Oracles for Privacy-Preserving Machine Learning

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science at George Mason University

By

Minh Quan Do
Bachelor of Science
George Mason University, 2019

Director: Dr. Foteini Baldimtsi, Professor
Department of Computer Science

Fall Semester 2022
George Mason University
Fairfax, VA

Dedication

I dedicate this thesis to all of the wonderful teachers who have helped me get to this point.

Acknowledgments

I would like to thank the following people who made this possible: Dr. Foteini Baldimtsi, Dr. Evgenios Kornaropoulos, and Dr. Giuseppe Ateniese.

Table of Contents

List of Figures

Abstract

ORACLES FOR PRIVACY-PRESERVING MACHINE LEARNING

Minh Quan Do

George Mason University, 2022

Thesis Director: Dr. Foteini Baldimtsi

Currently, the process of deploying machine learning models in production can leak information about the model such as model parameters. This leakage of information is problematic because it opens the door to a plethora of attacks that can compromise the privacy of the data used to train the model. In this thesis, we will introduce definitions for new primitives that are specifically designed for deploying machine learning models into production in such a way that guarantees the privacy of the model's parameters and the underlying dataset. We will also provide definitions for security, propose a scheme for deploying a model into production, and informally argue the security of our scheme.

Chapter 1: Introduction

On January 25th, 2017, Dr. Andrew Ng, former chief scientist at Baidu, co-founder of Coursera, and adjunct professor at Stanford University gave a lecture to the Stanford School of Business; in his lecture he proclaimed that AI is the "new electricity" [1]. Fast forward five years to today, we can definitively say that Dr. Andrew Ng's statements might be an understatement. In today's world, machine learning is used for a wide variety of applications; from recommending content and items on video streaming platforms and online retail, to predicting protein structures for drug discovery [2], to developing self-driving cars [3], and even defeating world players in board games like Go and Shogi [4].

At its core, machine learning is the process of enabling computers to "learn" how to conduct a specific task without being explicitly programmed to do so. For the most part, the *traditional approach* of developing a machine learning model has four main phases (we call these phases **the model development process**):

1. **Data collection**: aggregating data from various sources to create a dataset.

2. **Exploratory data analysis**: conducting basic data analysis to get a better understanding of the properties of the dataset.

3. **Data preparation and preprocessing**: data preparation includes cleaning and partitioning the dataset for training and testing purposes, removing outlier data samples, etc. Data preprocessing is the process of converting the raw data samples into feature vectors. This step could include dimensionality reduction, normalizing the attributes of the data samples to be within a certain range, cropping images, etc.

4. **Training**: training the model.

Once the model is trained, the process of using the trained model to make predictions on newly seen data is called *inference*. Generally speaking, inference is comprised of the following phases (we call these phases **the phases of inference**):

1. **Preprocessing**: converting raw data samples into feature vectors. Note that the preprocessing step used during inference is similar to the preprocessing step used in training but usually not exactly the same.

2. **Forward propagation**: conducting a forward pass of the feature vector through the model to output a prediction vector.

3. **Postprocessing**: convert the prediction vector into a more usable format. This could include converting the raw prediction vector into probabilities, or confidence scores, one-hot encoded vectors, etc.

The traditional approach of building machine learning models has some advantages because the data collection phase enables data science teams to have access to all available data. This means that the process of conducting exploratory data analysis, data preparation, and model training are streamlined due to ease of access to the underlying data. However, despite all the benefits of collecting and aggregating data, this approach is only appropriate when the underlying data is not sensitive in nature.

In the case of sensitive data, there are often tight regulations and security measures in place to control access to the data, which makes the process of data collection much more difficult. These challenges in dealing with sensitive data often come up in the healthcare setting where access to the electronic medical records that are used to build models to diagnose diseases [5] are controlled by the Health Insurance Portability and Accountability Act (HIPAA) [6]. Additionally, sensitive data is also prevalent in government, military, and intelligence agencies where machine learning is used for everything from drone warfare [7], to neutralizing cyberattacks [8], to surveying areas using satellite imagery [8], to detecting social unrest [8].

Due to the difficulties of collecting data in a traditional approach, a new field of research

called *privacy-preserving machine learning (PPML)* was developed to address the challenges of preserving the privacy of data during both the training and inference phase [9]. In Al-Rubaie et al.[9], the authors define three different roles *Data Owner (DO)*, *Computational Party (CP)*, and *Results Owner (RO)* that are involved in the development and usage of machine learning models. For our purposes, we also want to define another party called the *Model Owner (MO)*:

1. **Data Owner (DO)**: the party that owns the data
2. **Computational Party (CP)**: the party that performs the computation necessary to either train the model or conduct inference using the model.
3. **Model Owner (MO)**: the party that owns the parameters of the trained model.
4. **Results Owner (RO)**: the party that obtains the predicted labels computed by the MO.

Naturally if all roles are assumed by the same entity, then privacy is preserved; however, the issue of privacy arises when these roles are distributed across two or more entities. For example, if an entity collects its own data, trains its own model, and then uses the model to make predictions for internal purposes, then the entity assumes all of the roles and owns the entire process of development and usage of the model, thus, there are no concerns about other entities exposing the contents of its dataset or the trained model's parameters. However, in situations where an entity needs to rely on another entity's expertise and resources to train a model or use a trained model, then this becomes an issue because either the DO needs to hand the dataset over to the MO to train a model or conduct inference, or the MO needs to hand the model over to the DO so that the DO could use the model to conduct inference on its own data.

At the core of PPML are three questions that need to be addressed (we called these questions **the three fundamental questions of PPML**):

1. **How can the MO's model parameters be hidden from the DO? (*model privacy*).**

2. **How can the DO's data be hidden from the MO? (*data privacy*)**

3. **How can computation be executed efficiently and accurately without compromising model privacy and data privacy?**

In an effort to answer these question, many current PPML approaches require using cryptographic techniques like homomorphic encryption, garbled circuits [10], and differential privacy, [11] or hardware-based approaches like trusted execution environments [12], or novel collaborative machine learning techniques like federated learning and split learning to conduct inference. While most of these approaches address privacy issues that arise during the training phase and forward propagation, we observed that previous PPML approaches often do not provide any kind of strict definition of security for data collection, exploratory data analysis, data preparation, data preprocessing, and postprocessing of the labels.

With that said, the scope of the problem of securing all phases of training and inference is far too broad; thus, we do not plan to address all phases of training and inference. However, because many machine learning methods require the steps used for inference as a subset of the steps used in training (e.g., training a neural network requires a forward pass to compute the predicted labels and then a backward pass to update the parameters of the model), we believe that narrowing the scope of the problem to just securing inference can someday help us to solve the broader problem of conducting training securely.

Broadly speaking, the process of conducting inference can be viewed as a series of three algorithms (we will refer to these three algorithms as an *inference scheme*): a procedure for converting raw data samples into features vectors that are ready for inference (Preprocess), a procedure for deploying a trained model (M), and a procedure for converting the predicted labels of M into a more usable format (Postprocess). By extension, a *privacy-preserving inference scheme (PriPIS)* is simply an inference scheme that emphasizes the privacy of the data and the privacy of the model parameters.

Thus, the goal of this paper is to provide a theoretical construction of a privacy-preserving inference scheme, define the settings and threat models, and informally argue its security against various attacks.

Thesis Contribution. The contributions of this thesis can be summarized in the following ways:

1. *Privacy-Preserving Inference Scheme (PriPIS).* We propose a definition for a Privacy-Preserving Inference Scheme; a theoretical primitive specifically designed for conducting inference that emphasizes protecting the privacy of the DO's data and the MO's trained model parameters. Due to the variety of attacks that take advantage of the feature vectors and prediction vectors, we emphasize the need for securing the preprocessing and postprocessing stages in addition to securing the execution of the model. Thus, we designed a primitive to preserve the privacy of the DO's data during the preprocessing, forward propagation, and the postprocessing stage.

2. *Oracle Construction & User Interaction.* We provide a theoretical construction of a PriPIS and describe how a user would interact with the oracles. For the user interaction with the oracles, we devise a way to allow the user to verify the execution of the oracle using artifacts produced during training. We then informally argue the security of our construction against membership inference and model extraction attacks in a setting we call the *inference server setting* 4.2.1. Through the construction of the oracles, we demonstrate it is possible to decouple the role of the compute party from the roles of data owner, model owner, and results owner.

3. *Setting for PPML.* In Section 4.2, we argue that PPML techniques cannot be secure against attacks such as model extraction and membership inference if deployed into the wrong setting. The reason we defined requirements for the setting is because certain attacks on machine learning models such as model extraction and membership inference only require the adversary to be able to query the model (i.e., oracle access to the model) and to have access to the model's prediction vector. Since there

are many use cases where user requires oracle access and access to prediction vectors, an adversary can simply impersonate a user and conduct an attack; thus, no privacy-preserving machine learning technique is resistant against model extraction and membership inference in these settings because as of right now, it is still an open research question on how to distinguish between an ordinary user and an adversarial user. Thus, we would like to avoid settings where privacy is not guaranteed due to the possibility of model extraction and membership inference. We also emphasize the need for a more holistic view of PPML that encompasses not just the PPML technique itself, but also the setting it is deployed in. Furthermore, we define the requirements for a setting where a privacy-preserving inference scheme can be deployed and then we identified a practical setting that fulfills those requirements.

Outline. We will begin by reviewing some previous works in Chapter 2. From there, we will review some basic preliminary definitions for cryptography and machine learning in Chapter 3. Once the basic preliminaries are defined, we will define the interface of our privacy-preserving inference scheme in Chapter 4. Since there are no formal definitions for a secure framework for conducting inference, we will define our own interface, define the setting, and the parties involved in this setting. Afterwards we will develop our construction and the interactions that need to occur to work with our construction in Chapter 5. Lastly we will discuss directions for future research in Chapter 6.

Chapter 2: Related Work

2.1 Approaches

In this section we will review the related work on PPML. We will organize the relevant works in terms of the general approach used to obtain privacy. Overall, the PPML approaches span many fields; some approaches are based on cryptographic concepts, others originate from machine learning techniques, and some approaches even rely on hardware solutions. Each of these approaches have their strengths and weaknesses, and they address different parts of the model development process and the phases of inference but at the core, they all try to answer the the three fundamental questions of PPML.

2.1.1 Cryptographic Approaches

Popular cryptographic approaches to protecting data privacy during inference include approaches that use homomorphic encryption, garbled circuits, and multi-party computation.

Homomorphic Encryption

A homomorphic encryption scheme enables computations to be performed on encrypted data yielding a ciphertext containing the encrypted result [13]. A homomorphic encryption scheme can be either somewhat homomorphic or fully homomorphic (FHE). Schemes that are considered somewhat homomorphic can only enable either addition or multiplication operations to be performed on encrypted data; whereas schemes that are fully homomorphic can enable both addition *and* multiplication to be performed on encrypted data [13]. The first fully homomorphic encryption scheme was developed by Craig Gentry in 2009 and is used by many PPML approaches [14].

In 2019, Nandakumar et al. [15] sought to evaluate the feasibility of training neural networks on encrypted data. In their paper, Nandakumar et al. demonstrated that it was possible to train a neural network on encrypted data using FHE, achieve convergence, and achieve reasonable accuracy. As revolutionary as Nandakumar was in his approach, his work was limited because it only applied to a simple fully-connected 3-layer neural network. Later on, other works such as CryptoNets [16] and Hesamifard's approach [17] was able to successfully apply FHE to deep convolutional neural networks. However, despite the progress made in applying FHE to machine learning, there were still many problems that make it infeasible to use FHE to train and conduct inference on models, some of these problems include (but at are not limited to) the following:

- **FHE schemes support a limited number of mathematical operations**. Since FHE schemes only support addition, subtraction, multiplication, and division; this is a problem because many machine learning models often use non-linear activation functions (such as ReLU) that cannot be easily calculated using FHE.

- **FHE schemes introduce noise into the ciphertext and therefore requires bootstrapping to be ran periodically**. Although Nandakumar suggested ways to address this performance issue in his paper, the performance issue is magnified in the case of deep learning because deep neural networks contain more parameters, therefore requiring more operations to be performed to train those parameters, which leads to the FHE scheme introducing more noise into the network, and ultimately leads to more bootstrapping and more time spent on inference and training.

- **The message space must be in the integer domain**. Feature values and model parameters must be converted to integers before encryption can take place. This is a problem because machine learning tasks extensively utilize floating point numbers.

- **Ciphertext size**. Depending on the FHE scheme used and the security level of the scheme, the size of the ciphertext is considerably larger than the plaintext. This is significant problem because the most advanced deep learning models can be quite

large (GPT-3 has about 175 billion parameters, assuming each paramter is 4 bytes, that means the model requires about 700GB of storage space [18]).

Garbled Circuits

Another cryptographic approach involves the use of garbled circuits [10]. As described by Bellare et al., a garbling algorithm $\mathsf{Gb}(f) \rightarrow (F, e, d)$ is a randomized algorithm that takes as input a function $f : \{0,1\}^n \rightarrow \{0,1\}^m$ and outputs three functions (F, e, d). The *encoding function* e converts an *initial input* $x \in \{0,1\}^n$ into a *garbled input* $X = e(x)$. The *garbled function* F takes as input a garbled input X and computes a *garbled output* Y; i.e., $Y = F(X)$. The *decoding function* d then converts Y into the *final output* y; i.e., $y = d(Y) = f(x)$.

From the works we have reviewed, many PPML approaches use garbled circuits in addition to homomorphic encryption to perform training and inference. An example of how garbled circuits is used in tandem with homomorphic encryption is how GAZELLE [19] utilized FHE to compute linear operations (such as the matrix multiplication/addition in linear layers) and then used garbled circuits to compute non-linear activation functions (such as ReLU, sigmoid, and Tanh). However, the downside to using garbled circuits is that it is even more computationally expensive than FHE [20] [21]. Thus, approaches like Delphi [20] and Hasmifard's approach [17] have sought to eliminate the need for using garbled circuits by substituting commonly used non-linear activation functions with their polynomial approximations (e.g., quadratic approximation, numerical approximation, Taylor series approximation, Chebyshev polynomials, etc.).

Multi-Party Computation (MPC)

Another cryptographic approach involves the use of MPC protocols like secret sharing. Secret sharing splits a secret into multiple parts and distributes those parts among multiple parties. The shares must be combined in order to reconstruct the secret; meaning the individual shares alone are of no use. In the case of threshold secret sharing, only t number

of shares are required to reconstruct the secret, which makes secret sharing particularly useful for situations where not all parties holding secrets can be expected to participate in the act of secret sharing [22].

MPC protocols have been used to provide security during the inference process in distributed machine learning techniques like federated learning [23]. In federated learning, a central server collects weights and model updates from multiple clients' *local model* and then add those weights and updates to create the *global model*; the process of aggregating those weights and updates uses a secure aggregation protocol. In the secure aggregation protocol described by Bonawitz et al. [23], threshold secret sharing [22] was used to handle aggregation to account for situations where a client loses communication with the central server.

In summary, cryptographic approaches can be used to ensure data privacy and model privacy during preprocessing and forward propagation; however cryptographic approaches cannot guarantee data privacy during postprocessing because as soon as the labels are decrypted, many attack vectors can take advantage of the decrypted labels.

