Proof of Unlearning: Definitions and Instantiation

Jiasi Weng\textsuperscript{\textcopyright}, Member, IEEE, Shenglong Yao, Yuefeng Du\textsuperscript{\textcopyright}, Member, IEEE, Junjie Huang, Jian Weng\textsuperscript{\textcopyright}, Senior Member, IEEE, and Cong Wang\textsuperscript{\textcopyright}, Fellow, IEEE

Abstract—The “Right to be Forgotten” rule in machine learning (ML) practice enables some individual data to be deleted from a trained model, as pursued by recently developed machine unlearning techniques. To truly comply with the rule, a natural and necessary step is to verify if the individual data are indeed deleted after unlearning. Yet, previous parameter-space verification metrics may be easily evaded by a distrustful model trainer. Thus, Thudi et al. recently present a call to action on algorithm-level verification in USENIX Security’22. We respond to the call, by reconsidering the unlearning problem in the scenario of machine learning as a service (MLaaS), and proposing a new definition framework for Proof of Unlearning (PoUL) on algorithm level. Specifically, our PoUL definitions (i) enforce correctness properties on both the pre and post phases of unlearning, so as to prevent the state-of-the-art forging attacks; (ii) highlight proper practicality requirements of both the prover and verifier sides with minimal invasiveness to the off-the-shelf service pipeline and computational workloads. Under the definition framework, we subsequently present a trusted hardware-empowered instantiation using SGX enclave, by logically incorporating an authentication layer for tracing the data lineage with a proving layer for supporting the audit of learning. We customize authenticated data structures to support large out-of-enclave storage with simple operation logic, and meanwhile, enable proving complex unlearning logic with affordable memory footprints in the enclave. We finally validate the feasibility of the proposed instantiation with a proof-of-concept implementation and multi-dimensional performance evaluation.

Index Terms—Authentication, data integrity, trusted computing, machine learning (ML).

I. INTRODUCTION

MACHINE learning (ML) models deployed for prediction services are usually trained on user data, which can refer to the current machine learning as a service (MLaaS) paradigm. Particularly, a data owner can authorize a service provider to train an ML model over his/her data and later offer black-box prediction services with the trained model. But at a later time, the data owner might withdraw the authorization, i.e., sending a request to delete his/her data from the trained model, simply due to regret emotion [69], [84], or deterred by privacy attacks on trained models [16], [49], [66], [76]. Such right of data deletion can be legally protected by privacy regulations [79], namely “Right to be Forgotten” (RTBF) which is explicitly stated by the European Union’s General Data Protection Regulation (GDPR) [60] and the United States’s California Consumer Privacy Act (CCPA) [3]. More specifically, the U.K.’s Information Commissioner’s Office [1] and the Federal Trade Commission [22] recently clarify that complying with the deletion request requires retraining the model or deleting the model altogether.

Machine unlearning is a closely relevant concept, having a target model forget partial training data, but previous unlearning approaches [13], [15], [31], [32], [33], [37], [51], [85] might fail to achieve the RTBF compliance in a distrustful setting. First of all, the approaches often assume an honest server. However, the powerful server side is likely to not delete user data in reality, which is commonly reported [4], [12], [27], and is naturally wavering in users’ trust [46], [72]. Furthermore, a dishonest server can also strategically evade the verification metrics suggested by prior unlearning approaches, by launching forging attacks [67], [73], and therefore, Thudi et al. [73] call for action to audit unlearning, while leaving a blank space to be filled in. Last but not least, the nature of a black-box service manner might motivate the server to maliciously fork multiple models, which may invalidate previous unlearning approaches as well as existing black-box verified methods [40], [50], [70]. For instance, the server can fork an arbitrary model (not the target model in question), and claim having deleted data from the forking model, while still offering prediction services using the target model which never deletes data at all.

In light of the dishonest server, which can arbitrarily deviate from prior unlearning approaches, this work studies how to truly implement Proof of Unlearning (PoUL) for pursuing the RTBF compliance with \textit{end-to-end} assurances, \textit{i.e.}, closed-loop enforcement starting from pre-learning and prediction to unlearning and post-prediction.
Unlearning Problem: We clarify the unlearning problem specific for the off-the-shelf ML pipeline in MLaaS. Starting from a trained target model offering prediction services, when a data owner requests to delete a data point, the server should execute an unlearning process on the target model, yielding a newly predictive model which fully eliminates the effect of the data point in accordance to the unlearning goal of previous efforts [13], [15], [18], [21], [37].

But the server may behave dishonestly, which may forge an incorrect target model, or run an incorrect unlearning process or fork an inconsistent model for new predictions. We next define our PoUL to prevent such misbehavior with respect to a data point deletion.

Definitions of PoUL: Here involves two sequential phases aligned with the black-box service nature: (i) Setup phase. the server proves that a target model in question indeed learns the data point. (ii) Deletion phase. the server proves that a newly predictive model is yielded by a correct unlearning process which indeed removes the data point from the previous target model. Within the definition scope, we require that the server should simultaneously assure the correctness of the target model, the unlearning process and the newly predictive model, such that the data owner (as a honest verifier) can be convinced of the fulfillment of his/her unlearning problem.

PoUL is different from a recent excellent art [45], proof-of-learning (PoL), which facilitates proofs for a learning process based on the idea of re-execution. Specifically, PoL offers a verifier a document on learning trajectory for convincing him of the truth that the learning trajectory is used to generate a particular model with overwhelming probability. While the PoL is easy to understand and implement, it cannot be extended to implement the PoUL, due to different problem statements and threat models, figured out by Thudi et al. [73].

Recent attack examples [88] can efficiently forge learning trajectory to generate a fake but valid PoL proof, which further demonstrates the difficulty of implementing the PoUL. Lastly, our PoUL especially emphasizes that subsequent prediction services should be offered by an unlearned model as expected, which is not considered by the PoL.

To pursue practice-oriented solutions, we subsequently highlight practicality requirements needed for our PoUL: (a) Enabling generic models. The unlearning problem in MLaaS can happen in various models, which requires proof techniques to support generic computations involved by various model architectures. (b) Limited invasiveness. Proof techniques should be maximally compatible with the underlying learning algorithms, so as to maintain the original ML services quality. (c) Minimal overhead. Standing on the already workload-intensive ML pipeline, additional proving workloads should be as affordable as possible. Besides the server-side requirements, PoUL should also meet the following verifier-side requirements, such that the data owner can enjoy predictive services nearly as usual beyond verifying unlearning. (d) Concise proof. Generated proofs should be short enough to ease the storage cost of the data owner. (e) Efficient verification. Proofs should also be cheaply verified considering a usually thin verifier.

A. Solution Overview

Considering the above general PoUL framework, we will present our specific solution for its practical implementation, with the aim of fulfilling the deletion obligation effectively. Our defined PoUL is closely related to proof-based verifiable computation (VC) [80], particularly the concept of “authenticate first and prove later” proof systems. This leads us to devise a conceptual solution by combining dynamic authenticated data structures (ADS) [71] with proof-based verifiable computation. In this approach, a data owner can initially authorize their data to the server using a digest generated by ADS, such as RSA-based, pairing-based, or Merkle tree-like accumulators. The data owner then keeps track of specific correctness proofs issued by the server, which pertain to statements like “a specific computation is executed on this data matching the ADS, yielding that output”. To provide more detail, one option is to employ general-purpose and concise proof systems, such as succinct non-interactive arguments of knowledge (SNARK) [35]. However, these proof systems [80] may not be capable of handling extensive machine learning workloads efficiently. For example, even the latest succinct proof system [64] still takes 86GB memory to prove knowledge of $512 \times 512$ matrix multiplication, despite allowing concise proof. Other memory-efficient proof systems from interactive oracle proofs, “MPC-in-the-head” paradigm, garbled circuits and subfield Vector Oblivious Linear Evaluation (sVOLE), have to sacrifice communication efficiency or verification time, as summarized by a recent work [86]. Furthermore, off-the-shelf proof systems are typically not designed for training algorithms and often need modifications to adapt to the inherent cryptographic operations, such as arithmetic operations on finite fields through model quantization [90]. Recent efforts, like the one in [91], require alterations to the underlying linear model learning algorithms. In summary, the existing cryptographic solutions are either not scalable for general models, incompatible with the underlying learning algorithms, or cost-inefficient. Consequently, they do not align with our practicality goals.

From another perspective, trusted execution environments (TEEs) are quickly becoming trusted computing services offered by dominant service providers, e.g., Alibaba Cloud [8], IBM Cloud [43] and Microsoft Azure [6], and many application efforts [25], [29], [47] demonstrate that TEEs can assist the providers in performing with obligation compliance, e.g., enforcing data usage with GDPR compliance [61], [68]. Therefore, we are naturally encouraged to instantiate PoUL with the TEEs-backed offerings, and thereby enable native implementation and practical deployment on the server side. Also, we admit that recent TEEs-empowered ML works [11], [38], [39], [41], [42], [48], [52], [54], [74], [75], [87], [89] may help mitigate a certain issue within our PoUL, but we need new and holistic designs tailored for proving end-to-end RTBF compliance in MLaaS.

1) Our Unlearning Setting: We begin with Bourtoule et al.’s efficient retraining-based unlearning framework [13], towards enabling complete deletion [34]. Specifically, the server trains multiple chunked submodels in isolation...
over disjoint shards of data, and meanwhile, each chunked submodel is incrementally learned with non-overlapping slices of data points (named sliced data hereafter); when one data point is deleted, only particular submodels exactly trained on or impacted by the data point need to be retrained, which is therefore faster than retraining a complete model from scratch. Herein, we particularly denote that the lineage of a data point is starting from the slice this data point falls in to the submodels it impacts.

