

Investigating Privacy Attacks in the Gray-Box Setting to Enhance Collaborative Learning Schemes

Federico Mazzone
University of Twente
f.mazzone@utwente.nl

Ahmad Al Badawi
Duality Technologies
aalbadawi@dualitytech.com

Yuriy Polyakov
Duality Technologies
ypolyakov@dualitytech.com

Maarten Everts
University of Twente
Linksight
maarten.everts@utwente.nl

Florian Hahn
University of Twente
f.w.hahn@utwente.nl

Andreas Peter
University of Oldenburg
andreas.peter@uni-oldenburg.de

Abstract—The notion that collaborative machine learning can ensure privacy by just withholding the raw data is widely acknowledged to be flawed. Over the past seven years, the literature has revealed several privacy attacks that enable adversaries to extract information about a model’s training dataset by exploiting access to model parameters during or after training.

In this work, we study privacy attacks in the gray-box setting, where the attacker has only limited access — in terms of view and actions — to the model. The findings of our investigation provide new insights for the development of privacy-preserving collaborative learning solutions. We deploy SMARTCRYPTNN, a framework that tailors homomorphic encryption to protect the portions of the model posing higher privacy risks.

Our solution offers a trade-off between privacy and efficiency, which varies based on the extent and selection of the model components we choose to protect. We explore it on dense neural networks, where through extensive evaluation of diverse datasets and architectures, we uncover instances where a favorable sweet spot in the trade-off can be achieved by safeguarding only a single layer of the network. In one of such instances, our approach trains ~4 times faster compared to fully encrypted solutions, while reducing membership leakage by 17.8 times compared to plaintext solutions.

I. INTRODUCTION

The demand for more complex and accurate machine learning models in fields like image recognition and natural language processing has highlighted the need for extensive training data [78]. As a result, collaboration among data-owning entities has become crucial to leverage larger and more diverse datasets and enhance model performance. However, this collaborative approach raises privacy concerns, as sharing raw data can expose sensitive information, potentially violating privacy regulations and raising confidentiality concerns (for instance, this may be the case for hospitals, municipalities, insurance companies, etc.). To address this issue, collaborative learning solutions, like Federated Learning (FL) [48] and Split Learning (SL) [58], have been proposed. Those methods aim to protect privacy by enabling the training of a joint model without directly outsourcing the raw data.

The recent development of privacy attacks has made evident that solely withholding the training data is an insufficient strategy to ensure privacy protection in these scenarios [55],

[50], [25], [32], [77]. These attacks aim to retrieve information about the dataset a given model has been trained on by only having access to the model parameters (*white-box setting*), or to the associated prediction functionality (*black-box setting*). To mitigate such attacks, solutions based on Differential Privacy (DP) [65], [1], [49], Fully Homomorphic Encryption (FHE) [62], and MultiParty Computation (MPC) [72], [51], [71] have been introduced. However, these solutions come with trade-offs: DP introduces noise to protect privacy but may lead to accuracy loss; FHE provides strong privacy guarantees but has a high computation cost for deep arithmetic circuits that require bootstrapping, making it an unfeasible approach in many practical training contexts; MPC may require a large number of interactions between multiple parties and high communication bandwidth.

In this paper, we go beyond the conventional white- and black-box settings commonly explored in existing Privacy-Preserving Machine Learning (PPML) literature [55], and we explore a more flexible *gray-box setting* where the model is only *partially exposed* to the adversary. We investigate different classes of privacy attacks and analyze how the efficacy of the attacks is influenced by the limitations imposed on the adversary’s view and actions. For neural networks, which represent our main research focus, this gray-box setting can manifest as an attacker having access only to specific layers of the model, rather than the entire architecture. This particular setting occurs naturally in scenarios like SL, where the server only has access to the second split of the model. It is also relevant for non-collaborative scenarios, like the deployment of generative models: think for instance about the training of an AutoEncoder or a GAN, where only the decoder or the generator, respectively, is deployed to the public. From a privacy evaluation perspective, the gray-box setting can be seen as a generalization of such scenarios.

While prior work in the literature indirectly initiated the analysis of the gray-box setting for specific attacks or scenarios [55], [57], our aim is to provide a more comprehensive and methodical investigation. To this end, we consider multiple classes of privacy attacks, including membership inference, property inference, and model inversion [61]. By adapting state-of-the-art attacks to suit the gray-box setting, we assess their efficacy via both theoretical considerations and empirical

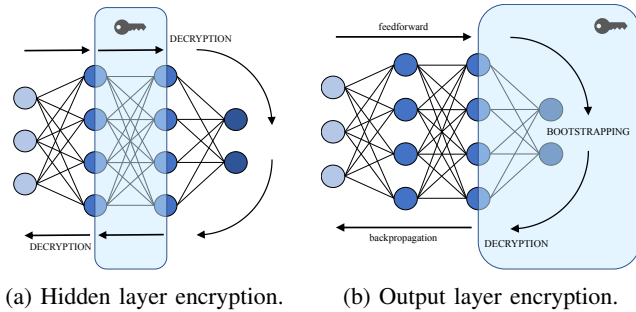


Fig. 1: Diagram representation of our approach.

evaluation. Our findings reveal that certain classes of attacks exhibit greater performance when specific layers of the model are accessible. For instance, membership inference attacks show a higher sensitivity towards the latter layers of the model, while model inversion techniques heavily rely on the availability of the initial layers.

Building upon these insights, we show how to leverage them to enhance collaborative learning solutions in terms of privacy, efficiency, and accuracy. We adopt a targeted defense strategy by applying homomorphic encryption to safeguard only the most vulnerable portions of the model (see Figure 1). This way, the resulting gradient updates align with the encryption status of the layers, ensuring that encrypted layers remain encrypted and the exposed layers remain in plaintext throughout the entire training process. The resulting approach offers higher privacy guarantees compared to plain FL, while concurrently achieving less computational and communication overhead when compared to fully encrypted solutions [62]. The freedom to choose the number of layers to protect introduces a novel trade-off between privacy and efficiency. In practical scenarios, this enables the final user to flexibly fine-tune the solution to meet specific efficiency requirements while sacrificing little privacy. Our investigation reveals that in certain instances encrypting only a few layers of the model is indeed enough to strongly inhibit most privacy attacks. For instance, encrypting just one layer was enough to reduce a membership inference leakage by 17.8 times. Notably, our approach becomes particularly efficient for deep models.