2.1.2 Perturbation

The difference between cryptographic approaches and perturbation approaches, is that cryptographic approaches aim to protect the privacy of the data *and* the security of the data; the ciphertext of the data ensures it is secure during transport as well as computation. Perturbation approaches on the other hand only provide privacy of the dataset and noise is introduced in a much less random manner than cryptographic approaches (e.g., noise introduced by differential privacy is usually gaussian noise [11]). Furthermore, in cryptographic approaches, the noise will eventually be removed from the computation by either running the Recrypt or the Decrypt algorithm [14]; whereas in perturbation approaches, computation is deliberately carried out *with* the noise; therefore, the noise introduced by perturbation approaches is never removed. Perturbation approaches can be grouped into two categories:

the *probability distortion approach* and the *value distribution approach* [24].

Probability Distortion Approach

Probability distortion approaches aims to anonymize the dataset by swapping the original raw data samples with a distorted version of the dataset that has the same frequency count statistics as the original data [25]. Probability distortion approaches are similar to the numerous data imputation and data generation techniques to used prior to training machine learning models. The only difference between data imputation/generation and probability distortion is that data imputation/generation approaches aim to enhance certain properties of the dataset to improve training whereas probability distortion approaches aim to enhance data privacy. Thus, probability distortion approaches can be used in the preprocessing stage of training and inference to enhance data privacy. Unfortunately, the downside to using probability distortion approaches is there are no strong security guarantees, meaning probability distortion approaches might only be able to provide a false sense of security, i.e., *security through obscurity*.

Value Distortion Approach

Value distortion approaches intend to introduce noise into the data to ensure data privacy. Most perturbation approaches use differential privacy to inject noise into features [11]. There has been numerous works using differential privacy to train models and conduct inference for a variety of tasks and data [26] [27] [28]. Al-Rubaie and Chang [9] noted that differential privacy is applied to machine learning in the following ways:

- **Input perturbation**: noise is added directly to the raw data samples. This approach ensures the raw data samples are differentially private and can be used to ensure data privacy during the preprocessing phase.
- **Algorithmic perturbation**: in Abadi et al. [26], differential privacy is applied to the stochastic gradient descent algorithm by adding noise to the gradients to ensure

privacy of the gradients during training. Thus, this approach can be used to ensure model privacy and data privacy during the forward propagation phase of inference.

- **Output perturbation**: adding noise to the output vector [29]. Depending on the use case, this technique could be useful for enhancing privacy during postprocessing.
- **Objective perturbation**: adding noise to the objective function during training [30]. This technique is only applicable during training and is not of interest to us as we are only concerned with techniques used during inference.

In summary, probability distortion approaches aim to enhance data privacy by swapping raw data samples for generated data samples in such a way where the statistical properties of the original dataset is retained. Probability distortion approaches can often be used in conjunction with data imputation/generation approaches during the preprocessing phase of inference and training. Value distortion approaches on the other hand aim to enhance data privacy and model privacy by introducing noise into the data, intermediate outputs, the final output, and objective function using differential privacy. The main downside of differential privacy is the noise that is introduced will also reduce the accuracy of the prediction vector; thus, there is always a tradeoff between accuracy and privacy when using value distortion approaches. However, depending on the setting, perturbation approaches can be used to enhance data privacy and model privacy in all phases of training and inference.

2.1.3 Privacy-Preserving Dimensionality Reduction (PPDR)

PPDR aims to enhance privacy through dimensionality reduction techniques like independent component analysis (ICA) [24], principle component analysis (PCA) [31], etc. In short, dimensionality reduction techniques like ICA and PCA use a *projection matrix* to project a high-dimensional feature vector into a lower-dimensional representation vector. PPDR takes advantage of this process by introducing noise into the projection matrix produced by ICA or PCA. Many of these PPDR approaches also leverage differential privacy to determine the amount of noise introduce into the projection matrix. Since ICA and PCA are

commonly used in the preprocessing phase of both training and inference, PPDR can also be used to protect data privacy during the preprocessing phase.

2.1.4 Hardware-based Approaches

Hardware-based approaches utilize secure hardware solutions called *Trusted Execution Environments (TEE)* also known as *secure enclaves*. TEEs have the following properties that make it suitable for executing computation on sensitive data [12]:

- **Separation Kernel**: the separation kernel divides the system up into multiple partitions and guarantees strong isolation between the partitions.
- **Data (spatial) separation**: data within one partition cannot be read or modified by other partitions.
- **Sanitization (temporal separation)**: shared resources cannot be used to leak information into other partitions.
- **Control of information flow**: communication between partitions cannot occur unless explicitly permitted.
- **Fault isolation**: security breach in one partition cannot spread to other partitions.

Examples of TEEs include Intel SGX, ARM TrustZone, Keystone Enclave [32], and the secure enclave in Apple's M1 System-on-a-Chip. One of the most comprehensive works that applied TEEs to machine learning is Ohrimenko et al. [33]. In their paper, Ohrimenko et al. developed data-oblivious algorithms for running support vector machines (SVM), matrix factorization, neural networks, k-means clustering, and decision trees on Intel SGX processors.

Assuming the TEEs are correctly implemented; using TEEs for machine learning allows for training and inference to be conducted without having to sacrifice fast computational runtime for security (such as in the case of cryptographic approaches) or accuracy for security (such as in the case of perturbation approaches). Furthermore, a hardware-based

approach is incredibly versatile and can be used to secure all parts of the training and inference process. With all of that said, the downside to using a TEE is it requires special hardware and if implemented incorrectly can still result in leakage of information outside the TEE [34].

2.1.5 Distributed Machine Learning Techniques

Another approach to conducting privacy-preserving machine learning is by using distributed machine learning techniques like federated learning [35] and split learning [36]. The main idea with distributed ML approaches is data privacy can be enhanced by bringing the model to the data instead of aggregating data like in the traditional model development process.

In the original federated learning paper, McMahan et al. [35] describes how federated learning is conducted using the following steps (note that we will refer to the central server as the model owner or MO and the clients as data owners or DOs):

1. A central server first decides how it will go about coordinating the training process. According to [37], the central server will decide on the model's architecture, which DOs get to train the model, which DOs' gradients get aggregated into the model, and what metrics and procedures to use to measure the performance of the model.

2. The MO broadcasts an untrained model to various DOs.

3. The DOs train the model locally using their data and send the the gradients back to the MO.

4. The MO would then aggregate the locally-computed gradients using a secure aggregation protocol [23].

5. Once the gradients are aggregated, the Federated Averaging algorithm [35] is used to update the model.

Once the training process is complete, the process of inference is as simple as taking the trained model and doing a forward pass of the data through the trained model, the authors

do not mention any steps to secure the model after it has been trained.

Because the DOs train the model locally, there is no need to transfer data to the MO. Thus, there is no need for the MO to create copies of the data for the sake of data locality, which reduces the amount of resources the MO needs to invest into data storage infrastructure. Additionally, because the secure aggregation protocol preserves the privacy and security of the DOs' data,the sensitive nature of the DOs' data is no longer a limiting factor to the process of developing machine learning models. Regardless of what security measures and regulations are in place, development of machine learning models will not be hindered by access to data and a model can always be trained without compromising the security and privacy of the underlying data.

In addition to preserving the privacy and security of data, FL can also enable the MO to build better models and apply machine learning to more use cases as well. This is often the case where data is more difficult to obtain where no individual DO has enough data to train an effective model; however collectively, there is enough data to train a model. In these situations, FL can enable the model to be trained on data acquired by multiple DOs and the MO will no longer have to settle for training a lower performing model because the MO does not have access to more data due to the DOs' security and privacy concerns.

Unfortunately, despite all of the advantages of FL, FL also has some disadvantages. Because the DOs have access to the weights of the public model, an adversarial DO can conduct model inversion attacks ([38], [39]) that enable it to reconstruct the data of other DOs and the MO by leveraging generative adversarial networks. Furthermore, it has even been proven that a malicious server can easily elude the secure aggregation protocol and recover model updates from the final aggregated value and can pinpoint the source of the recovered data to specific DOs in the federation [40].

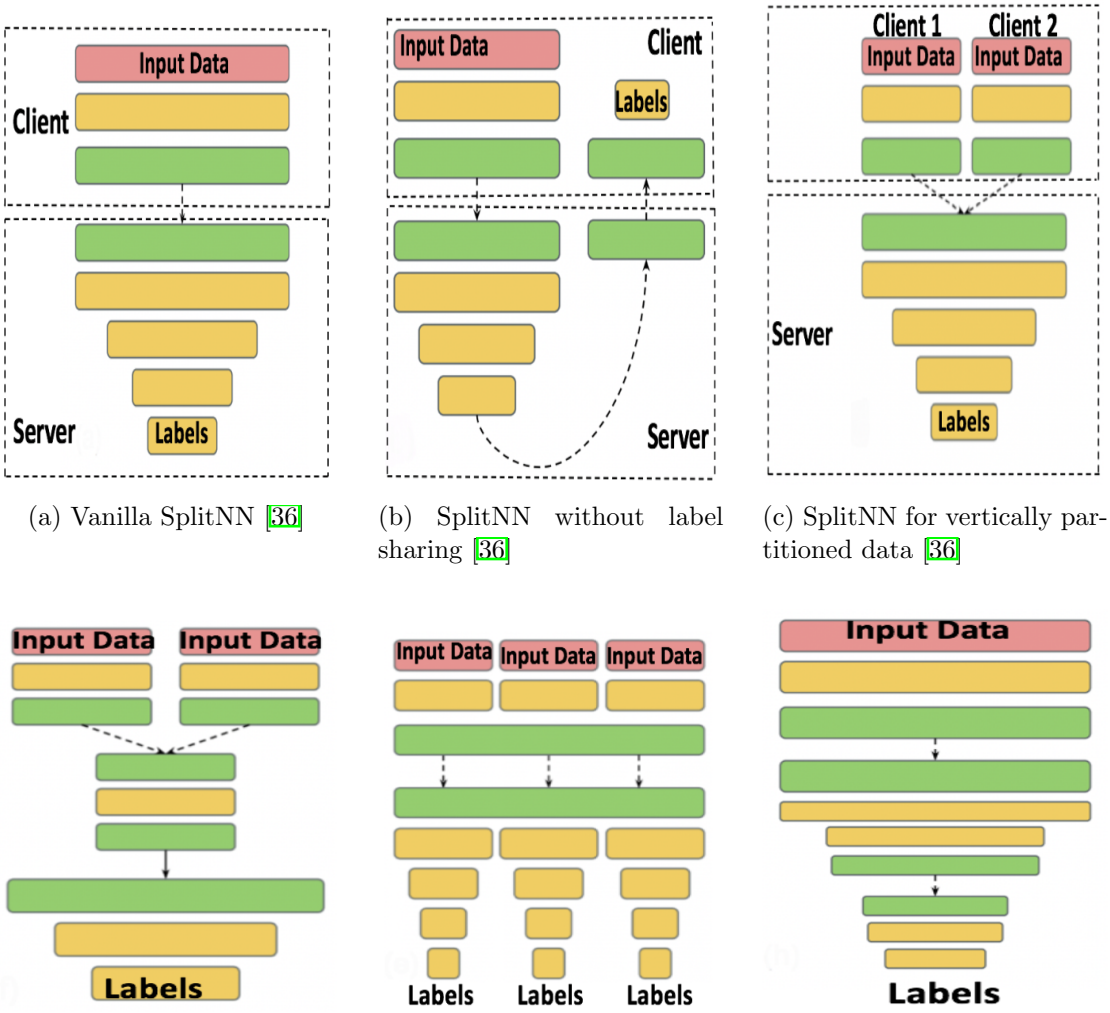
One of the most complex (and arguably creative) approaches to distributed learning would be the many variations of split learning [36] (Figure 2.1). These configurations of SplitNN can be tailored to perform learning and inference on a wide variety of settings depending on the task at hand, the sensitivity of the data, and the amount of compute

available for each party. The main idea with the various configurations of SplitNN can be summarized in the following steps:

1. Have the client (oftentimes the client is a data owner) conduct forward propagation on a partial model up to a specific layer called the "cut layer" (the output at the cut layer is call "smashed data").

2. Perform some sort of processing on the smashed data if necessary (e.g., like concatenating the outputs in Figures 2.1c, 2.1d, 2.1e) and then send that processed smashed data to another party (e.g., a server (like in Figures 2.1a, 2.1b, 2.1c, 2.1e) or another client (like in Figures 2.1d, 2.1f)).

3. Continue with forward propagation using the processed smashed data as input.



(a) Vanilla SplitNN [36]

(b) SplitNN without label sharing [36]

(c) SplitNN for vertically partitioned data [36]

(d) Extended vanilla SplitNN [36]

(e) SplitNN for multi-task output with multiple partitioned input [36]

(f) Multi-hop SplitNN [36]

Figure 2.1: Different configurations for SplitNN

The goal of these various configurations of SplitNN are 1) to enable machine learning to be conducted in a collaborative manner and 2) to eliminate the need for DOs to share its data during this collaborative training process. SplitNN does a great job of eliminating the need to share data, but that alone is not enough to guarantee the privacy of DOs' data. In fact, SplitNN provides only a form of *security through obscurity*, as it has been demonstrated that a malicious server or a malicious client can hijack the learning process

and bring the model to an insecure state where the malicious party can recreate the client's data without having any access to the smashed data [39]. Thus, despite all of the measures proposed by SplitNN to ensure the privacy of the training data, these measures are woefully inadequate as it only provides a false sense of privacy and security to the clients and the server.

In summary, neither SplitNN nor federated learning can provide data privacy and model privacy in any of the phases of training and inference. However, these approaches do point towards possible ways to operate in environments where data access is restricted and are worth noting as sensitive data often reside in environments that limit data transfers (e.g., top secret data used by government, military, and the intelligence community). Since PPML is often meant to be conducted on sensitive data, distributed machine learning techniques could be used in conjunction with other PPML approaches to train models and conduct inference in restrictive environments.

2.2 Attacks

Within the broad landscape of attacks on machine learning systems, Barreno et al. [41] states that the various attacks can be divided into two categories: *exploratory attacks* and *causative attacks*.

2.2.1 Exploratory Attacks

Exploratory attacks aim to compromise the privacy of the training dataset. Because the primary goal of privacy-preserving machine learning is preserving the privacy of the model parameters and the data, preventing exploratory attacks is the main goal. Examples of exploratory attacks include reconstruction attacks [42], model inversion attacks, and membership inference attacks [43].

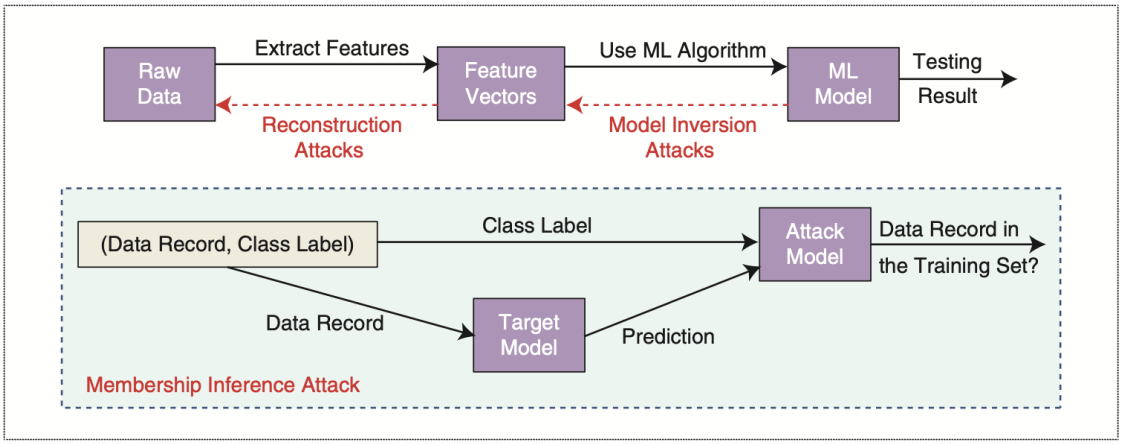


Figure 2.2: Different types of Exploratory Attacks

Reconstruction Attacks. As shown in Figure 2.2 (image retrieved from [9]), reconstruction attacks typically take place during the preprocessing phase where raw data samples are converted to features vectors [9]. An example of a reconstruction attack is the tecnique developed by J. Feng and A. K. Jain where a fingerprint image (the raw data) is reconstructed from its minutae representation (the features vector) as shown in Figure [42]. Fingerprint recognition systems will typically convert a greyscale representation (Figure 2.3a) into a minutae representation (Figure 2.3d) for ease of storage.

