FHorgEt: A Cryptographic Solution for Secure Machine Unlearning

David Balbás
IMDEA Software Institute
Madrid, Spain

Dario Fiore
IMDEA Software Institute
Madrid, Spain

Georgios Raikos
zkSecurity

Damien Robissout
Input-Output Engineering
Madrid, Spain

Claudio Soriente
GMV
Madrid, Spain

Abstract

Data regulations grant users the right to be forgotten, empowering them to control if and when their data is used in applications such as machine learning training. Machine unlearning offers a promising mechanism to enforce this right by enabling the removal of specific training data from models. Existing machine unlearning approaches, however, assume an honest server that correctly executes all unlearning requests. In practice, this assumption is too strong: nothing prevents a server from falsely claiming to have performed unlearning while secretly retaining the original model or continuing to use the data for training. Such behaviours remain possible even when unlearning requests are verifiable—for example, via zero-knowledge proofs—because the server may still keep copies of the data or model.

In this work, we argue that a security model for machine unlearning should capture data confidentiality throughout the lifecycle of a model, including training, inference, and unlearning. We introduce such a formalism and then present the first machine learning framework that provides cryptographic guarantees that unlearning requests are properly executed and that users' data is forgotten. We implement our framework using fully-homomorphic encryption (FHE) and secure multi-party computation (MPC), within a distributed setting where training, unlearning and inference requests are handled by a group of servers. Our constructions are secure in the honest-but-curious model if at least one of the servers is honest, and can be lifted against actively malicious servers following standard techniques. We also show, via a proof-of-concept implementation, that such a system does not add a significant overhead on top of FHE-based training.

Keywords

Machine unlearning, Machine learning security, Fully homomorphic encryption, Secure multi-party computation

1 Introduction

Machine unlearning enables the removal of specific training data from a machine learning model. This capability is crucial both for complying with data-protection regulations—where individuals may request that their personal data be deleted—and for purging harmful, low-quality, or toxic data that can degrade the model performance. In both scenarios, the objective of machine unlearning is to produce a model that behaves as if it were trained on the original dataset minus the items to be removed. A model owner has a natural incentive to unlearn bad or toxic data, since doing so can improve performance. However, when the model owner is asked to remove data solely to satisfy a regulatory request, it lacks any incentive to comply.

Consider a model M that is trained on data $\mathcal{D} = \{x_1, \ldots, x_n\}$ provided by n different users. When the i-th user asks for the unlearning of x_i, the model owner should apply the unlearning algorithm on M, thereby producing an updated model M'—that allegedly has no "memory" of x_i. However, this raises the following questions:

(1) How can one trust that x_i is not used later to train another model that behaves like M?
(2) How can one trust that M—the model trained with x_i—is deleted and no longer used for inference?

Existing machine unlearning frameworks, e.g. [7, 20, 21, 36], assume an honest server that duly executes deletion and unlearning requests in practice, thereby not considering these questions. A step forward was taken in a recent work by Eisenhofer et al. [16] which explores how to enable *verifiability* for unlearned models through verifiable computation techniques. Their approach consists of creating a *proof of unlearning* of input x_i from model M, resulting in a model M'. This proof can be verified both by the owner of x_i or by any third party, and it can be zero-knowledge to preserve the privacy of data and model. Unfortunately, even though the solution in [16] addresses verifiability, nothing prevents the model owner from deviating from an honest behaviour as reflected by questions (1) and (2). Hence, even with verifiable machine unlearning, data owners have no guarantees that their data will no longer be used in the machine learning (ML) application.

1.1 Our Contributions

The focus of this work is to address the problem—that we call secure machine unlearning—raised by the above questions (1) and (2). We do this with the following contributions.

First, we formalize the problem of secure machine unlearning by proposing a system architecture and a cryptographic definition. The goal is to capture the desired properties mentioned above, namely removing an item from a model in such a way that neither the item nor the model trained over that item can be used. Our formalization draws on the observation that the only way to guarantee such strong control over the data is to ensure that a system is never allowed unrestricted access to the data items used for training nor to the trained model. Intuitively, if a machine learning system has unrestricted access to a model M trained with item x, there is no way to guarantee that M is no longer used after a request to

unlearn x has been issued. Similarly, if the system has unrestricted access to x, there is no way to guarantee that x is not used to train a different model or to re-train M after the unlearning task. For this reason, our security model ensures that all data items remain private and that confidentiality is preserved across all stages of the machine learning pipeline, including training, inference and unlearning, except for what must be revealed through the basic functionality (e.g., inference outputs). Our model assumes that the machine learning system is distributed across several servers and ensures that, provided that at least one of them is honest, the confidentiality is preserved and the unlearning occurs effectively.

Second, we show how to realize secure machine unlearning protocols meeting our security notion through the aid of privacy-preserving cryptography. Specifically, we propose two solutions: one solely based on multiparty computation (MPC), and one based on fully homomorphic encryption (FHE) plus a minimal use of MPC. Following our model, our solutions do not only address the problem of unlearning but also provide methods to perform the main ML functionalities, namely training and inference, in a privacy-preserving manner. In fact, most subroutines of our protocols rely generically on MPC- or FHE-based training and inference protocols and thus they will benefit from future advances in the state of the art. We formalize our solutions in the honest-but-curious setting where all the servers are assumed to follow the protocol. Nevertheless, our solutions can be upgraded to achieve active security following standard techniques.

Our third contribution is the design of an instantiation of the FHE-based solution, its implementation, and its empirical evaluation. To this end, our main technical contributions are: (i) refine and adapt existing protocols for distributed versions of secret-key-based FHE operations, such as key generation, reencryption, and decryption; (ii) integrate a series of optimizations based on SISA [7] that crucially improve the scalability of the secure unlearning task. Our experiments indicate that an unlearning operation on a logistic regression model trained with 10,000 data points can be completed in under two hours on a single core—an overhead that may be acceptable in scenarios where unlearning requests occur sporadically.

We acknowledge that our secure machine unlearning solution presents certain practical challenges, largely due to the performance limitations of FHE and MPC in machine learning workloads. Nevertheless, our goal is to establish foundational principles for achieving machine unlearning in the presence of potentially adversarial servers and demonstrates that such a design is, in fact, feasible. Our solution shows how to achieve secure machine unlearning with costs that are closely related to those needed to perform secure training. Therefore, because our system is agnostic to the specific FHE or MPC instantiations, we expect that practical secure machine unlearning will emerge as these cryptographic primitives and their applications to secure ML continue to improve. Overall, our results illustrate that users can retain meaningful control over their data—even when it is used to train online machine learning models.

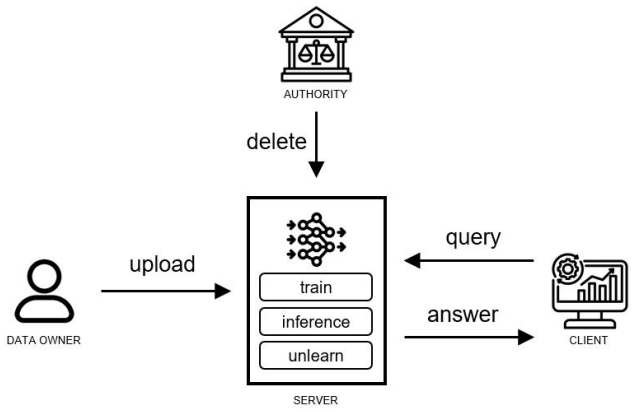

Figure 1: System model

2 Machine Unlearning

Figure 1 shows an overview of a machine learning application that allows unlearning. The central entity is a *server* that owns the machine learning model. One or more *data owners* upload data items and the server uses such data to *train* its model. One or more *clients* submit queries to the server that runs an *inference* algorithm on the trained model to provide answers. Finally, an *authority* may issue requests to delete from the model (the influence of) data items used for training; in this case, the server runs an *unlearning* algorithm to comply with the request. Each data owner could act as the authority for her own data. Alternatively, data owners may entrust third-parties with the authority of issuing unlearning requests.

In greater detail, we use M to denote the model trained on data $\mathcal{D}$ uploaded by data owners. The authority may issue requests to forget a set of data items referred to as the "forget set" $\mathcal{D}_f$. At this stage, the server runs an unlearning algorithm on M to obtain an updated model M' that should retain no information on $\mathcal{D}_f$. From this moment on, the server runs inference tasks on M'.

The literature features a number of techniques to "forget" $\mathcal{D}_f$. A natural means to unlearning is to re-train M from scratch using the "retain set" $\mathcal{D}_r \equiv \mathcal{D} \setminus \mathcal{D}_f$. Nevertheless, retraining-based methods may be impractical due to the the prohibitive training cost of large models. In 2021, Bourtoule et al., [7] introduced SISA—a retraining-based framework for machine unlearning that borrows from optimization techniques of distributed systems to minimize the overhead of the unlearning tasks. In SISA, the training dataset $\mathcal{D}$ is split in disjoint *shards* $\mathcal{D}_1, \ldots, \mathcal{D}_N$ and each shard $\mathcal{D}_i$ is used to train an isolated model M_i. At inference time, the input data is fed to each of the isolated models and their output is aggregated (e.g., the final output is the aggregate of the output vectors of the isolated models). In case of unlearning of data item $x \in \mathcal{D}_i$, SISA must only retrain model M_i while all other isolated models remain unchanged. The authors of [7] show that SISA incurs in reasonable overhead without significant accuracy loss compared to the baseline system where a single model is trained on $\mathcal{D}$ and unlearning is achieved by retraining.

Non-retraining based methods [42] seek to avoid the cost of retraining while producing similar results. Non-retraining

methods can be broadly split into two categories: exact unlearning and approximate unlearning. Exact unlearning [10] aims to obtain a model whose parameters have a distribution which is identical (with respect to the randomness of the training procedure) to that of the model trained on $\mathcal{D} \setminus \mathcal{D}_f$. Approximate unlearning deals with more relaxed definitions, such as those based on differential privacy [38]. One approach to approximate unlearning is weight adjustment [18], where the influence of the unwanted data is counteracted by directly tweaking the model's weights.

Recent work [29, 43] disputes the effective capability of retraining-free methods to unlearn items of the original training data-set, since a simple fine-tuning of the model obtained after unlearning, may restore its original—i.e., before unlearning–behavior.

3 Secure Machine Unlearning

Taking into consideration the system model of machine unlearning described in the previous section, here we present our extension for *secure* machine unlearning.

As mentioned in the previous section, machine unlearning relies on an honest server that duly fulfills the unlearning requests. We note, however, that a malicious server may refuse to run the unlearning task, e.g., to avoid the computational penalty associated with updating the model. Similarly, a malicious server may refuse to drop the original model M, so that prediction accuracy is not affected by the unlearning operation.

The only previous work that accounts for a malicious server is [16]. In particular, the authors of [16] leverage an interactive proof system for a server to prove that a model M' is the output of an unlearning function that takes as input a model M, a forget set $\mathcal{D}_f$, and optionally the original data set $\mathcal{D}$ used for training. The proof system is also used to prove that inference is carried out by using the updated model M'. As such, the machine unlearning framework of [16] provides cryptographic guarantees of the server behavior, but only as long as *all* clients check the proofs issued by the server. In other words, [16] does not prevent a server deviating from honest behavior by, e.g., (a) using M to answer inference queries, or (b) fine-tune M' as suggested by [29, 43] to restore M, or (c) re-train M with the original data-set $\mathcal{D}$ that includes $\mathcal{D}_f$. Ideally, an unlearning framework should guarantee that, once an unlearning request for $\mathcal{D}_f$ has been issued, the original model trained with data including $\mathcal{D}_f$ should not be used any further for inferences and no other model should be trained or fine-tuned with data that includes $\mathcal{D}_f$. To the best of our knowledge, no machine unlearning framework provides such guarantees of unlearning to data owners.

We also note that no unlearning framework can warrant such properties if unrestricted access to the data or the model is allowed. That is, if the server owns model M there is no means to prevent it from running inference tasks on M. Similarly, if the server has a copy of data $\mathcal{D}_f$, nothing prevents it from using $\mathcal{D}_f$ to train (or fine-tune) any model.

Hence, we use Multi-Party Computation (MPC) and Fully Homomorphic Encryption (FHE) and distribute the server functionality so to ensure that no task (training, inference and unlearning) can be unilaterally executed by a single server. Depending on the computational resources of the servers we consider two possible scenario. In one scenario, all servers use MPC to carry out training, inference and unlearning; in this scenario, one honest server is sufficient to ensure that unlearning requests are duly fulfilled. Another option is to separate the machine learning tasks from trust management by using a mix of MPC and FHE. In this scenario a powerful server—the *evaluator*—carries out ML-related operations, while the others—we call them *wardens*—manage cryptographic keys; unlearning requests are guaranteed as long as one warden is honest, even if the evaluator is not.

In what follows, we introduce a syntax for a secure machine unlearning scheme and a correctness definition. Then, we introduce a semi-honest security model where the servers are honest but curious and follow the protocol.

A remark on authentication/authorization. We treat the problem of authorization as orthogonal to our work. In particular, we assume that the servers implement some policy to determine which parties can upload data, send inference queries, or issue unlearning requests. For example, unlearning requests for data $\mathcal{D}_f$ may be issued only by the owner of $\mathcal{D}_f$ (i.e., data owners act as unlearning authorities for their own data) or by a third party that has been previously authorized by the owner of $\mathcal{D}_f$. Similarly, clients may send queries only if authorized by the data owners or upon payment to the servers. In a nutshell, given a PKI, any authorization logic can be integrated in our machine unlearning framework.

3.1 Syntax

We define a secure machine unlearning protocol as a tuple of cryptographic algorithms that are parametrized by an arbitrary set of clients, a set of data owners $\mathcal{U}$ and a set of m servers $S_1, \ldots, S_m$, which may or may not have the same role.[1] These algorithms are interactive between the servers, the clients, and data owners when needed. Moreover, we assume that all data owners, each uniquely identified by an identifier uid, are registered to a PKI with their public key $\mathsf{pk}_{\mathsf{uid}}$, and that the communication between all parties involved is authenticated.