Similar to other work [62], we only consider the semi-honest setting. In Section V-C, we see that active variants of privacy attacks are not feasible in this case, hence we exclude them from our in-depth analysis. Techniques developed against Split Learning, such as the ones described in [57], usually focus on model inversion in the malicious setting, and as such are not part of our investigation. A detailed description of our threat model can be found in Section III.

We carry out our investigation on diverse datasets for supervised classification tasks, spanning both tabular and image features. On those datasets, we assess both the privacy and efficiency of our partially encrypted FL strategy, observing how such a trade-off evolves based on the specific portion of the model that is protected. To empirically measure the computation and communication costs of our approach, we implement it in a framework named SMARTCRYPTNN, a prototype built upon the multiparty CKKS-RNS variant pro-

vided by the OpenFHE library [4]. Additionally, to ensure consistency with the work by Sav et al. [62], we simulate the protocol communication using MiniNet, allowing us to set different network delay and bandwidth constraints. To the best of our knowledge, we are the first to release an open-source implementation of FL under homomorphic encryption.

The main contributions of this work are listed below:

- We adapt different classes of privacy attacks to the gray-box setting and we assess how their efficacy scales with the portion of the model available to the adversary.
- We design a privacy-preserving collaborative training solution for deep learning that selectively encrypts only the layers of the model with higher privacy risk.
- We implement our solution and assess it on a variety of well-known datasets, making our framework available open-source at <https://github.com/FedericoMazzone/SmartCryptNN>.

II. BACKGROUND

In this section, we provide a brief description of some basic ML and PPML-related concepts and introduce the models and datasets used in our investigation.

A. Federated Learning

Federated Learning (FL) is a collaborative learning approach where multiple data-owning clients, P_1, \dots, P_N , jointly train a common model while keeping their data decentralized. Originally, Shokri and Shmatikov [65] introduced an FL scheme for neural networks where clients independently train local models on their datasets, while sharing portions of their model parameters with a global model hosted by a supporting server. The training process is asynchronous, with each client repeatedly downloading portions of the global model parameters, updating its local model, and then uploading portions of the gradients to the global model.

However, a more widely adopted approach for FL is Federated Averaging (FedAvg), as proposed by McMahan et al. [48]. In FedAvg, the training process occurs in synchronous rounds. In each round, a subset of clients is selected to participate. These clients perform a local training step on their own data, updating their model parameters. These locally trained models are then sent to the central server, where they are averaged together to obtain a new global model. The aggregated global model is then distributed back to all participating clients.

B. Multilayer Perceptrons

In this work, we focus on Multilayer Perceptrons (MLPs), for which we provide some notation as follows. For an MLP model f with L layers, we denote by w_i, b_i, ϕ_i the weights, biases, and activation function of layer i , respectively. We denote by l_i the output of layer i , that is $l_i = \phi_i(l_{i-1}w_i + b_i)$, where $l_0 := x$ is the input of the model. And we denote the intermediate linear application output as $u_i = l_{i-1}w_i + b_i$. For supervised classification tasks, the loss function is denoted by $L(f(x), y)$, where y is the ground-truth label associated to x , and can be minimized through Gradient Descent (GD) techniques. Below is a schematic representation of the computations performed during one step of the training process. More details can be found in Appendix A.

$$\begin{array}{ccc}
u_1 = l_0 w_1 + b_1 & \rightarrow \dots \rightarrow & u_L = l_{L-1} w_L + b_L \\
l_1 = \phi_1(u_1) & & l_L = \phi_L(u_L) \\
\uparrow & & \downarrow \\
l_0 = x & & L(l_L, y) \\
& & e_L = \partial L / \partial l_L \\
& & \downarrow \\
\nabla b_1 = e_1 \phi'_1(u_1) & & \nabla b_L = e_L \phi'_L(u_L) \\
\nabla w_1 = e_1 \phi'_1(u_1) l_0^T & \leftarrow \dots \leftarrow & \nabla w_L = e_L \phi'_L(u_L) l_{L-1}^T \\
(e_0 = e_1 \phi'_1(u_1) w_1^T) & & e_{L-1} = e_L \phi'_L(u_L) w_L^T
\end{array}$$

C. Multiparty Fully Homomorphic Encryption

Fully Homomorphic Encryption (FHE) enables the evaluation of unlimited-depth arithmetic circuits on encrypted data, utilizing a technique called bootstrapping for refreshing ciphertexts after homomorphic operations. In Multiparty Homomorphic Encryption (MHE), the secret key is shared among multiple parties, who can use their shares to distributively generate collective public/evaluation keys and perform distributed decryption and bootstrapping protocols. For our work, we use the threshold FHE version of the Cheon-Kim-Kim-Song (CKKS) scheme [17]. The threshold FHE construction follows the design of Asharov et al. [5], which was initially instantiated for the Brakerski-Gentry-Vaikuntanathan (BGV) scheme [11]. Mouchet et al. [53] applied this blueprint to the Brakerski / Fan-Vercauteren (BFV) [10], [24] and CKKS schemes, also introducing a distributed bootstrapping protocol based on threshold FHE. The distributed CKKS bootstrapping protocol was further developed in Sav et al. [62]. Additional information about threshold CKKS can be found in Appendix C.

To instantiate threshold CKKS, we use the OpenFHE library [4], which implements optimized CKKS variants proposed in [38] and threshold FHE extensions for CKKS. We use a N -out-of- N threshold scheme with additive sharing of the secret key, where all parties need to be present to perform decryption and bootstrapping. But the scheme can be easily modified to allow for arbitrary thresholds, i.e., t -out-of- N threshold FHE using Shamir secret sharing.

The CKKS scheme is well-suited for floating-point-like arithmetic and performs computations on vectors of real/complex numbers, allowing for Single Instruction, Multiple Data (SIMD) operations. In particular, the scheme works with residual polynomial rings of the form $R_q = \mathbb{Z}_q[x]/(x^n + 1)$, where n is a power of two. Note that due to the CKKS encoding, a plaintext can embed a vector of up to $n/2$ slots.

We use the alternate packing approach designed by Sav et al.[62] to encode matrices, and Chebyshev interpolation to approximate non-polynomial functions (more details in Appendix D).