(a) Greyscale image (b) phase image (c) skeleton image (d) minutae image

Figure 2.3: Fingerprint representations

(a) Reconstructed from phase image (b) Reconstructed from skeleton image

Figure 2.4: Fingerprint reconstruction [42]

For quite some time it was falsely believed that the minutiae representation does not contain sufficient information to reconstruct the original grayscale fingerprint image; but as J. Feng and A. K. Jain demonstrated in their paper, by first using the minutae representation to reconstruct either the skeleton representation (Figure 2.3c) or phase representation (Figure

2.3b), they can generate images that are almost identical to the original greyscale representation as shown in Figure 2.4. Thus, when looking at the wider context of protecting data privacy during the inference process, the existence of reconstruction attacks suggests that preprocessing alone is not enough to ensure privacy of the data and there must be security guarantees built into the preprocessing stage.

Model Inversion Attack. Another type of exploratory attack is a model inversion attack [44]. The aim of a model inversion attack is to recreate the feature vectors used to create an ML model by utilizing the responses received from that ML model. Despite this goal, model inversion attacks cannot actually recreate a sample from the training dataset.

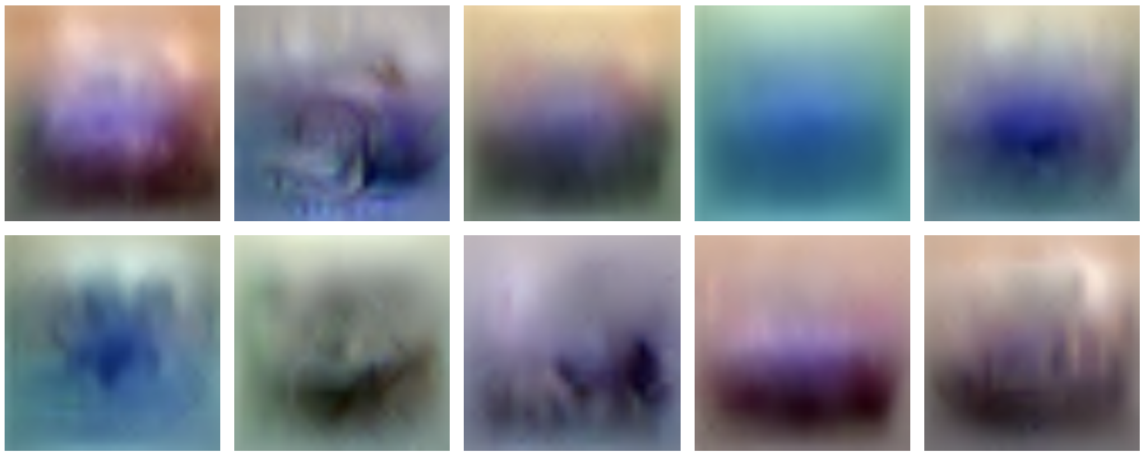


Figure 2.5: Images produced by model inversion. Model trained on the CIFAR-10 dataset. Top: airplane, automobile, bird, cat, deer. Bottom: dog, frog, horse, ship, truck. [43]

Figure 2.6: Images produced by model inversion based on rounding output vector to the nearest r [44]

As evidenced by the images in Figure 2.5, the images do not hold any resemblance to the objects in the class. However, even though model inversion may not be great at recreating the training images, model inversion could be used to extract statistical information about the training dataset to make other attacks (such as membership inference attacks) more effective. Thus, there are a few valuable insights that model inversion attacks teach us about preserving the privacy of the dataset: 1) the output vector leaks important information that could be used to compromise the privacy of the dataset, 2) the amount of information leaked largely depends on the format of the output vector (as evidenced by Figure 2.6) which implies there must be privacy-preserving mechanisms introduced into the postprocessing stage of inference, and 3) no party should have white-box access to the model when in execution.

Membership Inference Attack. Another type of exploratory attack is a membership inference attack. As shown in Figure 2.2, a membership inference attack is an exploratory attack that aims to determine if a given data point was present in the training dataset used to build a model [43]. One particularly famous membership inference attack is the *shadow models method* developed by Shokri et al. [45]. In their paper, Shokri et al. demonstrated that restricting the prediction vector to a single label, which is the absolute minimum a model must output to remain useful, is not enough to fully prevent membership inference.

Thus, the valuable insight here is that it is not enough to just secure the execution of preprocessing, forward propagation, and postprocessing; preserving privacy of the data also requires carefully crafting the setting in such a way that does not allow adversaries access to the prediction vectors.

Model Extraction Attacks. All of the previous exploratory attacks mentioned so far have been focused on compromising data privacy; however, there is a class of exploratory attack that is aimed at compromising model privacy called *model extraction attacks* [46]. In a model extraction attack, the aim is for the adversary to try to create a model that is similar (if not identical) to the target model. The adversary *only* has black-box access to the target model, it has no prior knowledge of the model parameters nor the training dataset. Thus, the valuable insight here is that when crafting the setting, we must be cognizant to ensure the role of the model owner and the results owner are assumed by the same party.

2.2.2 Causitive Attacks

Causative attacks are attacks that alter the training process through influence over the training data. The primary aim of causative attacks are to bring the model to a state of insecurity by tricking the model into misclassifying the points in a dataset (or just subset of the points in a dataset like a specific class(es) of points) thereby reducing the accuracy of the model. Examples of causative attacks include adversarial examples [47] where perturbations are introduced into an image to trick a model into misclassifying the image with high confidence.

In general, the primary goal of PPML is to defend against exploratory attacks. However, even though the objective of causitive attacks are not specifically aimed at revealing privacy, there are a class of causitive attacks that are aimed at making models more vulnerable to exploratory attacks like the Truth Serum Attack [48]. Thus, defending against causitive attacks is also important to the goal of preserving data privacy.

Chapter 3: Definitions

We will start by discussing the necessary cryptographic preliminaries, then we will discuss the preliminaries relevant to machine learning.

3.1 Cryptographic Preliminaries

Since our constructions are based on basic cryptographic primitives, we will start by reviewing the security definitions for these primitives.

Definition 1 (Private-Key Encryption Scheme)**.** *A private-key encryption scheme is composed of three algorithms [13]:*

- $\mathsf{KeyGen}(1^n) \to k$: *the key-generation algorithm which takes as input a security parameter* 1^n *and outputs a key* k*.*
- $\mathsf{Enc}_k(m) \to c$: *the encryption algorithm takes as input a key* k *and a plaintext message* m *and outputs a ciphertext* c*.*
- $\mathsf{Dec}_k(c) \to m$: *the decryption algorithm takes as input a key* k *and a ciphertext* c *and outputs the plaintext message* m*.*

Definition 2 (The CCA Indistinguishability Experiment (Private-Key Setting))**.** *Let* $\Pi = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Dec})$ *be a private-key encryption scheme, let* $\mathcal{A}$ *be a polynomial-time adversary, and let* n *be the security parameter; the CCA indistinguishability experiment* $\mathsf{Priv}^{\mathsf{CCA}}_{\mathcal{A},\Pi}$ *is defined like so [13]:*

$\mathsf{Priv}^{\mathsf{CCA}}_{\mathcal{A},\Pi}(n)$:

1. *Generate key* $k \leftarrow \mathsf{KeyGen}(1^n)$.

2. *Adversary $\mathcal{A}$ is given input 1^n and access to encryption oracle $\mathsf{Enc}_k(\cdot)$ and decryption oracle $\mathsf{Dec}_k(\cdot)$.*

3. *$\mathcal{A}$ outputs a pair of messages $m_0, m_1 \in \mathcal{M}$ where m_0, m_1 is of the same length, $\mathcal{M}$ is the message space associated with k, and $m_0 \neq m_1$.*

4. *A uniform bit $b \in \{0,1\}$ is chosen. The challenge ciphertext $c \leftarrow \mathsf{Enc}_k(m_b)$ is computed and given to $\mathcal{A}$.*

5. *$\mathcal{A}$ continues to have access to $\mathsf{Enc}_k(\cdot)$ and $\mathsf{Dec}_k(\cdot)$ however $\mathcal{A}$ cannot query $\mathsf{Dec}_k(\cdot)$ on the challenge ciphertext itself.*

6. *$\mathcal{A}$ outputs a bit b'.*

7. *The output of the experiment is 1 if $b = b'$ (i.e., the experiment succeeds and $\mathcal{A}$ guessed correctly) and 0 otherwise.*

Definition 3 (CCA-Security (Private-Key Setting))**.** *A private-key encryption scheme Π is considered indistinguishable under chosen-ciphertext attack (i.e. CCA-secure) if for all polynomial time adveraries $\mathcal{A}$ there is a negligible function negl such that [13]:*

$$\Pr\left[\mathsf{Priv}^{\mathsf{CCA}}_{\mathcal{A},\Pi}(n) = 1\right] \leq \frac{1}{2} + \mathsf{negl}(n)$$

Definition 4 (Message Authentication Code (MAC))**.** *A message authentication code (MAC) is composed of three algorithms [13]:*

- $\mathsf{KeyGen}(1^n) \rightarrow k$: *the key-generation algorithm which takes as input a security parameter 1^n and outputs a key k.*
- $\mathsf{Mac}_k(m) \rightarrow t$: *the tag-generation algorithm takes as input a key k and a message m and outputs a tag t.*
- $\mathsf{Vrf}_k(t, m) \rightarrow b$: *the verification algorithm takes as input a key k, a tag t, and a message m and outputs a bit $b \in \{0,1\}$. It holds that $\mathsf{Vrf}_k(\mathsf{Mac}_k(m), m) = 1$.*

Definition 5 (Diffie-Hellman Key Exchange Protocol)**.** *The Diffie-Hellman key exchange protocol is a way for two parties (we will refer to these parties as Alice and Bob) to agree on a shared secret key. The protocol proceeds as follows [13]:*

KeyExchange(1^n):

1. *Alice and Bob are both given security parameter* 1^n.
2. *Alice runs a group generation algorithm* $\mathcal{G}(1^n) \to (\mathbb{G}, q, g)$ *to generate group* $\mathbb{G}$ *with order* q *and generator* $g \in \mathbb{G}$.
3. *Alice chooses a uniform* $x \in \mathbb{Z}_q$*, computes* $h_A := g^x$*, and sends* $(\mathbb{G}, q, g, h_A)$ *to Bob.*
4. *Bob chooses a uniform* $y \in \mathbb{Z}_q$*, computes* $h_B := g^y$*, and sends* h_B *to Alice.*
5. *Bob outputs the key* $k_B := h_A^y$*, Alice outputs the key* $k_A := h_B^x$.

The key exchange protocol works because

$$k_A = h_B^x = (g^y)^x = g^{yx}$$

$$k_B = h_A^y = (g^x)^y = g^{xy}$$

3.2 Machine Learning Preliminaries

In this section, we will begin by defining the preliminaries related to machine learning concepts, then we will define our construction, and lastly we will define what it means for our construction to be secure in the presence of the attacks defined in section 2.2 2. We will first define the basic machine learning preliminaries and then we will provide more formal definitions for the four different roles we define in the introduction (link to introduction).

For our purposes, we restate the definition of a dataset provided by Yeom et al. [43] in Definition 6 as the *universal dataset*. We also borrowed the definition of a *model* and a *training dataset* from Yeom et al. [43] and made with a few small changes to the notation.

We will later on use the universal dataset to define other datasets. As an additive, we also define additional concepts related to datasets such as a *true label*.

Definition 6 (Universal Dataset)**.** *Let* $\mathbf{X}$ *be the set of all possible raw data samples and let* $\mathbf{Y}$ *be the set of all possible labels. We will refer to* $\mathbf{X} \times \mathbf{Y}$ *as the "universal dataset". A "data point" in a dataset is a tuple* $(x, y) \in \mathbf{X} \times \mathbf{Y}$ *where* $x \in \mathbf{X}$ *is a raw data sample and* $y \in \mathbf{Y}$ *is a label given to the raw data sample (sometimes, we will also refer to* y *as the "true label").*

Definition 7 (Machine Learning Model)**.** *We will define a machine learning model as a function that takes as input a set of parameters* P *and a feature vector* x' *and outputs a prediction vector* y_p*.*

$$M_P(x') \rightarrow y_p$$

If P *is randomly initialized, then we consider the model to be "untrained". If* P *has been tuned by some learning process, then we consider the model to be "trained". Sometimes we will use* M_P *to make predictions on a set of feature vectors, we denote this as* $M_P(X') \rightarrow Y_p$ *where* Y_p *is the set of prediction vectors.*

Definition 8 (Training dataset)**.** *The training dataset* (X_{train}, Y_{train}) *has the following characteristics:*

- $(X_{train}, Y_{train}) \subset \mathbf{X} \times \mathbf{Y}$
- *The data points in the training set is sampled i.i.d. from* $\mathbf{X} \times \mathbf{Y}$*. Thus, let* $\mathcal{D}_{train}$ *be the distribution of the training dataset and let* $\mathcal{D}$ *be the distribution of* $\mathbf{X} \times \mathbf{Y}$*, the distribution of data points in the training set must be similar to the distribution of* $\mathbf{X} \times \mathbf{Y}$*; thus,* $\mathcal{D}_{train} \sim \mathcal{D}$*.*
- *The dataset* (X_{train}, Y_{train}) *is be used to train model* M_P

The definition of an inference dataset, a preprocessing function, and a postprocessing function are definitions we defined on our own.

Definition 9 (Inference Dataset)**.** *The inference dataset* (X_{infer}, Y_{infer}) *has the following characteristics:*

- $(X_{infer}, Y_{infer}) \subset \mathbf{X} \times \mathbf{Y}$
- $(X_{infer}, Y_{infer}) \cap (X_{train}, Y_{train}) = \emptyset$
- *The data points in the training set is sampled i.i.d. from* $\mathbf{X} \times \mathbf{Y}$*. Thus, let* S_{infer} *be the distribution of the inference dataset,* $S_{train} \sim \mathcal{D}$*.*

Definition 10 (Preprocessing Function)**.** *A preprocessing function*

$$\mathsf{Preprocess}(x) \to x'$$

converts a raw data sample x *to a feature vector* x'*. Sometimes we will use a preprocessing function to convert an entire set of raw data samples into a set of feature vectors in which case we will denote this as* $\mathsf{Preprocess} : X \to X'$ *where* X' *is the set of feature vectors. We will sometimes refer to* (X', Y) *as a "preprocessed dataset".*

Definition 11 (Postprocessing Function)**.** *A postprocessing function*

$$\mathsf{Postprocess}(y_p) \to y'_p$$

converts a prediction vector y_p *into the final output vector* y'_p*. Sometimes we will use a postprocessing function to convert an entire set of prediction vectors into a set of feature vectors in which case we will denote this as* $\mathsf{Postprocess} : Y_p \to Y'_p$ *where* Y'_p *is the set of output vectors.*

We will now use Definitions 12-15 to provide more formal definition for the data owner, model owner, computation party, and results owner roles.

