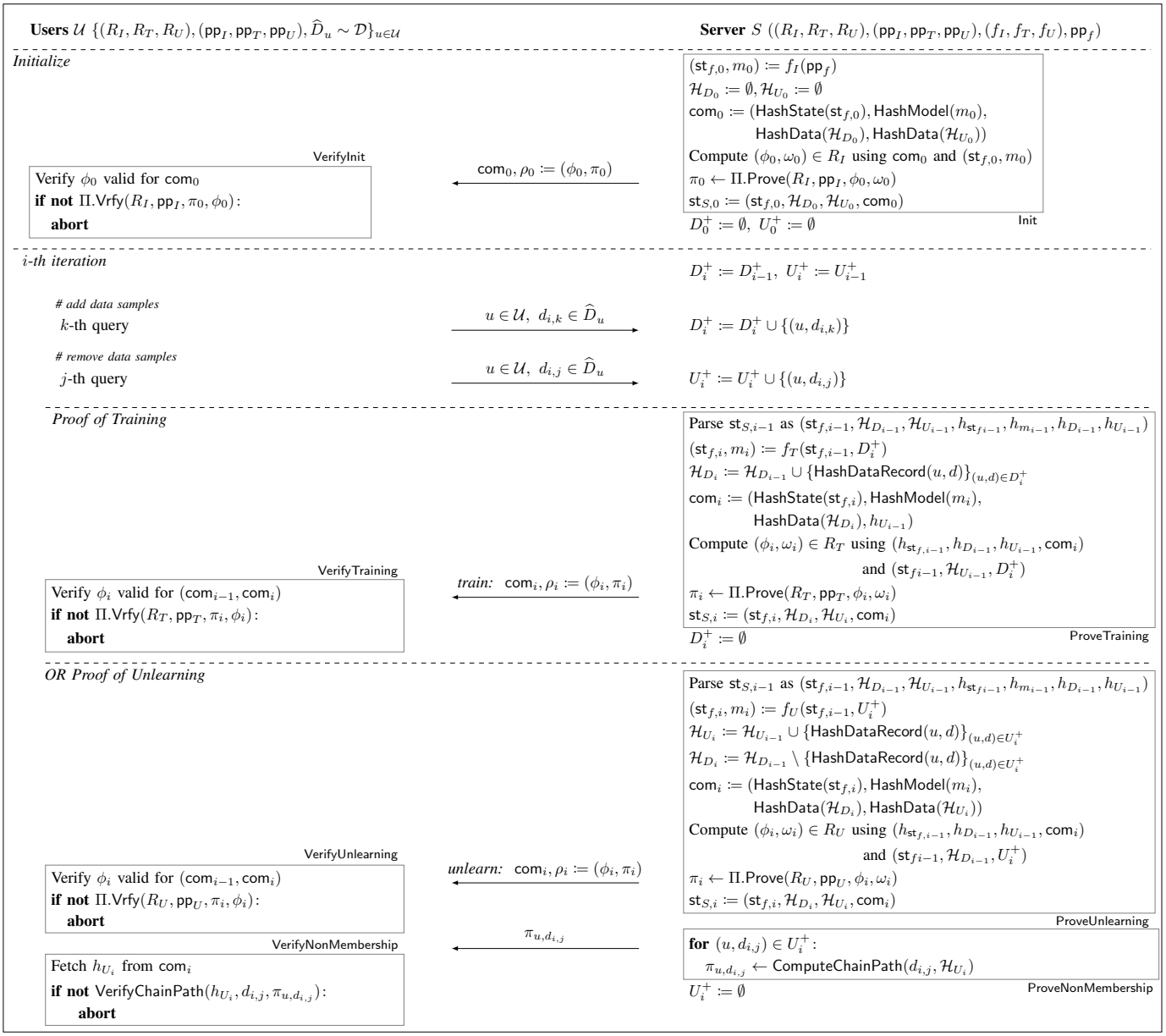
[67] Howard Wu, Wenting Zheng, Alessandro Chiesa, Raluca Ada Popa, and Ion Stoica. DIZK: A Distributed Zero Knowledge Proof System. In *USENIX Security Symposium*, 2018.

[68] Yinjun Wu, Edgar Dobriban, and Susan B. Davidson. DeltaGrad: Rapid Retraining of Machine Learning Models. In *International Conference on Machine Learning (ICML)*, 2020.