We work with a Residue Number System (RNS) instantiation of CKKS [38], which achieves the highest efficiency for CKKS among known variants of the scheme. Here, we briefly describe it at a high level. Given unique primes q_0, q_1, \dots, q_L , an RNS chain of moduli is built as $Q_i = \prod_{j=0}^i q_j$ for $i \in \{0, \dots, L\}$. A freshly encrypted ciphertext is a pair $(c_0, c_1) \in R_{Q_L}$. Then, after each multiplication, the ciphertext is rescaled to scale down the message and truncate the least significant bits, dropping the highest RNS limb, e.g., going

from R_{Q_L} to $R_{Q_{L-1}}$ after first rescaling. The maximum number of multiplications is given by $L - 1$. However, not all of these levels can be used for the main computation as the bootstrapping procedure consumes levels, too (even in the case of distributed bootstrapping).

The main drawbacks of MHE that often make it impractical in real-world contexts are the heavy computation costs and significant communication overhead, especially due to the need for distributed bootstrapping, which can become a bottleneck in practical implementations. This is particularly relevant in ML scenarios when evaluating deep multiplicative circuits like neural networks. Additionally, the inherent noise in the encryption scheme and the approximation of non-linear functions can lead to a decrease in the model’s accuracy. To address these challenges and enhance both efficiency and accuracy, our work focuses on reducing the number of layers that need to be encrypted.

D. Datasets and Models

We selected well-known public datasets widely used in the PPML literature: with Texas-100, Purchase-100, Locations our datasets include tabular data, and with AT&T, MNIST, EMNIST Letters, LFW our datasets include images. We refer to Appendix B for further details. This mix ensures a comprehensive evaluation of the privacy attacks and our prototype. We highlight that this work’s main goal is not to achieve the highest possible accuracy on the given datasets, but to investigate the efficacy of privacy attacks across different layers of our target model. To accomplish this, we deliberately subsampled some of the datasets, using a reduced dataset for training the models. By doing so, we aim to create a vulnerable model that is more susceptible to privacy attacks. As discussed in Section IV there are various ways to make a model vulnerable. Constraining the training set to a subset is compatible with real-world scenarios, where training parties might struggle with limited data availability. This approach also helps in expediting the experimental assessment of our encrypted training solution, as the experiments can be run within a reasonable timeframe given the reduced training data size.

As for the models, we primarily focus on MLP architectures, for compatibility with our FHE prototype, using the plain Stochastic Gradient Descent (SGD) optimizer and minimizing the Mean Squared Error (MSE) loss. Specifically, we train an MLP with two hidden layers of size 30 and 20 on the MNIST datasets, and 256, 128, and 64 on the Location dataset. For Purchase100 and Texas100, we adopt the MLP architecture proposed in [55], which consists of hidden layers with sizes 1024, 512, 256, and 128. Additionally, we train an MLP with two hidden layers of size 64 on the EMNIST Letters dataset, and a CNN on the LFW dataset. For the CNN, we adopt the architecture proposed in [50], which consists of three convolutional layers with 32, 64, and 128 filters, each with a 3x3 kernel and a max pooling layer, followed by two fully connected layers of size 256 and 2. For further details about the training settings, we refer to the repository.¹

¹<https://github.com/FedericoMazzone/SmartCryptNN>

III. THREAT MODEL

In this section, we briefly discuss our considered threat models. We distinguish between the information available to the adversary and the actions performed by the adversary.

A. Adversary’s View

PPML literature classifies privacy attacks depending on the adversary’s view of the model [55]. In black-box attacks, the adversary can only access the model’s output for arbitrarily chosen inputs but lacks information about model parameters. In white-box attacks, the adversary has full access to the model’s architecture, parameters, and hyperparameters used during training. This enables them to compute any function of the model parameters and any chosen input, including intermediate computations of the feedforward pass, i.e. output of intermediate layers. For labeled input, the adversary can hence compute the corresponding loss and the gradients for each layer.

We study attacks in the *gray-box setting*, which represents a more flexible threat model generalizing white-box scenarios for privacy-attacks, by considering intermediate adversary capabilities. In this gray-box setting, the adversary has only access to a subset of the model parameters and can compute any function limited to those parameters. Specifically, for the studied feed-forward neural networks, we consider a layer-wise granularity for this partial knowledge. This entails partitioning the model’s layers into two subsets: the *exposed layers* \mathcal{L}_E and the *secret layers* \mathcal{L}_S . For the exposed layers, the adversary has the same access as in the white-box setting. Thus, the adversary can compute the loss value for a given labeled example only if the last layer is exposed. Conversely, the adversary has no access to any parameters of secret layers.

B. Adversary’s Actions

In the scenarios where the adversary joins the training phase, PPML literature distinguishes between passive and active behaviour [55], [50] for privacy attacks. However, this taxonomy is inconsistent across prior works, and can cause confusion with terminology used in cryptography. Since in this paper we address both the ML and cryptographic aspects, we clarify our threat model as follows.

From a machine learning perspective, we closely follow the definition given by Nasr et al. [55] and distinguish between *ML-passive* and *ML-active* adversaries. An *ML-passive* adversary only observes passively the legitimate model updates and attempts to infer information by performing inference on the model, without changing anything in the local or global collaborative training procedure. In contrast, an *ML-active* adversary, influences the target model during training in order to coerce the data owners into unintentionally releasing more information through the model. The active adversary’s actions may include choosing specific artificially crafted inputs for the training procedure, or performing gradient ascents on specific inputs. We specifically use this definition in Section IV.

Note that this distinction differs from the cryptographic definition of passive (or semi-honest) and active (or malicious) adversaries. A *crypto-passive* adversary follows the protocol, while a *crypto-active* adversary can arbitrarily deviate from the

protocol. This cryptographic definition is more powerful than the machine learning definition of an active adversary, as the latter only allows for local manipulation of the model. In the cryptographic sense, a passive adversary is potentially allowed to feed arbitrary input to the model training procedure, but must adhere to the specified protocol. We specifically use this definition in Section V.

IV. PRIVACY ATTACKS IN THE GRAY-BOX SETTING

In this section, we assess existing privacy attacks in the gray-box setting, aiming to get insights into what portions of the model are more vulnerable to different kinds of attacks. In general, a privacy attack is a technique designed to extract information about the data that a particular model has been trained on. Privacy attacks are typically categorized based on the specific type of information they aim to extract. Following the taxonomy provided by Rigaki et al. in their survey [61], we classify privacy attacks into three categories: membership inference, model inversion, and property inference. To provide a comprehensive evaluation of privacy attacks, we considered all the aforementioned classes in our investigation. For each of these classes, we discuss how the attacks can operate in the gray-box setting described in Section III, and evaluate their effectiveness based on the degree of exposure of the model to the adversary. We select state-of-the-art attacks from the existing literature to experimentally support our assessment. These attacks often come with an ML-active variant, whose efficacy in the gray-box setting is highly dependent on how the secret layers are concealed and the limitations imposed on the adversary’s actions. We discuss the availability of their ML-active counterparts directly in Section V-C.