Definition 12 (Data Owner (DO))**.** *A data owner is a party that owns a partition of* $\mathbf{X} \times \mathbf{Y}$*, i.e., the DO owns dataset* (X_{DO}, Y_{DO}) *where* $X_{DO} \subset \mathbf{X}$ *and* $Y_{DO} \subset \mathbf{Y}$*.*

Definition 13 (Model Owner (MO))**.** *The model owner is the party that owns the parameters of trained model M_P after it has been train and retains ownership of the parameters while it is used for inference.*

Definition 14 (Computing Party (CP))**.** *The computing party is the party that carries out training and/or inference of the MO's model.*

Definition 15 (Results Owner (RO))**.** *The party that obtains the predicted labels Y_p and/or output labels Y'_p computed by trained model M_P.*

Chapter 4: Defining Privacy Preserving Inference

We will now define the core components of a privacy-preserving inference scheme (PPIS).

4.1 Definition of Privacy Preserving Inference

By researching the broad landscape of attacks covered in section 2.2, we were able to draw some important insights on the design of a privacy-preserving inference scheme.

Recall that reconstruction attacks use the feature vectors to recreate the raw data sample and there by poses a threat to the preprocessing stage of inference. The existence of reconstruction attacks teaches us that preprocessing alone is not a substitute for secure encryption. Therefore, we require that

1. A privacy-preserving inference scheme must ensure the privacy of the raw data samples and feature vectors during the preprocessing stage.

Furthermore, in model inversion attacks, the adversary will try to recreate a feature vector. Recall that model inversion attacks are possible because 1) the adversary has white-box access to the model, and 2) the output vector leaks information that could be used to compromise the privacy of the dataset. Thus, it is important that a privacy-preserving inference scheme ensures that

2. No party should have white-box access to the model when in execution.

3. The postprocessing stage of inference must be secure in order to avoid leaking information through the prediction vector.

Recall from the Introduction, inference is the process of taking a newly seen feature vector that the model was not originally trained on, and then conducting a forward pass

of that feature vector through the model to get a prediction vector. We will give the first definition of a privacy-preserving inference scheme (PriPIS) which we will denote as $\Pi_{ML} = (\mathsf{Setup}, \mathcal{O}_{\mathsf{Preprocess}}, \mathcal{O}_{M_P}, \mathcal{O}_{\mathsf{Postprocess}})$. The goal of a PriPIS is to conduct inference meanwhile also protecting the privacy of the MO's model parameters (e.g., the weights of a neural network) and the privacy of a DO's dataset from all other parties.

Definition 16 (Privacy-Preserving Inference Scheme)**.** *A privacy-preserving inference scheme is comprise of the following four algorithms:*

- $\mathsf{Setup}(M_P, \mathsf{Preprocess}, \mathsf{Postprocess}) \rightarrow (\mathcal{O}_{\mathsf{Preprocess}}, \mathcal{O}_{M_P}, \mathcal{O}_{\mathsf{Postprocess}})$: *The* Setup *algorithm takes as input a trained model M_P, preprocessing function* Preprocess*, and postprocessing function* Postprocess*, and outputs three oracles $\mathcal{O}_{\mathsf{Preprocess}}$, $\mathcal{O}_{M_P}$, and $\mathcal{O}_{\mathsf{Postprocess}}$. The* Setup *algorithm is responsible for creating and setting up the three oracles.*

- $\mathcal{O}_{\mathsf{Preprocess}}(x) \rightarrow (x', \sigma_{\mathsf{pre}})$: *The $\mathcal{O}_{\mathsf{Preprocess}}$ oracle takes as input a raw data sample x and outputs a feature vector x' and a signature σ_{pre}. Note that x' may or may not be encrypted depending on the setting. The $\mathcal{O}_{\mathsf{Preprocess}}$ oracle is responsible for ensuring the security of the execution of* Preprocess*, the privacy of the raw data sample and the feature vector, and providing a way to remotely attest* Preprocess *executed faithfully (this is what the signature σ_{pre} is used for).*

- $\mathcal{O}_{M_P}(x') \rightarrow (y, \sigma_{M_P})$: *The $\mathcal{O}_{M_P}$ oracle takes as input a feature vector x' and outputs a prediction vector y and a signature σ_{M_P}. Note that y may or may not be encrypted depending on the setting. The $\mathcal{O}_{M_P}$ oracle is responsible for the security of the execution of model M_P, the privacy of the feature vector x', the model parameters P, and the raw prediction vector y, and providing a way to remotely attest M_P executed faithfully (this is what the signature σ_{M_P} is used for).*

- $\mathcal{O}_{\mathsf{Postprocess}}(y) \rightarrow (y', \sigma_{\mathsf{post}})$: *The $\mathcal{O}_{\mathsf{Postrocess}}$ oracle takes as input a raw prediction vector y and outputs an "output vector" y' and a signature σ_{post}. Note that y' may or*

may not be encrypted depending on the setting. The $\mathcal{O}_{\mathsf{Postprocess}}$ oracle is responsible for ensuring the security of the execution of Postprocess*, the privacy of the raw prediction vector and the output vector, and providing a way to remotely attest* Postprocess *executed faithfully (this is what the signature σ_{post} is used for).*

4.1.1 Security Properties

We require the following properties for the three oracles:

- **Correctness:** the output of the oracle (whether sent in the clear or after decryption) must match the output of the algorithm it is running regardless if the algorithm is ran inside or outside the oracle.
- **Isolation:** the entire execution of the algorithm takes place inside the oracle, no party should be able see the intermediate outputs of any steps in the algorithm when the algorithm is running inside the oracle. No outside party can view any information inside the oracle.
- **Indistinguishability:** if the oracle were to require encryption of its input/output, then the encryption scheme used to send information to/from the oracle must be secure under chosen ciphertext attack (CCA-secure).
- **Verifiability:** the DO(s) must be able to verify the oracle executed the intended algorithm faithfully.

4.2 Security Model

The design of our setting is based on the lessons learned from our review of attacks in Section 2.2. Recall the main lesson learned from membership inference attacks is that restricting the prediction vector to a single label (the absolute minimum a model must output to remain useful) is not enough to prevent membership inference attacks; thus, we can imply there is no privacy-preserving inference scheme that is truly secure against membership inference

attacks if it is deployed in the wrong setting. Furthermore, since model extraction attacks do not require any prior knowledge of a model's parameters or training data, the key to preventing model extraction attacks lies in ensuring the roles of model owner and results owner are assumed by the same party. Therefore, we require that a setting must meet the following criteria in order to deploy a privacy-preserving inference scheme:

PPML Setting Requirements

1. The model must be trained using only the data from a single DO.
2. During inference, only the DO that owns the training dataset *and* the inference dataset can see the prediction vectors for the inference dataset.

The reason for these requirements is to develop a setting an adversarial DO or MO cannot benefit from conducting a membership inference attack or a model extraction attack. Simply speaking, if a party were to collect and train its own model, then there would be no reason to conduct a membership inference attack because it already knows the what samples are in its training dataset. Furthermore, there would be no reason for a party to conduct a model extraction attack because it already has the model.

4.2.1 Inference Server Setting

Thus far, the one setting we are particularly interested in that fulfilled the above criteria is the *inference server setting*. In the inference server setting, there are two parties: a user and a server; the user needs additional computational power to conduct inference on a larger scale but would like to protect the privacy of its training dataset and model parameters. The server will act as the computation party that provides the extra compute power.

The user has collected and cleaned its own training dataset and has assumed the role of data owner. In an effort to protect the privacy of its training dataset (or in an effort to obey restrictions on moving its data), the user utilizes a local compute instance to train a model on the training dataset it created beforehand, thereby also assuming the role of the model owner. Because the user is the sole owner of the training dataset and it has trained

a model locally using only its data, the inference server setting meets the first requirement in the PPML Setting Requirements. Additionally, the user wants to collect the prediction vectors for a set of inference data for its own purposes thereby assuming the role of results owner. Since the user is also the sole owner of the inference dataset in addition to assuming the role of results owner, then the second requirement in the PPML Setting Requirements is met. Thus, the inference server setting fulfils the criteria for deploying a privacy-preserving inference scheme.

This setting is applicable to research labs that train models that operate on top secret government or health data where the data cannot be moved off-premise due to access restrictions. In the government, defense, and intelligence communities, it is not uncommon for data to labeled as *sensitive compartmentalized information* and can only be accessed inside a Sensitive Compartmented Information Facility (SCIF). In the healthcare setting, there are often restrictions on accessing and moving patients' electronic medical records off-premise due to HIPAA. Thus, development teams in these settings must train their models in their local compute environments however their models are often deployed in situations where more compute power is needed to conduct inference on a large scale. In these situations, they need to rely on cloud service providers for extra compute power but would still require their models and datasets are kept private to all external parties including the cloud service provider. Therefore, the following security requirements must be enforced:

Inference Server Setting Security Requirements

- The server (the compute party) must not be able to see the data samples belonging to the user's training dataset.
- The server must not be able to see the user's model parameters.
- The server must not be able to see the prediction vectors computed by running the user's model on the user's inference dataset.

Threat model. In the inference server setting, the adversary has infiltrated the compute server and can see any data stored in the compute server's memory but not the data stored

in the memory address space allocated to the oracle; by extension, this implies the adversary cannot see the execution of the program inside the oracle. Additionally, the adversary has access to the transcript of the communication between the server and the user and can see the messages sent between the compute server and the user as well as the messages sent between the oracle and the user (note that the messages might be encrypted). Finally, since the adversary is on the server, it does not know the user's data or model parameters.

Chapter 5: Construction

We will start off by defining three subroutines that will be used to facilitate the user's interactions with the oracles. From there, we will use the interactions to construct the oracles and construct the algorithms ran by the user to interact with the oracles.

Random Sampling. The first algorithm, RandSample, will be used for randomly sampling from a dataset. RandSample takes as input two sequences of items D_1 and D_2 and an integer m which denotes the number of items the user wants to sample from D_1 and D_2 (note that D_1 and D_2 must have the same number of items and the items can be anything, in our case, items will be either raw data samples, feature vectors, prediction vectors, or output vectors). RandSample will randomly sample the same number of items from D_1 and D_2 to create the sequences d_1 and d_2 where $|d_1| = |d_2|$.

Algorithm 1 Random Sampling

Function RandSample(D_1, D_2, m):
Require: $|D_1| = |D_2|$
Require: $1 \leq m < |D_1|$
  Let $n = |D_1|$
  Let $I = \{1, 2, ..., n\}$
  Let $d_1 = ()$ be an empty sequence
  Let $d_2 = ()$ be an empty sequence
  for m times **do**
    Randomly sample $i \in I$
    Append $D_1[i]$ to d_1
    Append $D_2[i]$ to d_2
    Remove i from I
  end for
  return d_1, d_2

Mixing Datasets. The second algorithm Mix will be used to combine two datasets together in a random manner. Mix takes as input a sequence of items obtained during training (D_{train}) and a sequence of newly seen items to be used for inference (D_{infer}) (note that just like in RandSample, the items can be raw data samples, feature vectors, prediction vectors, or output vectors). Mix then contatenates D_{train} and D_{infer} together to create another sequence of items D. Mix will then shuffle the order of the items in D, store the indices of the training items in I, and then return the shuffled sequence of items D and the set of indices I.

Algorithm 2 Mix Datasets

Function Mix (D_{train}, D_{infer}):
  Let $I = ()$
  Let $D = D_{train}||D_{infer}$
  Shuffle D
  Let $i = 1$
  while $i < |D|$ **do**
    Let $d = D[i]$
    if $d \in D_{train}$ **then**
      Append i to I
    end if
    $i := i + 1$
  end while
  return D, I

Separating the datasets. The third algorithm, Separate will be used to separate the training items from the inference items from a sequence of items. Separate takes as input a sequence of items D and a list of indices I where the indices indicate the location of training items in D. Separate takes uses the list of indices to separate the training items from the inference items.

Algorithm 3 Separate Datasets

Function Separate(D, I):
  Let $D_{train} = ()$
  Let $D_{infer} = ()$
  Let $i = 1$
  while $i < |D|$ **do**
    Let $d = D[i]$
    if $i \in I$ **then**
      Append d to D_{train}
    else
      Append d to D_{infer}
    end if
    $i := i + 1$
  end while
  return D_{train}, D_{infer}

5.1 Interactions Between Oracles & Users

The interaction between the user and each of the three oracles all follow a similar workflow:

1. **User-side preparation**: Assuming the user has kept the artifacts computed when it ran Preprocess, M_P, and Postprocess during training; the user will randomly sample a few of the artifacts computed during training, mix those artifacts with the artifacts computed during inference, and send it to the oracle.

2. **Oracle execution**: The oracle will run the artifacts through the algorithm installed inside it and then send the outputs along with a tag over to the user for verification.

3. **User verification**: The user will categorize the outputs sent by oracle as either *training outputs* (outputs derived from the oracle applying its algorithm on training

artifacts) or *inference outputs* (outputs derived from the oracle applying its algorithm on inference artifacts). Theoretically, if the oracles executed properly, the training outputs the user got from the oracle should be exactly the same as the training outputs the user saw during training and the tags produced by the oracle can be verified by using the input, the algorithm, and its respective training output.