2) TEEs-Capable PoUL Implementation: Standing on the unlearning setting, we incorporate ADS with Intel Software Guard Extensions (SGX) [2], [23] on the server side, to issue proofs in the Setup and Deletion phases previously described. Our incorporation implies (i) an authentication layer for tracing which sliced data are learned or unlearned, and which chunked submodels are retrained or used to offer predictions; and (ii) a proving layer for attesting the execution correctness of the learning or prediction processes that exactly use particular authenticated data or submodels. However, it is challenging to implement the two-layer incorporation, due to the incompatible features between our unlearning setting and Intel SGX.

a) Challenge (i): Memory-efficient authentication: In the authentication layer, we require tracking which sliced data impact which chunked submodels, and supporting authenticated updates on the impacted submodels when a data point is deleted. But both data and submodels are often large-sized and the unlearning problem might cause a large number of submodels to be updated, which inevitably incurs large memory footprints far beyond the original memory limitation of SGX.1 Hence, we provide a memory-efficient authentication design tailored for the traceability from the sliced data to the updatable submodels while protecting the integrity.

b) Challenge (ii): Proof of stateful computation and fast verification: In the proving layer, the computations involving a chain of incremental retraining processes to update affected submodels, and subsequent prediction processes using the newly retrained model should be attested, so that a verifier is convinced of the fulfillment of data deletion. Yet, the computations are stateful, since the incremental retraining and prediction processes require previously generated states, e.g., submodels, which is in conflict with the SGX design of primarily protecting stateless computations. We extend the trust outside the SGX’s protected memory to securely save previous states, by typically letting the protected memory retain unique randomnesses (named seed) respective to each submodel for subsequent integrity checking. To reduce verification cost, we adopt a self-verification strategy. Concretely, we enable the verifier to only assert that an appointed submodel (i.e., a final one) matching a newly produced digest yields new predictions on a given test data, while the authenticity of all retrained submodels prior to the appointed submodel is verified by the enclave itself. Furthermore, in light of the need for monitoring the correctness of subsequent prediction services, an additional SGX-enabled auditor is considered.

1It is about 94 MB available to applications in a widely adopted version, and a recent release [5] supports 188 MB.

In summary, we make three-fold contributions as following:

- We propose a new two-phase definition framework for achieving PoUL, which adapts to Bourtoule et.’s generic unlearning algorithm (in IEEE S&P’21), and meanwhile, prevents recent forging attacks (in USENIX Security’22).
- Under the definition framework, we present an SGX-capable solution with newly customized designs, by logically integrating an authenticated layer for tracing the lineage of training data with a proving layer for auditing the correctness of model training.
- We give a proof-of-concept implementation for the SGX-capable PoUL, and evaluate the multifaceted performance in terms of storage cost, deletion time, learning/unlearning time and trained accuracy. The implementation code is available at https://github.com/Jamesyaoshenglou/unlearning-TEE.

II. BACKGROUND

A. Machine Unlearning

Machine unlearning starts from a concept of removing some training data from an already trained model [15]. According to the different understanding of the concept, existing approaches can be roughly classified into two groups, including approximate unlearning achieving data deletion from parameter level [31], [32], [33], [63] and exact unlearning achieving data deletion from algorithm level [13], [15], [18], [21], [37], [81]. The essential difference between the two groups is clarified by the Definition 1.

Definition 1: Let \( D = \{d_i\} \cup d_u, i \in \{1, \ldots, N\} \setminus u \) be a collection of \( N \) training samples. Let \( D_{-u} \) denote a collection of \( N - 1 \) training samples, namely \( D \setminus d_u \). Let \( D_M \) be the distribution of a model that has ever been trained over \( D \) and then unlearned \( d_u \) via a machine unlearning algorithm \( R \). Let \( D_M' \setminus d_u \) denote \( D_m \) to be the distribution of a model trained over \( D_{-u} \). Then denote \( R \) as an approximate unlearning algorithm, if the distribution \( D_M \) is approximately equal to \( D_M' \). Otherwise, denote \( R \) as an exact unlearning algorithm, if the distribution \( D_M \) is exactly equal to \( D_M' \).

Recently, the SISA framework [13] provides an exact unlearning method towards generic ML models, which fully erases the effect of deleted data. It is faster than retraining from scratch by trading storage for efficiency, and is followed by many promising unlearning work [18], [21], [37], [81]. Our work focuses on exact unlearning and aligns with the awesome SISA framework. While approximate unlearning is feasible in specific scenarios, it might provide a less robust compliance with the RTBF rule.

We next introduce a server-side ML pipeline under the SISA framework, as shown in Fig. 1. Suppose the server-side pipeline runs over the dataset authorized by a data owner and generates learned models, along with model checkpoints. At a later time, it complies with a deletion request of the data owner via unlearning. We introduce the pipeline with two parts: A) pre-learning and prediction stages, as well as B) unlearning and post-prediction stages. We denote \( N = \{1, \ldots, n\} \) and \( S = \{1, \ldots, s\} \).
Fig. 1. The server-side ML pipeline under the SISA framework. Multiple dashed rectangles represent that different shards of data are used for incrementally training constituent models in an isolated manner. Within one shard, the training data is further sliced into disjoint data slices, where the data slice in grey (i.e., $d_{s-1}$) influences or impacts the intermediate submodels in grey (i.e., $m_{s-1}$ and $m_s$). All constituent models (i.e., the last submodels) of individual shards produce predictions that are aggregated into final predictions.

1) Pre-Learning and Prediction Stages: There are four major operations. (a) Shard. In the first place, the data owner’s dataset is divided into $n$ non-overlapping data shards, denoted as $D_{i} \in \mathbb{N}$; (b) Isolation. Subsequently, stochastic gradient descent (SGD)-based training algorithm is applied to train a constituent model on each shard of data in isolation; (c) Slice. Inside each shard, the data $D_j$ is further sliced into $s$ disjoint data slices, represented by $D_{j} = \{d_{j,i}\}_{i \in \mathbb{N}}$, such that data slices are incrementally added for learning, and the produced submodels $\{m_j\}_{j \in \mathbb{N}}$ are saved during learning; (d) Aggregation. The final prediction is obtained by aggregating the predictions provided by the constituent models $m_{s,j}, j \in [N]$ of all shards. Herein, a data slice can contain a small set of non-overlapping data points. Note that the impact of each data point is restricted on relatively small-size submodels, rather than an entire model.

2) Unlearning and Post-Prediction Stages: On receiving the data owner’s request of deleting one data point $d_u \in D$, the following steps are executed: (a) find which shards this data point $d_u$ belongs to and which submodels it influences; (b) locate the shard and the slice associated to $d_u$, and delete the $d_u$ from the slice and the submodels influenced by $d_u$ within the shard; (c) re-execute the incremental training processes with the remaining data slices as pre-training.

Now we show a concrete example. We suppose the $d_u$ that will be deleted falls in the last but one slice $d_{s-1}$ of Fig. 1 and the submodels $m_{s-1}$ and $m_s$ are influenced by the $d_u$. Guided by the above steps, the $m_{s-1}$ and $m_s$ will be deleted, and subsequently, new $m_{s-1}$ and $m_s$ are relearned over the remaining data $\{d_1, \ldots, d_{s-2}, d_{s-1}\} \setminus \{d_u\}$ and $\{d_1, \ldots, d_{s-2}, d_{s-1}\} \setminus \{d_u, d_{s}\}$, respectively, in the same incremental manner. Finally, new predictions are produced by newly updated models and aggregated without any effect of the deleted data point. Notice, the server only needs to recompute the relatively small-scale submodels affected by the $d_u$, without the need to retrieve or relearn the entire model. As a result, this framework can be faster than retraining from scratch, varying from $2.45 \times$ to $4.63 \times$ [13].

Note that the SISA framework can also be categorized as a data-driven unlearning approach according to a recent machine unlearning survey [53]. Besides the category of data-driven unlearning, other two distinct categories are identified: (a) model-agnostic unlearning. The unlearning methods or frameworks are suitable to various ML models, e.g., Guo et al.’s idea [36]. The work may be extended to deep neural networks, despite that it only provides theoretic guarantees for a class of linear models. (b) model-intrinsic unlearning. The unlearning approaches are applied to a specific type of ML models. For instance, Chen et al. [21] and Wang et al. [81] focus on unlearning graph data from graph neural networks.

B. Trusted Execution Environments

Trusted Execution Environments (TEEs) provide a hardware-protected memory environment for sealed data and shielded execution of applications in an untrusted platform. Many off-the-shelf TEEs technologies, e.g., Intel SGX [2], [23], ARM TrustZone [9], become promising to implement secure applications with minimal performance compromise, in which Intel SGX is widely adopted, and thus our work adopts SGX.

1) Intel SGX Enclave: Intel SGX allows creating a secure memory region, named as enclave, to execute application codes with confidentiality and integrity guarantees, isolated from the outside platform which can be untrusted. It also provides a remote attestation mechanism by issuing a signature-based proof (Enhanced Privacy ID signature scheme [14]) on a requested quote, containing the measurement of the enclave’s code and data. With the proof, a remote client can assert the authenticity of the enclave identity and the truthfulness of code execution, after it subscribes to the Intel’s Attestation Service.