A. Membership Inference

Membership inference attacks aim to determine whether or not a given data point was part of the training set. These attacks exploit the intrinsic difference in the model’s behavior when performing prediction over known training data versus unseen data. Membership inference attacks reveal how much a model retains from its training data, helping to gauge the potential effectiveness of other privacy attacks such as data reconstruction but can also pose significant privacy risks on their own. Since the introduction of the first membership inference attack by Shokri et al. [66] in 2017, numerous studies have investigated the underlying causes of membership leakage in ML models. The primary contributing factor to membership leakage appears to be model overfitting or poor generalization [66], [74]. Several factors can exacerbate this issue, including a limited number of training samples [66], [31], high model complexity leading to overparametrization [55], and high feature dimensionality [66].

Gray-Box Setting: In the gray-box setting, the efficacy of membership inference attacks heavily depends on access to the last layers of the model. While the initial layers of a neural network tend to extract simple features from the input, enabling them to generalize well, the later layers specialize in detecting higher-level abstract features in the input, making them prone to overfitting and memorizing the specific training examples. For instance, in a CNN model trained for image classification, you can expect the first layers to learn more about edges and abstract shapes of the input image, while

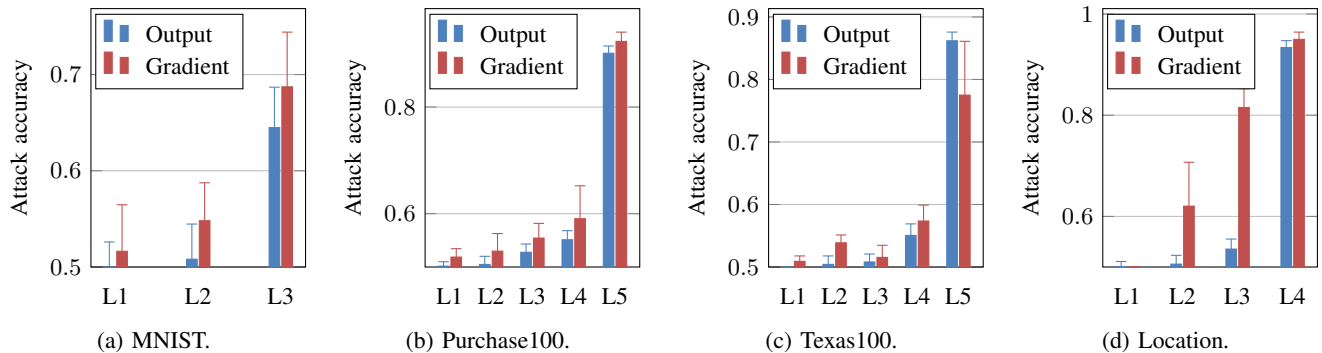


Fig. 2: Layer-wise accuracy of the white-box membership inference attack by Nasr et al. [55] against different datasets and models, exploiting both the layer’s output and gradient.

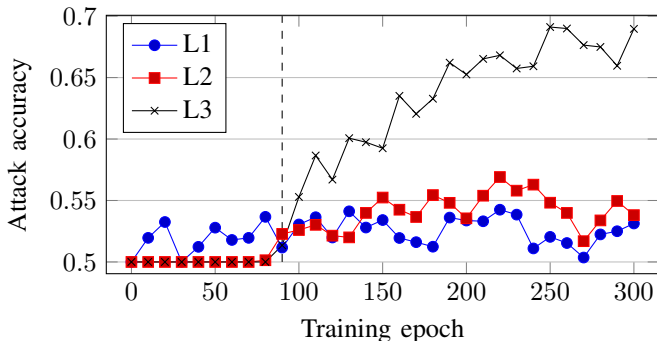


Fig. 3: Layer-wise accuracy of the membership inference attack by Nasr et al. [55] against intermediate models for the MNIST classification task. The model leaks more membership information as the number of training epochs grows. This behavior is particularly evident for the output layer.

the last layers more about intricate texture and artifacts within those shapes [75]. Moreover, as the neural network progresses to the later layers, the parameter capacity increases, causing the target model to store information about the exact training samples [55]. Therefore, if the last layers of the model are accessible, membership inference attacks tend to be stronger due to the higher degree of membership information leakage.

Experimental Assessment: To experimentally evaluate membership inference in the gray-box setting, we chose the white-box attack by Nasr et al. [55] due to its component-wise approach. Like other lines of work, their attack treats membership inference as a binary classification task, and trains a machine learning model to accomplish this task. We employ the supervised version, in which the attacker is assumed to know a portion of the private training dataset and uses this knowledge to perform supervised training of the attack model. Given a target data point, the attacker performs a feedforward pass of the model over it, computing hidden layer outputs, loss, and subsequent backpropagation to calculate gradients for each layer. These computed values, along with the true label, serve as input features for the attack model. While Nasr et al.’s work [55] primarily focuses on the combination of the last layers in the model, our investigation aims to assess the

privacy leakage of each individual layer of the model.

In line with the results by Nasr et al. [55] attacking the last layers, we observe a similar effect for our generalized setting: the combination of multiple (intermediate) layers does not leak significantly more membership information than the just the last of those layers. For instance, attacking layers 1, 3, and 4 of a given model does not provide a significant advantage over attacking just layer 4. Consequently, we simplified our experimental setting and attack each layer individually and do not expect significantly different accuracy compared to attacks that include any combination of previous layers.

The number of members and non-members is the same, resulting in a baseline attack accuracy of 50%. In Figure 2, we present the outcome of our experimental assessment on different datasets. We report the average and maximum attack accuracy over four runs. Our experimental results confirm the trend of the later layers of a model to leak more information compared to earlier layers. In particular, the very last layer leaks considerably more information than the others. This is especially evident in the cases of Purchase100 (Figure 2b) and MNIST (Figure 2a), where the attack accuracy for layer output increases from 55.12% to 90.10% (~7.8 times increase offset to the baseline) and from 50.81% to 64.49% (~17.8 times increase offset to the baseline), respectively, when passing from the second-to-last layer to the last layer.

Additionally, in line with the findings of Nasr et al. [55], our experiments confirm that the availability of gradients contributes to a higher attack accuracy.