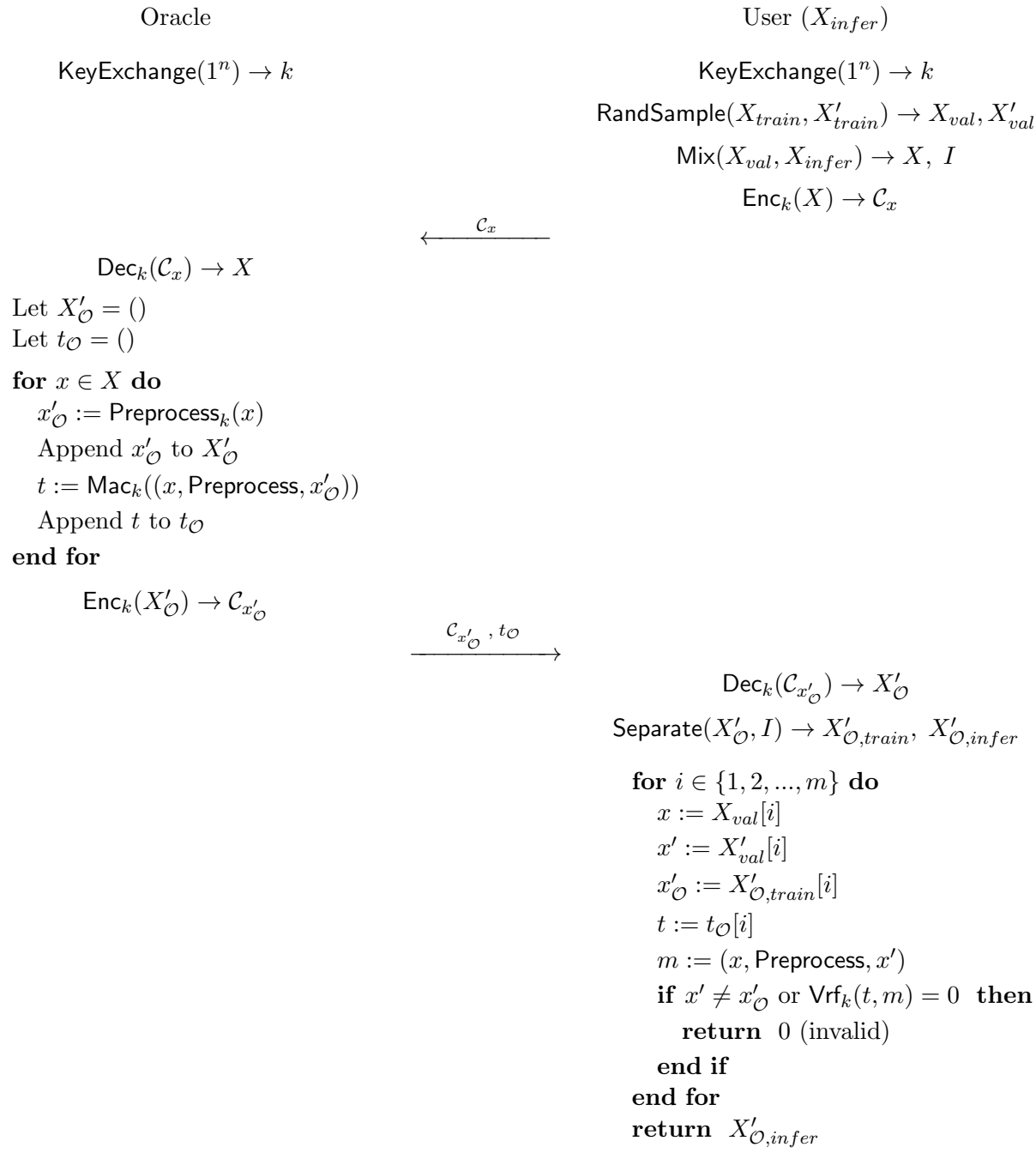


Figure 5.1: Preprocessor oracle interaction with oracle and user

In Figure 5.1, the prover is the server and the verifier is the user. The preprocessing oracle $\mathcal{O}_{\mathsf{pre}}$ is running on the server's compute instance and the oracle needs to be able to prove to the user it ran the Preprocess algorithm correctly. The user needs to verify that

the server's outputs are correct (thereby also verifying the Preprocess algorithm was ran correctly).

The interaction starts off with the user conducting key exchange with the oracle; this ensures the compute party (in real life this would be the cloud service provider) does not have access to the oracle and ensures the compute party cannot steal information from the oracle. The user then uses the RandSample 1 algorithm to randomly sample raw data samples from the user's training dataset (X_{train}) to produce X_{val}. The RandSample algorithm also randomly samples the corresponding features vectors computed when the user ran the Preprocess algorithm during training (X'_{train}) to produce X'_{val}. The randomly sampled raw data samples are mixed into the inference dataset to create dataset X and then all the raw data samples are encrypted and sent over to the oracle. In this phase, the oracle learns nothing about the user's data samples because the encryption scheme that is used to encrypt the raw data samples is indistinguishable (CCA-secure).

In the second part of the interaction, the oracle begins by first decrypting the ciphertext to get the raw data samples. Due to the oracle's isolation property, the server will not learn any information regarding the user's data through leakage. The oracle then runs the Preprocess algorithm and converts the raw data samples into feature vectors that are ready for inference. Once the feature vectors are computed, the oracle encrypts the features vectors, uses Mac to generate tags, and then sends the ciphertext and tags back to the server whom then sends it back to the user.

The third part of the interaction requires the to user to sort the feature vectors into $X'_{\mathcal{O},train}$ and $X'_{\mathcal{O},infer}$ where $X'_{\mathcal{O},train}$ is the set of feature vectors computed by the oracle running Preprocess on raw data samples from the training dataset and $X'_{\mathcal{O},train}$ is the set of feature vectors computed on the raw data samples in the inference dataset. The user then compares the feature vectors in $X'_{\mathcal{O},train}$ to the feature vectors computed during training (X'_{train}). If any of the feature vectors from $X'_{\mathcal{O},train}$ do not match the corresponding feature vector in X'_{train} or if Vrf reveals the tag to be invalid, the proof is invalid; otherwise, if all

the feature vectors match and all the tags are valid, the proof is valid.

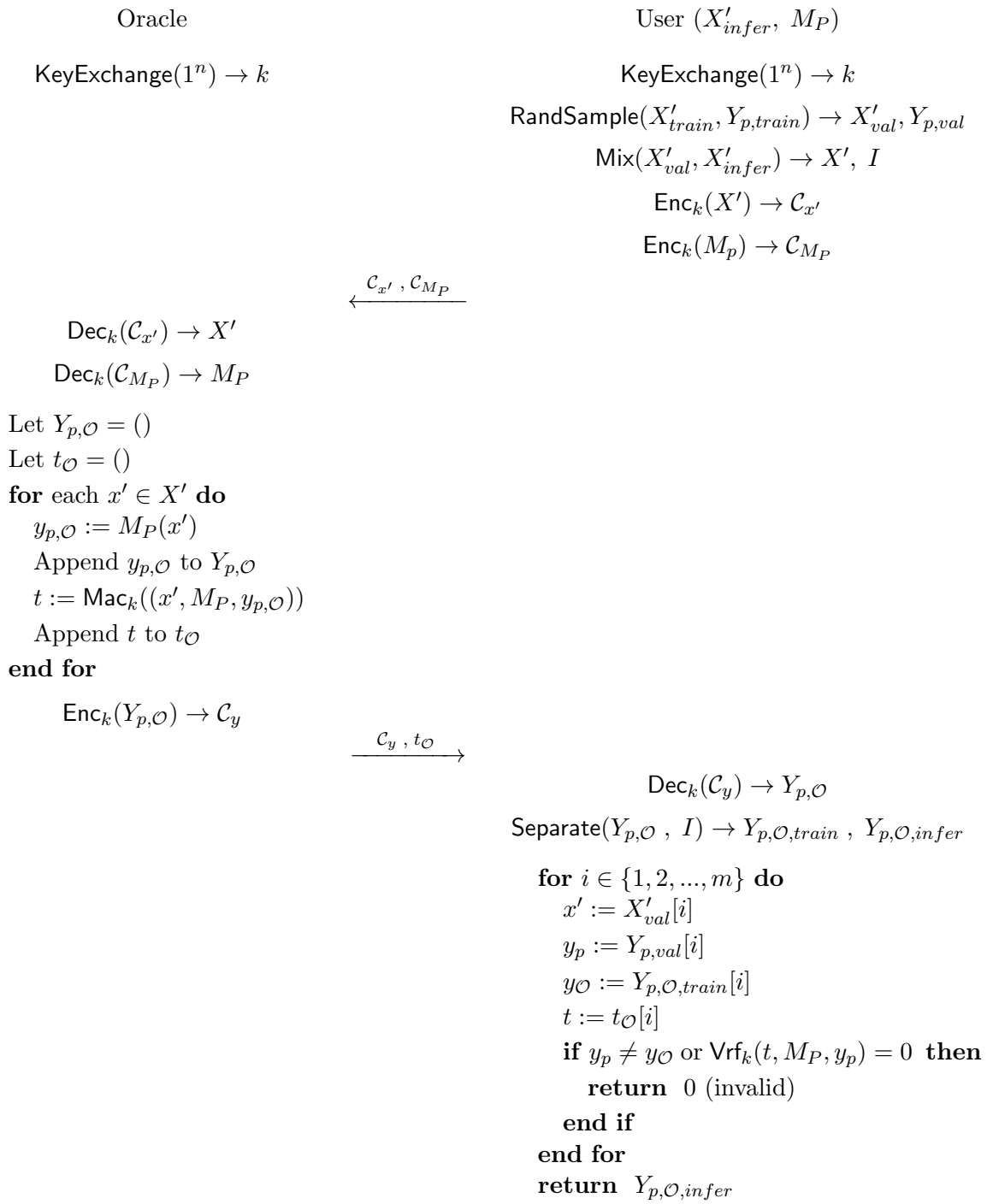


Figure 5.2: Model oracle interaction with oracle and user

The interaction between the user and the model oracle is very similar to the interaction between the user and the preprocessing oracle. The main difference between this interaction and the interaction with the preprocessing oracle is that the user will send the model parameters to the oracle in addition to feature vectors.

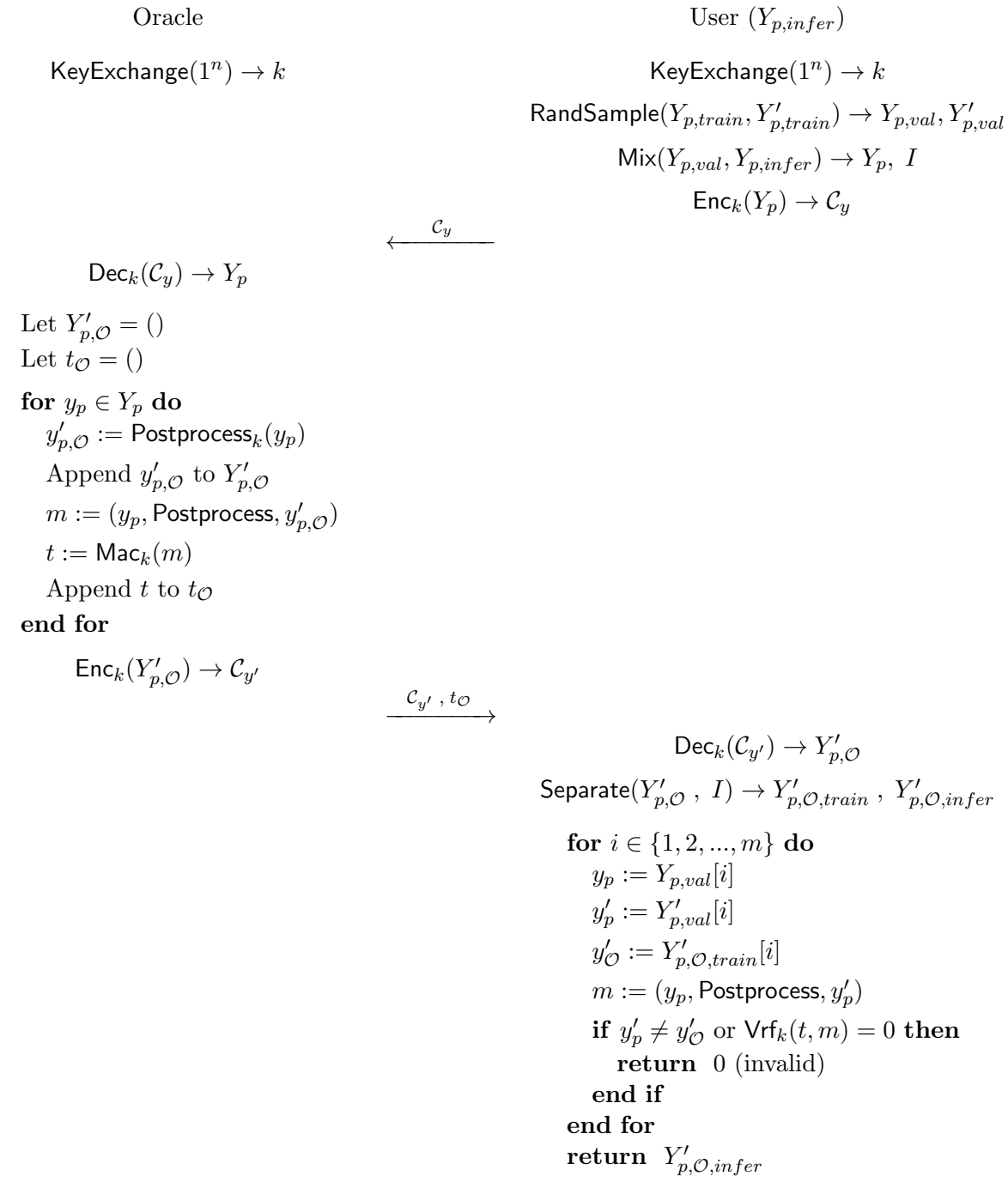


Figure 5.3: Postprocessor oracle interaction with oracle and user

The interaction between the user and the postprocessing oracle works exactly the same

as the preprocessing oracle; just replace the Preprocess algorithm with the Postprocess algorithm, the user's input is the prediction vectors, and the server's response is the output vector.

5.2 Construction of a Privacy-Preserving Inference Scheme

We will now give an example instantiation of our privacy-preserving inference scheme in the inference server setting. Recall an inference scheme is composed of four algorithms: $\Pi_{ML} = (\mathsf{Setup}, \mathcal{O}_{\mathsf{Preprocess}}, \mathcal{O}_{M_P}, \mathcal{O}_{\mathsf{Postprocess}})$. Because we want the interface of a privacy-preserving inference scheme to be as applicable to as many settings as possible, we do not want to consider the algorithms ran on the user side as part of a privacy-preserving inference scheme as these algorithms can differ greatly depending on the setting (e.g., the verification process might be much different in the settings described in the future works section). However, although the user side algorithms are not part of the inference scheme, it is important to note that these user-side algorithms are no less important to the interactive protocol than the algorithms in the inference scheme.

The overarching idea behind a privacy-preserving inference scheme is to run Preprocess, M_P, and Postprocess inside of an oracle to provide a secluded execution environment which we can ensure the three algorithms are executed correctly and the user's data is not leaked to the compute party. The Setup algorithm is first ran by the compute party to set up the three oracles one for each of the three algorithms. Once the oracles are set up, the algorithms are installed into the oracle. As mentioned before, the oracles must fulfill the security properties (section 4.1.1) of correctness, zero-leakage, indistinguishability, and verifiability, we can ensure the privacy of the user's data is ensured.

Setup. In real life, the Setup algorithm will be ran by the cloud service provider to create a partition (perhaps by using the separation kernel inside a TEE or using a separate compute server) and creates a secluded execution environment to set up the oracles.

Algorithm 4 Set Up Oracles

Function $\mathsf{Setup}(M_P, \mathsf{Preprocess}, \mathsf{Postprocess})$:

1: Compute party creates $\mathcal{O}_{\mathsf{Preprocess}}$, $\mathcal{O}_{M_P}$, and $\mathcal{O}_{\mathsf{Postprocess}}$
2: Compute party installs $\mathsf{Preprocess}$ into $\mathcal{O}_{\mathsf{Preprocess}}$, install the model architecture of M_P without the trained parameters into $\mathcal{O}_{M_P}$, and install $\mathsf{Postprocess}$ into $\mathcal{O}_{\mathsf{Postprocess}}$

Preprocessing Oracle. The preprocessing oracle begins by using a key exchange protocol to first agree on a private encryption key with the user (we will refer to the encryption key as k). The user then uses a CCA-secure private key encryption scheme to encrypts its dataset of raw data samples using the key k like so: $\mathsf{Enc}_k(X) \to \mathcal{C}_x$ where $\mathcal{C}_x = \{c_1, ..., c_n\}$ is the set of ciphertexts when n raw data samples are encrypted. The user then sends $\mathcal{C}_x$ over to the oracle. The oracle then runs the decryption algorithm to obtain the raw data samples from the user and proceeds to run the data samples through the $\mathsf{Preprocess}$ algorithm, then encrypts and sends the feature vectors back to the user.