Definition 1 (Secure Machine Unlearning). *A **secure machine unlearning scheme** is a tuple of interactive protocols* $(\mathsf{Setup}, \mathsf{Init}, \mathsf{Upload}, \mathsf{Train}, \mathsf{Infer}, \mathsf{Unlearn})$ *that are run by a set of servers $S_1, \ldots, S_m$, clients, and data owners. All algorithms except* Setup *are* stateful*, in the sense that each server holds* a state st_i *which gets* updated after executing each protocol. *The scheme is parametrized by a training data space $\mathcal{X}$, a query data space $\mathcal{Q}$, a data owner space $\mathcal{U}$, and a model space $\mathcal{M}$.*

- $\mathsf{Setup}(1^\lambda) \to \mathsf{pp}$: *On input a security parameter 1^λ, outputs public parameters* pp*. We assume that these are an implicit input to all other algorithms.*
- $\mathsf{Init}() \to \{\mathsf{st}_i\}_{i\in[m]}$: *This algorithm runs among the servers. The interactive initialization protocol outputs states $\mathsf{st}_1, \ldots, \mathsf{st}_m$ for each server $S_1, \ldots, S_m$, respectively. Each server's state is implicitly an input to all other algorithms.*

[1] Looking ahead, in our FHE-based solution there is a distinguished server, denoted by S_0, which performs all computations over encrypted data.

$\mathsf{Upload}(\mathsf{uid}, x) \to \mathsf{xid}$: *This algorithm runs between one data owner and the servers. On data owner input a unique identifier* $\mathsf{uid} \in \mathcal{U}$*, and private*[2] *input data* $x \in \mathcal{X}$*, the interactive upload protocol outputs a unique input identifier* xid.

$\mathsf{Train}(M, \mathcal{D}; r) \to \mathsf{fid}$: *This algorithm runs between the servers. On input a starting model*[3] $M \in \mathcal{M}$ *with an associated training algorithm* T_M*, a dataset* $\mathcal{D} = \{\mathsf{xid}_i\}_{i\in[n]}$*, and explicit randomness* r*, the interactive training protocol updates the states with the trained model and outputs a public model identifier* fid.

$\mathsf{Infer}(\mathsf{fid}, q) \to y$: *This algorithm runs between a client and the servers. On client input a model identifier* fid *and a query* $q \in \mathcal{Q}$*, the inference protocol outputs an inference value* y*. We remark that this captures both the case in which the input and output of inference are public and the case in which they are private to the client; the only difference is whether the client is corrupted or not.*

$\mathsf{Unlearn}(\mathcal{D}_f) \to b$: *This protocol is run between the servers. On input an unlearning dataset* $\mathcal{D}_f = \{\mathsf{xid}_i\}_{i\in[k]}$*, the interactive unlearning protocol accepts* $(b = 1)$ *or rejects* $(b = 0)$*.*

Definition 2 (Correctness). *We say that a machine unlearning scheme is* δ*-correct if for any* $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$*, server states* $\{\mathsf{st}_i\}_{i\in[m]} \leftarrow \mathsf{Init}()$*, datasets* $\mathcal{D} = \{x_1, \ldots, x_n\} \subset \mathcal{X}$ *and* $\mathcal{D}_f \subset \mathcal{D}$*, users* $\{\mathsf{uid}_1, \ldots, \mathsf{uid}_n\} \subset \mathcal{U}$*, model* $M \in \mathcal{M}$ *with associated training function* T_M*, and query* $q \in \mathcal{Q}$*, it holds that*

$$\Pr\left[\, y = y' \;\middle|\; \begin{array}{l} \mathsf{xid}_i \leftarrow \mathsf{Upload}(\mathsf{uid}_i, x_i), \forall\, i \in [n] \\ \mathsf{fid} \leftarrow \mathsf{Train}(M, \mathcal{D}; r) \\ y \leftarrow \mathsf{Infer}(\mathsf{fid}, q) \\ M_{\mathcal{D}} \leftarrow T_M(\mathcal{D}; r) \\ y' \leftarrow M_{\mathcal{D}}(q) \end{array} \right] \geq 1 - \delta.$$

and that

$$\Pr\left[\, \begin{array}{c} y = y' \\ \wedge \quad b = 1 \end{array} \;\middle|\; \begin{array}{l} \mathsf{xid}_i \leftarrow \mathsf{Upload}(\mathsf{uid}_i, x_i), \forall\, i \in [n] \\ \mathsf{fid} \leftarrow \mathsf{Train}(M, \mathcal{D}; r) \\ b \leftarrow \mathsf{Unlearn}(\mathcal{D}_f) \\ y \leftarrow \mathsf{Infer}(\mathsf{fid}, q) \\ M_{\mathcal{D}} \leftarrow T_M(\mathcal{D} \setminus \mathcal{D}_f; r) \\ y' \leftarrow M_{\mathcal{D}}(q) \end{array} \right] \geq 1 - \delta.$$

We refer to the first property as inference correctness. *The second property is denoted by* unlearning correctness.

We note that less strict definitions of inference and unlearning correctness could be introduced to account for possible precision loss caused by the implementation of the protocols, or to allow for approximate unlearning. Instead of requiring exact equality, such definitions could instead require that $\Delta(y, y') \leq \epsilon$ with overwhelming probability, with Δ being a distance function and $\epsilon > 0$ a parameter of the definition.

Figure 2 represents a general overview of the syntax and the interactions between the servers and the other parties of the system, such as data owners, clients, and authorities. The implementation of each block can be subjected to change depending on the underlying schemes used (e.g. MPC or FHE) but the functionalities should be maintained.

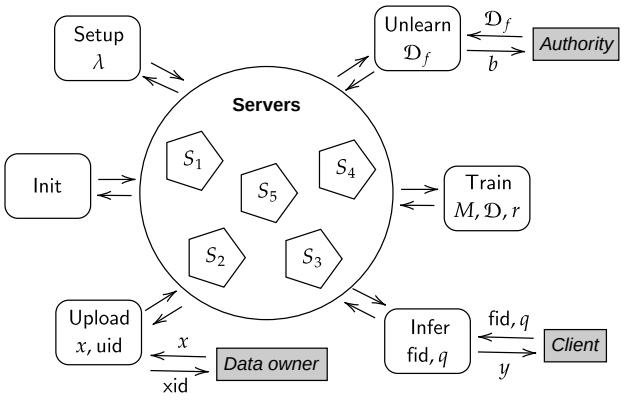

Figure 2: Overview of the secure machine unlearning syntax.

3.2 Security Model

We define a game-based security model inspired by the indistinguishability-based security definition of functional encryption, while taking into account the multi-server setting. In this model, the adversary $\mathcal{A}$ is allowed to corrupt up to t out of the m servers. A corruption reveals the internal state st_i of each server throughout the protocol, but does not allow the adversary to control those servers, as they honestly follow the execution of the protocol. Hence, our security model captures *semi-honest adversaries*. Moreover, the adversary has access to all the transcripts that result from the interaction of the servers, as usual in semi-honest security models for e.g. MPC.

We model security via a game $\mathsf{PRIV}_{m,t,b}$ between an adversary $\mathcal{A}$ and a challenger, where the adversary has access to multiple oracles that allow them to drive the protocol execution. We state the game, which is parametrized by the number of servers m, a bound on the number of corrupt servers t, and a challenge bit $b \in \{0, 1\}$, in Figure 3. The game captures three main security properties:

- *Client data confidentiality throughout the session.* This is captured via a semantic security definition where the adversary controls all the data in the system except for two inputs x_0, x_1, for which only x_b is uploaded and used by the servers to train models. The goal of $\mathcal{A}$ is to learn b. Naturally, inference queries can leak information about training data, and in particular about b. This is a natural and correct behaviour of inference. Our definition captures that this must be *the only allowed leakage* by disabling all $\mathcal{O}\mathsf{Infer}$ queries that result into different outputs for the models trained with x_b and x_{1-b}. Adversaries that perform queries which induce leakage are disallowed, again drawing inspiration from functional encryption definitions [4].
- *Semi-honest security for up to t corrupted servers.* This is captured by letting the adversary corrupt t servers for which it gains read access throughout the entire game.

[2] Naturally, one can extend this algorithm to support public data upload as well. As this does not have an impact on security, we omit this possibility from the syntax and consider, without loss of generality, that all uploaded data is private.

[3] One example of a starting model would be a pre-trained model that is fine-tuned with the dataset $\mathcal{D}$. One can also start with an untrained model with e.g. empty or random weights.

- *Forward security for unlearned data.* This is captured as the adversary gains access to the states of all servers after the unlearning phase, including all of their current key material. Hence, any scheme which satisfies this definition must ensure that the key material encrypting data to be unlearned is erased securely after the unlearning phase.

The latter property is quite strong insofar as giving the adversary access to all server states implies that it can decrypt all model parameters, resulting in white-box access to the unlearned model. For this notion to hold, one needs that the unlearning algorithm not only produces indistinguishable output distributions with respect to a trained model, but also indistinguishable model *parameter* distributions. Therefore, in Figure 3 we also define a weaker notion of forward security where the adversary only gets black-box access to the model after unlearning, which we capture by enabling access to the $\mathcal{O}\mathsf{Infer}$ oracle.

We remark that our definitions implicitly consider corrupted clients, as the adversary chooses the inference queries and sees its outputs. Extending this to a more fine-grained setting where only some of the client queries are adversarial is left for future work.

We say that the scheme is t-out-of-m private if the advantage of any PPT adversary $\mathcal{A}$ in the game, which is given by the expression below, is negligible:

$$\mathsf{Adv}^{\mathrm{priv-m,t}}_{\mathcal{A}}(\lambda) = \left|\Pr[\mathcal{A}^{\mathsf{PRIV}}_{m,t,0} = 1] - \Pr[\mathcal{A}^{\mathsf{PRIV}}_{m,t,1} = 1]\right| \leq \mathsf{negl}(\lambda)$$

On malicious security. We remark that the above game can capture malicious security by simply letting the adversary control the corrupted servers in C.

4 Our Secure Machine Unlearning Protocols

4.1 Overview

As noted earlier, if the server holds M—trained on a dataset that includes data to be forgotten x—no mechanism can guarantee that M will no longer be used for inference after a request to unlearn x has been issued. Therefore, we depart from the single-server model and we leverage a distributed multi-server approach to construct secure machine unlearning protocols. We propose two solutions. The first solution is solely based on MPC, whereas the second is based on fully homomorphic encryption along with a minimal use of MPC for key management.

MPC-based. Machine learning applications using multiple servers have been introduced in the past in the context of machine learning, mainly to provide data and model confidentiality [9]. In a nutshell, the model and the data are secret-shared across two or more servers and all servers run an MPC protocol to either train the model or carry out an inference task.

In our MPC-based solution, which we describe in detail in Section A, several servers hold shares of the model M and the dataset $\mathcal{D}$ used for training. Upon receiving an inference request on data z, the servers run an MPC protocol to compute $y = M(z)$. Upon receiving an unlearning request for data x, the servers compute an updated model M'; from this moment, all inference requests will be carried out by using (shares of) M'. As long as one server is honest, no further inference task will use the original model M—thereby effectively unlearning x.

Note that the above idea fits unlearning frameworks based on re-training [7] as well as those that do not access the original training dataset [18]. In the first case, the servers train model M' on $\mathcal{D} \setminus \{x\}$—where both M' and the training data is secret-shared. In case unlearning is carried out without the original dataset, the servers use an MPC protocol to compute M' by only using (shares of) M and x.

Our MPC-based design favours servers with high computational power and low latency communication links. This is because the MPC task may require demanding computation and large amounts of data exchanged among the servers in order to run the (re-)training algorithms in MPC.

FHE-based. Our second solution is based on fully homomorphic encryption and favours scenarios where most servers have lower computational resources and/or high-latency communication links. In this design, we rely on one party with large computational power, which we call evaluator or main server. This entity runs the learning, unlearning and inference task over FHE-encrypted data and model. We present this solution in Section 4.2 and in Figure 4.

The evaluator does not possess any key material: we leverage a group of two or more additional servers, that we call "wardens", to hold shares of the decryption keys. In a nutshell, given an encrypted model $E(\mathsf{pk}, M)$ under public key pk the server (or evaluator) homomorphically computes $c = E(\mathsf{pk}, M(z))$ and submits the ciphertext to the wardens. Then, the wardens use their key-shares to decrypt c and obtain $M(z)$.

Given an unlearning request for data x, the wardens jointly compute a new key pair $(\mathsf{pk}', \mathsf{sk}')$, such that each warden holds a share of sk'. Hence, the server and the wardens jointly compute an updated model $E(\mathsf{pk}', M')$ encrypted under key pk'. From this moment, inference requests must be carried out by the server using $E(\mathsf{pk}', M')$. In particular, the wardens delete the shares of the former sk. As long as one warden is honest, M cannot be used to run inference tasks.

Discussion. In the rest of the paper, we focus on the FHE-based solution as we find it more advantageous in practice, since it requires substantially less communication (and availability) between the different servers.

4.2 FHE-based Secure Machine Unlearning

We start by recalling the syntax of fully homomorphic encryption, which is the main building block of this solution.

Definition 3 (Fully Homomorphic Encryption). *A fully homomorphic encryption scheme is a tuple of algorithms defined as follows. The scheme is parametrized by a message space $\mathcal{X}$, a ciphertext space C, and a function space M.*

$\underline{\mathsf{Setup}(1^\lambda) \to \mathsf{pp}:}$ *On input the security parameter 1^λ, output public parameters* pp.

$\underline{\mathsf{KeyGen}(\mathsf{pp}) \to (\mathsf{sk}, \mathsf{pk}):}$ *On input public parameters* pp*, output a secret key* sk *and a public key* pk.