[69] Lingchen Zhao, Qian Wang, Cong Wang, Qi Li, Chao Shen, and Bo Feng. VeriML: Enabling Integrity Assurances and Fair Payments for Machine Learning as a Service. *IEEE Transactions on Parallel and Distributed Systems (TPDS)*, 2021.

A Additional Algorithms and Full Protocol

We instantiate the protocol Φ_f for the triple of admissible functions $f = (f_I, f_T, f_U)$ with two primitives: a SNARK Π, and a hash function Hash. Additional algorithms are described below.



$\underline{\mathsf{HashData}(\mathcal{H})}$
$\Psi \coloneqq \mathsf{Hash}(d_\emptyset)$
for $h_d \in \mathcal{H}$:
  $\Psi \coloneqq \mathsf{Hash}(\Psi, h_d)$
return Ψ

$\underline{\mathsf{AppendHashData}(h, \mathcal{H})}$
$\Psi \coloneqq h$
for $h_d \in \mathcal{H}$:
  $\Psi \coloneqq \mathsf{Hash}(\Psi, h_d)$
return Ψ

$\underline{\mathsf{HashDataRecord}(u, d = (x, y))}$
$h_d \coloneqq \mathsf{Hash}(u)$
for $x_j \in x$:
  $h_d \coloneqq \mathsf{Hash}(h_d, \mathsf{Hash}(x_j))$
$h_d \coloneqq \mathsf{Hash}(h_d, \mathsf{Hash}(y))$
return h_d

$\underline{\mathsf{HashModel}(m = [w_0, \ldots, w_n])}$
$h_m \coloneqq \mathsf{Hash}(m[0])$
for $w_i \in m[1\colon]$:
  $h_m \coloneqq \mathsf{Hash}(h_m, \mathsf{Hash}(w_i))$
return h_m

$\underline{\mathsf{HashState}(\mathsf{st}_f)}$
$h_{\mathsf{st}_f} \coloneqq \mathsf{Hash}(\mathsf{st}_f[0])$
for $s_i \in h_{\mathsf{st}_f}[1\colon]$:
  $h_{\mathsf{st}_f} \coloneqq \mathsf{Hash}(h_{\mathsf{st}_f}, s_i)$
return h_{st_f}

$\underline{\mathsf{VerifyChainPath}(u, d, h_U, \pi_{u,d})}$
 # recompute hash Ψ from path $\pi_{u,d}$
$\Psi \coloneqq \mathsf{Hash}(\pi_{u,d}[0], \mathsf{HashDataRecord}(u, d))$
for node **in** $\pi_{u,d}[1\colon]$:
  $\Psi \coloneqq \mathsf{Hash}(\Psi, \mathrm{node})$
 # verify final hash
return $[\![\Psi = h_U]\!]$

$\underline{\mathsf{ComputeChainPath}(u, d, \mathcal{H}_U)}$
$h_d \coloneqq \mathsf{HashDataRecord}(u, d)$
$idx_d \coloneqq \mathcal{H}_U.\mathsf{index}(h_d)$
if $idx_d = \perp$:
  return $\perp$
 # get intermediate hash from chain below d
$\Psi \coloneqq \mathsf{HashData}(\mathcal{H}_{U\,:\,idx_d})$
$\pi_{u,d} \coloneqq [\Psi]$
 # add path from d
for $h_d \in \mathcal{H}_{U_{idx_d+1}\colon}$:
  $\pi_{u,d}.\mathsf{append}(h_d)$
return $\pi_{u,d}$

B CIRCUITS

Based on circuits C_I and C_T, we prove correct execution of admissible functions for initialization and proof of training.

C_I(**public** $h_{\mathsf{st}_{f,0}}, h_{m_0}, h_{D_0}, h_{U_0}$, **private** $\mathsf{st}_{f,0}, m_0$)

```
# Check input for initialization
if h_{st_{f,0}} ≠ HashState(st_{f,0}) or
   h_{m_0} ≠ HashModel(m_0) or
   h_{D_0} ≠ HashData(∅) or
   h_{U_0} ≠ HashData(∅):
    return false
return true
```