Algorithm 5 Preprocessing Oracle

Function $\mathcal{O}_{\mathsf{Preprocess}}(\mathcal{C}_x)$:

1: Run $\mathsf{KeyExchange}(1^n)$ with user to agree on key k
2: Run $\mathsf{Dec}_k(\mathcal{C}_x) \to X$ where $X = \{x_1, ..., x_n\}$ is the raw data samples in plaintext
3: Let $X'_{\mathcal{O}} = ()$
4: Let $t_{\mathcal{O}} = ()$
5: **for** $x \in X$ **do**
6: $\quad x'_{\mathcal{O}} := \mathsf{Preprocess}_k(x)$
7: $\quad$ Append $x'_{\mathcal{O}}$ to $X'_{\mathcal{O}}$
8: $\quad t := \mathsf{Mac}_k((x, \mathsf{Preprocess}, x'_{\mathcal{O}}))$
9: $\quad$ Append t to $t_{\mathcal{O}}$
10: **end for**
11: Run $\mathsf{Enc}_k(X'_{\mathcal{O}}) \to \mathcal{C}_{x'_{\mathcal{O}}}$ where $\mathcal{C}_{x'_{\mathcal{O}}}$ is the ciphertext after encrypting the feature vectors
12: **return** $\mathcal{C}_{x'_{\mathcal{O}}}$

To interact with the oracle, the user must first run $\mathsf{Preprocess.Prepare}$ to pass data to the oracle. Once the oracle has completed execution, the user must then run $\mathsf{Preprocess.Verify}$ to validate the outputs of the oracle.

Algorithm 6 Preprocessing Oracle Preparation Algorithm

Function $\mathsf{Preprocess.Prepare}(\mathcal{C}_{x'_{\mathcal{O}}}, I, X_{train}, X'_{train}, X_{infer})$:

$\mathsf{KeyExchange}(1^n) \rightarrow k$
$\mathsf{RandSample}(X_{train}, X'_{train}) \rightarrow X_{val}, X'_{val}$
$\mathsf{Mix}(X_{val}, X_{infer}) \rightarrow X,\ I$
$\mathsf{Enc}_k(X) \rightarrow \mathcal{C}_x$
return $\mathcal{C}_x$, I

Algorithm 7 Preprocessing Oracle Verification Algorithm

Function $\mathsf{Preprocess.Verify}(\mathcal{C}_{x'_{\mathcal{O}}}, t_{\mathcal{O}}, I, X_{val}, X'_{val})$:

$\mathsf{Dec}_k(\mathcal{C}_{x'_{\mathcal{O}}}) \rightarrow X'_{\mathcal{O}}$
$\mathsf{Separate}(X'_{\mathcal{O}}, I) \rightarrow X'_{\mathcal{O},train},\ X'_{\mathcal{O},infer}$
for $i \in \{1, 2, ..., m\}$ **do**
  $x := X_{val}[i]$
  $x' := X'_{val}[i]$
  $x'_{\mathcal{O}} := X'_{\mathcal{O},train}[i]$
  $t := t_{\mathcal{O}}[i]$
  $m := (x, \mathsf{Preprocess}, x')$
  if $x' \neq x'_{\mathcal{O}}$ or $\mathsf{Vrf}_k(t, m) = 0$ **then**
    return 0 (invalid)
  end if
end for
return $X'_{\mathcal{O},infer}$

Model oracle. Once again, the oracle begins by using a key exchange protocol to first agree on a private encryption key k with the user. The user uses a private key encryption scheme and the key k to encrypt the feature vectors and the model parameters:

$$\mathsf{Enc}_k(X') \to \mathcal{C}_{x'}$$

$$\mathsf{Enc}_k(M_P) \to \mathcal{C}_{M_P}$$

where $\mathcal{C}_{x'}$ is the set of ciphertexts after encrypting n feature vectors and $\mathcal{C}_{M_P}$ is the encrypted model. The user sends $\mathcal{C}_{x'}$ and $\mathcal{C}_{M_P}$ to the oracle. The oracle then decrypts $\mathcal{C}_{x'}$ and $\mathcal{C}_{M_P}$ to obtain the feature vectors and the model, conducts inference, encrypts the prediction vectors, and sends the encrypted prediction vectors back to the user.

Algorithm 8 Model Oracle

Function $\mathcal{O}_{M_P}(\mathcal{C}_{x'})$:

Run $\mathsf{Dec}_k(\mathcal{C}_{x'}) \to X'$ where $X' = \{x'_1, ..., x'_n\}$ is the feature vectors in plaintext
Run $\mathsf{Dec}_k(\mathcal{C}_{M_P}) \to M_P$ where M_P is the model with trained parameters P in plaintext
Let $Y_{p,\mathcal{O}} = ()$
Let $t_{\mathcal{O}} = ()$
for each $x' \in X'$ **do**
    $y_{p,\mathcal{O}} := M_P(x')$
    Append $y_{p,\mathcal{O}}$ to $Y_{p,\mathcal{O}}$
    $t := \mathsf{Mac}_k((x', M_P, y_{p,\mathcal{O}}))$
    Append t to $t_{\mathcal{O}}$
end for
Run $\mathsf{Enc}_k(Y_{p,\mathcal{O}}) \to \mathcal{C}_y$ where $\mathcal{C}_y$ is the encrypted prediction vectors
return $\mathcal{C}_y$

The user uses the following algorithms to interact with the oracle:

Algorithm 9 Model Oracle Preparation Algorithm

Function $\mathsf{Model.Prepare}(\mathcal{C}_{x'_{\mathcal{O}}}, I, X'_{train}, Y_{p,train}, X'_{infer})$:

$\mathsf{KeyExchange}(1^n) \rightarrow k$
$\mathsf{RandSample}(X'_{train}, Y_{p,train}) \rightarrow X'_{val}, Y_{p,val}$
$\mathsf{Mix}(X'_{val}, X'_{infer}) \rightarrow X',\ I$
$\mathsf{Enc}_k(X') \rightarrow \mathcal{C}_{x'}$
$\mathsf{Enc}_k(M_p) \rightarrow \mathcal{C}_{M_P}$
return $\mathcal{C}_{x'}, \mathcal{C}_{M_P}$

Algorithm 10 Model Oracle Verification Algorithm

Function $\mathsf{Model.Verify}(\mathcal{C}_y, t_{\mathcal{O}}, I, X'_{val}, Y_{p,val})$:

$\mathsf{Dec}_k(\mathcal{C}_y) \rightarrow Y_{p,\mathcal{O}}$
$\mathsf{Separate}(Y_{p,\mathcal{O}}\ ,\ I) \rightarrow Y_{p,\mathcal{O},train}\ ,\ Y_{p,\mathcal{O},infer}$
for $i \in \{1, 2, ..., m\}$ **do**
  $x' := X'_{val}[i]$
  $y_p := Y_{p,val}[i]$
  $y_{\mathcal{O}} := Y_{p,\mathcal{O},train}[i]$
  $t := t_{\mathcal{O}}[i]$
  if $y_p \neq y_{\mathcal{O}}$ or $\mathsf{Vrf}_k(t, M_P, y_p) = 0$ **then**
    return 0 (invalid)
  end if
end for
return $Y_{p,\mathcal{O},infer}$

The oracle begins by using a key exchange protocol to first agree on a private encryption key k with the user. The user uses a private key encryption scheme and the key k to encrypt the prediction vectors: $\mathsf{Enc}_k(Y) \rightarrow \mathcal{C}_y$. where $\mathcal{C}_y = \{c_y, ..., c_y\}$ be the set of ciphertexts when n prediction vectors are encrypted. Just as in the preprocessing oracle, the postprocessing oracle decrypts, runs the prediction vectors through the Preprocess algorithm, encrypts the output vectors, and then sends it back to the user.

Algorithm 11 Postprocessing Oracle

Function $\mathcal{O}_{\mathsf{Postprocess}}(\mathcal{C}_y)$:

1: Run $\mathsf{Dec}_k(\mathcal{C}_y) \rightarrow Y_p$ where $Y_p = \{y_1, ..., y_n\}$ is the prediction vectors in plaintext
2: Let $Y'_{p,\mathcal{O}} = ()$
3: Let $t_{\mathcal{O}} = ()$
4: **for** $y_p \in Y_p$ **do**
5: $\quad y'_{p,\mathcal{O}} := \mathsf{Postprocess}_k(y_p)$
6: $\quad$ Append $y'_{p,\mathcal{O}}$ to $Y'_{p,\mathcal{O}}$
7: $\quad m := (y_p, \mathsf{Postprocess}, y'_{p,\mathcal{O}})$
8: $\quad t := \mathsf{Mac}_k(m)$
9: $\quad$ Append t to $t_{\mathcal{O}}$
10: **end for**
11: Run $\mathsf{Enc}_k(Y'_{\mathcal{O}}) \rightarrow \mathcal{C}_{y'}$ where $\mathcal{C}_{y'}$ is the ciphertext after encrypting the output vectors
12: **return** $\mathcal{C}_{y'}$

In a very similar fashion to the preprocessing and model oracles, the user must use the following two algorithms to interact with the postprocessing oracle.

Algorithm 12 Postprocessing Oracle Preparation Algorithm

Function Postprocess.Prepare:

$\mathsf{KeyExchange}(1^n) \rightarrow k$
$\mathsf{RandSample}(Y_{p,train}, Y'_{p,train}) \rightarrow Y_{p,val}, Y'_{p,val}$
$\mathsf{Mix}(Y_{p,val}, Y_{p,infer}) \rightarrow Y_p,\ I$
$\mathsf{Enc}_k(Y_p) \rightarrow \mathcal{C}_y$
return $\mathcal{C}_y$, I

Algorithm 13 Postprocessing Oracle Verification Algorithm

Function Postprocess.Verify:

$\mathsf{Dec}_k(\mathcal{C}_{y'}) \to Y'_{p,\mathcal{O}}$
$\mathsf{Separate}(Y'_{p,\mathcal{O}}\ ,\ I) \to Y'_{p,\mathcal{O},train}\ ,\ Y'_{p,\mathcal{O},infer}$
for $i \in \{1, 2, ..., m\}$ **do**
  $y_p := Y_{p,val}[i]$
  $y'_p := Y'_{p,val}[i]$
  $y'_{\mathcal{O}} := Y'_{p,\mathcal{O},train}[i]$
  $m := (y_p, \mathsf{Postprocess}, y'_p)$
  if $y'_p \neq y'_{\mathcal{O}}$ or $\mathsf{Vrf}_k(t, m) = 0$ **then**
    return 0 (invalid)
  end if
end for
return $Y'_{p,\mathcal{O},infer}$

5.3 Security Analysis

Correctness. Correctness of the verification algorithm is trivial as the oracle will produce identical results to the user running Preprocess , M_P, or Postprocess outside of the oracle (this is described in the correctness requirement of the oracle).

Soundness. Soundness of the verification algorithm is as follows: let the output of Preprocess , M_P, or Postprocess be a string of bits, suppose the oracle has a failure probability of p (i.e., assuming Verify returns 1, the probability the oracle fails to produce the correct string of bits is p). The probability of any string being incorrect can be represented by the

following binomial distribution:

$$P(\text{any string being incorrect}) = \binom{n}{r} p^r (1-p)^{n-r} = \binom{1}{1} p^1 (1-q)^{1-1} = p$$

suppose the oracle returns n strings to the user Essentially, let n be the number of strings sampled and r be the number of strings that are incorrect in our set of sampled strings; in this case, we can set $n = 1$ because we are only sampling 1 string from the dataset, and we can set $r = 1$ because we are interested in the case where only 1 string is incorrect, we can say that the probability that any of the strings being incorrect in the verification algorithm is also p.

We can think of the verification algorithm as randomly sampling a subset of size m from the n strings in the inference set. Since the verification algorithm requires all m strings to be correct in order to return 1, then the probability the verification algorithm will return 1 is as follows:

$$P(\text{all } m \text{ samples are valid}) = (1-p)^m$$

To account for the fact the samples are distributed randomly throughout the dataset; we can think of this as calculating for the number of ways the verification algorithm can sample m samples.

$$\text{number of ways to sample a subset of size } m \text{ from } n \text{ samples} = \binom{n}{m}$$

Thus, the probability of the verification algorithm return 1 is

$$P(\text{all samples are valid}) = \binom{n}{m} (1-p)^m = \frac{n!}{(n-m)!m!}(1-p)^m$$

When looking at the final equation, it is obvious to see that the more training samples are sent to the oracle relative to the n, the more secure the verification algorithm becomes. A larger m means the probability all m samples are valid decreases and the verification algorithm will be less likely to return 1, which means it is much more difficult for the oracle to get away with making mistakes.

Resistance against ML attacks. The interactions are secure against reconstruction attacks because the user's data has been encrypted before it got to the adversary which resides in the server; thus, the adversary cannot obtain any information about the underlying feature vectors because the encryption scheme used to encrypt the feature vector is CCA-secure. The interactions are also secure against model inversion attacks, membership inference attacks, and model extraction attacks because the adversary never has access to the prediction vector, and because the user trained its model exclusively on its own data. This means that the only information a model inversion attack could yield are the feature vectors, which the user can easily compute by running the raw data samples through the Preprocess algorithm. A membership inference attack is also pointless because the only information it can yield is whether or not a data sample belongs to the training dataset, which the user already knew. Lastly, the only information that can be gained from a model extraction attack are the model parameters, which the user already knew.

Security in the inference server setting. Since all messages between the oracle and the user are encrypted, the server cannot observe the user's dataset, nor the model's parameters, nor the prediction vectors being passed, therefore, the interaction fulfills all three requirements in the Inference Server Setting Security Requirements.

Security of the Oracles. Recall in Section 4.1.1, the oracles must fulfill four properties: correctness, isolation, indistinguishability, and verifiability. Since the oracle is simply running the algorithm without adding any extra security measures that would decrease the accuracy of the model (such as differential privacy), the correctness property of the oracles

holds. For this implementation of the oracles, we make the assumption that the isolation property holds. Since we require the use of a CCA-secure private-key encryption scheme, the indistinguishability property is fulfilled. Verifiability is fulfilled by using the Mac algorithm for message authentication as well as the preparation and verification algorithm on the user's side.

5.4 Efficiency

The efficiency of the interaction and of the oracles depends on two factors: the number of samples in the inference dataset and the number of training samples sent to the oracle that must be verified. Thus, the runtime of the oracles will increase linearly to the number of of samples in the inference dataset; and the runtime of the verification algorithm will also increase linearly to the number of training samples.

5.5 Ways to Implement Oracles

The easiest way to implement oracle would be for the compute party to dedicate an entire server, with its own file storage, ram, GPU/CPU processors, etc. to running the oracles. The compute party (i.e., a cloud service provider) acts as a resources allocator and assigns servers to compute jobs and then reclaim those servers once compute jobs finish. The advantage of this approach is that hardware-level isolation can be guaranteed; however, the downside is that resource allocation may not be efficient as the compute party will have to allocate an entire server for even small tasks.