2) Formal Functionality Modeling: Shi et al. [58] formalize the functionality of an SGX enclave, involving enclave initialization, enclave operations and attestation. With the functionality, Tramèr et al. [77] introduce an enclave-empowered “commit-and-prove” functionality executed by a prover equipped with a transparent enclave and a verifier. The transparent enclave is considered with integrity property, but with minimal confidentiality assumptions, e.g., random number generators and the signing key, not including the programs running in the enclave.

3) Attack Threats on SGX Enclave: Integrity violation attacks on the SGX enclave, such as forking, replacing, relocation and rollback attacks will be mitigated by our designs (see Section V-B). However, we do not address existing side-channel attacks breaking integrity by stealing secret keys [19], [78], aware of recent countermeasures protecting secret-dependent memory accesses against such side-channel attacks. Control flow attacks [7], [24] are also outside of our scope, since the attacks escape from the off-the-shelf static remote attestation mechanisms. In addition, Denial-of-Service attacks [44], e.g., shutting down the enclave applications, are not our concern.

C. Data Structure for Fast Membership Testing

We elaborate here a space-optimized and high-speed data structure, namely, cuckoo filter [26], designed for membership query from a usually large data set with low and controllable
false positive rates. Different from Bloom filters, it supports $O(1)$ element deletion, not merely addition. Compared with cuckoo hash tables, it only stores short and constant-size fingerprints of elements.

Concretely, a cuckoo filter consists of an array of buckets, in which each bucket contains multiple entries, e.g., 4 or 8. A fingerprint of an element can be filled in two possible buckets. This is determined by two hash functions derived from standard cuckoo hashing [57] as introduced in the following. For example, given a new element $e$ to be inserted and its fingerprint $f_e$ (e.g., truncated PRESENT ciphertext with 16 bits as [65] applied), an alternative location $h_1$ of bucket $B$ is found by calculating a 64-bit hash value $h_1 = \text{Hash}(e)$. The hash functions can adopt the CityHash [26]. If it is empty, the fingerprint of this element is inserted into $B[h_1]$. Otherwise, another location $h_2$ is found by calculating $h_2 = h_1 \oplus \text{Hash}(f_e)$, where the original element $e$ is not needed to be retrieved. Next, a location displacement method is called, if $B[h_2]$ is also not empty. Specifically, the element at $h_2$ would be replaced with $f_e$, and the element is placed in its alternative location. This displacement process is repeated until an empty bucket is found or a displacement threshold (e.g., 500) is reached.

### III. Problem Statement

We observe that a dishonest sever can arbitrarily deviate from fulfilling a data deletion request of a data owner during unlearning. Also, there not exists an approach to enforcing the behavior of the server during unlearning. The observation motivates us to establish trust for the data owner. This section will clarify the threats and deletion assumptions we are concerned about, for ease of understanding our definitions of PoUL in the next section.

#### A. Threat Scenario

On top of the ML pipeline previously described, we consider the server can misbehave arbitrarily, by forking multiple models due to the non-transparent nature of the black-box service manner, or forging unlearned models with off-the-shelf forging attacks [67], [73]. We note prior auditing methods [30], [40], [50], [70] are not concerned about the misbehaviors. Concretely, we summarize the following three cases of misbehaviors in the pipeline assisted with Fig. 2:

- At the prediction stage, the server might substitute a correct target model and avoid deleting data from the target model. The so-called correct target model is denoted as the model which was ever trained using the data to be deleted.
- At the unlearning stage, the server might not correctly execute an unlearning process.
- At the post-prediction stage, the server might not deploy a correct unlearned model for subsequent service.

For data owner, we consider he/she is honest throughout the process, only having a black-box access to the models. His/Her data collected by the server for learning and his/her test data for predictions are also considered benign. Data confidentiality against the server is not considered. Lastly, we let the data owner communicate with the server via an authenticated point-to-point communication channel.

#### B. Deletion Assumptions

We focus on the problem of deleting training data from a target model on the server site. We note that data deletion from memory on physical medium addressed by previous arts [55], [59] is outside of our consideration. We are not concerned about data or model copy, simply due to monetary storage costs, and do not consider deleting copied data or model. We assume the training data is non-overlapping similar to the SISA framework. We focus on a single data deletion request, but our design can be extended to delete multiple data pieces. Relying on the SISA framework’s technical characteristic, multiple data deletion requests should be assumed irrespective of the actual models (e.g., a never unlearned model or an unlearned model) in question [13], and adaptive deletion [37] is out of our scope. Additionally, recent inference attacks [20], [28] on unlearned models are out of consideration.

### IV. Definitions on Proof of Unlearning

We are ready to describe the definitions on PoUL, along with three correctness properties and five practicality goals. Intuitively, we require the server to generate proofs of correct unlearning, and given the proofs, the data owner has confidence in asserting if the server complies with his/her deletion request or not. Notice, our PoUL definition is not only agnostic to the ML model architectures, but also to the unlearning techniques employed.

To begin with, we denote some notations. We let an indexed collection of learning data $D = \{d_i\}_{i \in [N]}$, including a data point $d_u$ that will be deleted. We denote $\mathbb{M}_0$ as a public model with initial weights, and $F/G$ as public learning/prediction algorithms. We let a test data $x$ and a prediction $p$. We mark $[m_u]$ as the affected part of a model by its learning data $d_u$. We use $\text{SR}$ and $\text{DO}$ to denote the server and the data owner, respectively. We also use $\pi$ to define a proof component generated by the $\text{SR}$.

#### A. Definition Framework

There are two phases composing the definition framework for PoUL. In a setup phase, the $\text{SR}$ offers a proof attesting that the currently predictive model is a true target model responding to a test challenge from the $\text{DO}$. In a later deletion phase, the $\text{SR}$ retains a proof of truth that the newly predictive
model is a truly unlearned model from the above target model excluding the deleted data piece \(d_u\), responding to a new test challenge. Particularly, the two phases involve a chain of interactive procedures as following.

1) Setup Phase: (\(S_1\)-Initialize) The DO uploads his/her data \(D\) along with an authentic data digest to the SR; The DO specifies a learning algorithm \(F\) and an initial model \(M_0\) with the SR; The SR then learns a model \(M_1\) from \(M_0\) with \(F\) taking the data \(D\) as input. Notice, the learned \(M_1\) now is the target model we describe before, and its digest \(H(M_1)\) is published. (\(S_2\)-Challenge) The DO sends a test data \(t\) to the currently predictive model for prediction query. (\(S_3\)-Prove) The SR responds to the query with a prediction \(p\) and a proof component \(\pi\). (\(S_4\)-Verify) With the proof \(\pi\) and prediction \(p\), and the above model digest \(H(M_1)\), the DO verifies the correctness of the prediction. Specifically, he/she would reject it with high probability, if the prediction is not yielded by the target model \(M_1\), denoted as a statement \(p \leftarrow G(M_1, t) \land M_1 \leftarrow F(M_0, D)\).

2) Deletion Phase: (\(D_1\)-Unlearning) The DO sends a request of deleting the data \(d_u\) from the target model \(M_1\); the SR executes an unlearning process of \(F\) on the \(M_1\) and \(D\backslash d_u\), yielding an unlearned model \(M_2\) whose digest \(H(M_2)\) is public. (\(D_2\)-Challenge) At a later time, the DO challenges the newly predictive model with a new test data \(t'\). (\(D_3\)-Prove) The SR returns a prediction \(p'\), along with a new proof component \(\pi'\). (\(D_4\)-Verify) With the proof \(\pi'\) and prediction \(p'\), and the digests \(H(M_1)\) plus \(H(M_2)\), the DO verifies the prediction correctness, and rejects it with high probability, if the prediction \(p'\) is not offered by a correctly unlearned model, denoted as a statement \(p' \leftarrow G(M_2, t') \land M_2 \leftarrow F(M_1\backslash\{m_u\}, D\backslash d_u)\).

3) Logical Components: Essentially, there are two-layer components to be realized for supporting the setup and the deletion phases in our scoped PoUL:

a) Authentication layer: The data owner authenticates his/her data (e.g., data digest) in the first place and delegates the data to the server, along with the authenticated result for tracking operations on the data in a future time, e.g., using or not using the data for training. Relying on the authentication layer, the intermediate models and the final model yielded by a learning process, should also be authenticated saved (and updated), such that a later unlearning process or a prediction process is ensured to use the previously authenticated models.

b) Proving layer: This layer works jointly with the authentication layer. Particularly, the server convinces the data owner that a learning process, either in the \(S_1\)-Initialize or the \(D_1\)-Unlearning procedure, is executed as expected, with the learning data matching the previously authenticated data. Besides learning, the server also assures the correctness of a predictive model, which exactly means that a prediction process in the \(S_3\)-Prove procedure (resp. \(D_3\)-Prove) uses a most recent model yielded by the \(S_1\)-Initialize procedure (resp. \(D_1\)-Unlearning).

In the following subsections, we define three correctness properties and five practicality goals that should be satisfied by PoUL.

B. Correctness Properties

1) Target Model Correctness: A currently predictive model \(M'_1\) yielding a prediction \(p\) respective to a test data \(t\), is a correct target model \(M_1\), if the statement \(p \leftarrow G(M'_1, t) \land M'_1 = H^{-1}(M_1) \land M_1 \leftarrow F(M_0, D)\) is true, where \(G, F\) and \(M_0\) are publicly known.

2) Unlearning Correctness: The unlearning process for deleting the \(d_u\) from the target model \(M_1\) is correct, if \(M_2 \leftarrow F(M_1\backslash\{m_u\}, D\backslash d_u)\) is true, where \(\{m_u\}\) included in \(M_1\) is the part exactly impacted by the \(d_u\).