$\underline{\mathsf{Enc}(\mathsf{pk}, x) \to c:}$ *On input a public key* pk *and a message x, output a ciphertext c.*

$\underline{\mathsf{Dec}(\mathsf{sk}, c) \to x:}$ *On input a secret key* sk *and a ciphertext c, output a decrypted message x.*

Game $\mathsf{PRIV}_{m,t,b}$:

- The challenger sets up public parameters $\mathsf{pp} \leftarrow \mathsf{Setup}(1^\lambda)$, server states $\{\mathsf{st}_i\}_{i\in[m]} \leftarrow \mathsf{Init}()$, and a set of registered data owners with identifiers $\mathsf{uid} \in \mathcal{U}$.
- $\mathcal{A}(\mathsf{pp}, \mathcal{U})$ outputs a set $C \subseteq [m]$ of corrupted servers such that $|C| \leq t$, and two inputs $x_0, x_1 \in \mathcal{X}$ and a data owner $\mathsf{uid}^* \in \mathcal{U}$. The adversary gains *read* access to st_c for each $c \in C$ throughout the game. The adversary can also observe all messages resulting from the execution of the protocols among the servers.
- The challenger runs $\mathsf{xid}^* \leftarrow \mathsf{Upload}(\mathsf{uid}^*, x_b)$ and sends xid^* to $\mathcal{A}$.
- **Phase 1:** then, $\mathcal{A}$ can query the following oracles:
  $\underline{O\mathsf{Upload}(\mathsf{uid} \in \mathcal{U}, x \in \mathcal{X})}$**:** The challenger runs $\mathsf{xid} \leftarrow \mathsf{Upload}(\mathsf{uid}, x)$ and returns xid to $\mathcal{A}$.
  $\underline{O\mathsf{Train}(M \in \mathcal{M}, \mathcal{D}; r)}$**:** The challenger checks that every $\mathsf{xid} \in \mathcal{D}$ has been queried to $O\mathsf{Upload}$.
  Then, it runs $\mathsf{fid} \leftarrow \mathsf{Train}(M, \mathcal{D} \cup \{\mathsf{xid}^*\}; r)$ and outputs fid to $\mathcal{A}$. It also runs another training *locally* as $\bar{M}_{\mathsf{fid}} \leftarrow T_M(\mathcal{D} \cup \{x_{1-b}\}; r)$, storing the resulting model.
  $\underline{O\mathsf{Infer}(\mathsf{fid}, q)}$**:** The challenger runs $y \leftarrow \mathsf{Infer}(\mathsf{fid}, q)$, and it runs $y' \leftarrow \bar{M}_{\mathsf{fid}}(q)$ on the *locally trained* model. If $y = y'$, then it outputs y to $\mathcal{A}$. Otherwise, it outputs $\perp$ and aborts the game immediately.
- **Phase 2:** The challenger runs $\mathsf{Unlearn}(\mathsf{xid}^*)$. Then,
  - For the *strong forward security* property, the challenger reveals the state of all servers $\mathsf{st}_1, \ldots, \mathsf{st}_m$ to $\mathcal{A}$.
  - For the *weak forward security* property, the challenger gives $\mathcal{A}$ oracle access to $O\mathsf{Infer}$ again.
- Finally, $\mathcal{A}$ outputs a guess b' of b.

Figure 3: Semi-honest security game $\mathsf{PRIV}_{m,t,b}$ for a machine unlearning scheme, parametrized by a bit b and a bound on the number of corrupt servers t. The game captures two flavours of forward security which differ only in Phase 2.

$\underline{\mathsf{Eval}(\mathsf{pk}, f, \{c_i\}_{i\in[n]}) \to c}$: *On input a public key* pk*, a function* f*, and a set of ciphertexts* $\{c_i\}_{i\in[n]}$ *output an evaluated ciphertext* c*.*

In our solution, we need an FHE scheme in which the secret key is secret-shared among m parties and a set of secure distributed protocols for key generation, reencryption, and decryption. We define these protocols below.

Definition 4 (FHE with distributed key generation). *Let* $\mathcal{P} = \{P_1, P_2, \ldots, P_m\}$ *be a set of* m *parties. A fully homomorphic encryption scheme admits distributed key generation if there exists a secure MPC protocol* $\Pi_{\mathsf{DistKeyGen}}$ *for the following algorithm.*

$\underline{\mathsf{DistKeyGen}(\mathsf{pp}) \to (\mathsf{pk}, [\mathsf{sk}]_i)}$: *On input public parameters* pp*, at the end of the interaction each party* P_i *possesses the generated public key* pk *and an* m*-out-of-*m *secret share of the corresponding secret key* $[\mathsf{pk}]_i$*.*

Definition 5 (FHE with distributed reencryption). *A fully homomorphic encryption scheme admits distributed reencryption if it supports the following algorithms.*

$\underline{\mathsf{ReEncShare}([\mathsf{sk}], [\mathsf{sk}'], c) \to [u_c^{\mathsf{sk}\to\mathsf{sk}'}]}$: *On input two secret key shares* $[\mathsf{sk}]$, $[\mathsf{sk}']$ *of different secret keys and a ciphertext* c*, output a reencryption share* $[u_c^{\mathsf{sk}\to\mathsf{sk}'}]$*.*

$\underline{\mathsf{ReEnc}(\{[u_c^{\mathsf{sk}\to\mathsf{sk}'}]_i\}_{i\in[m]}) \to c'}$: *On input* m *reencryption shares* $\{[u_c^{\mathsf{sk}\to\mathsf{sk}'}]_i\}_{i\in[m]}$ *for a ciphertext* c *encrypted under* sk*, output an updated ciphertext* c' *encrypted under* sk'*.*

Definition 6 (FHE with distributed decryption). *A fully homomorphic encryption scheme admits distributed decryption if it supports the following algorithms.*

$\underline{\mathsf{DecShare}([\mathsf{sk}], c) \to [v_c^{\mathsf{sk}}]}$: *On input a secret key share* $[\mathsf{sk}]$ *and a ciphertext* c*, output a decryption share* $[v_c^{\mathsf{sk}}]$*.*

$\underline{\mathsf{DistDec}(\{[v_c^{\mathsf{sk}}]_i\}_{i\in[m]}) \to x}$: *On input* m *decryption shares* $\{[v_c^{\mathsf{sk}}]_i\}_{i\in[m]}$ *for a ciphertext* c *encrypted under* sk*, output the message* x *encrypted in* c*.*

4.2.1 Protocol roles. For our FHE-based solution, we outline the following roles:

- **Data owners,** found in GDPR [1] as data subjects, are the owners of the data processed by the system. As such, they upload data items and may issue data unlearning requests.
- **Wardens,** tasked with key management. This involves generating the joint FHE key pair, performing distributed reencryption and decryption, and deleting the shares of previous keys as part of the unlearning process. We denote them by $S_1, \ldots, S_m$.
- **Evaluator,** denoted by S_0, tasked with performing the expensive FHE operations of the system, such as training, inference, and unlearning without retraining.
- **Clients,** the parties submitting inference requests.

We observe that the security model presented in Section 3.2 applies to this FHE-based approach as well. The servers of the security model correspond to the key wardens, since all three security properties are fulfilled by their role. Namely, by the semantic security of the FHE scheme, the m-out-of-m secret sharing of the secret key held by the wardens, and the reencryption and deletion of the old key shares. As for the evaluator S_0 defined above, it functions merely as a unit for outsourced FHE computations. Thus, under a semi-honest security assumption, it is not included in the security model since it can be corrupted by the adversary. We note that S_0 does not necessarily need to be a fixed entity in our construction, but can be a cluster, a set of servers, or even a different computing entity each time.

4.2.2 Protocol description. Our FHE-based solution involves n data items, m wardens $S_1, \ldots, S_m$, and an evaluator S_0. We describe our protocol formally in Figure 4. At a high level, our solution works as follows. The wardens jointly generate a public FHE key pk^e such that its corresponding secret key is secret-shared among them, each keeping a share $[\mathsf{sk}^e]_j$. Then, data owners encrypt their data to pk^e to upload it to the system; data items are tracked using a unique identifier xid, and we denote the resulting ciphertexts by c_{xid}. The evaluator S_0 can then train a model M by computing on the encrypted dataset, obtaining a model identifier fid and the corresponding model ciphertext c_{fid}. Multiple models can be trained, resulting in different ciphertexts under pk^e.

If an unlearning with retraining request is issued for xid, the wardens proceed as follows. First, they generate a new FHE key pk^{e+1}. Then, they reencrypt all data item ciphertexts (except for the data item corresponding to xid) under the new key pk^{e+1}. Then, the server retrains all models under the new key pk^{e+1}, obtaining updated ciphertexts c^*_{fid}. Finally, the wardens erase the shares $[\mathsf{sk}^e]_j$ of the old key. Note that this guarantees forward security for unlearned data as long as a single warden is not corrupted and runs the key erasure.

For unlearning requests not based on retraining, the evaluator must first update the model, by homomorphically computing the unlearning algorithm, and then it collaborates with the wardens to move model and data to the new key pk^{e+1}. To this end, the wardens reencrypt both the model ciphertexts c_{fid} and the data item ciphertexts c_{xid} (except the ones corresponding to unlearned data). Note that the key update and reencryption mechanisms essentially define a series of epochs. New epochs are triggered by unlearning requests, and each key pk^e corresponds to a single epoch e.

4.2.3 Security Analysis. Our FHE-based construction is a secure machine unlearning scheme following the definition of Figure 3. For retraining-based unlearning, the scheme satisfies the strong notion of forward security for up to $m-1$ corrupted wardens. We state the main result in the following theorem, which we prove in Section B.

Theorem 1. *Let Π be our FHE-based construction with retraining-based unlearning from Figure 4. Let $\mathcal{A}$ be a static PPT adversary in the semi-honest security game $\mathsf{PRIV}_{m,m-1,b}$ with strong forward security which makes at most q oracle queries. If FHE is a secure IND-CPA encryption scheme and if all distributed sub-protocols are one-shot MPC protocols secure against $m-1$ semi-honest corruptions, then $\mathsf{Adv}^{\mathrm{priv-m,m-1}}_{\mathcal{A},\Pi}(\lambda) = \mathsf{negl}(\lambda)$.*

For unlearning without retraining, the scheme cannot satisfy strong forward security as the unlearning algorithm does not guarantee that the updated parameters are independent of the unlearned data. Weak forward security holds if the unlearning algorithm ensures that the distribution of the model outputs is statistically close to the one of the retraining-based method. We claim that similar results hold for our MPC-based constructions, although we do not formally prove this.

Achieving Active Security. While our constructions are described in the semi-honest setting, we can strengthen them to be secure against active adversaries following standard techniques. First, one needs to rely on an MPC framework with security in the malicious setting, such as MP-SPDZ [15, 26]. Second, the server must prove that it performs the FHE evaluations correctly. This can be enforced using zero-knowledge proofs of correct evaluation, following approaches such as [11]. These modifications incur a substantial performance overhead, in particular due to the cost of zero-knowledge proofs over FHE computations.

5 Our Instantiation

We present our concrete instantiation of the secure machine unlearning solutions presented in the previous section. In particular, we instantiate the FHE-based solution using retraining as the unlearning method, and we implement a proof of concept based on that. First, we describe the instantiations of all the building blocks of the generic FHE-based solution and discuss the modifications made to the protocols to tailor them to our application. Later, we introduce a series of optimizations that improve the scalability of the system.

5.1 FHE-based solution architecture

Architecture. We use an architecture of $m+1$ servers, consisting of m wardens and a server tasked with performing the homomorphic FHE operations. All servers are required to be online for the system to function, and we assume the servers to be semi-honest, allowing all servers but one warden to be corrupted.

FHE scheme. Our protocol is scheme-agnostic, only requiring that large depth computations such as ML training are supported and threshold functionality can be implemented. We choose the scheme TFHE [12] for our proof-of-concept implementation due to the availability of the software libraries ConcreteML [44], implementing the machine learning functionality, and Threshold-FHE [45], implementing the threshold functionality.

In TFHE, data items are encrypted as standard Learning-With-Errors (LWE) ciphertexts. For a message $x \in \mathbb{Z}_p$ and ciphertext modulus q, the ciphertext will be of the form $c := (\mathbf{c_1}, c_2) = (\mathbf{a}, b)$ where:

$$b = \mathbf{a} \cdot \mathbf{sk} + \Delta \cdot x + e \pmod{q} \tag{1}$$

with $\mathbf{a} \in \mathbb{Z}_q^\ell$, $\Delta = q/p$, $\mathbf{sk} \in \{0,1\}^\ell$ being the secret key, and e being a noise term.

For a public key version of the scheme, we use the methods described in [24]. We note that both the generation of the public key, and the encryption algorithm, consist of linear operations. We observe that despite the optimizations found in these methods, the ciphertexts resulting from public key encryption will be of the form presented above.

Noise control. TFHE uses bootstrapping to keep the noise level under control after evaluating a function over ciphertexts (e.g. during training). This ensures that evaluated ciphertexts remain decryptable. As this is part of the standard FHE functionality, we do not detail how bootstrapping works. We remark, however, that in our implementation bootstrapping is carried out regularly.

Secret sharing. The secret sharing scheme used is an additive m-out-of-m secret sharing over a ring $\mathbb{Z}_q$. A secret s is split into m shares, with $[s]_i$ being the share of party i. The secret is reconstructed as $s = \sum_{i=1}^{m} [s]_i$. Concretely, we instantiate with the rings $\mathbb{Z}_{2^{64}}$ or $\mathbb{Z}_{2^{128}}$.

$\underline{\mathsf{Setup}(1^\lambda)}$**:**

- Output $\mathsf{pp} \leftarrow \mathsf{FHE.Setup}(1^\lambda)$.

$\underline{\mathsf{Init}()}$**:**

- Wardens $S_1, \ldots, S_m$ run the distributed key generation protocol $\Pi_{\mathsf{DistKeyGen}}(\mathsf{pp})$.
- Each S_j obtains $([\mathsf{sk}^e]_j, \mathsf{pk}^e) \leftarrow \mathsf{DistKeyGen}(\mathsf{pp})$, where we set $e = 1$ for the first epoch.
- Output pk^e.

$\underline{\mathsf{Train}(M, \mathcal{D}; r)}$**:**

- S_0 retrieves ciphertexts $C = \{c_{\mathsf{xid}}\}_{\mathsf{xid} \in \mathcal{D}}$ corresponding to $\mathcal{D} = \{\mathsf{xid}_i\}_{i \in [n]}$.
- S_0 generates a unique model identifier fid for M.
- S_0 evaluates the training function $c_{\mathsf{fid}} \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}^e, T_{\mathsf{fid}}, C; r)$.
- S_0 stores c_{fid} and outputs fid.