C_T(**public** $h_{\mathsf{st}_{f,i}}, h_{\mathsf{st}_{f,i-1}}, h_{m_i}, h_{D_i}, h_{D_{i-1}}, h_{U_i}, h_{U_{i-1}}$, **private** $\mathsf{st}_{f,i-1}, \mathcal{H}_{U_{i-1}}, D_i^+$)

```
# Check input set of hashed unlearnt data records
if h_{U_{i-1}} ≠ HashData(H_{U_{i-1}}):
    return false
# Update and check set of hashed training data records and unlearnt data records
H_{D_i^+} := {HashDataRecord(u, d)}_{(u,d)∈D_i^+}
if h_{D_i} ≠ AppendHashData(h_{D_{i-1}}, H_{D_i^+}) or h_{U_i} ≠ h_{U_{i-1}} or H_{U_{i-1}} ∩ H_{D_i^+} ≠ ∅:
    return false
# Check input state, perform training and check outputs
if h_{st_{f,i-1}} ≠ HashState(st_{f,i-1}):
    return false
(st_{f,i}, m_i) := f_T(st_{f,i-1}, D_i^+)
if h_{st_{f,i}} ≠ HashState(st_{f,i}) or h_{m_i} ≠ HashModel(m_i):
    return false
return true
```

C COMPLETENESS PROOF

Proof (of Theorem 1). We need to prove the three properties in Definition 1 capturing the initialization, the proof of training and the proof of unlearning which includes the proof of non-membership.

Initialization. First, running Init yields the initialized state $\mathsf{st}_{f,0}$ and model m_0 obtained by executing f_I. Using the hash function to commit to those two values and additionally the empty sets $\mathcal{H}_{D_0}$ and $\mathcal{H}_{U_0}$, an instance $(\phi_0, \omega_0) \in R_I$ can be derived and the SNARK proof π_0 can be created using $\Pi.\mathsf{Prove}$. By correctness of the relation and completeness of the SNARK, ϕ_0 will be valid for com_0 and $\Pi.\mathsf{Vrfy}(R_I, \mathsf{pp}_I, \pi_0, \phi_0) = 1$.

Proof of Update. For the second property, recall the inputs and outputs of $\mathsf{ProveTraining}$ and $\mathsf{ProveUnlearning}$. The state $\mathsf{st}_{S,i-1}$ contains the state $\mathsf{st}_{f,i-1}$, the set $\mathcal{H}_{D_{i-1}}$ of hashed data records, the set $\mathcal{H}_{U_{i-1}}$ of hashed unlearnt data records and the previous commitment com_{i-1}. In the proof of training, the new state $\mathsf{st}_{f,i}$ and the new model m_i are computed by running function f_T on $\mathsf{st}_{f,i-1}$ and the set D_i^+ of data records to be added. The new sets $\mathcal{H}_{D_i}$ and $\mathcal{H}_{U_i}$ are computed from the previous ones and updated with D_i^+. The commitment is computed by hashing the four components.

Then the training instance $(\phi_i, \omega_i) \in R_T$ can be derived and the SNARK proof π_i computed. The proof attests (cf. Appendix B) that (1) m_i was computed correctly since f_T was executed, (2) the training data does not contain unlearnt record since the hashes of the new data records are not contained in $\mathcal{H}_{U_i}$, and (3) the set of unlearnt data records has not changed since the commitments to the unlearnt data records are the same. By correctness of the relation and perfect completeness of the SNARK, we have $\Pi.\mathsf{Vrfy}(R_T, \mathsf{pp}_T, \pi_i, \phi_i) = 1$.

Note that there exists a special case where the server is unable to create a proof although the datasets are valid. This is the case whenever there exist two distinct data records $(u, d) \in D_i^+$, $(u', d') \in U_i$, where $U_i = \bigcup_{j \in [i]} U_j^+$ is the dataset implicitly contained in $\mathcal{H}_{U_i}$, such that $\mathsf{HashDataRecord}(u, d) = \mathsf{HashDataRecord}(u', d')$. However, we only require computational completeness and assume that the datasets are provided by a PPT adversary. Then this translates to finding a collision for the hash function which happens with negligible probability if the hash function is collision-resistance. Hence, $\mathsf{VerifyTraining}$ will output 1 with probability $1 - \mathsf{negl}(\lambda)$.

Proof of Unlearning. Completeness for the proof of unlearning proceeds similar. The new state $\mathsf{st}_{f,i}$ and the new model m_i are computed by running function f_U on $\mathsf{st}_{f,i-1}$ and the set U_i^+ of data records to be deleted. The new sets $\mathcal{H}_{D_i}$ and $\mathcal{H}_{U_i}$ are computed from the previous ones and updated by removing and appending U_i^+, respectively. The commitment is computed by hashing the four components.