3) New Model Correctness: A newly predictive model \(M'_2\) yielding a new prediction \(p'\) for a new test data \(t'\), is a correct new model with respect to the deleted \(d_u\), if the statement \(p' \leftarrow G(M'_2, t') \land M'_2 = H^{-1}(M_2)\) is true.

Specifically, in the setup phase, we require guaranteeing the target model correctness, such that the server cannot fake a target model that never learns the data owner’s data, and claim unlearning from it in the later deletion phase. In the deletion phase, unlearning correctness and new model correctness should be ensured, such that the server cannot execute an incorrect unlearning process, and cannot fork an arbitrary model for subsequent predictions while claiming the fulfillment of unlearning. If the above correctness properties are satisfied, the data owner can be convinced that his/her deletion request is truly addressed by the server.

C. Practicality Goals

1) Generic Model Supports: Data deletion requests can occur in generic ML model scenarios, as previous unlearning approaches [13] supported. The definition of PoUL should cover generic ML models.

2) Limited Invasiveness: We desire that proof generation has little modification effect on the underlying learning and unlearning pipelines, and should not compromise model accuracy.

3) Minimal Overhead: Proof generation should not incur additional unaffordable overhead to the already intensive workloads.

4) Concise Proof: Proof size should be short, compared to giving a PoL document on training trajectory [45].

5) Efficient Verification: Verification cost is expected to be small and constant, faster than re-executing and validating partial training trajectory as [45].

V. OUR DESIGNS

We present our SGX-protected designs for the PoUL on top of the SISA unlearning framework, with a partition of an authentication layer and a proving layer. We mainly place the large authenticated storage with simple operation logic outside the enclave, and preserve complex provable execution logic but minimal storage costs in the enclave. This section begins with two challenges of implementing PoUL, considering the server equipped with an SGX enclave. In Section V-B, we customize data structures for authenticating learning data and intermediate submodels. In Section V-C, we integrate the customized data structures with a proof protocol for realizing PoUL.
A. Challenges and Solutions

Although Intel SGX provides the integrity property that is important to our scenario problem, its inherent limitations make us encounter challenges in order to efficiently implement PoUL. We next introduce two main challenges, with respect to the authentication layer and the proving layer we need.

1) Authentication Layer: How to efficiently track the lineage from authenticated learning data to learned intermediate submodels while supporting authenticated deletion?

It is challenging due to the incompatibility of memory-constrained SGX enclave and memory-intensive workloads. Combining with the SISA training process in Section II-A, one possibility of implementing the authentication layer is to move data and submodels outside the enclave (e.g., persistent memory or disks), and meanwhile, adopt appropriate compact ADS to authenticate them as well as maintain the lineage from the data to the corresponding submodels. In particular, we may leverage two same-size Merkle hash trees (MHTs) to separately package data and the corresponding intermediate submodels, aligning with sliced indices in SISA. Their lineage is preserved, by making the location of a data slice in one MHT consistent with that of the earliest intermediate submodel influenced by the data slice within another MHT. Lastly, the two entire MHTs reside out of the enclave and their constant-size roots are stored in the enclave for integrity check. Such adoption, however, is not efficient, in light of the costs to update the two MHTs at worst-case unlearning of SISA. That is, (op₁) deleting a data piece from the first data slice of a shard, and simultaneously, (op₂) updating a sequential of submodels affected by the data piece. We suppose n_d and n_m leaves in the two MHTs. Then, op₁ consumes n_d × log n_d + log n_d hash evaluations for checking the integrity of n_d data slices and updating the first data slice. op₂ needs 2×n_m × log n_m hash evaluations for integrity check and update on the n_m submodels. To assure correctness, both op₁ and op₂ also need to be operated in the secure enclave, and thus unfortunately it is not cost-efficient.

Solution: We rethink that the necessity of adopting the above ADS is aimed to track the data lineage correlated with the intermediate submodels while supporting dynamic updates. To relieve the strict configuration on the ADS and avoid tree traversals, we let the enclave itself achieve the data lineage tracking with correctness guarantees, by designing highly memory-efficient data structures (i.e., compact index list and filter) and cheap pointers interlinking data slices with correlated submodels out of the enclave. Besides, data and submodels are authentically saved outside the enclave, and are protected against potential integrity attacks, with our strategical data structure designs in the enclave.

2) Proving Layer: How to enable efficient verifiability on the correctness of predictive models and incremental learning which involve stateful computations, using previously authenticated submodels and learning data?

This challenge is caused by the conflict between our essential requirements on attesting stateful computations, e.g., multiple incremental learning processes, and the SGX’s protection primarily on stateless in-memory computations, which cannot allow a low and constant verification cost. To be specific, we require the enclave to execute each incremental learning process, taking as input a most recent submodel and newly incremental learning data, and finally yielding a new submodel. A verifier then can assert the correctness of the new submodel by verifying a signature. Due to the incremental nature, the verifier who asserts the correctness of a predictive model (i.e., the final submodel) needs to assert that all submodels learned prior to this final submodel are correct. For example, when the data owner wants to verify if a newly predictive model is not trained on her data piece originally falling in the most beginning data slice, she needs to verify the correctness starting from the most beginning submodel till the final one. Such verification overhead linearly depends on the number of submodels affected by the data slice.

Solution: We enable a constant verification cost by letting the verifier only assert the correctness of the final submodel, but it in turn requires extending the SGX’s protection to stateful computations. Therefore, we consider a self-verification method. That is, before verifying the final model, the authenticity of all prior submodels is verified by the enclave itself, by securely retaining unique randomnesses (named seed) respective to each submodel. To further assert the correctness of subsequent predictions, we additionally introduce a trusted enclave, serving as an auditing enclave. The auditing enclave is responsible to send challenges to the previous enclave for execution, checkpoint the attestation proofs with regard to prediction correctness, and verify the proofs. Once the auditing enclave catches an incorrect execution, it can generate an alert report, along with an attestation proof. Resorting to the auditing enclave, the data owner only needs to verify the auditing enclave’s attestation proof to determine whether subsequent predictions are offered by a correct model. It is also noteworthy that the introduction of the auditing enclave can support third-party auditing.

B. Designing Data Structures for Authentication Layer

To overcome the first challenge, we customize data structures inside and out of the enclave for our logical authentication layer, as Fig. 3 shown. We specifically adopt highly memory-efficient index list and cuckoo filter in the enclave, and use cheap pointers to interlink data slices with correlated submodels out of the enclave. With the data structures, we are aimed to (a) authentically store learning data and submodels out of the enclave, (b) efficiently track the lineage from data slices to learned submodels, (c) support fast deletion and update on the submodels upon receiving a request of deleting a data point, while maintaining the traceable lineage.
1) Data Structures: There are two out-enclave structures and two compact in-enclave data structures.

- data_store: a list for storing sliced learning data and the corresponding message authentication codes (MACs). Its basic unit is respective to a single data point. We note that multiple disjoint data points compose one data slice. The linkability between two neighboring submodels is enforced by the key_list introduced later.
- model_link: a linked list for storing submodels and the integrity MACs. Its basic unit is for one submodel which is learned over the learning data added with one new data slice. The linkability between two neighboring submodels is enforced by the key_list introduced later.
- key_list: a list for storing the keys respective to each data point. Its purpose is to efficiently fetch data and submodels and let data slices and submodels store in a correct order outside the enclave.
- filter: a succinct structure for packaging data points while supporting deletion and membership query. It is operated in the enclave at the very beginning to authenticate data. It supports fast deletion on a data point, and tells a learning program that this data point is not allowed to be used.

2) Detailed Field Description: We now correspondingly describe concrete fields of the filter, key_list, data_store and model_link. We note that each data point is represented as a form of $<\text{kid}, \text{data}>$, where kid is an identifier of the data with a consensual hash function, e.g., non-cryptographic xxHash function.

filter: it is a stateful cuckoo filter residing within the enclave, filled with the fingerprints of the data points. The fingerprints are generated as the form of $\text{Enc}(\text{kid}|\text{data}|\text{eid})$, where $\text{Enc}$ can be a lightweight encryption algorithm used by [65], and $\text{eid}$ is the identity of an initialized enclave. The fingerprints can further be truncated by selecting optimized parameters. For example, we use 8 bits per data point for the Purchase dataset. When a request of deleting a data point is raised, the data fingerprint is removed from the cuckoo filter. Moreover, the filter helps the enclave in checking if a loaded data point is matched with the respective kid, besides checking its MAC.

key_list: it maintains a list (or skip list) of keys about the data points, where each key entry has five fields as following. (1) kid serves as the data index. The orders of the indices enable us to determine the orders of storing and restoring data points (and data slices) plus submodels out of the enclave, and fast fetch them into the enclave in learning and unlearning via pointers. We can see two pointer fields *data and *model later. (2) tag is a binary field for indicating whether the data point is inserted into or deleted from the cuckoo filter. With the tag, we can avoid repeatedly querying and inserting data points in the filter from scratch, especially in case of deleting a data point from one data slice and relearning over the remaining data. More importantly, we can use the tag to mitigate false positives caused by the existence of certain data point that just shares the same fingerprint with the deleted data point, when looking up the deleted data point later. (3) *data is used to find the corresponding data point. (4) *model is used to connect a data slice with the corresponding submodel which is just trained over the training data that just recently adds this data slice. Note that one data slice can contain multiple data points, and we let its last data point’s key be responsible for linking the corresponding submodel. (5) seed is prepared for storing the submodels with freshness guarantees. This seed is generated when a new submodel is built within the enclave, which is filled with a unique and unpredictable randomness. We assign the fresh seed to the key of the last data point within a data slice for ease of retrieval, resorting to the fact that the key is responsible for linking with the submodel we describe before.

data_store and model_link: each entry of the data_store stores a data point and its MAC $\text{dmac}$, and similarly, each entry of the model_link stores a submodel and the integrity MAC $m_s_mac$. We can check a data point’s integrity by verifying its MAC and then querying filter with $\text{Enc}(\text{kid}|\text{data}|\text{eid})$. When resorting a submodel, we check the integrity by retrieving the respective seed and rebuilding its MAC $m_s_mac$, so as to verify the authenticity of the submodel. We note that the submodels and the integrity MACs can be stored in disks or persistent memory, while during learning certain submodel, its previous submodel can reside in DRAM memories.