$\underline{\mathsf{Unlearn}(\mathcal{D}_f)}$ **(with retraining):**

- S_0 checks if every $\mathsf{xid} \in \mathcal{D}_f$ exists and contributed to some model fid. Else, abort and output $b = 0$.
- Wardens $S_1, \ldots, S_m$ run the distributed key generation protocol $\Pi_{\mathsf{DistKeyGen}}(\mathsf{pp})$. Each S_j obtains $([\mathsf{sk}^{e+1}]_j, \mathsf{pk}^{e+1}) \leftarrow \mathsf{DistKeyGen}(\mathsf{pp})$.
- For every $\mathsf{xid} \in \mathcal{D} \setminus \mathcal{D}_f$, wardens compute reencryption shares: $[u_{\mathsf{xid}}^{e \to e+1}]_j \leftarrow \mathsf{ReEncShare}([\mathsf{sk}^e]_j, [\mathsf{sk}^{e+1}]_j, c_{\mathsf{xid}})$.
- S_0 aggregates shares to get reencrypted ciphertexts: $c_{\mathsf{xid}} \leftarrow \mathsf{ReEnc}(\{[u_{\mathsf{xid}}^{e \to e+1}]_j\}_{j \in [m]})$.
- All wardens delete their shares of sk^e.
- S_0 retrains all models and stores the ciphertexts $c_{\mathsf{fid}} \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}^{e+1}, T_{\mathsf{fid}}, \{c_{\mathsf{xid}}^{e+1}\}_{\mathsf{xid} \in \mathcal{D}_{\mathsf{fid}} \setminus \mathcal{D}_f})$, where $\mathcal{D}_{\mathsf{fid}}$ is the dataset used to train fid in epoch e.
- Output $b = 1$.

$\underline{\mathsf{Upload}(\mathsf{uid}, x)}$**:**

- uid generates a unique identifier xid and sends xid and $c_{\mathsf{xid}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}^e, x)$ to S_0.
- S_0 accepts if xid has not been used before, otherwise rejects the update.

$\underline{\mathsf{Infer}(\mathsf{fid}, q)}$**:**

- S_0 evaluates $c_y \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}^e, I_{\mathsf{fid}}, (c_{\mathsf{fid}}, q))$ where I_{fid} is the inference function of model fid.
- Wardens $S_1, \ldots, S_m$ each compute a decryption share: $[v_y^{\mathsf{sk}^e}]_j \leftarrow \mathsf{DecShare}([\mathsf{sk}^e]_j, c_y)$ and send them to the client.
- The client reconstructs the output: $y \leftarrow \mathsf{DistDec}([v_y^{\mathsf{sk}^e}]_j)$.

$\underline{\mathsf{Unlearn}(\mathcal{D}_f)}$ **(without retraining):**

- S_0 checks if every $\mathsf{xid} \in \mathcal{D}_f$ exists and contributed to some model fid. Else, abort and output $b = 0$.
- For every model fid, S_0 computes the unlearning function U_{fid}: $c_{\mathsf{fid}} \leftarrow \mathsf{FHE.Eval}(\mathsf{pk}^e, U_{\mathsf{fid}}, (c_{\mathsf{fid}}, \{c_{\mathsf{xid}}\}_{\mathsf{xid} \in \mathcal{D}_f}))$.
- Wardens $S_1, \ldots, S_m$ run the distributed key generation protocol $\Pi_{\mathsf{DistKeyGen}}(\mathsf{pp})$. Each S_j obtains $([\mathsf{sk}^{e+1}]_j, \mathsf{pk}^{e+1}) \leftarrow \mathsf{DistKeyGen}(\mathsf{pp})$.
- Wardens compute reencryption shares:
  - For every $\mathsf{xid} \in \mathcal{D} \setminus \mathcal{D}_f$, compute $[u_{\mathsf{xid}}^{e \to e+1}]_j \leftarrow \mathsf{ReEncShare}([\mathsf{sk}^e]_j, [\mathsf{sk}^{e+1}]_j, c_{\mathsf{xid}})$.
  - For every fid, compute $[u_{\mathsf{fid}}^{e \to e+1}]_j \leftarrow \mathsf{ReEncShare}([\mathsf{sk}^e]_j, [\mathsf{sk}^{e+1}]_j, c_{\mathsf{fid}})$.
- All wardens delete their shares of sk^e.
- S_0 aggregates shares to get reencrypted ciphertexts:
  - $\mathsf{xid} \in \mathcal{D} \setminus \mathcal{D}_f$, compute $c_{\mathsf{xid}} \leftarrow \mathsf{ReEnc}(\{[u_{\mathsf{xid}}^{e \to e+1}]_j\}_{j \in [m]})$.
  - For every fid, compute $c_{\mathsf{fid}} \leftarrow \mathsf{ReEnc}(\{[u_{\mathsf{fid}}^{e \to e+1}]_j\}_{j \in [m]})$.
- Output $b = 1$.

Figure 4: Our FHE-based construction of a secure machine unlearning protocol.

Threshold TFHE.. TFHE has an extension supporting threshold decryption described in [5] and implemented in [45]. We adapted the threshold functionality from [45] to suit our semi-honest security model and changed the secret sharing scheme to a m-out-of-m additive sharing. This allows us to achieve security against $m - 1$ corrupted wardens. Furthermore, we extended the library by implementing our reencryption functionality.

Distributed key generation. For m wardens, the shares of a secret key $\mathbf{sk} \in \{0,1\}^\ell$ will be of the form $[\mathbf{sk}]_i = ([sk_1]_i, \ldots, [sk_\ell]_i)$ where $\forall j \in [\ell], \sum_{i=1}^m [sk_j]_i \in \{0,1\}$. To form these secret key shares, we draw from bit shares generated in an offline preprocessing, running the protocol `MPC.GenBits` from [5]. Since we use the less complex access structure of m-out-of-m additive sharing, we can instantiate it with the ring $\mathbb{Z}_{2^k} \cong GR(2^k, 1)$, $k \in \{64, 128\}$.

The public key $\mathbf{pk}$ is generated by aggregating partial public key shares $[\mathbf{pk}]_i$, computed according to [24] by each warden using their secret key shares $[\mathbf{sk}]_i$. This computation is possible owing to the linear properties of the additive secret sharing scheme.

Similarly, a key-switching key ksk and a bootstrapping key bk are generated by aggregating partial key-switching key shares $[ksk]_i$ and bootstrapping key shares $[bk]_i$, by performing linear operations on the previously generated secret key shares. We note, however, that in the case of the bootstrapping key, multiplication of secret key shares is also required. Thus, Beaver triples [2] are used and further interaction is needed [5]. The Beaver triples are also generated during the offline preprocessing.

The number and types of the keys generated depends on the parameters of the TFHE scheme as they are needed to realize the FHE computation of the scheme. More details on that can be found in [5]. Among them, of particular interest is the additional secret key $\bar{\mathbf{sk}}$, generated with larger parameters and used in the distributed decryption protocol.

Distributed decryption. To perform distributed decryption of the inference result without compromising the secret key, we use the noise flooding technique for the TFHE scheme, as detailed in [5].

The distributed decryption protocol among m wardens for an LWE ciphertext c proceeds as follows:

(1) The ciphertext c is switched to a ciphertext $\bar{c} := (\bar{\mathbf{c}}_1, \bar{c}_2)$ with larger parameters using a special bootstrapping operation (`TFHE.SwitchSquash` in [5]). This switch to larger parameters is necessary to prevent decryption errors due to the additional noise introduced by the noise flooding technique. Let Q be the new ciphertext modulus, and $\bar{\Delta} = Q/p$. The new ciphertext is of the form:

$$\bar{c} := (\bar{\mathbf{a}},\ \bar{\mathbf{a}} \cdot \bar{\mathbf{sk}} + \bar{\Delta} \cdot x + \bar{e} \pmod{Q}).$$

(2) Let $\bar{\chi}$ be the noise distribution for noise flooding. Warden i samples $e_{f,i} \leftarrow \bar{\chi}$ and computes the decryption share for $\bar{c}$ as:

$$[v_{\bar{c}}]_i := \bar{\mathbf{c}}_1 \cdot [\bar{\mathbf{sk}}]_i + e_{f,i} \pmod{Q}.$$

(3) To decrypt $\bar{c}$, the shares are aggregated as:

$$\sum_{i=1}^{m} [v_{\bar{c}}]_i = \bar{\mathbf{a}} \sum_{i=1}^{m} [\bar{\mathbf{sk}}]_i + \sum_{i=1}^{m} e_{f,i} = \bar{\mathbf{a}} \cdot \bar{\mathbf{sk}} + \sum_{i=1}^{m} e_{f,i}.$$

Using this aggregation of the shares, one can decrypt the ciphertext as follows:

$$x^* := \bar{c}_2 - (\bar{\mathbf{a}} \cdot \bar{\mathbf{sk}} + \sum_{i=1}^{m} e_{f,i})$$

$$= \bar{\mathbf{a}} \cdot \bar{\mathbf{sk}} + \bar{\Delta} \cdot x + \bar{e} - \bar{\mathbf{a}} \cdot \bar{\mathbf{sk}} - \sum_{i=1}^{m} e_{f,i} = \bar{\Delta} \cdot x + (\bar{e} - \sum_{i=1}^{m} e_{f,i}).$$

The message is obtained by scaling and rounding:

$$x = \lceil (x^* \pmod{Q}) \cdot \bar{\Delta}^{-1} \rfloor.$$

Distributed reencryption. To perform distributed reencryption among m wardens of an LWE ciphertext $c := (\mathbf{c}_1, c_2)$ under secret key $\mathbf{sk}$ to a ciphertext c' under a key $\mathbf{sk}'$, the protocol proceeds as follows:

(1) Let χ be the noise distribution for the TFHE scheme. Warden i samples $e_i \leftarrow \chi$ and computes the reencryption share for c as:

$$[u]_i = -\mathbf{c}_1 \cdot [\mathbf{sk}]_i + \mathbf{c}_1 \cdot [\mathbf{sk}']_i + e_i \pmod{q}$$

(2) To compute the reencrypted ciphertext c', the shares are aggregated as:

$$\sum_{i=1}^{m} [u]_i = -\mathbf{a} \cdot \sum_{i=1}^{m} [\mathbf{sk}]_i + \mathbf{a} \cdot \sum_{i=1}^{m} [\mathbf{sk}']_i + \sum_{i=1}^{m} e_i$$

$$= -\mathbf{a} \cdot \mathbf{sk} + \mathbf{a} \cdot \mathbf{sk}' + \sum_{i=1}^{m} e_i$$

Using the aggregation of the shares, the new ciphertext will be $c' := (\mathbf{c}_1, c_2')$, where:

$$c_2' := c_2 - \mathbf{a} \cdot \mathbf{sk} + \mathbf{a} \cdot \mathbf{sk}' + \sum_{i=1}^{m} e_i$$

$$= \mathbf{a} \cdot \mathbf{sk} + \Delta \cdot x + e - \mathbf{a} \cdot \mathbf{sk} + \mathbf{a} \cdot \mathbf{sk}' + \sum_{i=1}^{m} e_i$$

$$= \mathbf{a} \cdot \mathbf{sk}' + \Delta \cdot x + (e + \sum_{i=1}^{m} e_i) \pmod{q}$$

5.2 Optimizations

An important optimization of the protocol can be made by using the SISA framework [7] mentioned earlier. Recall that SISA splits the dataset into N disjoint shards $D_1, \ldots, D_N$ which are used to train N distinct model instances $M_1, \ldots, M_N$. For inference, one runs inference on each model and the final result is computed using an aggregation method on their outputs. This approach reduces the complexity of retraining-based unlearning by limiting the required retraining to only one shard.

Applying SISA to our solution requires some changes to the scheme introduced in Figure 4. A straightforward way to implement the SISA strategy would be to have all the models and shards encrypted under the same secret key. However, after each unlearning request, *all* models and shards would need to be reencrypted under a new key to prevent leakage of the unlearned data. This would be expensive and would not benefit much of the SISA sharding. A preferable efficient approach is to have each model and shard encrypted under a different key. Doing so means that, whenever unlearning is triggered, only the models and the shard concerned by the request need to change their secret key.

An issue that comes up in this case is though that during inference, each model produces results encrypted under a different key, and thus aggregation cannot be performed directly on those results. Letting the client perform the aggregation limits the defences a system can employ against leakage from black-box attacks, such as the restriction of the output vector to the top k classes to protect against member inference attacks [37]. Our solution consists in introducing a an additional *inference key*. Then, we reencrypt the outputs of each model to this inference key. The wardens can then compute aggregation on those inference results and perform a distributed decryption to reveal the output. This inference key needs to be changed (i.e., deleted and replaced by a fresh one) after each unlearning operation to make sure that the inference results under each shard's model (before the aggregation) are never leaked. The exact changes to the protocols of Figure 4 to implement SISA are described in detail in Section C, where we also argue that this approach preserves security.

6 Implementation

We implemented our FHE-based solution of Section 5 in Python and Rust. Our code is available open-source.[4] Our implementation relies on the Threshold-FHE [45] and Concrete-ML [44] libraries by Zama. These libraries provide implementations of distributed protocols and useful scripts for ML applications based on the TFHE scheme.[5] In particular, we adapted and extended the Threshold-FHE library to fit our use case, by adding: the ciphertext reencryption functionality for TFHE, as outlined in section 5.1; additive m-out-of-m secret sharing, instead of Shamir's secret sharing; the implementation of the Galois ring $\mathbb{Z}_{2^k} \cong GR(2^k, 1)$, $k \in \{64, 128\}$ for compatibility, tailoring existing protocols, such as the PRSS, for the semi-honest security setting. As part of the distributed key generation protocol,

[4] https://github.com/graikos/FHorgEt-threshold-fhe

[5] While Concrete-ML has a functionality to import TFHE parameters from TFHE-rs, it currently lacks an extension to import parameters from Threshold-FHE (planned as future work by Zama). We benchmarked the components based on Concrete-ML using slightly different parameters from TFHE-rs; we expect this to have negligible impact on the timings.

in order to generate the shared keys, we need to generate shared random bits, which in turn requires Beaver triples over the ring $\mathbb{Z}_{2^k}$. We did not implement the Beaver triple generation over $\mathbb{Z}_{2^k}$, as one can reuse existing works and implementations, such as [13, 26].