The unlearning instance $(\phi_i, \omega_i) \in R_U$ is derived and the SNARK proof π_i that is computed attests (cf. Figure 3) that (1) m_i was computed correctly since f_U was executed, (2) the training data does not contain unlearnt record since we removed the records in U_i^+ from $\mathcal{H}_{D_i}$, and (3) previous set of unlearnt data records is a subset of the updated set since we added the records in U_i^+ to $\mathcal{H}_{U_i}$ to which we commit. By correctness of the relation and perfect completeness of the SNARK, we have $\Pi.\mathsf{Vrfy}(R_U, \mathsf{pp}_U, \pi_i, \phi_i) = 1$ and $\mathsf{VerifyUnlearn}$ will output 1 with probability 1.

Finally, consider the algorithm $\mathsf{ProveNonMembership}$. If a data record (u, d) was unlearnt in iteration i, then its hash is present in $\mathcal{H}_{U_i}$. The proof of non-membership $\pi_{u,d}$ consists of the chain path to (u, d) in the chain of $\mathcal{H}_{U_i}$. Let com_i be the commitment for this iteration, then by correctness of the tree path algorithm, $\mathsf{VerifyNonMembership}$ will output 1 with probability 1. □

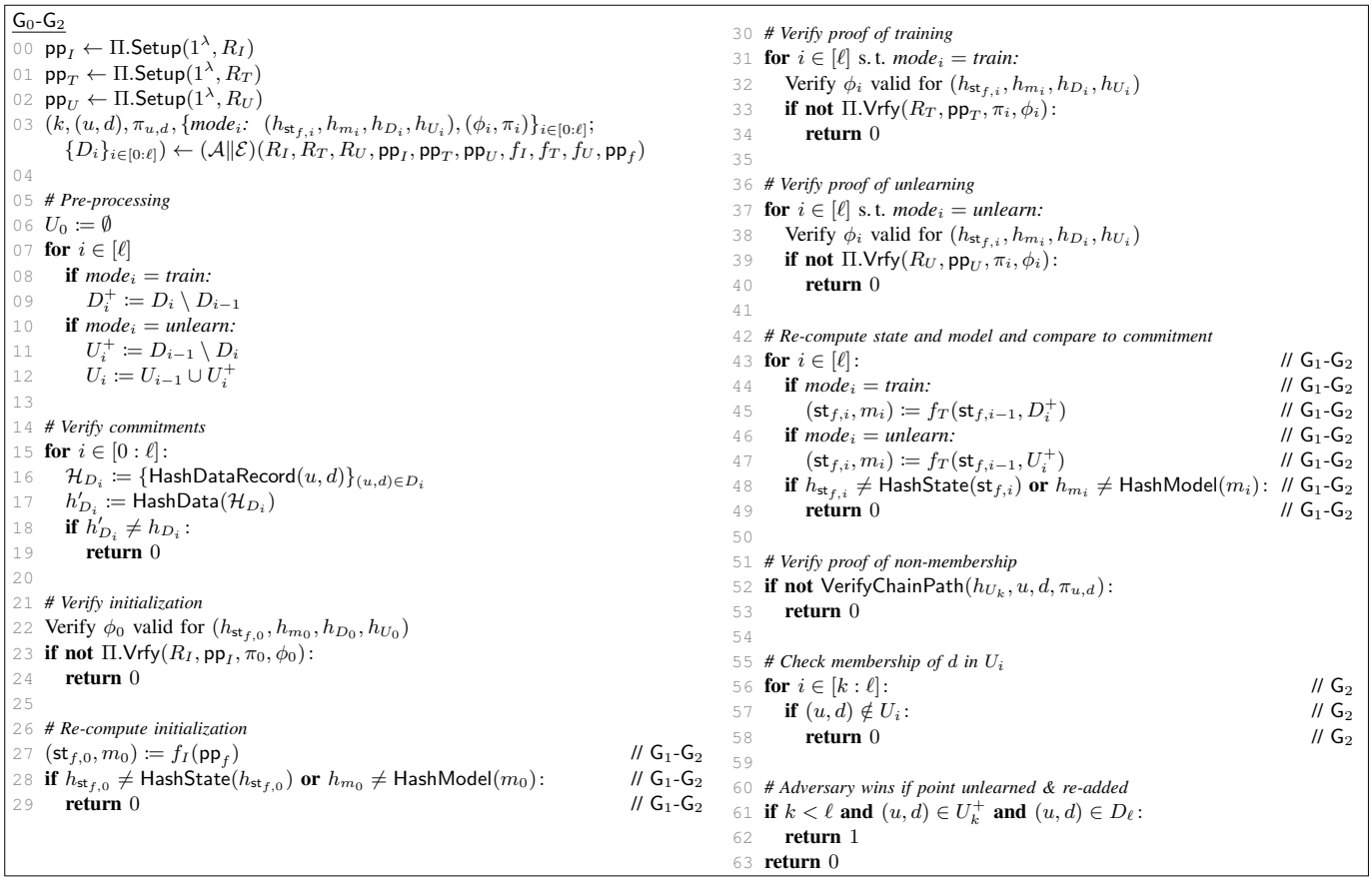


Fig. 4: *Games* G_0-G_2 *for the proof of Theorem 2.* We prove unlearning security for our instantiated protocol Φ_f in Appendix A, where $f = (f_I, f_T, f_U)$ and hyperparameter pp_f are fixed by the participating parties and determine relations R_I, R_T and R_U.

D Security Proof

Proof (of Theorem 2). Let $\mathcal{A}$ be an adversary against unlearning security (as defined in Figure 2) of our instantiation. We will first argue that for all $\mathcal{A}$ there exists an extractor $\mathcal{E}$ that outputs the underlying datasets D_i. This follows directly from the knowledge soundness of the SNARK for relations R_I, R_T and R_U. For this, look at the private inputs to the circuits in Figure 3 and Appenix B which translate to the witness. Initialization gives us that $D_0 = \emptyset$. The proof of training inputs D_i^+ and the proof of unlearning inputs U_i^+ such that we can extract $D_i = D_{i-1} \cup D_i^+$ if $\mathit{mode}_i = \mathit{train}$ and $D_i = D_{i-1} \setminus U_i^+$ if $\mathit{mode}_i = \mathit{unlearn}$.

We continue with the sequence of games given in Figure 4.