3) Mitigating Integrity Violation Attacks: We consider potential threats violating the integrity of out-enclave data and submodels. We include the enclave identity within each fingerprint to protect against forking attacks, so as to prevent the server from scheduling multiple enclave instances to conduct the same unlearning task. We also use the enclave filter to protect each data point against replacing attacks, e.g., replacing a data point with one not matching a particular kid. As for model checkpoints, we generate fresh and fresh seeds respective to each newly generated submodel, and associate the in-enclave seeds to their integrity MACs. With the enforcement, relocation attacks and rollback attacks, e.g., using a valid submodel from a different or stale address to replace the real stored one, are mitigated.

C. Designs on Proving Layer

We are ready to implement the proving layer, by integrating the previous data structures with SGX enclaves to address the second challenge. We start by using the data structures to make the data and submodels operate in an authenticated manner, and meanwhile, scheduling an execution enclave to enforce the correctness of the learning and prediction computations associated with the data and submodels. It enables a data owner to assert the correctness of the associated computations on deleting a specific data point, by challenging a newly predictive model and verifying attestation proofs. Later, we introduce an auditing enclave to monitor the subsequent correctness of prediction services, aiming to relieve the data owner’s verification cost.

1) The ProtPULSGX Protocol: From a commit-and-prove perspective, we leverage an enclave to ensure that learning processes are fed with the learning data consistent with pre-committed data as a form of in-enclave filter. Similarly, we enforce that the later prediction process uses a newly yielded submodel matching the last committed submodel in model_link. Next, in Fig. 4, we elaborate the protocol based on Tramèr et al.’s “commit-and-prove” functionality
executing the steps in line 15-24, and the last submodel is queried by new test data. The execution processes are similar to that in the setup phase, with the main difference that the new model \( h'_d \) does not use the data point \( d_u \).

2) Verifying Correctness Properties: With the \( \text{ProPUL}_\text{SGX} \) protocol, the data owner can assert whether the server offers a fake target model, or executes an incorrect unlearning process, or uses an incorrect unlearned model for new predictions, with respect to the deletion on \( d_u \in D \). The correctness properties are verified based on some necessary assumptions: the SGX enclave guarantees integrity and partial confidentiality (i.e., sealing randomness and secret keys); the associated signature scheme \( \Sigma \) satisfies unforgeable under chosen message attacks (EU-CMA) security; the hash functions used for generating the enclave identity and integrity MACs are collision-resistant; lastly, all programs running inside the enclave are bug-free.

a) Verifying target model correctness: The data owner can verify that the predictive model is indeed the incrementally learned model \( h'_d \) over the data \( D \), with the following three steps. First, the data owner can verify the authenticity of the filter \( c \) and the key list, namely, correctly packing her data, by asserting the validity of \( \sigma_d \) and querying the data point membership \( d_u \) with the filter, see line 32. Second, the data owner can assert that the last submodel \( h_{\text{model}} \) is incrementally learned over the data existing in the filter \( c \), on the promise of the validity of \( \sigma_d \), the integrity of on-the-fly batches of data and the previous submodels, and the confidentiality of randomly generated \( \text{seed} \). This step relies on the attestation proof \( \sigma_m \) and be verified in line 34. Finally, the data owner asserts that the predictive model is the above submodel \( h_{\text{model}} \) by verifying \( \sigma_p \) in line 36.

b) Verifying unlearning correctness: The data owner firstly can verify the validity of a new attestation proof \( \sigma'_d \) on the updated filter and key list after the deletion operations. Particularly, the deletion operations on the key list are twofold: for the \( d_u \)'s key entry, the changes include \( \text{tag}=0 \), *data unlinked to \( d_u \), *model unlinked to the affected submodel, and \( \text{seed} = \text{NULL} \); for other key entries after the \( d_u \)'s key entry in the key list, the *model fields are unlinked to the submodels, and the \( \text{seed} \) fields are set NULL. Next, all affected submodels are relearned on the remaining data until the last submodel \( h'_d \) is obtained, which is enforced by the updated filter and the key list. The data owner secondly verifies that the new last submodel \( h'_d \) is incrementally relearned over the data exactly excluding \( d_u \), by asserting the corresponding signature-based proof \( \sigma'_m \) against the updated filter \( c' \). Note that the correctness of all affected submodels prior to \( h'_d \) are enforced and verified by the enclave itself, with help of our authentication layer.

c) Verifying new model correctness: With the last submodel \( h'_d \) which is newly generated, the data owner can determine whether it is used for prediction on her given test data, by verifying the new signature \( \sigma'_p \) on given new test data \( t^* \) and \( h'_d \).

D. Extension Designs

We proceed to describe some approaches to extend the previous designs to support third-party auditing, multiple data owners and batch data deletion.
1) Introducing an Auditing Enclave: We introduce an additional enclave as a trusted auditor to monitor the correctness of the execution enclave and offer the evidences of incorrect execution for post verification. It derives from our considerations that making the data owner always assert the correctness of each prediction is impractical. Besides, the previous designs might not be scalable to support third-party regulators to audit unlearning, if they have no access to the data owners’ data. We note that the auditing enclave can work in a centralized or decentralized manner, inspired by Paccagnella et al.’s work [56].

The auditing enclave can be setup on the server’s machine and configured with auditing programs, after the interactions between the data owner and the server. The auditing programs define the auditing logic, including (1) reading the inputs and outputs of ECALL functions invoked by the execution enclave, (2) interacting with Intel’s verification server, (3) logging the verification results, and (4) generating reports in the case of verification failure. At the beginning, the auditing enclave can run a key exchange protocol with the execution enclave, so as to build a secure communication channel [10]. Then, the auditing enclave is stably scheduled when using models for predictions.

Specifically, it can involve the following four steps starting from receiving a data owner’s test challenge. (a) A data owner securely transmits test data to the auditing enclave. This test data has previously been marked for deletion by the data owner. The auditing enclave, governed by defined logic (1), is also capable of reading the inputs and outputs of the execution enclave. (b) Subsequently, the auditing enclave conducts various checks. This includes verifying the membership of the test data within the cuckoo filter inside the execution enclave concerning the program marked as prog$. Simultaneously, it confirms whether the current predictive model accurately processes the test data, aligned with the program prog$. The results of these verifications are logged in accordance with mentioned program logic (2) and (3). (c) In addition, the auditing enclave validates whether the predictive model aligns with the one originally trained on the data stored in the cuckoo filter, following the program prog$. This assertion result is also logged. (d) Finally, the auditing enclave generates a comprehensive verification report using logic (4). This report amalgamates the query result from step (a) with the two verification results from step (b) and (c). It serves as a transparent record of the verification process, ensuring clarity and convincing evidence of data deletion. This report can be shared with the data owner or any third parties interested in confirming the deletion of data.

2) Supporting Multiple Data Owners: We consider that the data used to train a predictive model can come from multiple data owners. In such a scenario, data owners cannot mutually access the data of others, and all of them do not trust the server who trains the model. To implement PoUL, we start by sequentially authenticating the data owners’ data with a unique in-enclave filter in the $\text{S1}_{\text{Initialize}}$ procedure. Suppose the first data owner, with her identity $\text{ow}_1$, uploads her data to the server’s execution enclave. The enclave generates the fingerprint of each data point in the form of $\text{Enc}(\text{kid}||\text{data}||\text{eid}||\text{ow}_1)$. Herein, kid is generated by xxHash($\text{ow}_1||\text{data}$). After traversing her data, the second data owner who prepares for uploading data asserts the current states of the execution enclave, by verifying $(\text{sid}, h_c, \text{prog}_c, \text{prog}_l)$, where $h_c$ is the digest of the current filter in the enclave. The above processes repeat until the last data owner’s data is traversed, and the filter at the current time packs all data owners’ data. Next, suppose a Deletion phase is invoked for complying with one data owner’s deletion request which contains the corresponding kids of the data point to be deleted. The execution enclave can retrieve kid = xxHash($\text{ow}_1||\text{data}$), and delete the associated data and submodels. Notice, we consider that the training data points are likely to be overlapping in the multi-owner scenario. Therefore, both keys and fingerprints should include the unique identity information of data owners for identifiability, as described above.