7 Experimental Evaluation

We ran our experiments in a virtual machine running Debian GNU/Linux running a 2017 Xeon-Gold-6154 processor with 8 cores at 3GHz and with 32 GB of RAM. In our measurements, we used only one core to run the wardens and 8 cores for the evaluator, thereby modeling that wardens can be implemented by moderately powerful machines. The distributed protocols were executed on the same machine simulating a network with high latency of around 50ms and a transfer rate of 500Mbit per second to model realistic scenarios in which the wardens are located in different networks (e.g., over the Internet). We remark that we do not fully exploit the parallelism that would be available for most of these algorithms in a realistic scenario.

To benchmark our system, we implemented an ML application based on a logistic regression model for binary classification.

We ran a variety of experiments to measure the scalability of the system with respect to: the number of wardens (from 3 to 7), the size of the data type (unsigned integers between 4 and 64 bits, denoted u4, u8, ..., u64), the size of the dataset, and the impact of the SISA optimization. All the reported timings are the median over 10 executions. First, we present our main results fixing 3 wardens and the u8 data type. Next, in Section 7.1.1 we discuss the impact of the SISA optimization on the performance of our system. Finally, in Section 7.1.2 we discuss the performance impact of increasing wardens and data types.

We used logistic regression with the Iris dataset [17], which is a classical benchmark dataset composed of 150 data points that are small float values (quantized with u8 integers) representing flower descriptions. The data points in this dataset are composed of 4 features to describe the flowers from which we only use 2, following the example present in the ConcreteML library. Furthermore, in order to test the system with larger datasets we used random datasets with a varying number of elements, $\{500, 1000, 5000, 10000\}$, of 30 features each.

Summary of Results. From our experimental campaign, we draw the following conclusions. The overall system overhead arises primarily from three components: distributed key generation (DKG), FHE-based training, and distributed reencryption. From these, the DKG phase incurs the largest overhead in our benchmarks, but is model-independent and can be run in a preprocessing phase. During the online phase, the cost introduced by reencryption is very small compared to training and also model-independent. As a result, the relative share of reencryption overhead is expected to decrease in scenarios involving more complex architectures (e.g., deep neural networks). Therefore, we argue that the dominant cost of unlearning is actually FHE-based training, which is the only cost that depends on model complexity. From this perspective, our evaluation confirms that secure machine unlearning can be achieved with performances that are similar to those of secure training, and expect that future advances in FHE-based ML training will translate to improvements to the cost of unlearning. Separately, the impact of SISA is very significant, reducing the cost of unlearning by a factor of (roughly) the number of SISA shards used. The following section provides additional details about our results.

7.1 Evaluation Results

We report the performance of the crucial protocols in our system, namely initialization (i.e., distributed key generation), upload (i.e., encryption), training, inference, and unlearning. Our goal is to show the relative costs of each of the components of our solution, in particular to provide insights into the overhead of the unlearning's cost relative to bare-bones FHE-based training.

Initialization. The initialization phase consists of running the DKG protocol among the wardens. For 3 wardens, the DKG takes around 2h20m, of which 3m26s are for key generation and the remaining time is for generating shared random bits (not including the cost of generating Beaver triples). The DKG outputs a public key of 1.2GB of size, available to every party in the system, and shares of the secret key, each around 17KB, available to the wardens.

We remark that recent work [8] improves the asymptotic costs of shared random bit generation. As bit generation is scheme-agnostic, our scheme could rely on this algorithm for better efficiency. Unfortunately, no implementation is available and we leave its benchmarking for future work.

Upload. The cost for a client to upload a data item is that of encrypting the data. It depends on the data type and benefits significantly from the use of multiple cores. As we report in Table 1, this cost is rather practical:

Data type	u4	u8	u16	u32	u64
Time (ms), single core	93	169	347	675	1384
Time (ms), 8 cores	23	27	47	92	207

Table 1: Time to encrypt different data types using TFHE.

Unlearning. We recall that unlearning a data item mainly consists of three main operations (in addition to deletion): DKG between the wardens, distributed reencryption (which in turn requires each warden to generate reencryption shares for the data items to be kept, and the evaluator to aggregate these shares), training on encrypted data at the evaluator. Since the DKG protocol is data independent, we let the wardens run it in a preprocessing phase and thus we do not include its cost as part of the unlearning operation (see above for the DKG cost of this preprocessing).

In Figure 5, we show the cost of unlearning one data item in a regression model built from datasets of varying size: 150 data points for the Iris dataset and $\{500, 1,000, 5,000, 10,000\}$ data points for the random dataset. We emphasize that the unlearning cost is largely dominated by the training algorithm on the encrypted dataset. For instance, after deletion, for a dataset of 999 u8 datapoints and 3 wardens, training takes around 1 hour and 34 minutes whereas distributed reencryption takes about 3 minutes.

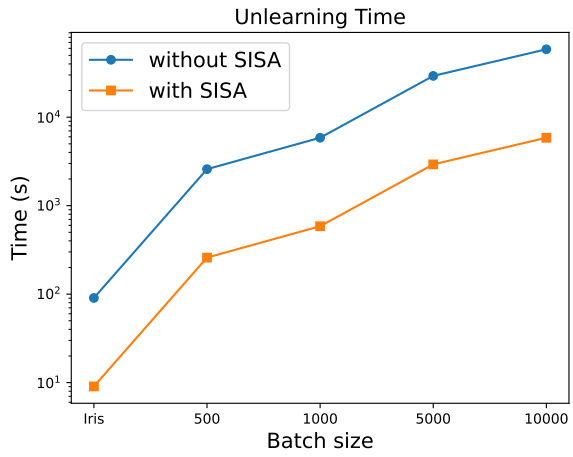

Figure 5: Unlearning time for datasets of varying size with and without SISA with 10 shards, using 3 wardens.

Training and Inference. The cost of training the logistic regression model is essentially the same as that reported as part of unlearning. The inference cost is given by that of running the inference algorithm on encrypted data at the evaluator—about 3.95ms for the Iris dataset with 2 features, and 9.75ms for the random dataset with 30 features—plus the execution of distributed decryption by the wardens, about 4.5s.

7.1.1 On the impact of SISA. We implemented the SISA optimization in the context of logistic regression. First, we tested SISA with 10 shards on plaintext data for training and inference based on logistic regression, using majority vote between the shard models. We did not observe significant changes in the accuracy of the model thus validating it as a viable option for regression-based models. Then we evaluated the performance of using SISA. We observed no change in training (as the evaluator runs more trainings but of smaller size each) and upload. For initialization, unlearning and inference, see below.

Initialization. When using SISA, one must run one DKG for each shard. Thus, the initialization cost grows linearly with the number of shards (though all these DKG could be fully parallelized since each execution is independent).

Unlearning. To observe the benefit of SISA in the unlearning protocol, we refer to Figure 5. For a dataset of n elements split in $N = 10$ shards, timing includes: the cost of generating the reencryption shares for $n/N - 1$ ciphertexts by the 3 wardens; the cost of aggregating the $3(n/N - 1)$ reencryption shares at the evaluator; the cost of re-training the model with the new dataset of $n/N - 1$ points. The timings also include the network cost of transferring the ciphertexts from the evaluator to the wardens and to receive back the reencryption shares. On the other hand, we did not include the cost of generating fresh epoch keys and inference key (i.e., 11 DKGs), as these DKG are data-independent and could be executed by the wardens offline.

As one can observe, the SISA optimization (with 10 shards) reduces the unlearning cost by roughly a factor of 10. Also, as in the case of unlearning with a single shard, the cost is largely dominated by re-training as the distributed reencryption is rather cheap compared to it, around 30% for the Iris dataset and 3% for the random dataset.

We note that this gain is possible in our logistic regression benchmarks because we validated that it is sufficient to train the model on one shard with n/N points while keeping a comparable level of accuracy. This is possible thanks to the simplicity of the model. For more complex models, training on one shard may require more iterations. We leave this exploration as future work.

Inference. With SISA, after computing the encrypted votes of N inference evaluations, the wardens must perform N distributed reencryptions, an aggregation (which is a cheap evaluation of a sum of ciphertexts in our use case), and a distributed decryption. Therefore, the computational overhead of SISA on inference grows linearly with the number of shards. However, in practice this overhead has a small impact since the single distributed decryption dominates. Concretely, in the case of 10 shards in the random dataset benchmark, inference takes a total of 6.6s, which is only 2.1s slower than without SISA.

7.1.2 Scaling with more wardens and larger data types. Figure 6 illustrates how the running time of the main distributed operations of our protocol changes when increasing the number of wardens and the data type. We observe that the number of wardens has little to no impact on the execution time. Therefore, we believe that scaling our system to support many wardens will not pose a significant performance challenge. On the other hand, the size of the datatype does impact the computation time, as larger elements (and thus ciphertexts) need to be processed in FHE. As shown in Figure 6, this increase is linear in the number of bits of the datatype except for the reencryption share generation, for which it grows logarithmically.

8 Related Work

A growing body of research has explored the use of FHE to enable privacy-preserving machine learning. A substantial portion of this work focuses on performing inference over encrypted inputs using pre-trained models. Notable early work includes CryptoNets [19], which demonstrated inference on encrypted MNIST images using leveled FHE and nGraph-HE [3], an extension to the Intel nGraph Deep Learning compiler that accommodates for scenarios where data is encrypted. Other work focuses on boolean neural networks [35], discretized neural networks [6], and deep convolutional neural networks [28].

In contrast, relatively few works address both training and inference over encrypted data due to the computational costs and difficulties associated with, e.g., encrypted gradient descent. MLaaS [23] uses the client for ciphertext refresh. CryptoDL [22] proposed approximating activation functions with low-degree polynomials to enable encrypted training of deep learning models on small datasets. Nandakumar et al., [31] show a number of optimizations - ranging from ciphertext packing to network simplification - to show feasibility of training over encrypted data. We note that optimization proposed in [22, 31] can be, in principle, also applied to our framework to speed up machine learning tasks.

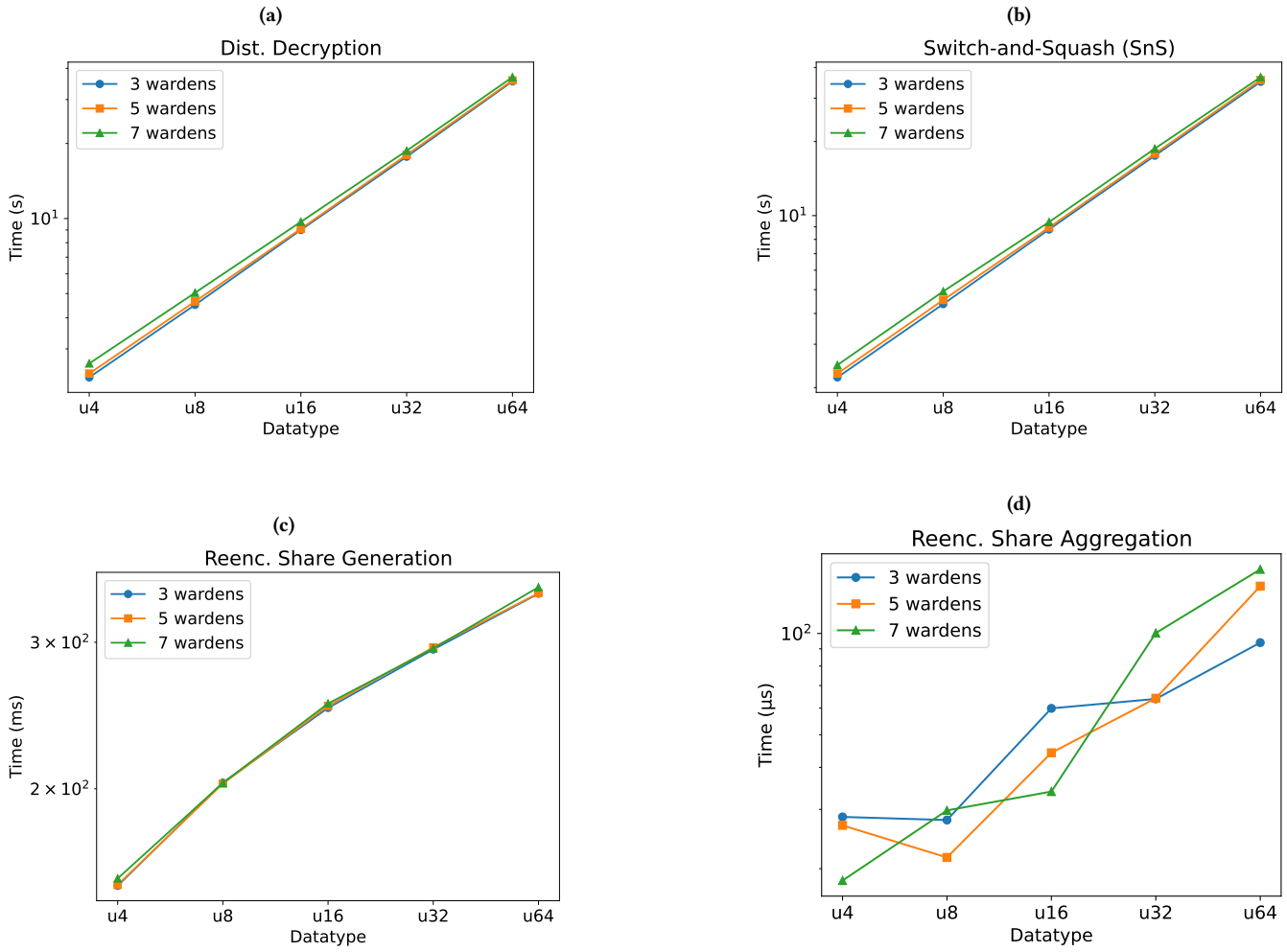

Figure 6: (a) End-to-end Distributed Decryption of a ciphertext, including transfer of ciphertext from evaluator to wardens; (b) Switch-and-Squash subprocedure, included in Distributed Decryption; (c) Generation of reencryption shares, including transfer of ciphertext to wardens, and transfer of shares back to evaluator; (d) Aggregation of reencryption shares run on a single core, assuming shares and ciphertext are loaded in memory. All plots use logarithmic y-axis except for (c).