Attacking Intermediate Models: A model acquires more information about its training data the more training iterations it undergoes, thus leaking progressively more information as it approaches the end of the training process. To assess how the membership leakage changes across the epochs, we use the attack by Nasr et al. [55] against the intermediate models. Specifically, in Figure 3, we present the attack accuracy against the model trained on the MNIST dataset. We carry out the attack at 10-epoch intervals, targeting each layer within the model independently. Our experiment reveals a consistent upward trend in attack accuracy with respect to the number of training epochs, especially for the later layers. Notably, a significant deviation from the attack baseline appears only from epoch 90.

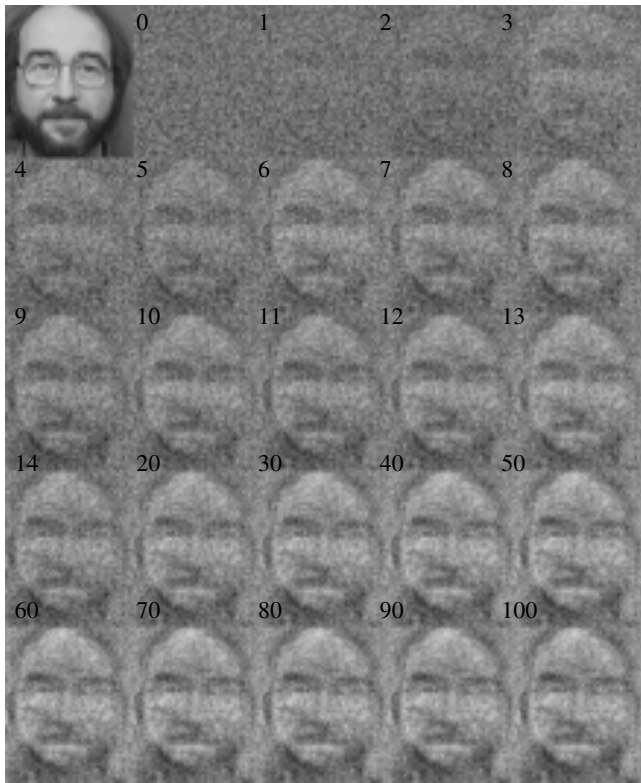


Fig. 4: Reconstruction of a face in the AT&T dataset performed at different training epochs. The first picture is a class representative, while the number at the top-left of each picture denotes the corresponding training epoch of the model.

B. Model Inversion

Model inversion, also known as reconstruction attack, aims to recreate training samples and, in some cases, their associated labels. There are two main types of model inversion attacks: those that aim to reconstruct actual training samples [77], [73] and those that aim to craft a class representative [25], [32]. The latter type is particularly useful when all samples associated with a given label are similar, such as faces of the same person, or when the attacker has no prior knowledge about what a specific label encodes. The effectiveness of model inversion attacks has been shown to increase with the target model’s level of overfitting [74] and its predictive power, as measured by loss minimization [76]. To mitigate these attacks, one suggested approach is to partially prune the gradients before updating the model [77].

One of the first model inversion attacks on neural networks was developed by Fredrikson et al. [25]. The attacker crafts a dummy input for the target model and then uses gradient descent to optimize the dummy input. The high-level idea is that, instead of fitting the model parameters to the input, the attacker computes the gradient of the loss function with respect to the input and fits the latter to the model parameters. In contrast, other model inversion attacks use generative models to construct class representative. For instance, Hitaj et al. [32] proposed a method based on Generative Adversarial Networks (GANs). In this approach, the attacker designs a generator G with the purpose of producing examples for a specific

class y , using the target model itself as the discriminator. The generator takes noise x_ϵ as input and generates $x_y = G(x_\epsilon)$, intended to represent class y . The parameters of G , denoted as θ_G , are optimized to minimize $L(f(x_y), y)$, which indicates how confidently the model classifies x_y as y . Another model inversion attack by Zhu et al. [77] assumes that the attacker has access to the gradients $\nabla L(f(x), y)$ computed by a training party for a data point (x, y) , e.g., in a collaborative learning setting when the attacker corrupts the supporting server (with no secure aggregation ongoing) or if the attacker corrupts all parties but the target one. The attacker initializes a dummy data point (x', y') , computes the corresponding gradients $\nabla L(f(x'), y')$, and minimizes the distance between these dummy gradients and the original gradients, which in turn brings the dummy input (x', y') closer to the original (x, y) . To solve the minimization problem, the attacker differentiates $\|\nabla L(f(x'), y') - \nabla L(f(x), y)\|$ with respect to (x', y') and uses GD to find a local minimum (x', y') .

Gray-Box Setting: All these attacks share the common requirement of computing the derivative of the target model’s loss with respect to the model input: $\partial L / \partial x$, which can be written as $\frac{\partial L}{\partial x} = \frac{\partial L}{\partial l_1} \frac{\partial l_1}{\partial x}$, where l_1 is the output of the first layer. To compute this derivative, the attacker then needs access to the first layer’s gradients and parameters.

Blocking access to the first layer straightforwardly prevents the attack by Fredrikson et al. [25]. This limitation also prevents backpropagation from the target model to the generator for GAN-based approaches like [32], and hinders the ability to solve the gradient difference minimization problem in the case of the attack by Zhu et al. [77]. We conclude that denying the attacker access to the first layer of the model appears to be sufficient in preventing these specific types of model inversion attack, thus no experimental assessment is needed in this case. However, we refrain from making a general claim, as there might still be potential attacks that can circumvent this limitation, and leave this as future research direction.

Attacking Intermediate Models: We also evaluated the effectiveness of model inversion attacks against intermediate training models. In Figure 4, we display the reconstruction of a face from the AT&T dataset performed during different training epochs. We use the model inversion attack by Fredrikson et al. [25]. Following their work, we train an MLP with one hidden layer with 3000 nodes, sigmoid activation function, and a softmax output layer, using the SGD optimizer and crossentropy loss function. Similarly to membership inference, our experiments reveal that as the number of training epochs increases, the reconstructed class representative becomes more and more visually similar to the corresponding training examples. However, we notice that the attack works already well even after just a few epochs. This happens since a model inversion attack works well as soon as the model is generalizing well enough.

C. Property Inference

Property inference attacks aim to extract properties about the training samples that are uncorrelated to the learning task at hand. For example, in a face recognition task, where the goal is gender classification, the attacker might try to infer whether people in the training dataset are wearing sunglasses.

Similarly, for a model designed for handwriting recognition, the attacker may attempt to determine the font used to write the messages (e.g., cursive or block letters).