Game G_0. Let G_0 be the original game $\mathsf{GameUnlearn}$ and $\mathcal{E}$ be the extractor. Recall that the adversary must output a sequence of tuples $(k, (u, d), \pi_{u,d}, \{\mathit{mode}_i\colon\ \mathsf{com}_i, \rho_i\}_{i\in[0:\ell]})$ for some $\ell \in \mathbb{N}$, where $\mathsf{com}_i = (h_{\mathsf{st}_{f,i}}, h_{m_i}, h_{D_i}, h_{U_i})$ and $\rho_i = (\phi_i, \pi_i)$ for $i \in [0 : \ell]$. We iterate over the winning conditions and return 0 as soon as one of them is violated (cf. Figure 4). For book-keeping we also compute all sets of unlearnt data points U_i, as well as the sets D_i^+, U_i^+ from D_i as described for the extractor. Note that this is only a conceptual change at this point and we have

$$\Pr[\mathsf{G}_0 \Rightarrow 1] = \Pr[\mathsf{GameUnlearn}_{\mathcal{A},\mathcal{E},\Phi_f,\mathcal{D}}(1^\lambda) \Rightarrow 1] \ .$$

Game G_1. In G_1, we compute the state $\mathsf{st}_{f,i}$ and the model m_i for each iteration from the corresponding datasets by applying f_I, f_T and f_U. We then check whether the hashes of state and model correspond to $h_{\mathsf{st}_{f,i}}$ and h_{m_i} in the commitment. If this is not the case, the game outputs 0. We claim

$$|\Pr[\mathsf{G}_1 \Rightarrow 1] - \Pr[\mathsf{G}_0 \Rightarrow 1]| \leq \mathsf{negl}(\lambda) \ .$$

To prove the claim we argue in the following steps:

- First, π_i proves that the adversary knows a state $\mathsf{st}'_{f,i}$ and a dataset $D_i^{+\prime}$ for each proof of training (or a dataset $U_i^{+\prime}$ for each proof of unlearning) such that model m'_i was computed by applying function f_T (or function f_U) to state $\mathsf{st}'_{f,i-1}$ and dataset $D_i^{+\prime}$ (or dataset $U_i^{+\prime}$). It also proves that the commitment aligns with the inputs. Since the functions are deterministic, we thus have $h_{\mathsf{st}_{f,i}} = \mathsf{HashState}(\mathsf{st}'_{f,i})$ and $h_{m_i} = \mathsf{HashData}(m'_i)$ as well as $h_{D_i} = \mathsf{HashData}(\mathcal{H}'_{D_i})$, where $\mathcal{H}'_{D_i}$ is the set of all hashed data records in D'_i.
  By soundness of the SNARK, the adversary can only forge a proof for an invalid statement with negligible probability, so we can assume the proof was generated honestly with a witness. By knowledge soundness, the extractor is able to compute this witness such that $D'_i = D_i$.
- Second, we claim that then $\mathsf{st}'_{f,i} = \mathsf{st}_{f,i}$ and $m'_i = m_i$ are the actual state and model used for the next iteration. This is true unless the adversary finds a collision in the hash function such that $\mathsf{HashState}(\mathsf{st}'_{f,i}) = \mathsf{HashState}(\mathsf{st}_{f,i}) = h_{\mathsf{st}_{f,i}}$ or $\mathsf{HashModel}(m'_i) = \mathsf{HashModel}(m_i) = h_{m_i}$, which we assume to happen only with negligible probability.