Another hardware-based approach would be to use a trusted execution environment like Intel SGX, ARM TrustZone, Keystone Enclave, etc [12][32]. The TEE approach is likely to be more efficient than allocating an entire server but it still has its downsides. One of the downsides to using TEEs is that it has been shown that TEEs are vulnerable to side-channel attacks [34]. Although it is important to note that we are not sure how relevant these side-channel attacks are to the privacy-preserving machine learning setting as the side-channel

attacks take advantage of the enclave's memory access patterns; thus, while the adversary may be able to infer details about the algorithm being ran, the individual data samples, model parameters, etc might still be secure.

In addition to hardware-based approaches, software-based approaches could also be used to implement the oracles. One particular software-base approach is to use a virtual TEE like Open-TEE [49]. In short, a virtual TEE is a software implementation of a TEE that is meant to be used as a tool to make it easier for application developers to develop and debug applications for TEEs. Once an application has been developed and tested on a virtual TEE, it can be compiled to run on an actual hardware TEE. Although virtual TEEs are meant to be used as a tool for TEE application development, we do believe it can be used to implement and run oracles without the need to be compiled to run on hardware TEEs.

Another software-based approach would be to use *secure encrypted virtualization (SEV)* to provide isolation of virtual machines and containers [50]. The idea behind SEV such as AMD SEV is to use a trusted hypervisor to provide isolation. The benefit of using SEV is that it simplifies software development. The downside is that using a hypervisor increases the attack surface because it increases the amount of hardware and software that needs to be trusted thereby increasing the likelihood of vulnerabilities introduced through the trusted hypervisor.

Chapter 6: Discussion and Extensions

In this section, we will begin by discussing how our privacy-preserving inference scheme improves upon currently existing privacy-preserving machine learning literature. Afterwards, we will discuss future avenues for research, as well as the impacts of relaxing the criteria for settings in which privacy-preserving inference schemes are be deployed.

6.1 Relations to Related Work

First and foremost, we have developed a theoretical framework for machine learning inference that emphasizes protecting the privacy of the dataset and the model parameters. Our approach is an attempt to view the problem of securing inference in a much more holistic way that also included securing preprocessing and postprocessing. This is in contrast to the approaches we covered in section 2.1 where the primary goal was to ensure the privacy of the data during forward propagation and generally ignored the leakage that resulted from the prediction vectors and the preprocessing stage.

The privacy-preserving inference scheme we developed has also enabled us to emphasize the separation of the ownership of compute resources and the ownership of the model, algorithms, and data. In much of PPML literature, the compute party often also assumes the role of model owner because it is responsible for carrying out training and inference [9]; we show that it is possible and preferable to decouple model ownership and ownership of compute resources by using the isolation property of the oracles.

By decoupling model ownership and ownership of compute resources, we show it is possible to create a setting in which privacy-preserving inference schemes can be deployed safely where typical machine learning attacks are not applicable. Furthermore, we also emphasized that preserving privacy also includes deploying the model into the right setting,

no inference scheme is privacy-preserving if deployed in the wrong setting. Lastly, we also defined the criteria a setting must meet in order to deploy a privacy-preserving inference scheme and identified a setting that met those criteria.

6.2 Future Research Directions

Thus far, we have looked at a setting (the inference server setting) where the adversary existed on the compute party and only one party assumed the party of DO, MO, and RO. Looking towards the future, we would like to look into ways to protect against adversaries sitting on the user's side. We are also interested in expanding the use of oracles to conduct training as well as other settings where there are multiple parties collaborating with the oracle (i.e., multiple DOs, MOs, and ROs).

6.2.1 User-Side Adversaries

When looking at situations where the adversary exists on the user's server, then membership inference and model extraction attacks become an issue. In cases where the user is careful to encrypt its model parameters and raw data samples but the adversary can still see the feature vectors and prediction vectors, the adversary can still conduct membership inference attacks and model extraction attacks to compromise the privacy of the user's training dataset and model. Note that in this setting, there is an adversary sitting inside the user's compute environment, but the user itself is not adversarial.

Since many powerful machine learning attacks use the prediction vector to leak information about the model parameters and data, we would like to look at ways to create what we call *secure label encoding (SLE)*. The main goal of developing SLE techniques is to develop a label encoding format that can better mitigate model extraction attacks and membership inference attacks than the one-hot encoding format. As mentioned in [46] and [44], this problem is still an open research problem, and since we cannot find any literature on this topic, we believe conducting research on SLE techniques could provide great benefits to privacy-preserving machine learning.

6.2.2 Oracles for Training

Thus far, we have only looked into ways to preserve privacy for inference. In the future, we would like to use the same oracles to train models as well. We believe the principle of isolating the execution of the training process inside an oracle could achieve the same level of privacy in a similar setting (i.e., a setting where an adversary has resides inside the compute party but cannot view the execution inside the oracle). To conduct training, we believe a few adjustments would have to be made to the Prepare and Verify algorithms since it currently uses the outputs from training; however, we believe it is possible to execute forward propagation and backpropagation inside an oracle just like in inference.

6.2.3 Relaxing Ownership Requirements

In retrospect, the requirements of our oracles and the requirements of our setting are very strict, thus, we want to look at ways to relax these requirements and explore what attacks are applicable to situations where ownership of data, model, and results is not assumed by a single party.

Let $Train(TD) \rightarrow M_P$ be an algorithm that takes as input a training dataset $TD = (X_{train}, Y_{train})$ and outputs a trained model M_P. Let $Infer(ID, M_P) \rightarrow y_p$ be an algorithm that takes as input a set of data samples $ID = X_{infer}$ (i.e., the data samples of the inference dataset, $\forall x \in X_{infer}$, $x \notin X_{train}$) and a trained model M_P and outputs a set of prediction vectors y_p.

In situations where the user does everything internally, the user collects its own training data and does not share the training data with anyone else, trains a model exclusively on its own training data, and uses its own model to conduct inference on its own inference data, we can model the workflow like so:

User $(TD,\ ID)$
$Train(TD) \rightarrow M_P$
$Infer(ID, M_P) \rightarrow y_p$

Figure 6.1: All-Internal Setting

In the All-Internal setting, data privacy and model privacy is guaranteed against all machine learning attacks because the user assumes all four roles (data owner (DO), model owner (MO), results owner (RO), and compute party (CP)).

The inference server setting that we covered previously can be modeled like so:

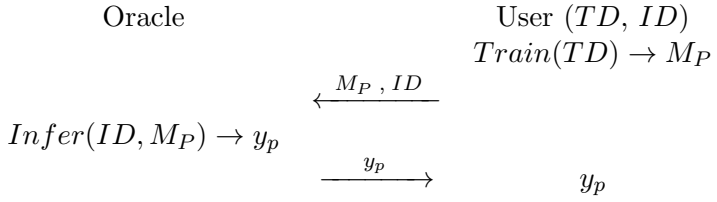


Figure 6.2: Inference Setting Workflow

Recall that for the inference server setting, we required the user to own both the training and inference dataset and conduct training locally; the only part of the workflow that was not conducted by the user is inference, which was conducted by the oracle. Once again, recall that the user's data is secure during computation because the communication between the user and the oracle is encrypted (assume all communication is encrypted for all of the following diagrams) and the oracle's isolation property ensures that no data is being leaked

from the oracle. Since the user owns the training and inference dataset, the trained model, and the prediction vector, the user assumes all roles except for CP.

Using oracles, we believe the user can also safely outsource computation to handle training securely like in the following two settings:

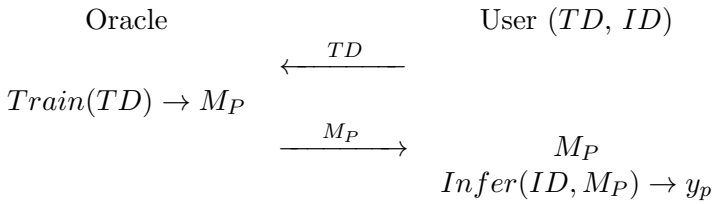


Figure 6.3: Training Server Setting Workflow

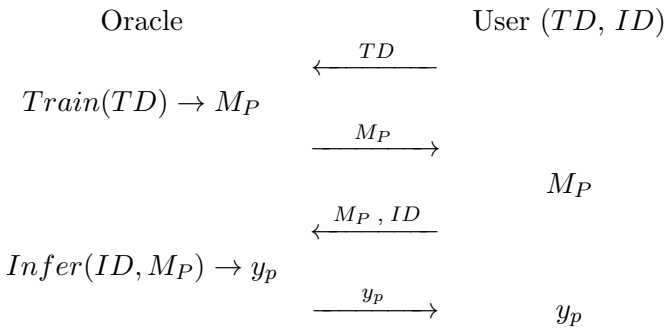


Figure 6.4: Training & Inference Server Setting Workflow

In both of the above settings (Figure 6.3 and Figure 6.4) privacy is still retained because ownership of the training dataset, inference dataset, model, and results is still assumed by one user.

However problems arise when we start to look at settings with multiple model owners, data owners, and results owners. In the following setting (Figure 6.5), multiple data owners pool their datasets together inside an oracle to build a model. Each user is a data owner and owns its respective partition of the training dataset and owns its own inference dataset

which is not shared with the oracle.

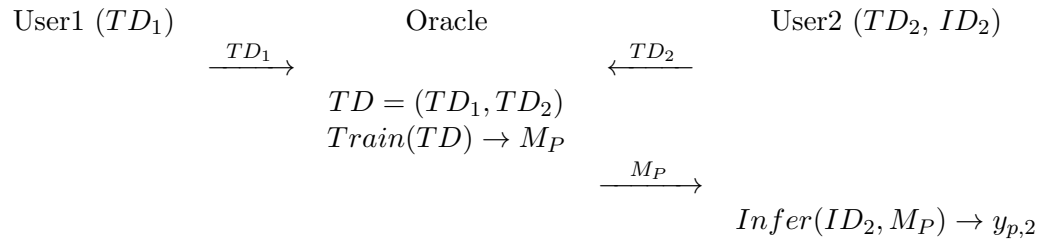


Figure 6.5: Collaborative Training Setting Workflow

This setting violates the first criterion of the PPML Setting Requirements: *the model must be trained using only the data from a single DO*. In this case, User2 can conduct membership inference attacks and compromise the privacy of User1's dataset. Essentially, User2 can conduct a membership inference attack and if the attack reveals that a data sample is in the training dataset but is not in the User2's training partition, then User2 will know that it is in another user's training partition.

Settings where model ownership is transferred from one user to another are also vulnerable to membership inference attacks.

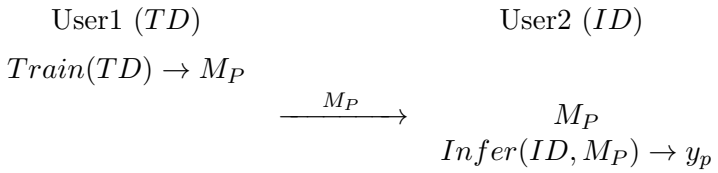


Figure 6.6: Training Server Setting Workflow

Because this setting violates the second criterion of the PPML Setting Requirements: *during inference, only the party that owns the training dataset and the inference dataset can see the prediction vectors for the inference dataset*, although User1 did not hand its dataset over to User2, User2 can still use the model sent over by User1 to compromise the privacy of

User1's training dataset.

In settings where there are multiple model owners, data privacy is also not guaranteed. A good example of such a setting is the federated learning setting.

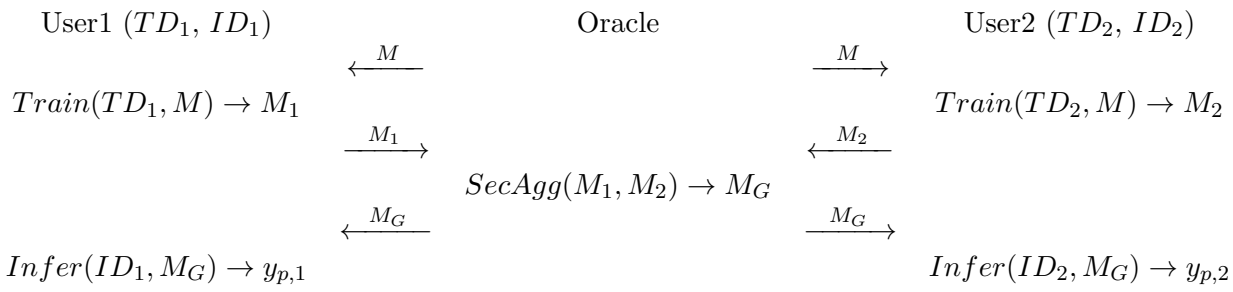


Figure 6.7: Federated Learning Setting Workflow

The main disadvantage of federated learning is that it breaks the first criterion of the PPML Setting Requirements thereby leading to a plethora of attacks that allow users in the federation to compromise the privacy of other user's data [38], [51], [52]. Furthermore, since federated learning also violates the second criterion of the PPML Setting Requirements, there is reason to believe the prediction vectors computed using the global model could enable model extraction to be used to steal other user's local models. We have yet to see a model extraction attack successfully conducted in the federated learning setting, however we suspect it is possible for users to steal the parameters of other user's local models as there have been very powerful attacks that can elude the secure aggregation protocol used by federated learning [40]. Furthermore, Kairouz et al. [53] also mentioned it is an open question on what other protections would need to be put into place to protect against model inversion attacks in the federated learning setting.

In the typical machine-learning-as-a-service setting, User1 has trained a model and is now using its trained model as a service to make predictions for other users (User2 in this instance). In this setting, User2 can conduct either a membership inference attack to compromise the privacy of User1's training dataset or compromise the privacy of User1's

trained model parameters by conducting a model extraction attack.

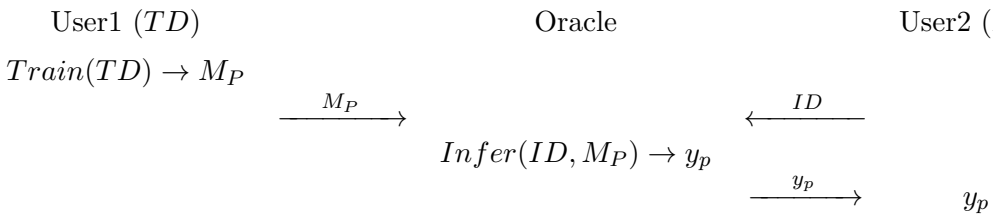


Figure 6.8: Machine-Learning-as-a-Service (MLaaS) Setting Workflow

According to Tramèr et al. [46], any setting where the results owner is not the same party as the model owner, the results owner can use a model extraction attack to compromise the security of the MO's model parameters. Furthermore, an adversarial results owner can use model extraction as a precursor to conduct membership inference attacks.

Model Validation Setting

Due to the numerous challenges in aggregating data, we were curious if there were any machine learning settings that still protected the privacy of the data and model parameters but still allowed for multiple DOs, MOs, and/or ROs. One of the settings we came up with that we believe might fit these characteristics is a setting we coin *the model validation setting*.

In the model validation setting, there are three parties: a user, a server, and other data owners (we will refer to all other DOs as one party). Just like in the inference server setting, the user has collected and cleaned its own training dataset and assumed the role of data owner and also uses its local compute instance to train a model on the training dataset thereby also assuming the role of the model owner. However, in this setting, the user needs to validate the performance of its model against a broader set of data it does not have. Therefore, in the model validation setting, the user is interested only in metrics like testing accuracy derived from testing the model against other data owners. Thus, in this case,

there is no results owner because nobody needs to see the prediction vector on a particular data sample. The server acts as the compute party and is responsible for aggregating data from other DOs, testing the model against the aggregated data, and ultimately calculating the statistics for the user.