The underlying conditions and factors that enable property inference attacks are not yet fully understood [61]. It remains unclear what specific characteristics or vulnerabilities in a model make it susceptible to such attacks. Surprisingly, these attacks have shown effectiveness even on well-generalized models, and the relationship between their efficacy and overfitting is still unclear [27], [50]. It has been suggested that sharing only a small portion of the gradients, as in the collaborative approach by Shokri and Shmatikov [65], may contribute to mitigate these attacks [50].

To carry out a property inference attack, the adversary needs access to samples both with and without the property they want to infer. They calculate the gradients of the target model for both types of samples and trains a binary classifier to distinguish between the gradients of samples with the property of inference and samples without it. In collaborative learning, the adversary can obtain the gradients of honest parties by computing the difference between two subsequent model updates. However, if the adversary only has access to aggregated data from other parties, the attack may become less effective as the number of honest parties increases.

Gray-Box Setting: Due to the lack of a clear understanding of the underlying causes of property inference leakage, it is challenging to predict how such attacks will scale in the gray-box setting. The general idea is that the less gradients are exposed to the attacker, the less information is available for inference. However, it remains unclear whether specific types of layers (e.g., convolutional or fully-connected) or their positions in the model contribute to higher or lower information leakage.

Experimental Assessment: To assess this category of attacks in the gray-box setting, we build upon the white-box property inference attack proposed by Melis et al. [50] and adapt it to target only a subset of the gradients. This attack works in batches, aiming to determine whether a batch of data points possesses the target property or not. In our variant of the attack, we feed to the attack model only the gradients computed with respect to parameters in the exposed layers.

We conduct the assessment on two datasets: Labeled Faces In the Wild (LFW) and EMNIST letters. For the LFW dataset we train a CNN model following the architecture provided in [50], with 3 convolutional and 2 fully connected layers, using gender as main classification task, and race:black as inference task, which has been reported to yield the highest attack rate. While for the EMNIST letter dataset, we train a custom MLP model with 3 layers, using the standard 26 letters classification as the main task, and the letter case (upper or lower) as the inference task. The gradients of the exposed layers are fed to a Random Forest classifier with 50 trees, and the attack accuracy is averaged over multiple instances of the attack model. For both datasets, we use batches of size 32, which are balanced with respect to the inference property, resulting in an attack accuracy baseline of 50%.

For the CNN model trained on LFW, we did not find any specific patterns indicating whether some layers are more or less susceptible to inference than others. The attack exhibited

TABLE I: Property inference attack by Melis et al. [50] against 5-layer CNN trained on the LFW dataset. The attack accuracy is reported per layer, demonstrating high variability across multiple training attempts of the same target model.

Run	Conv. 1	Conv. 2	Conv. 3	Dense 1	Dense 2
1	66.88	81.44	88.13	95.06	81.69
2	60.25	74.31	72.00	53.19	56.13
3	63.31	69.63	79.06	66.25	55.88
4	86.69	93.44	91.75	99.88	96.00

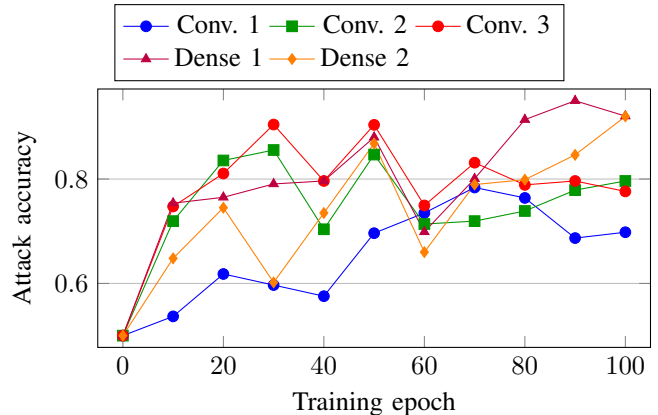


Fig. 5: Layer-wise accuracy of a property inference attack against intermediate models for the LFW classification task.

significant variability when conducting multiple iterations of training the target model and conducting repeated testing. In Table I, we report the attack accuracy layer-wise for multiple attempts of the experiment, revealing no consistent vulnerability or resistance of any layer across the runs. On the other hand, for the MLP model trained on the EMNIST letters dataset, the attack did not achieve accuracy significantly above the baseline, regardless of the choice of the layers to attack (including combinations of multiple layers).

Attacking Intermediate Models: We also assessed how property inference varies across training epochs. In Figure 5, we report the attack accuracy of the property inference attack by Melis et al. [50] against intermediate training models for the LFW classification task. The attack is performed every 10 training epochs and it targets each layer of the model individually to get better insights. The model exhibits a general upward trend for information leakage as the number of training epochs grows. However, this trend appear not to be consistent, and the leakage is already substantial from the start.

V. PARTIALLY ENCRYPTED MODELS

In this section, we leverage the insights from the investigation in Section IV to design a flexible collaborative learning protocol that allows users to trade off privacy for efficiency. Our approach involves using an FHE scheme to encrypt the most vulnerable parts of the model and performing federated training on this partially encrypted model. The level of privacy protection is determined by the selection of layers to be encrypted (*secret layers* \mathcal{L}_S), while the remaining layers are left in plaintext (*exposed layers* \mathcal{L}_E). The more layers we

encrypt, the less information potential adversaries can access, thus enhancing privacy, but it also leads to more computations performed under encryption, thus reducing efficiency. This flexibility allows our approach to achieve greater privacy than employing standard FL [48], while achieving a more practical level of efficiency than fully encrypted solutions [26], [62].

When performing feedforward and backpropagation on the model, we need to be careful about how to switch from secret to exposed layers and vice versa. Computations are conducted under encryption whenever an encrypted layer is encountered, possibly invoking bootstrapping to refresh intermediate computations. When an exposed layer is encountered, a decryption is called to allow continuing the training pass in plaintext (see Figure 1).

A. Protocol Description

We describe the protocol for MLP models, trained with SGD optimizer, and MSE loss, but it can be easily generalized to any feed-forward model. Including simple momentum-based optimizers such as Nesterov Accelerated Gradient is straightforward and only requires an additional weight update. While adaptive optimizers such as AdaGrad [21], RMSProp [68], and Adam [40] may require additional care due to the division by the rescaling coefficient. For adapting the protocol to include convolutional layers, we refer to [62].