3) Enabling Batch Data Deletion: To delete data in a batch fashion can be necessary, since proving the desired correctness properties with respect to one data point deletion is already cost-consuming for a service provider. With the concern, the server might wait for deletion requests from the data owners, and record the corresponding kid, and after a waiting period, he executes a Deletion phase in batch, starting from the kid at the foremost location. We argue such a processing is reasonable, since there often allows a time period to comply with the deletion requests, e.g., one month regulated by the U.K.’s Information Commissioner’s Office.²

VI. IMPLEMENTATION AND EVALUATION

We implement our SGX-capable PoUL instantiation with the logically authentication layer and proving layer. Based on the implementation, our evaluation answers the questions as follows: (Q₁) how is the additional storage cost for PoUL, standing on the already storage workloads of the SISA framework? (Q₂) how does our authentication layer perform, compared against an MHT-based implementation? (Q₃) how are the learning/unlearning time complexity and trained accuracy in the SGX-capable PoUL?

A. Implementation and Setup

We begin with the implementation idea of the SISA framework, and run the training process over a single shard of data in an SGX enclave. We omit other shards’ training processes, since they can be securely executed in the same manner, and thus we measure prediction accuracy of a single-shard model.

1) SGX Enclave: We install SGX SDK of version Linux 2.16 to initialize the SGX enclave environment. We also utilize the Intel’s SGX DNNL Library with version 1.1.1 to bootstrap our training tasks. All codes are run in hardware mode. Besides, we configure the heap size to 90,000,000 (nearly 2,304 MB) to utilize memory resources external to the SGX enclave through page swapping.

2) In-Enclave Programs: As we described in Section V-C, the programs are implemented in C++ majorly including (1) prog$ for building a cuckoo filter by using Fan et al.’s public

²https://ico.org.uk/for-organisations/guide-to-data-protection/
library, (2) \texttt{prog}_k for generating a list for data keys, (3) \texttt{prog}, for implementing learning and unlearning a specific model guided by the SISA framework, and (4) \texttt{prog}_p, for implementing a specific prediction process by restoring a newly trained model via \texttt{prog}.

3) Model and Dataset: Following the SISA framework's open source library, we adopt a model with two fully connected (FC) layers and re-implement it in C++. Each FC layer is followed with an activation layer, and the output layer uses one-hot encoding for two classes. The model is trained with a mini-batch stochastic gradient descent (SGD) algorithm, and evaluated over the Purchase [62] dataset. The Purchase dataset is divided into a training set with 280367 data points and a test set with 31152 data points.

4) Setup Configuration: We configure the setup information about training the model on the Purchase dataset using the SISA framework. Concretely, we divide the Purchase dataset into 5 shards, and each shard contains 56073 data points. We then further slice one-shard data points with a range of slice size in \{1, 3, 6, 12\}. For a training case with a fixed slice size, we adaptively select the training parameters, such as the batch size, epoch number and learning rate, leading to a trained model with best accuracy.

Besides, our experiments are carried out on an Ubuntu 20.04 server equipped with Intel® Xeon(R) CPU E3-1505M v5 @ 2.80GHz ×8 CPU and 14.6 GiB of RAM.

### B. Evaluation

Firstly, we measure the storage costs of our customized data structures, when building the SISA training framework over the Purchase dataset. We select appropriate parameters, such as the size of each item in the filter and the size of each hidden layer in a model, which are the main impact factors for the storage costs. On the Purchase dataset with 56073 data points in one shard, we let the filter have $2^{16}$ buckets and the fingerprint size of each item be 12 bits, such that the false positive rate (FPR) for membership query is low enough, nearly 0.003. As a result, the filter filled with the one-shard data leads to 192 KB, as presented in Table I. We can also lower the fingerprint size till 8 bits to obtain a smaller-size filter, but we suffer from a higher FPR value, about 0.041. Next, the key_list is in about 2.85 MB size, in which each entry needs 416 bits. The last two storage sizes are majorly dominated by the data scale (i.e., 600 nodes in the input layer) and the number of nodes in the FC layer (i.e., 128 nodes), on the case of using precision training with 32-bit floating point numbers. They are the storage workloads already needed by the SISA training. Therefore, our authentication layer additionally incurs around 3.00 MB storage workloads for training on the Purchase dataset, answering the Q1 question.

$$\text{TABLE I}$$

**STORAGE SIZE OF CUSTOMIZED DATA STRUCTURES**

| Dataset  | Filter key_list | Data Store model_link |
|----------|-----------------|-----------------------|
| Purchase | 192 KB 2.85 MB  | 133.17 MB 1.81 MB     |

Secondly, we make a further effort to evaluate the runtime performance of our authenticated layer, compared to a baseline implementation based on an MHT. As we introduced before, we setup a cuckoo filter inside the enclave for authenticating one shard of data points and later authentically deleting data points. Here, we compare the performance of insertion, query and deletion operations with that of an MHT using the SHA-256 hash function, over the Purchase dataset. For demonstrating a best-case comparison, we let the fingerprint size be only 8 bits. To the end, we deduce the time consumption for inserting, querying and deleting one data point in average over the cuckoo filter and MHT, as summarized in Table II. To respond to the Q2 question, our implementation with respect to one data point can achieve about 185, 360 and 340 times time savings in insertion, query and deletion, respectively.

![Fig. 5. Impact of unlearning one data point.](image)

We are ready to evaluate the learning/unlearning time and accuracy in the SGX-capable setting for answering the Q3 question. On top of our setup configuration with six sequential slices, we measure the unlearning time respective to the positions of the deleted data point, i.e., the slice it falls in. We also evaluate the individual learning time of each slice before unlearning for comparison. It is easy to observe from Fig. 5(a) that learning time turns longer as the number of slices increases, due to that more data need to be trained. As for unlearning, it consumes more time when the deleted data point falls in the slice with smaller indices, since more impacted submodels need to be retrained. During the training processes, we notice that model checkpoint and restoring in such an SGX setting take negligible time, compared to the learning/unlearning time, as shown in Fig. 5(b). Besides the time consumption, we also compare the corresponding trained accuracy in Fig. 5(c). We use the trained accuracy 95.14% before unlearning as a baseline (shorten as BL in the figure). We discover from the results that the case of single data point deletion may not influence the original trained accuracy, which might be aligned with a recent study’s conclusion [79]. It is noteworthy that due to the memory restriction of the SGX enclave, the trained accuracy mentioned before is obtained by carefully tuning hyperparameters during the SGD training. Fig. 6 demonstrates the improvement of trained accuracy when (a) the batch size is fixed with 1000 while the epoch

---

3https://github.com/efficient/cuckoofilter

4https://github.com/cleverhans-lab/machine-unlearning
number increases till 22; and (b) the epoch number is fixed with 20 while the batch size decreases to 1000. As a result, we obtain the baseline accuracy in 95.14% by training with 22 epochs and a batch size of 1000.

VII. ANALYSIS OF RELATED WORK

A. Auditing Unlearning

Auditing unlearning implies that a data owner or third-party regulators can examine if the revoked training data indeed do not exist in an unlearned model residing on a server. That may relate to a long line of work in auditing the integrity of remotely stored data [17], [82], [83], but the focuses here are placed on the model derived from data. Existing methods [30], [40], [50], [70] to audit unlearning essentially examine the existence or absence of certain fingerprint of deleted data left on an unlearned model, independent of unlearning techniques. The fingerprint effect either merely relies on original training data [40], [50], or is strengthened via backdoored triggers [70] and recent watermarking and fingerprinting strategies [30]. The examination is generally performed by means of black-box querying [40], [50], [70], aligned with the MLaaS paradigm that offers black-box APIs to users for querying server-side models. Although the SISA unlearning method can also be audited by letting an auditor re-execute the unlearning process with the submodel checkpoints and the corresponding dataset, the auditing manner cannot protect the intellectual property of the submodels, and may not support efficient verification as mentioned in Section IV-C.

Different from prior efforts, our work opens a new auditing perspective, in light of a dishonest server who can conduct forging and forking attacks. For example, the server can forge a model which has an indistinguishable model distribution with that of the target model in question (consider deep neural network models), but the model is never trained on requested data that will be deleted [67], [73]. Then, the server forks the target model in question using the forging model, by resorting to the black-box service manner, and claim having deleted data from the forging model while still offering prediction services with the target model. In such case, neither previous verification metrics via comparing model distribution, nor the above auditing methods via probing fingerprint, can faithfully tell if the requested data is indeed deleted.

Our established definition framework stands independently of unlearning methods, even though we solely provide a specific example based on SISA. For a more comprehensive explanation, aside from the quantization-based k-means deletion method proposed by Ginart et al. [31], we can also use an authenticated data structure (i.e., a cuckoo filter inside the SGX enclave) to package data points, and remain the lineage of data points and the centroid during training inside the enclave. When receiving deletion requests, the enclave could retrain or recompute related centroids to reflect deletion operations. It is important to note that an essential principle of providing an instantiation is the simultaneous enforcement of the properties of target model correctness, unlearning correctness, and new model correctness, as previously defined in Section IV-B.

B. TEEs-Based Verifiable ML

In outsourced ML scenarios, TEEs become a practical tool to enforce computation correctness, either in the training process [11], [39], [41], [42], [52], [54], [87], [89] or the inference process [38], [48], [74], [75]. Our work departs from prior implementations merely for inference, but closely relates to the arts, namely [11], [39], [41], [87], [89] for implementing an integrity-protected training (and prediction) pipeline. Among these arts, very few of them immediately support authentically tracking, storing and updating the lineage of training data in intermediate models, which are particularly required to realize our PoUL.