Beyond purely FHE-based methods, other works use MPC or a mix of MPC and FHE for training [25, 33, 34] or for training and inference [14, 27, 30, 40, 41].

A natural alternative to cryptography-based machine unlearning is to consider the use of TEEs (trusted execution environments), which provide hardware-based isolation and attestation that sensitive data is handled securely. These technologies allow model owners to perform training, inference, and deletion within the trusted environment, without exposing plaintext data [32, 39]. While this is possible in principle, running a main server in a TEE would require a massive trusted computing base, which is discouraged. On the plus side, one can leverage simpler encryption (no need for FHE) and some guarantees that the computation was carried out honestly. On the minus side, TEE is a hardware assumption based on trust, whereas FHE (or MPC, or other cryptography-based privacy-enhancing technologies) are solutions that provide provably secure guarantees. In our FHE-based scheme, a middle-ground solution is to run the wardens in TEEs to increase confidence in their honesty. This would not require a huge trusted computing base, but does not prevent the use of FHE in the main server.

Acknowledgments

This work was done while Damien Robissout and Georgios Raikos were at IMDEA Software Institute.

This work is supported by the PICOCRYPT project that has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (Grant agreement No. 101001283), partially funded by

the grants PID2022-142290OB-I00 and CEX2024-001471-M, funded by MCIN/AEI/10.13039/501100011033/ FEDER, UE, and partially funded by Ministerio de Universidades (FPU21/00600).

The authors used generative AI-based tools to revise the text, improve flow and correct typos, grammatical errors, and awkward phrasing. The authors also used AI to help solve compilation errors in figures.

References

[1] 2016. Regulation (EU) 2016/679 of the European Parliament and of the Council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation) (Text with EEA relevance). , 88 pages.

[2] Donald Beaver. 1992. Efficient Multiparty Protocols Using Circuit Randomization. In *Advances in Cryptology — CRYPTO '91*, Joan Feigenbaum (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 420–432.

[3] Fabian Boemer, Anamaria Costache, Rosario Cammarota, and Casimir Wierzynski. 2019. nGraph-HE: A Graph Compiler for Deep Learning on Homomorphically Encrypted Data. In *Proceedings of the 16th ACM International Conference on Computing Frontiers*. 3–13.

[4] Dan Boneh, Amit Sahai, and Brent Waters. 2011. Functional Encryption: Definitions and Challenges. In *TCC 2011 (LNCS, Vol. 6597)*, Yuval Ishai (Ed.). Springer, Berlin, Heidelberg, 253–273. https://doi.org/10.1007/978-3-642-19571-6_16

[5] Carl Bootland, Kelong Cong, Daniel Demmler, Tore Kasper Frederiksen, Benoit Libert, Jean-Baptiste Orfila, Dragos Rotaru, Nigel P. Smart, Titouan Tanguy, Samuel Tap, and Michael Walter. 2025. Threshold (Fully) Homomorphic Encryption. Cryptology ePrint Archive, Paper 2025/699. https://eprint.iacr.org/2025/699

[6] Florian Bourse, Michele Minelli, Matthias Minihold, and Pascal Paillier. 2018. Fast Homomorphic Evaluation of Deep Discretized Neural Networks. In *Advances in Cryptology - CRYPTO 2018 - 38th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 19-23, 2018, Proceedings, Part III (Lecture Notes in Computer Science, Vol. 10993)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, 483–512. https://doi.org/10.1007/978-3-319-96878-0_17

[7] Lucas Bourtoule, Varun Chandrasekaran, Christopher A. Choquette-Choo, Hengrui Jia, Adelin Travers, Baiwu Zhang, David Lie, and Nicolas Papernot. 2021. Machine Unlearning. In *42nd IEEE Symposium on Security and Privacy, SP 2021, San Francisco, CA, USA, 24-27 May 2021*. IEEE, 141–159. https://doi.org/10.1109/SP40001.2021.00019

[8] Zvika Brakerski, Offir Friedman, Avichai Marmor, Dolev Mutzari, Yuval Spiizer, and Ni Trieu. 2025. Threshold FHE with Efficient Asynchronous Decryption. Cryptology ePrint Archive, Paper 2025/712. https://eprint.iacr.org/2025/712

[9] José Cabrero-Holgueras and Sergio Pastrana. 2021. SoK: Privacy-Preserving Computation Techniques for Deep Learning. *Proc. Priv. Enhancing Technol.* 2021, 4 (2021), 139–162. https://doi.org/10.2478/POPETS-2021-0064

[10] Yinzhi Cao and Junfeng Yang. 2015. Towards Making Systems Forget with Machine Unlearning. In *2015 IEEE Symposium on Security and Privacy*. IEEE Computer Society Press, 463–480. https://doi.org/10.1109/SP.2015.35

[11] Ignacio Cascudo, Anamaria Costache, Daniele Cozzo, Dario Fiore, Antonio Guimarães, and Eduardo Soria-Vazquez. 2025. Verifiable Computation for Approximate Homomorphic Encryption Schemes. In *CRYPTO 2025, Part VII (LNCS)*. Springer, Cham, 643–677. https://doi.org/10.1007/978-3-032-01907-3_21

[12] Ilaria Chillotti, Nicolas Gama, Mariya Georgieva, and Malika Izabachène. 2020. TFHE: Fast Fully Homomorphic Encryption Over the Torus. *Journal of Cryptology* 33, 1 (Jan. 2020), 34–91. https://doi.org/10.1007/s00145-019-09319-x

[13] Ronald Cramer, Ivan Damgård, Daniel Escudero, Peter Scholl, and Chaoping Xing. 2018. SPD $\mathbb{Z}_{2^k}$: Efficient MPC mod 2^k for Dishonest Majority. In *CRYPTO 2018, Part II (LNCS, Vol. 10992)*, Hovav Shacham and Alexandra Boldyreva (Eds.). Springer, Cham, 769–798. https://doi.org/10.1007/978-3-319-96881-0_26

[14] Anders P. K. Dalskov, Daniel Escudero, and Marcel Keller. 2021. Fantastic Four: Honest-Majority Four-Party Secure Computation With Malicious Security. In *30th USENIX Security Symposium, USENIX Security 2021, August 11-13, 2021*, Michael D. Bailey and Rachel Greenstadt (Eds.). USENIX Association, 2183–2200. https://www.usenix.org/conference/usenixsecurity21/presentation/dalskov

[15] Ivan Damgård, Valerio Pastro, Nigel P. Smart, and Sarah Zakarias. 2012. Multiparty Computation from Somewhat Homomorphic Encryption. In *CRYPTO 2012 (LNCS, Vol. 7417)*, Reihaneh Safavi-Naini and Ran Canetti (Eds.). Springer, Berlin, Heidelberg, 643–662. https://doi.org/10.1007/978-3-642-32009-5_38

[16] Thorsten Eisenhofer, Doreen Riepel, Varun Chandrasekaran, Esha Ghosh, Olga Ohrimenko, and Nicolas Papernot. 2025. Verifiable and Provably Secure Machine Unlearning. In *2025 IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*. 479–496. https://doi.org/10.1109/SaTML64287.2025.00033

[17] Ronald A Fisher. 1936. The use of multiple measurements in taxonomic problems. *Annals of eugenics* 7, 2 (1936), 179–188.

[18] Jack Foster, Stefan Schoepf, and Alexandra Brintrup. 2024. Fast Machine Unlearning without Retraining through Selective Synaptic Dampening. In *Thirty-Eighth AAAI Conference on Artificial Intelligence, AAAI 2024, Thirty-Sixth Conference on Innovative Applications of Artificial Intelligence, IAAI 2024, Fourteenth Symposium on Educational Advances in Artificial Intelligence, EAAI 2014, February 20-27, 2024, Vancouver, Canada*, Michael J. Wooldridge, Jennifer G. Dy, and Sriraam Natarajan (Eds.). AAAI Press, 12043–12051. https://doi.org/10.1609/AAAI.V38I11.29092

[19] Ran Gilad-Bachrach, Nathan Dowlin, Kim Laine, Kristin Lauter, Michael Naehrig, and John Wernsing. 2016. CryptoNets: Applying Neural Networks to Encrypted Data with High Throughput and Accuracy. In *International Conference on Machine Learning (ICML)*. 201–210.

[20] Aditya Golatkar, Alessandro Achille, and Stefano Soatto. 2020. Eternal Sunshine of the Spotless Net: Selective Forgetting in Deep Networks. In *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 9301–9309. https://doi.org/10.1109/CVPR42600.2020.00932

[21] Chuan Guo, Tom Goldstein, Awni Hannun, and Laurens Van Der Maaten. 2020. Certified data removal from machine learning models. In *Proceedings of the 37th International Conference on Machine Learning (ICML'20)*. JMLR.org, Article 359, 11 pages.

[22] Ehsan Hesamifard, Hassan Takabi, Mahdi Ghasemi, and Richard Wright. 2017. CryptoDL: Deep Neural Networks over Encrypted Data. *arXiv preprint arXiv:1711.05189* (2017).

[23] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N. Wright. 2018. Privacy-preserving Machine Learning as a Service. *Proc. Priv. Enhancing Technol.* 2018, 3 (2018), 123–142. https://doi.org/10.1515/POPETS-2018-0024

[24] Marc Joye. 2024. TFHE Public-Key Encryption Revisited. In *Topics in Cryptology – CT-RSA 2024*, Elisabeth Oswald (Ed.). Vol. 14643. Springer Nature Switzerland, Cham, 277–291. https://doi.org/10.1007/978-3-031-58868-6_11

[25] Chiraag Juvekar, Vinod Vaikuntanathan, and Anantha Chandrakasan. 2018. GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In *USENIX Security 2018*, William Enck and Adrienne Porter Felt (Eds.). USENIX Association, 1651–1669. https://www.usenix.org/conference/usenixsecurity18/presentation/juvekar

[26] Marcel Keller. 2020. MP-SPDZ: A Versatile Framework for Multi-Party Computation. In *ACM CCS 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM Press, 1575–1590. https://doi.org/10.1145/3372297.3417872

[27] Brian Knott, Shobha Venkataraman, Awni Y. Hannun, Shubho Sengupta, Mark Ibrahim, and Laurens van der Maaten. 2021. CrypTen: Secure Multi-Party Computation Meets Machine Learning. In *Advances in Neural Information Processing Systems 34: Annual Conference on Neural Information Processing Systems 2021, NeurIPS 2021, December 6-14, 2021, virtual*, Marc'Aurelio Ranzato, Alina Beygelzimer, Yann N. Dauphin, Percy Liang, and Jennifer Wortman Vaughan (Eds.). 4961–4973. https://proceedings.neurips.cc/paper/2021/hash/2754518221cfbc8d25c13a06a4cb8421-Abstract.html

[28] Eunsang Lee, Joon-Woo Lee, Junghyun Lee, Young-Sik Kim, Yongjune Kim, Jong-Seon No, and Woosuk Choi. 2022. Low-Complexity Deep Convolutional Neural Networks on Fully Homomorphic Encryption Using Multiplexed Parallel Convolutions. In *International Conference on Machine Learning, ICML 2022, 17-23 July 2022, Baltimore, Maryland, USA (Proceedings of Machine Learning Research, Vol. 162)*, Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvári, Gang Niu, and Sivan Sabato (Eds.). PMLR, 12403–12422. https://proceedings.mlr.press/v162/lee22e.html

[29] Jakub Łucki, Boyi Wei, Yangsibo Huang, Peter Henderson, Florian Tramèr, and Javier Rando. 2025. An Adversarial Perspective on Machine Unlearning for AI Safety. *Transactions on Machine Learning Research* (2025). https://doi.org/10.48550/arXiv.2409.18025

[30] Payman Mohassel and Peter Rindal. 2018. ABY^3: A Mixed Protocol Framework for Machine Learning. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, CCS 2018, Toronto, ON, Canada, October 15-19, 2018*, David Lie, Mohammad Mannan, Michael Backes, and XiaoFeng Wang (Eds.). ACM, 35–52. https://doi.org/10.1145/3243734.3243760

[31] Karthik Nandakumar, Nalini K. Ratha, Sharath Pankanti, and Shai Halevi. 2019. Towards Deep Neural Network Training on Encrypted Data. In *IEEE Conference on Computer Vision and Pattern Recognition Workshops, CVPR Workshops 2019, Long Beach, CA, USA, June 16-20, 2019*. Computer Vision Foundation / IEEE, 40–48. https://doi.org/10.1109/CVPRW.2019.00011

[32] Konstantina Papafragkaki and Giorgos Vasiliadis. 2025. InferONNX: Practical and Privacy-Preserving Machine Learning Inference Using Trusted Execution Environments. In *Detection of Intrusions and Malware, and Vulnerability Assessment*, Manuel Egele, Veelasha Moonsamy, Daniel Gruss, and Michele Carminati (Eds.). Springer Nature Switzerland, Cham, 25–43.

[33] Deevashwer Rathee, Mayank Rathee, Nishant Kumar, Nishanth Chandran, Divya Gupta, Aseem Rastogi, and Rahul Sharma. 2020. CrypTFlow2: Practical 2-Party Secure Inference. In *CCS '20: 2020 ACM SIGSAC Conference on Computer and Communications Security, Virtual Event, USA, November 9-13, 2020*, Jay Ligatti, Xinming Ou, Jonathan Katz, and Giovanni Vigna (Eds.). ACM, 325–342. https://doi.org/10.1145/3372297.3417274

[34] Bita Darvish Rouhani, M. Sadegh Riazi, and Farinaz Koushanfar. 2019. XONN: XNOR-based Oblivious Deep Neural Network Inference. In *27th USENIX Security Symposium (USENIX Security 19)*. 1501–1518.

[35] Amartya Sanyal, Matt J. Kusner, Adrià Gascón, and Varun Kanade. 2018. TAPAS: Tricks to Accelerate (encrypted) Prediction As a Service. In *Proceedings of the 35th International Conference on Machine Learning, ICML 2018, Stockholmsmässan, Stockholm, Sweden, July 10-15, 2018 (Proceedings of Machine Learning Research, Vol. 80)*, Jennifer G. Dy and Andreas Krause (Eds.). PMLR, 4497–4506. http://proceedings.mlr.press/v80/sanyal18a.html

[36] Ayush Sekhari, Jayadev Acharya, Gautam Kamath, and Ananda Theertha Suresh. 2021. Remember what you want to forget: algorithms for machine unlearning. In *Proceedings of the 35th International Conference on Neural Information Processing Systems (NIPS '21)*. Curran Associates Inc., Red Hook, NY, USA, Article 1383, 12 pages.