Global Training: The protocol involves N training parties P_1, \dots, P_N and a central server S , whose role can potentially be taken by any P_i . The parties want to jointly train an MLP model, thus they agree on the model depth L , architecture, activation functions, training hyper-parameters, and on the set of layers to encrypt \mathcal{L}_S . The central server initiates the protocol by coordinating the FHE setup and key-generation phase, at the end of which P_1, \dots, P_N have their own private shares, and all actors possess the corresponding public key, relinearization, and rotation keys, which allow all parties to perform the necessary homomorphic operations. The central server initializes the model f in plaintext, by generating random weight matrices w_1, \dots, w_L and bias vectors b_1, \dots, b_L of appropriate sizes, according to the distribution given by the chosen initialization technique. Then, it uses the public key to encrypt the parameters w_i, b_i corresponding to the secret layers $L_i \in \mathcal{L}_S$, after proper encoding (see Appendix D).

At this point, the training starts, and proceeds as in standard FL. The central server broadcast the partially encrypted model to P_1, \dots, P_N . Each party P_i performs a certain number of local training iterations, and sends back the updated local model to the central server. The central server finally aggregates the local models, by averaging the parameters, using homomorphic addition and scalar multiplication by $1/N$ for the ones in \mathcal{L}_S . These steps are repeated for a fixed number of iterations E_g or until some convergence condition is satisfied (see Protocol 1).

Local Training: The local training subroutine, presented in Protocol 2, involves each party P_i performing E_l model updates locally before updating the central model. During each local update, a batch of size B is sampled from the local training set D_i . For each example in the batch, one training step is performed, which includes feedforward and backpropagation to compute gradients. These gradients are then averaged across the batch sample and used to perform a

Protocol 1 Global Training

ACTORS: central server S , training parties P_1, \dots, P_N

- 1: Parties agree on model depth L , architecture a , activation functions ϕ_j , learning coefficient η , batch size B , parameter initialization technique ModelInit, number of global iterations E_g , and the set of layers to encrypt \mathcal{L}_S
- FHE SCHEME SETUP:**
- 2: All actors collaborate to run the FHE setup and key generation, resulting in collective public key pk
- MODEL INITIALIZATION:**
- 3: S initializes the model in plaintext $(w_1, b_1), \dots, (w_L, b_L) \leftarrow \text{ModelInit}(L, a)$
- 4: S encrypts parameters in secret layers:
- 5: **for** $j = 1 \rightarrow L$ **do**
- 6: **if** $L_j \in \mathcal{L}_S$ **then**
- 7: $w_j \leftarrow \text{Encrypt}(pk, w_j)$
- 8: $b_j \leftarrow \text{Encrypt}(pk, b_j)$
- 9: **end if**
- 10: **end for**
- FEDERATED TRAINING**
- 11: **for all** $g = 1 \rightarrow E_g$ **do**
- 12: S sends $(w_1, b_1), \dots, (w_L, b_L)$ to the parties
- LOCAL TRAINING:**
- 13: Each P_i runs E_l local model updates, getting the local model $(w_1^i, b_1^i), \dots, (w_L^i, b_L^i)$ and sends it to S (see Protocol 2)
- AGGREGATION:**
- 14: S aggregates the local models:
- 15: **for** $j = 1 \rightarrow L$ **do**
- 16: $w_j \leftarrow \frac{1}{N} \sum_{i=1}^N w_j^i$ (HE eval. if $L_j \in \mathcal{L}_S$)
- 17: $b_j \leftarrow \frac{1}{N} \sum_{i=1}^N b_j^i$ (HE eval. if $L_j \in \mathcal{L}_S$)
- 18: **end for**
- 19: **end for**

local model update. After E_l updates, the local model is sent to the central server, which proceeds to the aggregation step. We can add L2 regularization at the cost of an additional plaintext-scalar multiplication, by multiplying the weight matrices by $1 - \eta\lambda/B$ just before line 8, where λ is the weight decay coefficient.

Protocol 2 Local Training

- 1: **for** $e = 1 \rightarrow E_l$ **do**
- 2: Sample $(x_1, y_1), \dots, (x_B, y_B)$ from local dataset
- 3: **for** $b = 1 \rightarrow B$ **do**
- 4: Compute gradients $(\nabla w_1^b, \nabla b_1^b), \dots, (\nabla w_L^b, \nabla b_L^b)$ by performing one training pass on (x_b, y_b) (Protocol 3)
- 5: **end for**
- 6: Update local model:
- 7: **for** $j = 1 \rightarrow L$ **do**
- 8: $w_j \leftarrow w_j - \frac{\eta}{B} \sum_{b=1}^B \nabla w_j^b$ (HE eval. if $L_j \in \mathcal{L}_S$)
- 9: $b_j \leftarrow b_j - \frac{\eta}{B} \sum_{b=1}^B \nabla b_j^b$ (HE eval. if $L_j \in \mathcal{L}_S$)
- 10: **end for**
- 11: **end for**

Protocol 3 outlines one training pass of our approach. To feedforward an input through a partially encrypted model, we begin by feeding the vector in plaintext starting from the input layer. When an encrypted layer is encountered, the

computation proceeds under encryption, with bootstrapping being called when necessary (we omit bootstrapping calls from the protocol description since their call frequency depends on the FHE parameters). As soon as an exposed layer is reached, a distributed decryption is invoked. The same process is followed during the backpropagation step. Note that if the last layer is encrypted, the loss is also computed under encryption.

Protocol 3 One Training Pass

```

1:  $l_0 \leftarrow x$ 
   FEEDFORWARD:
2: for  $j = 1 \rightarrow L$  do
3:    $u_j \leftarrow l_{j-1}w_j + b_j$  (HE eval. if  $L_j \in \mathcal{L}_S$ )
4:   if  $j < L \wedge L_j \in \mathcal{L}_S \wedge L_{j+1} \in \mathcal{L}_E$  then
5:      $u_j \leftarrow \text{Decrypt}(u_j)$ 
6:   end if
7:    $l_j \leftarrow \phi_j(u_j)$  (HE eval. if  $(j = L \wedge L_L \in \mathcal{L}_S)$ 
   or  $(j < L \wedge L_j, L_{j+1} \in \mathcal{L}_S)$ )
8: end for
   BACKPROPAGATION:
9:  $e_L \leftarrow y - l_L$  (HE eval. if  $L_L \in \mathcal{L}_S$ )
10:  $\nabla b_L \leftarrow e_L \phi'_L(u_L)$  (HE eval. if  $L_L \in \mathcal{L}_S$ )
11:  $\nabla w_L \leftarrow \nabla b_L l_{L-1}^T$  (HE eval. if  $L_L \in \mathcal{L}_S$ )
12: for  $j = L - 1 \rightarrow 1$  do
13:    $e_j \leftarrow \nabla b_{j+1} w_{j+1}^T$  (HE eval. if  $L_{j+1} \in \mathcal{L}_S$ )
14:   if  $L_j \in \mathcal{L}_E \wedge L_{j+1} \in \mathcal{L}_S$  then
15:      $e_j \leftarrow \text{Decrypt}(e_j)$ 
16:   end if
17:    $\nabla b_j \leftarrow e_j \phi'_j(u_j)$  (HE eval. if  $L_j, L_{j+1} \in \mathcal{L}_S$ )
18:    $\nabla w_j \leftarrow \nabla b_j l_{j-1}^T$  (HE eval. if  $L_j, L_{j-1} \in \mathcal{L}_S$ 
   or  $L_j, L_{j+1} \in \mathcal{L}_S$ )
19: end for