This setting is relevant to situations where researchers have limited access to data in which it may have enough data to train a model but does not have enough data to validate its model. In other words, the user does not have enough data to create both a training dataset and an inference dataset; in which case, the user is forced to use all data samples for training and must rely on other DOs to validate its model. This setting is especially applicable to healthcare as the datasets tend to have fewer data samples, biased (e.g., perhaps there are too many patients of the same race), and are often stored in silos [54] [55]. The model validation setting is relevant to this scenario because the user likely has a small dataset that likely contains bias and needs to evaluate its model on other DOs' datasets to ensure the model is not biased but the user cannot aggregate datasets from other DOs because of data access restrictions.

The setting can be described in the following diagram:

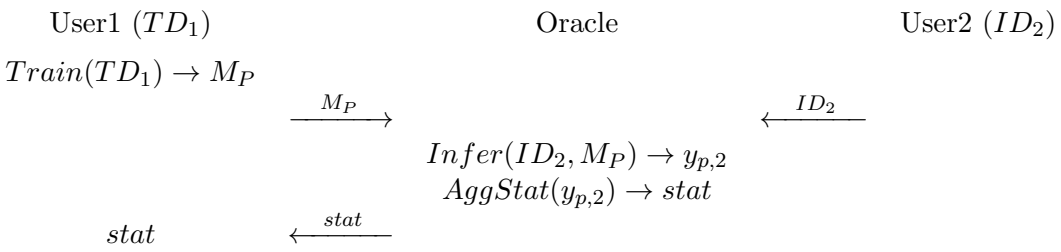


Figure 6.9: Model Validation Setting Workflow

In the diagram, User1 trains a model using its own training dataset TD_1 and then sends the model to the oracle. The oracle collects data from User2 and uses it for inference. The prediction vectors are aggregated and used to compute some statistical metric *stat* that is then sent back to User1. We believe the aggregation of the prediction vectors should be

enough to ensure the privacy of User2's data. More research will need to be conducted to prove this.

In this setting, the following security requirements must be enforced:

- No other party can view a DO's raw data samples (i.e., other DO's and CP should be able to view the DO's raw data samples).
- No other party other than the MO should be able to retrieve the model parameters.
- No party can see the prediction vectors.

This setting does not violate the first criterion of the PPML Setting Requirements because the model is trained using only the data of a single user. The second criterion is also fulfilled because even though no DO owns both the training and inference datasets, no DO has to the prediction vector either; therefore, membership inference nor model extraction is possible.

Threat model. In the model validation setting, the adversary has infiltrated the compute server and can see any data stored in the compute server's memory and can see the data sent between the compute server, the user, and the other DOs. The adversary can also see the output of any oracle that runs on the server.

Chapter 7: Conclusion

In this thesis, we proposed a theoretical primitive, which we call a privacy-preserving inference scheme, that is designed to conduct machine learning inference in a way that protects the privacy of the underlying user's data. We emphasize the need for a more holistic view of privacy-preserving machine learning that emphasized privacy-preservation during the preprocessing, inference, and postprocessing stages. We also emphasized that privacy-preserving machine learning solutions will always be vulnerable to attacks like membership inference and model extraction unless deployed into the right setting. We then proceeded to define the requirements for a setting to be deemed appropriate to deploy privacy-preserving machine learning solutions and identified a setting that fulfills these requirements (the inference server setting). Lastly we developed a theoretical construction of a privacy-preserving inference scheme, described how it worked in the inference server setting, and informally argued why it is secure in the inference server setting.

Overall, we showed that by using the isolation property of the oracles in our privacy-preserving inference scheme primitive, a privacy-preserving inference scheme can be resilient to attacks on machine learning models if deployed in the right setting; thereby, suggesting that it is possible to decouple ownership of computational resources from ownership of the model, results, and data. In the future, we would like to provide an actual implementation of our construction and look into ways to relax the strict assumptions of our setting.

Bibliography

[1] A. Ng, "Andrew ng: Artificial intelligence is the new electricity - youtube," Jan 2017. [Online]. Available: https://www.youtube.com/watch?v=21EiKfQYZXc

[2] J. Jumper, R. Evans, A. Pritzel, T. Green, M. Figurnov, O. Ronneberger, K. Tunyasuvunakool, R. Bates, A. Žídek, A. Potapenko *et al.*, "Highly accurate protein structure prediction with alphafold," *Nature*, vol. 596, no. 7873, pp. 583–589, 2021.

[3] "Artificial intelligence &; autopilot." [Online]. Available: https://www.tesla.com/AI

[4] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "A general reinforcement learning algorithm that masters chess, shogi, and go through self-play," *Science*, vol. 362, no. 6419, pp. 1140–1144, 2018.

[5] J. R. Ayala Solares, F. E. Diletta Raimondi, Y. Zhu, F. Rahimian, D. Canoy, J. Tran, A. C. Pinho Gomes, A. H. Payberah, M. Zottoli, M. Nazarzadeh, N. Conrad, K. Rahimi, and G. Salimi-Khorshidi, "Deep learning for electronic health records: A comparative review of multiple deep neural architectures," *Journal of Biomedical Informatics*, vol. 101, p. 103337, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1532046419302564

[6] Centers for Medicare & Medicaid Services, "The Health Insurance Portability and Accountability Act of 1996 (HIPAA)," Online at http://www.cms.hhs.gov/hipaa/, 1996.

[7] P. Scharre, *Army of None: Autonomous Weapons and the Future of War*. Old Saybrook, CT: Tantor Audio, 2018.

[8] M. Roth, "Artificial intelligence at the cia - current applications," Nov 2019. [Online]. Available: https://emerj.com/ai-sector-overviews/artificial-intelligence-at-the-cia-current-applications/

[9] M. Al-Rubaie and J. M. Chang, "Privacy-preserving machine learning: Threats and solutions," *IEEE Security & Privacy*, vol. 17, no. 2, pp. 49–58, 2019.

[10] M. Bellare, V. T. Hoang, and P. Rogaway, "Foundations of garbled circuits," in *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, ser. CCS '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 784–796. [Online]. Available: https://doi.org/10.1145/2382196.2382279

[11] C. Dwork, "A firm foundation for private data analysis," *Communications of the ACM*, January 2011. [Online]. Available: https://www.microsoft.com/en-us/research/publication/a-firm-foundation-for-private-data-analysis/

[12] M. Sabt, M. Achemlal, and A. Bouabdallah, "Trusted execution environment: what it is, and what it is not," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1. IEEE, 2015, pp. 57–64.

[13] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2020.

[14] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[15] K. Nandakumar, N. Ratha, S. Pankanti, and S. Halevi, "Towards deep neural network training on encrypted data," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, 2019, pp. 0–0.

[16] R. Gilad-Bachrach, N. Dowlin, K. Laine, K. Lauter, M. Naehrig, and J. Wernsing, "Cryptonets: Applying neural networks to encrypted data with high throughput and accuracy," in *Proceedings of The 33rd International Conference on Machine Learning*, ser. Proceedings of Machine Learning Research, M. F. Balcan and K. Q. Weinberger,

Eds., vol. 48. New York, New York, USA: PMLR, 20–22 Jun 2016, pp. 201–210. [Online]. Available: https://proceedings.mlr.press/v48/gilad-bachrach16.html

[17] E. Hesamifard, H. Takabi, and M. Ghasemi, "Deep neural networks classification over encrypted data," in *Proceedings of the Ninth ACM Conference on Data and Application Security and Privacy*, 2019, pp. 97–108.

[18] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell, S. Agarwal, A. Herbert-Voss, G. Krueger, T. Henighan, R. Child, A. Ramesh, D. M. Ziegler, J. Wu, C. Winter, C. Hesse, M. Chen, E. Sigler, M. Litwin, S. Gray, B. Chess, J. Clark, C. Berner, S. McCandlish, A. Radford, I. Sutskever, and D. Amodei, "Language models are few-shot learners," 2020. [Online]. Available: https://arxiv.org/abs/2005.14165

[19] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *27th USENIX Security Symposium (USENIX Security 18)*, 2018, pp. 1651–1669.

[20] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa, "Delphi: A cryptographic inference service for neural networks," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020, pp. 2505–2522.

[21] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 19–38.

[22] A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, p. 612–613, nov 1979. [Online]. Available: https://doi.org/10.1145/359168.359176

[23] K. Bonawitz, V. Ivanov, B. Kreuter, A. Marcedone, H. B. McMahan, S. Patel, D. Ramage, A. Segal, and K. Seth, *Practical Secure Aggregation for Privacy-Preserving Machine Learning*. New York, NY, USA: Association for Computing Machinery, 2017, p. 1175–1191. [Online]. Available: https://doi.org/10.1145/3133956.3133982

[24] K. Liu, H. Kargupta, and J. Ryan, "Random projection-based multiplicative data perturbation for privacy preserving distributed data mining," *IEEE Transactions on Knowledge and Data Engineering*, vol. 18, no. 1, pp. 92–106, 2006.

[25] C. K. Liew, U. J. Choi, and C. J. Liew, "A data distortion by probability distribution," *ACM Trans. Database Syst.*, vol. 10, no. 3, p. 395–411, sep 1985. [Online]. Available: https://doi.org/10.1145/3979.4017

[26] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang, "Deep learning with differential privacy," in *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*, 2016, pp. 308–318.

[27] A. D. Sarwate and K. Chaudhuri, "Signal processing and machine learning with differential privacy: Algorithms and challenges for continuous data," *IEEE signal processing magazine*, vol. 30, no. 5, pp. 86–94, 2013.

[28] Z. Ji, Z. C. Lipton, and C. Elkan, "Differential privacy and machine learning: a survey and review," *arXiv preprint arXiv:1412.7584*, 2014.

[29] K. Chaudhuri, A. D. Sarwate, and K. Sinha, "A near-optimal algorithm for differentially-private principal components." *Journal of Machine Learning Research*, vol. 14, 2013.

[30] K. Chaudhuri, C. Monteleoni, and A. D. Sarwate, "Differentially private empirical risk minimization." *Journal of Machine Learning Research*, vol. 12, no. 3, 2011.

[31] X. Jiang, Z. Ji, S. Wang, N. Mohammed, S. Cheng, and L. Ohno-Machado, "Differential-private data publishing through component analysis," *Transactions on data privacy*, vol. 6, no. 1, p. 19, 2013.

[32] D. Lee, D. Kohlbrenner, S. Shinde, D. Song, and K. Asanović, "Keystone: An open framework for architecting tees," *arXiv preprint arXiv:1907.10119*, 2019.

[33] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa, "Oblivious Multi-Party machine learning on trusted processors," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 619–636. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/ohrimenko

[34] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," in *2015 IEEE Symposium on Security and Privacy*, 2015, pp. 640–656.

[35] H. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," Reading, Massachusetts, 1993.

[36] P. Vepakomma, O. Gupta, T. Swedish, and R. Raskar, "Split learning for health: Distributed deep learning without sharing raw patient data," *arXiv preprint arXiv:1812.00564*, 2018.

[37] K. Bonawitz, H. Eichner, W. Grieskamp, D. Huba, A. Ingerman, V. Ivanov, C. Kiddon, J. Konečnỳ, S. Mazzocchi, H. B. McMahan *et al.*, "Towards federated learning at scale: System design," *arXiv preprint arXiv:1902.01046*, 2019.

[38] B. Hitaj, G. Ateniese, and F. Perez-Cruz, "Deep models under the gan: information leakage from collaborative deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 603–618.

[39] D. Pasquini, G. Ateniese, and M. Bernaschi, "Unleashing the tiger: Inference attacks on split learning," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 2113–2129.

[40] D. Pasquini, D. Francati, and G. Ateniese, "Eluding secure aggregation in federated learning via model inconsistency," *arXiv*, 11 2021.

[41] M. Barreno, B. Nelson, R. Sears, A. D. Joseph, and J. D. Tygar, "Can machine learning be secure?" in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, 2006, pp. 16–25.

[42] J. Feng and A. K. Jain, "Fingerprint reconstruction: From minutiae to phase," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 2, pp. 209–223, 2011.

[43] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, "Privacy risk in machine learning: Analyzing the connection to overfitting," 2017. [Online]. Available: https://arxiv.org/abs/1709.01604

[44] M. Fredrikson, S. Jha, and T. Ristenpart, "Model inversion attacks that exploit confidence information and basic countermeasures," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 1322–1333. [Online]. Available: https://doi.org/10.1145/2810103.2813677

[45] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," 2016. [Online]. Available: https://arxiv.org/abs/1610.05820

[46] F. Tramèr, F. Zhang, A. Juels, M. K. Reiter, and T. Ristenpart, "Stealing machine learning models via prediction APIs," in *25th USENIX Security Symposium (USENIX Security 16)*. Austin, TX: USENIX Association, Aug. 2016, pp. 601–618. [Online]. Available: https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/tramer

[47] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," 2015.

[48] F. Tramèr, R. Shokri, A. S. Joaquin, H. Le, M. Jagielski, S. Hong, and N. Carlini,

"Truth serum: Poisoning machine learning models to reveal their secrets," 2022. [Online]. Available: https://arxiv.org/abs/2204.00032

[49] B. McGillion, T. Dettenborn, T. Nyman, and N. Asokan, "Open-tee – an open virtual trusted execution environment," in *2015 IEEE Trustcom/BigDataSE/ISPA*, vol. 1, 2015, pp. 400–407.

[50] J. Ménétrey, C. Göttel, M. Pasin, P. Felber, and V. Schiavoni, "An exploratory study of attestation mechanisms for trusted execution environments," 2022. [Online]. Available: https://arxiv.org/abs/2204.06790

[51] L. Melis, C. Song, E. D. Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," 2018.

[52] M. Nasr, R. Shokri, and A. Houmansadr, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, may 2019. [Online]. Available: https://doi.org/10.1109%2Fsp.2019.00065

[53] P. Kairouz, H. B. McMahan, B. Avent, A. Bellet, M. Bennis, A. N. Bhagoji, K. Bonawitz, Z. Charles, G. Cormode, R. Cummings *et al.*, "Advances and open problems in federated learning," *Foundations and Trends® in Machine Learning*, vol. 14, no. 1–2, pp. 1–210, 2021.

[54] M. J. Willemink, W. A. Koszek, C. Hardell, J. Wu, D. Fleischmann, H. Harvey, L. R. Folio, R. M. Summers, D. L. Rubin, and M. P. Lungren, "Preparing medical imaging data for machine learning," *Radiology*, vol. 295, no. 1, pp. 4–15, Apr 2020, 32068507[pmid]. [Online]. Available: https://pubmed.ncbi.nlm.nih.gov/32068507

[55] G. Varoquaux and V. Cheplygina, "Machine learning for medical imaging: methodological failures and recommendations for the future," *npj Digital Medicine*,

vol. 5, no. 1, p. 48, Apr 2022. [Online]. Available: https://doi.org/10.1038/s41746-022-00592-y

Biography

Minh Quan Do received a Bachelors of Science in Bioengineering from George Mason University in December 2019, and he will soon be graduating from George Mason University with a Masters of Science in Computer Science. He has 2 years of industry experience as a lead machine learning engineer at the Leidos Innovation Center's AI/ML Accelerator. As of October 2022, he has completed a temporary position at CGI Federal as a blockchain engineer where he researched ways to utilized zero-knowledge proofs to develop attribute-based access control systems (ABAC).