Game G_2. In G_2, we check whether the data record (u, d) output by $\mathcal{A}$ is contained in the underlying datasets U_i of the k-th and all subsequent iterations. We will show that

$$|\Pr[\mathsf{G}_2 \Rightarrow 1] - \Pr[\mathsf{G}_1 \Rightarrow 1]| \leq \mathsf{negl}(\lambda) \ .$$

For this we first look again at the SNARK proof π_i and the underlying circuits. If a proof of training is performed, the adversary must prove that $h_{U_i} = h_{U_{i-1}}$. This implies—assuming no hash collision occurs—that $\mathcal{H}_{U_i} = \mathcal{H}_{U_{i-1}}$ and $U_i = U_{i-1}$. If a proof of unlearning is performed, the SNARK proof ensures that $\mathcal{H}_{U_{i-1}} \subset \mathcal{H}_{U_i}$ and thus $U_{i-1} \subset U_i$, again using collision-resistance of the hash function. Thus, if $(u, d) \in U_k$, it must also be true that $(u, d) \in U_{k+1}$, ..., $(u, d) \in U_\ell$. By soundness of the SNARK, the adversary cannot prove a false statement, so the above claims must hold.

We also know that $(u, d) \in U_k$ since the proof of non-membership consists of the path from the hashed data record (u, d) to the hash h_{U_k} contained in the k-th commitment. Since the adversary can only win if the proof verifies successfully, we know that in this case the hash value of (u, d), in the following denoted by $h_{u,d} \coloneqq \mathsf{HashDataRecord}(u, d)$, must be a node in the hash chain constructed from $\mathcal{H}_{U_k}$. Unless the adversary finds another data record (u', d') such that $\mathsf{HashDataRecord}(u', d')$ maps to the same hash value $h_{u,d}$—which happens with negligible probability—the record (u, d) must be contained in U_k.

Finally, we show that $\Pr[\mathsf{G}_2 \Rightarrow 1] \leq \mathsf{negl}(\lambda)$. For this, recall that π_i also attests that no unlearnt data point is contained in the dataset, in particular that the intersection $\mathcal{H}_{D_i} \cap \mathcal{H}_{U_i}$ is empty. Together with the fact that the commitments h_{D_i} and h_{U_i} are constructed from $\mathcal{H}_{D_i}$ and $\mathcal{H}_{U_i}$ (due to soundness of the SNARK), the hashed datasets must have been obtained from the corresponding dataset D_i and U_i (unless the adversary has found a collision in the hash function). Combining with previous results, this implies that $D_i \cap U_i = \emptyset$ for all $i \in [\ell]$. As shown above, we know that $(u, d) \in U_\ell$. The final winning condition requires that $(u, d) \in D_\ell$. This cannot be the case since it would contradict the fact that the intersection of the two sets is empty, which proves the final claim.

Collecting the probabilities yields

$$\Pr[\mathsf{GameUnlearn}_{\mathcal{A},\mathcal{E},\Phi_f,\mathcal{D}}(1^\lambda)] \leq \mathsf{negl}(\lambda) \ ,$$

which concludes the proof of Theorem 2. □

E Scalability to Benchmark Datasets

We compute the proof of training for different datasets from the PMLB benchmark suite [52]. Size refers to #data points × #features and R1CS refers to #constraints.

Dataset	**Size**		**R1CS**	**Π.Prove**	**Π.Vrfy**
Creditscore	100	6	3,986,308	2m 22s	0m 47s
Patient	88	8	4,579,718	2m 28s	0m 53s
Cy Young	92	10	5,903,988	3m 16s	1m 9s
Corral	160	6	6,347,236	3m 43s	1m 15s
Lawsuit	264	4	7,190,981	4m 7s	1m 27s
Breast cancer	286	9	16,514,048	9m 25s	3m 18s
Monk3	554	6	21,841,281	13m 36s	4m 32s