[37] Reza Shokri, Marco Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership Inference Attacks Against Machine Learning Models. In *2017 IEEE Symposium on Security and Privacy (SP)*. 3–18. https://doi.org/10.1109/SP.2017.41

[38] Anvith Thudi, Hengrui Jia, Ilia Shumailov, and Nicolas Papernot. 2022. On the Necessity of Auditable Algorithmic Definitions for Machine Unlearning. In *USENIX Security 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 4007–4022. https://www.usenix.org/conference/usenixsecurity22/presentation/thudi

[39] Jean-Baptiste Truong, William Gallagher, Tian Guo, and Robert J Walls. 2021. Memory-efficient deep learning inference in trusted execution environments. In *2021 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 161–167.

[40] Sameer Wagh, Shruti Tople, Fabrice Benhamouda, Eyal Kushilevitz, Prateek Mittal, and Tal Rabin. 2021. Falcon: Honest-Majority Maliciously Secure Framework for Private Deep Learning. *Proc. Priv. Enhancing Technol.* 2021, 1 (2021), 188–208. https://doi.org/10.2478/POPETS-2021-0011

[41] Jean-Luc Watson, Sameer Wagh, and Raluca Ada Popa. 2022. Piranha: A GPU Platform for Secure Computation. In *31st USENIX Security Symposium, USENIX Security 2022, Boston, MA, USA, August 10-12, 2022*, Kevin R. B. Butler and Kurt Thomas (Eds.). USENIX Association, 827–844. https://www.usenix.org/conference/usenixsecurity22/presentation/watson

[42] Heng Xu, Tianqing Zhu, Lefeng Zhang, Wanlei Zhou, and Philip S. Yu. 2023. Machine Unlearning: A Survey. *ACM Comput. Surv.* 56, 1, Article 9 (aug 2023), 36 pages. https://doi.org/10.1145/3603620

[43] Xiaoyu Xu, Xiang Yue, Yang Liu, Qingqing Ye, Haibo Hu, and Minxin Du. 2025. Unlearning Isn't Deletion: Investigating Reversibility of Machine Unlearning in LLMs. *CoRR* abs/2505.16831 (2025).

[44] Zama. 2022. Concrete ML: a Privacy-Preserving Machine Learning Library using Fully Homomorphic Encryption for Data Scientists. https://github.com/zama-ai/concrete-ml.

[45] Zama. 2025. threshold-fhe: Threshold MPC protocols for FHE. https://github.com/zama-ai/threshold-fhe.

A MPC-based Secure Machine Unlearning

Protocol roles. For our MPC-based solution, we outline the following roles:

- **Data owners,** found in GDPR [1] as data subjects, are the owners of the data processed by the system. As such, they upload data items and may issue data unlearning requests.
- **Servers,** tasked with running the MPC protocols for training, inference, and unlearning.
- **Clients,** defined as the parties submitting inference requests.

We do not enforce a strict distinction between roles. As noted in Section 3.1, for instance, the inference functionality may be publicly accessible, meaning data owners, or servers, can act as clients and submit inference requests as well. Similarly, data owners may also take part in the MPC protocols as servers.

Protocol description. Our MPC-based solution involves n data items and s servers. Let $\mathcal{D} = \{\mathsf{xid}_i\}_{i\in[n]}$ be the dataset. We include a formal description in Figure 7. In the figure, we use the following syntax for the MPC protocols:

$\underline{\mathsf{Setup}(1^\lambda) \to \mathsf{pp}:}$ On input the security parameter 1^λ, output public parameters pp.

$\underline{\mathsf{Share}(x) \to ([x]_j)_{j\in[m]}:}$ On input x, output shares $[x]_j$ for every $j \in [m]$.

$\underline{\mathsf{Compute}(F, [x]_j) \to [y]_j:}$ On input a function F and an input share $[x]_j$, output an output share $[y]_j$. This algorithm is run interactively among the m servers.

$\underline{\mathsf{Rec}(([x]_j)_{j\in[m]}) \to x:}$ On input a set of shares $([x]_j)_{j\in[m]}$, output the reconstructed x.

Data upload. To implement $\mathsf{Upload}(\mathsf{uid}, x)$, data owner with identifier $\mathsf{uid} \in \mathcal{U}$ secret-shares x using an m-out-of-m secret sharing scheme. It distributes the shares $[x]_j$ for $j \in [m]$ to the m servers, and receives a unique identifier xid as a response. [6]

Training. To implement $\mathsf{Train}(M, \mathcal{D}; r)$ for a model M over a dataset $\mathcal{D} = \{\mathsf{xid}_i\}_{i\in[n]}$:

(1) Each server j holds shares of the data items $[x_i]_j$ corresponding to xid_i for all $i \in [n]$.
(2) All servers run an MPC protocol to train the model, obtaining secret shares of the trained model $[M]_j$.

Inference. To implement $\mathsf{Infer}(x)$ for a model M on input x, we proceed as follows:

(1) The client secret shares x for the servers, distributing $[x]_j$ for $j \in [m]$. Note that if x is not private, this step is not required.
(2) The servers retrieve the share $[M]_j$ of the model and run an MPC protocol with inputs $[M]_j$, $[x]_j$ for each j, receiving their own share of the output $[y]_j$.
(3) The servers send their shares $[y]_j$ to the intended recipient of the result, or they broadcast them, in case of publicly available inference results.
(4) A recipient collects each $[y]_j$ for every $j \in [m]$ and reconstructs the output y.

Unlearning. To implement $\mathsf{Unlearn}(\mathsf{xid})$ for a model M and an item identified by xid, obtaining a new model G:

(1) The servers verify that the item identified by xid exists and has been used to train M. If not, the servers abort, producing $b = 0$, and the protocol exits.
(2) The servers retrieve the shares $[M]_j$ and $[x_{i^*}]_j$ identified by xid and run the unlearning algorithm in MPC. In the event of unlearning with retraining, they also need to reuse $[x_k]_j$ for every other $k \in [n]$. The servers compute shares $[M']_j$ of the retrained model and produce $b = 1$.
(3) The servers delete their shares $[M]_j$, $[x_{i^*}]_j$.

[6] Depending on the context, unique identifiers could be generated by the data owners or the servers, using, for example, a pseudorandom generator or an auto-incrementing counter. While a concrete instantiation is beyond the scope of this work, it is crucial that the identifiers are (a) unique and (b) do not leak information about the private input x.

$\underline{\mathsf{Setup}(1^\lambda)}$**:**

- Output $\mathsf{pp} \leftarrow \mathsf{MPC.Setup}(1^\lambda)$.

$\underline{\mathsf{Train}(M, \mathcal{D}; r)}$**:**

- Each server S_j retrieves $D_j = \{[x_{\mathsf{xid}}]_j\}_{\mathsf{xid} \in \mathcal{D}}$ corresponding to $\mathcal{D} = \{\mathsf{xid}_i\}_{i \in [n]}$.
- Servers S_j agree on a unique model identifier fid for M. For instance, server S_1 creates fid.
- The servers jointly compute the training function, where S_j obtains a model share $[M_{\mathsf{fid}}]_j \leftarrow \mathsf{MPC.Compute}(T_{\mathsf{fid}}, D_j)$.
- S_j stores $[M_{\mathsf{fid}}]_j$ and outputs fid.

$\underline{\mathsf{Unlearn}(\mathcal{D}_f)}$ **(with retraining):**

- S_j checks if every $\mathsf{xid} \in \mathcal{D}_f$ exists and contributed to some model fid. Else, abort and output $b = 0$.
- S_j retrieves the shares $[x_{\mathsf{xid}}]_j$ for every $\mathsf{xid} \in \mathcal{D}_f$.
- Let $D_j = \{[x_{\mathsf{xid}}]_j : \mathsf{xid} \in \mathcal{D} \setminus \mathcal{D}_f\}$. S_j jointly runs training on D_j with the other servers, obtaining $[M'_{\mathsf{fid}}]_j \leftarrow \mathsf{MPC.Compute}(T_{\mathsf{fid}}, D_j)$ for every fid where some $\mathsf{xid} \in \mathcal{D}_f$ was used.
- If the computation succeeds, S_j deletes the following shares:
  - The former model shares $[M_{\mathsf{fid}}]_j$ for every retrained fid.
  - Every share $[x_{\mathsf{xid}}]_j$ for every $\mathsf{xid} \in \mathcal{D}_f$.
- If the protocol succeeds, output $b = 1$. Otherwise, output $b = 0$.

$\underline{\mathsf{Upload}(\mathsf{uid}, x)}$**:**

- uid generates a unique identifier xid.
- uid secret-shares x as $([x]_1, \ldots, [x]_m) \leftarrow \mathsf{MPC.Share}(x)$.
- uid sends $(\mathsf{xid}, [x]_j)$ to every $j \in [m]$.

$\underline{\mathsf{Infer}(\mathsf{fid}, q)}$**:**

- Server S_j jointly runs $[y]_j \leftarrow \mathsf{MPC.Compute}(I_{\mathsf{fid}}, ([M_{\mathsf{fid}}]_j, q))$ with all servers, where I_{fid} is the inference function of model fid.
- S_j sends $[y]_j$ to the client.
- The client reconstructs the output as $y \leftarrow \mathsf{MPC.Rec}(([y]_j)_{j \in [m]})$.

$\underline{\mathsf{Unlearn}(\mathcal{D}_f)}$ **(without retraining):**

- S_j checks if every $\mathsf{xid} \in \mathcal{D}_f$ exists and contributed to some model fid. Else, abort and output $b = 0$.
- S_j retrieves the shares $[x_{\mathsf{xid}}]_j$ for every $\mathsf{xid} \in \mathcal{D}_f$.
- For every model fid in which some $\mathsf{xid} \in \mathcal{D}_f$ was used, S_j jointly computes the unlearning function $[M'_{\mathsf{fid}}]_j \leftarrow \mathsf{MPC.Compute}(U_{\mathsf{fid}}, \{[x_{\mathsf{xid}}]_j\}_{\mathsf{xid} \in \mathcal{D}_f})$.
- If the computation succeeds, S_j deletes the following shares:
  - The former model shares $[M_{\mathsf{fid}}]_j$ for every retrained fid.
  - Every share $[x_{\mathsf{xid}}]_j$ for every $\mathsf{xid} \in \mathcal{D}_f$.
- If the protocol succeeds, output $b = 1$. Otherwise, output $b = 0$.

Figure 7: Our MPC-based construction of a secure machine unlearning protocol.

B Security Proof of our FHE-based Construction (Theorem 1)

We prove that our construction is a secure machine unlearning scheme with *strong forward security* for a corruption threshold of up to $m-1$ servers (one of them being the main server S_0), as defined in Figure 3. The proof proceeds via a series of game hops where the different games are described below. All games run between a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ and a challenger. Note that the adversary cannot corrupt all of the wardens.

- Game $(0, b)$ is the original security game $\mathsf{PRIV}_{m,m-1,b}$ in Figure 3, where x_b is uploaded and later unlearned.
- In Game $(1, b)$, we additionally abort the game if there is any non-unique xid or fid.
- In Game $(2, b)$, we additionally modify the $\mathcal{O}\mathsf{Train}$ oracle such that it trains both models M_{fid} and $\bar{M}_{\mathsf{fid}}$ locally (i.e., M_{fid} on $\mathcal{D} \cup \{\mathsf{xid}_b\}$ and $\bar{M}_{\mathsf{fid}}$ on $\mathcal{D} \cup \{\mathsf{xid}_{1-b}\}$). We remark that the Train algorithm is still executed in a distributed way.
- In Game $(3, b)$, we additionally replace the $\mathcal{O}\mathsf{Infer}$ oracle such that inference is executed locally on M_{fid}, and the adversary is given the output of $M_{\mathsf{fid}}(q)$. We note that the Infer protocol is still run, but its output is ignored.
- In Game $(4, b)$, we additionally replace all the distributed protocols from the construction in Figure 4 (key generation, share redistribution, encryption and decryption) by their corresponding simulators. Recall that all these protocols are secure multi-party computation protocols in the semi-honest model for a corruption threshold of up to $m - 1$ servers.
- In Game $(5, b)$, we replace all model ciphertexts c_{fid} corresponding to the first phase (i.e., before the reencryption triggered by the unlearning request) by encryptions of zero.

Next, we analyse the advantage of any PPT adversary $\mathcal{A}$ between each pair of games. We do so via a series of lemmas.

Lemma 1. *For any PPT adversary $\mathcal{A}$, we have that*

$$\left|\Pr[\mathcal{A}_b^{G0} = 1] - \Pr[\mathcal{A}_b^{G1} = 1]\right| = 0$$

Proof. The winning probability of the adversary does not change, as in our semi-honest setting the server S_0 follows the protocol. S_0 simply (a) creates unique fid for every model, and (b) rejects data uploads if xid has been used. □

Lemma 2. *For any PPT adversary $\mathcal{A}$, we have that*

$$\left|\Pr[\mathcal{A}_b^{G1} = 1] - \Pr[\mathcal{A}_b^{G2} = 1]\right| = 0$$

Proof. The view adversary also does not change between this pair of games due to the correctness of the FHE protocols, which guarantee that the output of running Infer locally is the same as the output of running it over (distributed) FHE when both the server and the wardens follow the protocol. In case the correctness of the FHE protocol is not perfect, but holds with all-but-negligible probability, both games would be statistically close as all parties involved follow the protocol. □

Lemma 3. *For any PPT adversary $\mathcal{A}$, we have that*

$$\left|\Pr[\mathcal{A}_b^{G2} = 1] - \Pr[\mathcal{A}_b^{G3} = 1]\right| = 0$$

Proof. The argument is identical to the proof of the previous lemma and follows by the correctness of FHE. □

Lemma 4. *For any PPT adversary $\mathcal{A}$ which makes up to q oracle queries in the game,*

$$\left|\Pr[\mathcal{A}_b^{G3} = 1] - \Pr[\mathcal{A}_b^{G4} = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. In short, the view of the adversary does not change as it cannot corrupt more than $m - 1$ servers. As all the MPC functionalities are secure for up to $m - 1$ corrupted servers, the step follows by the security of the distributed algorithms in the semi-honest model.