1) Comparison: TrustFL [89] and Plinius [87] are potential to meet our requirements, but they are still not satisfactory. TrustFL is a TEE+GPUs solution for integrity-preserving federated learning, with the main ideas including (1) individual participants conduct all training iterations on a GPU processor and authentically stores all training intermediate models in alignment with authenticated batches of data; (2) a co-located TEE verifier can randomly re-execute several successive iterations of training for correctness checking after training. To authentically store models outside the TEE, it adopts an MHT by packing each intermediate model as a leaf, while preserving the tree root inside the enclave for verification (suppose n leaves in the tree). When the TEE restores an intermediate model, it needs to compute O(log n) hashes along the path from the leaf with respect to this intermediate model to the root, so as to verify the integrity. Such design is not suitable to realize the PoUL, since we require updating a large number of intermediate models upon receiving a single deletion request, which needs verifying their hashes and computing new hashes in the memory-limited TEE. As for Plinius [87], it focuses on storing models in the processor-accessible persistent memory (PM) collocated with a TEE. The authors study how to efficiently copy model checkpoints from the secondary storage to the persistent memory, with the eventual aim of making the TEE fast restore the model checkpoints. Although it does not demand designing efficient data structures to store model checkpoints with dynamic deletion supports, the idea of incorporating efficiently accessible PM is complementary to our work.

2) Challenging Issues of Realizing PoUL by Combining GPUs With TEEs: To make our TEEs-based PoUL instantiation salable to real-life deep neural networks, a suggestive direction is to remove workloads from CPUs-embedded TEEs to GPUs, as promoted by existing works on the training
phase [11], [39], [52], [89]; Yet, it is challenging and to be answered by more ongoing rigorous efforts with the previous lessons. This line of works usually provides probabilistic verification on the integrity of computations delegated to distrustful GPUs by leveraging additional integrity-enhancing algorithms, either delegating all training iterations, e.g., TrustFL [89] and GINN [11] or solely partial linear computations, e.g., Goten [52] and DarKnight [39]. In addition to integrity-enhancing algorithms, GINN also creatively uses gradient clipping to defend against a fine-gained tampering attack on a single SGD-update step. Despite the inspiring efforts, following their lessons to realize PoUL needs more exploration, in order to guarantee integrity and practical performance like accuracy, storage overhead and communication complexity caused by frequent system calls. One big issue is simultaneously ensuring unlearning accuracy and integrity, considering the following two lessons: First, Freivalds’ algorithm for integrity checking [11], [52], [75] requires carefully mapping the original floating-point numbers into fixed-point ones for maintaining accuracy. Second, the adoption of gradient clipping demands appropriately adjusting hyperparameters like clipping rate and learning rate for retaining performance, as evaluated by GINN. Thus, combining GPUs with TEEs for implementing PoUL requires exploring the answer to how the above defences against integrity violation impact unlearning accuracy in the SISA setting.

VIII. Conclusion and Future Work

We respond to the need for auditing unlearning in light of the latest attacks on unlearning. We define the new problem on Proof of Unlearning from the perspective of VC, which is allowed to be realized using various VC techniques. As an initial effort to push forward auditable unlearning, we propose a native implementation based on Intel SGX. Standing on top of the initial effort, we collate future work directions as following: (i) Scaling to specialized hardware accelerators. We pursue more efficient auditing in the TEEs setting empowered with specialized hardware accelerators, by incorporating the available optimizing compilers or adapting to the accelerators with the TEEs capability. (ii) Enabling flexible deletion. We seek to explore auditing the compliance of deleting inappropriate sensitive attributes, e.g., race or gender, which can be included in a group of data points, rather than individual data points. To support deleting the group of data is necessary, as the ML fairness problems become increasingly noticeable (e.g., in face recognition system). (iii) Protecting data content and privacy. To fully respect privacy regulations, auditing unlearning may stand on the privacy-preserving machine learning/unlearning pipeline, in which data content and data privacy are protected. Towards this direction, extending our SGX-capable PoUL to work jointly with data-oblivious implementations and differential privacy mechanisms will be explored.

APPENDIX

A. Impact of Varying Number of Slices

Recall that we fix the number of slices at 6 for our previous evaluation. We now vary the number of slices in one shard, and observe if it impacts the training time and accuracy on the Purchase dataset. We divide the one-shard training data into 1, 3, 6 and 12 slices, and correspondingly execute the incrementally training processes over them. As Fig. 7 demonstrated, the number of slices makes a really little effect on both the accuracy and the training time.

B. Experiments Over the Image Dataset

We here present the experimental results that use the LeNet convolutional neural network model running over the MNIST dataset. The MNIST dataset contains 60K training samples and 10K test samples. We use the 60K training samples to simulate one shard of data. The shard of data is further divided into six slices of data and each slice contains 10K samples. Note that we conduct the experiments in an Ubuntu 16.04 server equipped with a CPU of 3.40GHz, 16 GB RAM and a GPU of Nvidia GTX-1080.

We first demonstrate the storage size of different data structures for the 60K samples in TABLE III. Note that we use the same filter with \(2^{16}\) buckets as our previous experiments to package the samples, and thus we do not present it in the table. Overall, we also require around 3.00 MB (actually 3.17 MB) for the authentication layer in this experiment.

Next, we assess the insertion time, query time, and deletion time associated with managing a data sample using the cuckoo
of slices within the range of changes in model accuracy. Initially, we modify the number of slices, the epoch size and batch size, to observe consistently low.

Additionally, we tune training parameters, including the number of slices, the epoch size and batch size, to observe changes in model accuracy. Initially, we modify the number of slices within the range of \{1, 2, 3, 9, 12\}. Subsequently, we set the batch size to 256 while adjusting the epoch size. Following this, we keep the epoch number fixed at 5 and incrementally increase the batch size up to 256. The corresponding accuracy variations are illustrated in Fig. 8 and Fig. 9. These experimental results enable us to identify and select the most suitable parameters for achieving high model accuracy.


table IV

| Structures   | Insertion | Query     | Deletion |
|--------------|-----------|-----------|----------|
| Cuckoo filter| 0.020     | 0.020     | 0.023    |

Table IV: Runtime Performance with the Cuckoo Filter (µ s)

Fig. 9. Accuracy improvement via training parameter tuning over MNIST.

The evaluation results are presented in TABLE IV. It is evident from the results that the operational costs remain consistently low.

REFERENCES

[1] Guidance on the AI Auditing Framework. Accessed: 2020. [Online]. Available: https://ico.org.uk/about-the-ico/ico-and-stakeholder-consultations/ico-consultation-on-the-draft-ai-auditing-framework-guidance-for-organisations/

[2] (2015). Intel(r) Software Guard Extensions (Intel SGX). [Online]. Available: https://software.intel.com/sites/default/files/332680-002.pdf

[3] (2018). California Consumer Privacy Act (CCPA). [Online]. Available: https://oag.ca.gov/privacy/ccpa

[4] (2018). Facebook Blames a Bug for Not Deleting Your Deleted Videos. [Online]. Available: https://newyork.cbslocal.com/2018/04/03/facebook-deleted-videos-bug/

[5] (2020). 10th Generation Intel Core Processor Families Data Sheet. [Online]. Available: https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/10th-gen-core-families-datasheet-vol-1-datasheet.pdf

[6] (2023). Microsoft Azure Machine Learning (Azure ML). [Online]. Available: https://studio.azureml.net

[7] T. Abera et al., “C-FLAT: Control-flow attestation for embedded systems software,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Oct. 2016, pp. 743–754.

[8] Alibaba. Alibaba Cloud. Accessed: 2021. [Online]. Available: https://www.alibabacloud.com/product/ecb

[9] T. Alves and D. Felton, “Trustzone: Integrated hardware and software security,” Cherry Hinton, Cambridge, U.K., ARM White Paper, 2004, vol. 3, no. 4.

[10] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, “Innovative technology for CPU based attestation and sealing,” in Proc. HASP, vol. 13, 2013, pp. 1–7.

[11] A. Asvadishirejini, M. Kantarcioğlu, and B. Malin, “GINN: Fast GPU-TEE based integrity for neural network training,” in Proc. 12th ACM Conf. Data Appl. Secur. Privacy, Apr. 2022, pp. 1–15.

[12] Dyslexic Atheist. (2020). Instead of Deleting Account, New York Times Appends ‘1000’ to Username and Email Address. [Online]. Available: https://news.ycombinator.com/item?id=23005060

[13] L. Bouroule et al., “Machine unlearning,” in Proc. IEEE Symp. Secur. Privacy (SP), May 2021, pp. 141–159.

[14] E. Brickell and J. Li, “Enhanced privacy ID from bilinear pairing for hardware authentication and attestation,” Int. J. Inf. Privacy. Secur. Integrity, vol. 1, no. 1, pp. 3–33, 2011.

[15] Y. Cao and J. Yang, “Towards making systems forget with machine unlearning,” in Proc. IEEE Symp. Secur. Privacy, May 2015, pp. 463–480.

[16] N. Carlini, C. Liu, U. Erlingsson, J. Kos, and D. Song, “The secret sharer: Evaluating and testing unintended memorization in neural networks,” in Proc. USENIX Secur., 2019, pp. 267–284.

[17] B. Chen and R. Curtmola, “Robust dynamic provable data possession,” in Proc. ACM Conf. Comput. Commun. Secur., 2019, pp. 623–635.

[18] C. Chen, F. Sun, M. Zhang, and B. Ding. “Recommendation unlearning,” in Proc. ACM Web Conf., Apr. 2022, pp. 2768–2777.

[19] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai, “SgxPecture: Stealing Intel secrets from SGX enclaves via speculative execution,” in Proc. IEEE Eur. Symp. Secur. Privacy, Jun. 2019, pp. 142–157.