Formally, we prove the lemma via a series of hybrid games. Throughout the game, we have up to q executions of MPC protocols, corresponding to the q queries made by the adversary to the different game oracles. We define hybrids $H_0, \ldots, H_q$ such that in H_i, executions $1, \ldots, i$ are simulated (by the corresponding MPC simulator) whereas the remaining ones are executed normally. Note that Game $(3, b)$ corresponds to hybrid $H_{0,b}$ and Game $(4, b)$ corresponds to hybrid $H_{q,b}$. Also, note that in hybrid H_i, the long-term view of the adversary at the time of the $(i + 1)$-th execution contains the views of executions $1, \ldots, i$, which are all simulated. This is key to enable a reduction to stand-alone (one-shot) MPC security.

Now, consider an adversary $\mathcal{A}$ that interpolates between $H_{i,b}$ and $H_{i+1,b}$, this is, that outputs a different bit in $H_{i,b}$ and in $H_{i+1,b}$. Then, we build an adversary $\mathcal{B}$ against the security of the MPC protocols as follows. The reduction $\mathcal{B}$ proceeds in three phases: 1) simulate the prefix (executions $1, \ldots, i$), 2) use its MPC challenger to handle the $(i + 1)$-th execution, and 3) honestly execute the remaining suffix (executions $i + 2, \ldots, q$).[7] More precisely, we construct a PPT adversary $\mathcal{B}$ against the stand-alone security of the $(i + 1)$-th MPC invocation Π_{i+1} as follows.

- $\mathcal{B}$ internally runs $\mathcal{A}$ and emulates the overall game environment for $\mathcal{A}$. Whenever $\mathcal{A}$ issues oracle queries that correspond to distributed protocol invocations Π_j with $j \leq i$, $\mathcal{B}$ answers those queries by running the corresponding simulator $\mathcal{S}_{\Pi_j}$ on appropriate auxiliary inputs (see below).
- While simulating the first i invocations, $\mathcal{B}$ keeps track of the adversary's *simulated* long-term state

  $$\sigma_i := (\rho_{\mathcal{A}}, \mathsf{view}_1, \ldots, \mathsf{view}_i),$$

  where each view_j is the simulated corrupted-party view returned by $\mathcal{S}_{\Pi_j}$ and $\rho_{\mathcal{A}}$ denotes $\mathcal{A}$'s internal randomness. The simulator outputs are recorded exactly as they would appear to $\mathcal{A}$ in hybrid H_i.
- When $\mathcal{A}$ triggers the $(i + 1)$-th distributed invocation Π_{i+1}, $\mathcal{B}$ forwards the description of Π_{i+1} and the auxiliary input σ_i to its own MPC challenger. The challenger returns (depending on its internal challenge bit) the *real* corrupted view resulting from an honest execution of Π_{i+1}, or a *simulated* view produced by the challenger's simulation oracle.
- $\mathcal{B}$ supplies the view obtained from its MPC challenger to $\mathcal{A}$ as the transcript that the corrupted parties would see in the $(i + 1)$-th invocation.
- For any subsequent invocations Π_j with $j > i+1$, $\mathcal{B}$ continues the simulation of the global game exactly as in hybrids H_i and H_{i+1}: it honestly executes those protocols for all honest parties. Concretely, because $\mathcal{B}$ controls the environment it can run honest-party code (it has access to all public parameters and can sample internal randomness as required).
- Eventually $\mathcal{A}$ outputs a guess bit b'. $\mathcal{B}$ outputs the same bit as its own guess to its MPC challenger.

Observe that $\mathcal{B}$ perfectly matches the view presented to $\mathcal{A}$ in the following two cases:

- If the MPC challenger returns a *real* corrupted view for Π_{i+1}, then the joint distribution of the messages and local states given to the corrupted parties during the $(i + 1)$-th invocation is exactly the same as in hybrid H_i (where Π_{i+1} is run honestly). Moreover, because $\mathcal{B}$ simulated the prefix invocations $1, \ldots, i$ using the same simulators as in H_i, the global distribution presented to $\mathcal{A}$ is identical to H_i.
- If the MPC challenger returns a *simulated* view for Π_{i+1} (i.e., the output of the simulator $\mathcal{S}_{\Pi_{i+1}}$ on input σ_i), then the corrupted parties' view at invocation $(i + 1)$ is exactly as in hybrid H_{i+1} (where the $(i + 1)$-th invocation is simulated).

Thus, $\mathcal{B}$'s interaction with its MPC challenger perfectly implements either hybrid H_i or H_{i+1} for $\mathcal{A}$, depending on whether the challenger returned a real or simulated view. Therefore, the distinguishing advantage of $\mathcal{A}$ between these two hybrids is carried over to $\mathcal{B}$'s distinguishing advantage in the one-shot MPC security experiment. The lemma then follows by a standard hybrid argument over q. □

Lemma 5. *For any PPT adversary $\mathcal{A}$, we have that*

$$\left|\Pr[\mathcal{A}_b^{G4} = 1] - \Pr[\mathcal{A}_b^{G5} = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. In this step, we carry out a reduction to the semantic security of the FHE scheme. Again, we should formalize this step as a series of hybrid sub-games which iterate over the maximum number of oracle queries that the adversary can do, as this is an upper bound for the number of different models (and therefore model ciphertexts) used in the game. For simplicity, we illustrate the reduction for a single model identifier fid, as the hybrid argument follows in a standard manner.

Let $\mathcal{A}$ be an adversary against game $(4, b)$ which does not win in game $(5, b)$. We describe a PPT adversary $\mathcal{B}$ for the IND-CPA security game of FHE as follows:

- $\mathcal{B}$ receives FHE public parameters pp and a public key pk from its own challenger.
- $\mathcal{B}$ simulates an execution of Game $(4, b)$ for the m servers to $\mathcal{A}$ (running the ideal functionalities of all the distributed algorithms on pk) and allowing $\mathcal{A}$ to corrupt up to $m - 1$ servers.
- When $\mathcal{B}$ receives a $\mathcal{O}\mathsf{Upload}(\mathsf{uid}, x)$ query, it runs $c_{\mathsf{xid}} \leftarrow \mathsf{FHE.Enc}(\mathsf{pk}, x)$

[7] To clarify, we are assuming that for every distributed protocol instance Π (key generation, share redistribution, encryption, decryption, ...) the standard stand-alone semi-honest security guarantee holds. That is, there exists a PPT simulator $\mathcal{S}_\Pi$ such that for any (polynomial-size) auxiliary input aux given to the corrupted parties, the joint distribution of the corrupted parties' real view in a real execution of Π with auxiliary input aux is computationally indistinguishable from the output of $\mathcal{S}_\Pi(\mathsf{aux})$. We explicitly rely on the fact that the simulator may take arbitrary auxiliary input (which models prior transcripts, long-term keys, and any state carried across invocations).

- When $\mathcal{B}$ receives the $O\mathsf{Train}(m, \mathcal{D}; r)$ query, it runs training both locally and by running the ideal functionality among the servers. Then, it forwards both a zero string and M_{fid} to its challenger, which returns ciphertext c^*. Then, $\mathcal{B}$ returns c^* to $\mathcal{A}$.
- $\mathcal{B}$ continues the simulation, and when $\mathcal{A}$ outputs a guess b', $\mathcal{B}$ forwards the guess b' to its own challenger.

Note that the simulation carried out by $\mathcal{B}$ is perfect as it has all data and models in the clear, and so it can answer all oracle queries perfectly. Therefore, the advantage of $\mathcal{B}$ upper bounds the difference in advantage of $\mathcal{A}$ between games $(4, b)$ and $(5, b)$, and the lemma follows. □

Lemma 6. *For any PPT adversary $\mathcal{A}$, we have that*

$$\left|\Pr[\mathcal{A}_b^{G5} = 1] - \Pr[\mathcal{A}_{1-b}^{G5} = 1]\right| = \mathsf{negl}(\lambda).$$

Proof. Between games $(5, b)$ and $(5, 1-b)$, we can also carry out a reduction to the semantic security of FHE, which works as before except that the challenge ciphertext is now c_{xid^*}, which corresponds to x_b. To carry out the reduction, note that the only information about x_b that is given to the adversary is c_{xid^*}, since:

- x_b is never used in Stage 2 of the security game, as it is unlearned before Stage 2 starts.
- All Stage 1 model ciphertexts are encryptions of zero due to the previous game-hop.
- All $O\mathsf{Infer}$ queries are answered locally and do not depend on the bit b, as the game aborts if they do.
- All key material leaked to the adversary corresponds to Stage 2 and does not contain any information of the Stage 1 ciphertexts, as we are running the simulator of the distributed reencryption protocol.

Hence, the reduction can perfectly simulate these games without the knowledge of x_b, concluding the security analysis. □

The theorem follows by collecting the results from all the previous lemmas.

C Description of the SISA Optimization

For simplicity, we present the SISA optimization assuming a single model M, where copies of this model are trained with the dataset shards. This approach can be easily extended to support multiple models operating over the same dataset, by having instances of the different models for every shard. It can also support models over different datasets, at the cost of duplicate ciphertexts in case of dataset overlap.

The modifications applied to the protocol of Figure 4 are as follows:

Init. The wardens generate N joint FHE key pairs, one for each shard. Each warden j receives $\left\{([sk^{M_l}]_j, pk^{M_l}) \leftarrow \Pi_{\mathsf{DistKeyGen}}(\mathsf{pp})\right\}_{l\in[N]}$. They also generate an initial inference FHE key pair and receive $([sk^{I_0}]_j, pk^{I_0}) \leftarrow \Pi_{\mathsf{DistKeyGen}}(\mathsf{pp})$. A counter for unlearning requests $z \leftarrow 0$ is initialized.

Data upload. Data owner encrypts x under the public key pk^{M_l}, where l is the appropriate shard.

Training. The modified training functionality proceeds as follows:

- The dataset $\mathcal{D} = \{\mathsf{xid}_i\}_{i\in[n]}$ is split into N disjoint shards.
- For each shard l, the corresponding set of ciphertexts $\mathcal{D}_l$ is retrieved.
- Each model instance M_l is trained by running $c_{M_l} \leftarrow \mathsf{FHE.Eval}(pk^{M_l}, T, \mathcal{D}_l)$.

Inference. The modified functionality $\mathsf{Infer}(x)$ on input x proceeds as follows:

- The client sends x to the server.
- For each shard l, the model ciphertext c_{M^l} is retrieved. The inference function is evaluated on each model instance M_l as $c_{y^l} \leftarrow \mathsf{FHE.Eval}(pk^{M_l}, I, (c_{M_l}, x))$.
- All wardens compute reencryption shares:

$$\left\{[u_l^{sk^{M_l}\to sk^{I_z}}]_j \leftarrow \mathsf{ReEncShare}([sk^{M_l}]_j, [sk^{I_z}]_j, c_{y^l})\right\}_{j\in[m],l\in[N]}.$$

- The server aggregates the shares and obtains $\left\{c_l^{I_z} \leftarrow \mathsf{ReEnc}(\{[u_l^{sk^{M_l}\to sk^{I_z}}]_j\}_{j\in[m]})\right\}_{l\in[N]}$.
- Let A be the aggregation functionality for SISA. The server computes the inference result as $c_y \leftarrow \mathsf{FHE.Eval}(pk^{I_z}, A, \{c_l^{I_z}\}_{l\in[N]})$.
- The distributed decryption of c_y proceeds as in Figure 4.

Unlearning with retraining. The modified functionality $\mathsf{Unlearn}(\mathsf{xid})$ with retraining proceeds as follows:

- The server identifies the corresponding shard l for xid and verifies it exists and has been used to train M_l. If not, the server aborts, returning $b = 0$, and the protocol exits.
- The protocol proceeds as in Figure 4, updating only the model M_l for shard l and its corresponding keys instead of M.
- The wardens generate a new inference FHE key pair and receive $([sk^{I_{z+1}}]_j, pk^{I_{z+1}}) \leftarrow \Pi_{\mathsf{DistKeyGen}}(\mathsf{pp})$.
- All wardens delete their shares $[sk^{I_z}]_j$ and increment the counter $z \leftarrow z + 1$.

C.1 Security Analysis of SISA

It is not hard to extend our security analysis to capture the SISA optimizations. For this, we can follow the same steps in the security argument in Section B, except that not all shards are reencrypted, but only the one that corresponds to the unlearning requests is. Besides, we have to take into consideration the new inference key which does not exist in the previous protocol.

The modifications with respect to the game hops in the previous argument are as follows:

- Games $(0, b)$, $(1, b)$, $(2, b)$ and $(3, b)$ are identical, taking into account that inference is carried out (also locally by the challenger) via aggregation of the different shards. We denote the shard that contains the challenge input xid^* by l^*, which is known to the adversary.
- In Game $(4, b)$, we replace all the distributed protocols from the construction in Figure 4 (key generation, share redistribution, encryption and decryption) by their corresponding simulators, *including the reencryption steps for the inference key.*
- In Game $(5, b)$, we replace the model ciphertext $c_{M_{l^*}}$ corresponding to the first phase (i.e., before the reencryption triggered by the unlearning request) *and to the shard l^** by an encryption of zero. Besides, *we also replace the ciphertexts $c_{l^*}^{I_z}$*

corresponding to the outputs of the shard l^* upon an inference request by encryptions of zero.

Between games $(3, b)$ and $(4, b)$, the argument is exactly as before, as we are simply replacing an additional functionality by its simulator.

Between games $(4, b)$ and $(5, b)$, the argument also follows by the semantic security of FHE, except that we have to proceed in two steps: first for the model ciphertext, and then for the outputs of the shard l^* upon an inference request. We note that all other ciphertexts (corresponding to shards $l \neq l^*$) do not depend on the value of b, so they can remain unchanged across the proof.

Finally, between games $(5, b)$ and $(5, 1 - b)$, note that the shares of the former inference key are deleted, so all the material leaked to the adversary after unlearning corresponds to Stage 2 ciphertexts. The argument then follows as before.