[20] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang. “When machine unlearning jeopardizes privacy,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Nov. 2021, p. 896–911.

[21] M. Chen, Z. Zhang, T. Wang, M. Backes, M. Humbert, and Y. Zhang. “Graph unlearning,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Nov. 2022, pp. 499–513.

[22] Federal Trade Commission. California Company Settles FTC Allegations it Deceived Consumers About Use of Facial Recognition in Photo Storage App, Federal Trade Commission, America, 2021.

[23] V. Costan and S. Devadas. Intel SGX Explained. [Online]. Available: https://eprint.iacr.org/2016/086.pdf

[24] G. Desouy et al., “LO-FAT: Low-overhead control flow ATtestation in hardware,” in Proc. 54th ACM/EDAC/IEEE Design Autom. Conf. (DAC), Jun. 2017, pp. 1–6.

[25] H. Duan, C. Wang, X. Yuan, Y. Zhou, Q. Wang, and K. Ren. “LightBox: Full-stack protected stateful middlebox at lightning speed,” in Proc. ACM SIGSAC Conf. Comput. Commun. Secur., Nov. 2019, pp. 2351–2367.

[26] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, “Cuckoo filter: Practically better than Bloom,” in Proc. 10th ACM Int. Conf. Emerg. Netw. Experiments Technol., Dec. 2014, pp. 75–88.

[27] Conner Forrest. (2017). Dropbox Bug Sends Years-Old Deleted Files Back to User Accounts. [Online]. Available: https://www.techrepublic.com/article/dropbox-bug-sends-years-old-deleted-files-back-to-user-accounts/

[28] J. Gao, S. Garg, M. Mahmoody, and P. N. Vasudevan. “Deletion inference, reconstruction, and compliance in machine (un) learning,” in Proc. PETS, 2022, pp. 1–31.

[29] M. Gao, H. Dang, and E.-C. Chang. “TEEKAP: Self-expiring data capsule using trusted execution environment,” in Proc. Annua. Comput. Secur. Appl. Conf., Dec. 2022, pp. 235–247.

[30] X. Gao et al., “VeriFi: Towards verifiable federated unlearning,” 2022, arXiv:2205.12709.

[31] A. Gintal and M. Guan, “Making ai forget you: Data deletion in machine learning,” in Proc. NIPS, 2019, pp. 1–14.

[32] A. Golatkar, A. Achille, and S. Soatto, “Forgetting outside the box: Scrubbing deep networks of information accessible from input-output observations,” in Proc. ECCV, 2020, pp. 383–398.

[33] A. Golatkar, A. Achille, and S. Soatto, “ Eternal sunshine of the spotless net: Selective forgetting in deep networks,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. (CVPR), Jun. 2020, pp. 9301–9309.

[34] S. Greengard, “Can AI learn to forget?” Commun. ACM, vol. 65, no. 4, pp. 9–11, Apr. 2022.

[35] J. Groth, “Short pairing-based non-interactive zero-knowledge arguments,” in Advances in Cryptology—ASIACRYPT, Berlin, Germany. Singapore: Springer, 2010, pp. 321–340.

[36] C. Guo, T. Goldstein, A. Hannun, and L. Van Der Maaten, “Certified data removal from machine learning models,” in Proc. Int. Conf. Mach. Learn., 2020, pp. 3832–3842.

[37] V. Gupta, C. Jung, S. Neel, A. Roth, S. Sharifi-Malvajerdi, and C. Waite, “Adaptive machine unlearning,” in Proc. NIPS, 2021, pp. 1–12.

[38] L. Hanzlik et al., “MLCapsule: Guarded offline deployment of machine learning as a service,” in Proc. IEEE/CVF Conf. Comput. Vis. Pattern Recognit. Workshops (CVPRW), Jun. 2021, pp. 3295–3304.
[39] H. Hashemi, Y. Wang, and M. Annavaram, “DarkKnight: An accelerated framework for privacy and integrity preserving deep learning trusted hardware,” in Proc. 54th Annu. IEEE/ACM Int. Symp. Microarchitect. Oct. 2021, pp. 212–224.
[40] Y. Huang, X. Li, and K. Li, “EMA: Auditng data removal from trained models,” in Proc. MICCAI, 2021, pp. 793–803.
[41] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel, “Chiron: Privacy-preserving machine learning as a service,” 2018, arXiv:1803.05961.
[42] N. Hynes, R. Cheng, and D. Song. “Effective deep learning on multi-source private data,” 2018, arXiv:1807.06689.
[43] IBM. IBM Cloud Data Shield. Accessed: 2020. [Online]. Available: https://www.ibm.com/cloud-data-shield.
[44] Y. Jang, J. Lee, S. Lee, and T. Kim, “SGX-bomb: Locking down the processor via Rowhammer attack,” in Proc. 2nd Workshop Syst. Trust. Executed, 2017, pp. 1–6.
[45] H. Jia et al., “Proof-of-learning: Definitions and practice,” in Proc. IEEE INFOCOM Conf. Comput. Commun. May 2022, pp. 1039–1056.
[46] Y. Liu, L. Xu, X. Yuan, C. Wang, and B. Li, “The right to be forgotten in the internet is a fool’s errand. Accessed: 2015. [Online]. Available: https://www.fastcompany.com/3048871/why-deleting-personal-information-on-the-internet-is-a-fools-errand.
[47] A. Thudi, H. Jia, I. Shumailov, and N. Papernot, “On the necessity of auditabile algorithmic definitions for machine unlearning,” in Proc. USENIX Secur. 2022, pp. 4007–4022.
[48] K. Grover, S. Tople, S. Shinde, R. Bhagwan, and R. Ranjjee, “Privacy-preserving machine learning as a service,” 2018, arXiv:1810.06002.
[49] F. Tramer and D. Boneh, “Slalom: Fast, verifiable and private execution of neural networks in trusted hardware,” in Proc. ICML, 2018, pp. 1–19.
[50] F. Tramer, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, “Stealing machine learning models via predictio APIs,” in Proc. USENIX Secur., 2016, pp. 601–618.
[51] F. Tramer, F. Zhang, H. Lin, J.-P. Hubaux, A. Juels, and E. Shi, “Sealed-glass proofs: Using transparent enclaves to prove and sell knowledge,” in Proc. IEEE Symp. Secur. Privacy, Apr. 2017, pp. 19–34.
[52] S. van Schaik. (2020). SGAXE: How SGX Fails in Practice. [Online]. Available: https://saxgxe.com/files/SGAXe.pdf.
[53] E. F. Villaronga, P. Kieseberg, and T. Li, “Humans forget, machines remember: Artificial intelligence and the right to be forgotten,” Comput. Law Secur. Rev., vol. 34, no. 2, pp. 304–313, Jun. 2018.
[54] M. Walifsh and A. J. Blumberg, “Verifying computations without reexecuting them,” Commun. ACM, vol. 58, no. 2, pp. 74–84, Jan. 2015.
[55] C.-L. Wang, M. Huai, and D. Wang, “Inductive graph unlearning,” in Proc. USENIX Secur. 2023, pp. 1–25.
[56] C. Wang, Q. Wang, K. Ren, and W. Lou, “Ensuring data storage security in cloud computing,” in Proc. 17th Int. Workshop Quality Service, Jul. 2009, pp. 1–9.
[57] C. Wang, Q. Wang, K. Ren, and W. Lou, “Privacy-preserving public auditing for data storage security in cloud computing,” in Proc. IEEE INFOCOM, Mar. 2010, pp. 1–9.
[58] Y. Wang, G. Norcie, S. Komanduri, A. Acquisti, P. G. Leon, and L. F. Cranor, “‘I regretted the minute I pressed share’ a qualitative study of regrets on Facebook,” in Proc. SOUPS, 2011, pp. 1–16.
[59] A. Warnecke, L. Pirch, C. Wressnegger, and K. Rieck, “Machine unlearning of features and labels,” 2021, arXiv:2108.11577.
[60] C. Weng, K. Yang, J. Katz, and X. Wang, “Wolverine: Fast, scalable, and communication-efficient zero-knowledge proofs for Boolean and arithmetic circuits,” in Proc. IEEE Symp. Secur. Privacy (SP), May 2021, pp. 1074–1091.
[61] P. Yuhala, P. Felber, V. Schiavoni, and A. Tchana, “Plinius: Secure and persistent machine learning model training,” in Proc. 51st Annu. IEEE/ACM Int. Conf. Dependable Syst. Netw. (DSN), Jun. 2021, pp. 52–62.
[62] R. Zhang, J. Liu, Y. Ding, Q. Wu, and K. Ren, “‘Adversarial examples’ for proof-of-learning,” in Proc. IEEE Symp. Secur. Privacy, May 2022, pp. 1408–1422.
[63] X. Zhang, F. Li, Z. Zhang, Q. Li, C. Wang, and J. Wu, “Enabling execution assurance of federated learning at untrusted participants,” in Proc. IEEE INFOCOM Conf. Comput. Commun., Jul. 2020, pp. 1877–1886.
[64] L. Zhao, Q. Wang, C. Wang, Q. Li, C. Shen, and B. Feng, “VeriML: Enabling integrity assurances and fair payments for machine learning as a service,” IEEE Trans. Parallel Distrib. Syst., vol. 32, no. 10, pp. 2524–2540, Oct. 2021.
[65] W. Zheng, R. A. Pope, J. E. Gonzalez, and I. Stoica, “Helen: Maliciously secure cooperative learning for linear models,” in Proc. IEEE Symp. Secur. Privacy (SP), May 2019, pp. 724–738.