```

Unless we are at the last layer of the model, decrypting just after the linear transformation at line 5 is optimal. To show this, let us consider the case we are at the end of a group of encrypted layers, that is we are at layer L_j for some $j < L$, with $L_j \in \mathcal{L}_S$ and $L_{j+1} \in \mathcal{L}_E$. If instead of decrypting u_j , we perform an additional step under encryption, and decrypt after the evaluation of $\phi_j(u_j)$, then an adversary could just invert the activation function if bijective (e.g., sigmoid) or still get information about u_j for most of the activation functions commonly used. If we keep going under encryption for a step further, and decrypt for instance after the next linear transformation u_{j+1} in order to keep l_j private, then, since $L_{j+1} \in \mathcal{L}_E$, we would need to invoke a decryption for ∇w_{j+1} , which depends on l_j . However, an adversary could easily retrieve l_j given ∇w_{j+1} and ∇b_{j+1} , which is also in plaintext since $L_{j+1} \in \mathcal{L}_E$. Similar remarks hold for the decryption of the error in the backpropagation phase at line 15.

Consequently, when encrypting a single non-output layer, the corresponding layer output and gradient cannot be protected. Thus, to protect the initial or central portions of the model, at least two consecutive layers need to be encrypted, and even in that case, the gradient of the bias of the last layer of such group will be exposed. One approach to address this limitation is to omit the bias parameters on that specific layer.

An additional argument against encrypting only one non-output layer is the potential for an adversary to reconstruct the encrypted parameters. If the adversary can gather enough

input and output pairs (x, y) , where $y = wx + b$, they could retrieve the values of w and b by solving a system of linear equations with the layer parameters as the unknowns. Encrypting two consecutive layers (or the last one), already makes the system of equation significantly harder to solve. The system will involve many more variables and the activation function of the first layer as well, which is typically non-linear. As additional measure, lowering the precision of the FHE scheme and introducing additional noise in the ciphertext can further complicate the reconstruction of the encrypted layers.

Prediction: After completing the training phase, the partially encrypted model can be directly used for predictions. However, cooperation among the training parties remains necessary for distributed bootstrapping and decryption calls. Prediction queries can be initiated by any of the training parties or an external entity. If the querier is one of the training parties, they can locally conduct the feedforward step and seek assistance from the others only for bootstrapping and decryption, including the potential output decryption if the last layer is also encrypted. If the querier is an external entity, additional precautions are needed due to the presence of exposed layers. To ensure the privacy of the query, the querier encrypts their input with the collective public key of the FHE scheme, and sends the encrypted input to one of the training parties. The selected party will then perform the feedforward pass on behalf of the querier. In this case, operations on the exposed layers must be adapted to work under encryption. While matrix multiplication and bias addition remain straightforward, the activation functions need to be approximated to be homomorphically evaluated, leading to a potential loss of accuracy. Moreover, unlike during training, the output of a group of adjacent layers should not be decrypted.² The final output of the model then remains encrypted, regardless of whether the last layer is private or not. At this point, the training parties can send the decryption shares of the output to the querier, who can then reconstruct the output in clear, or a *key-switch* [53] to the public key of the querier can be performed.

Delayed Encryption: We can leverage the investigation of privacy attacks on intermediate training models to optimize our solution. From Section IV, we know that some attacks start to be effective only after some number of training epochs. An optimization for our solution consists of starting the federated learning process fully in plaintext, and encrypting (some of) the layers only once the model becomes vulnerable. The number of layers to encrypt can be dynamically adjusted during the training process. We will discuss this more in details in Section V-E.

B. Efficiency Analysis

Compared to a fully encrypted approach, partially encrypted models offer significant efficiency advantages, including reduced number of computations under encryption, lighter model updates, and fewer communication rounds. To simplify the analysis, we assume all layers to have same order of magnitude sizes, and plaintext size and operation costs to be negligible with respect to their encrypted counterpart.

²It is possible to perform plaintext operations in exposed central layers, when they are sufficiently distant from both the input and output layers to lower the possibility of reconstructing the query input or output.

In general, since computations are carried out in plaintext in the exposed layers, we expect a lower-bound for the gain in computational complexity to be at least linear in the number of exposed layers relative to the total number of layers in the model, i.e. $|\mathcal{L}_E|/L$. Additionally, we can avoid the homomorphic evaluation of the last activation function in each group of encrypted layers (but the ones containing the last layer). Moreover, as the training parties decrypt their computations at the end of each encrypted block, the need for distributed bootstrapping decreases or even disappears. This results in gains in computation efficiency, communication size, and communication rounds, as each bootstrapping process typically requires one round-trip of communication. This advantage is amplified by the fact that bootstrapping in the CKKS scheme needs the ciphertext to have some levels left, further increasing the computational overhead in fully encrypted models. Depending on the number of contiguous encrypted layers, even faster somewhat HE schemes can be adopted.

Regarding the communication size during model update and broadcast, we again observe a linear gain in $|\mathcal{L}_E|/L$, as the model parameters corresponding to exposed layers are sent in plaintext. On the other hand, during the aggregation phase, fewer parameters need to be averaged under encryption, leading to an additional gain in terms of computational complexity. Finally, we note that in the optimized version of our solution, the performance gain factor $|\mathcal{L}_E|/L$ changes according to the number of exposed layers across the training epochs. We provide detailed execution time and communication size of a specific use case in Section V-F.

C. Security Analysis

In this section, we sketch a security proof for the encrypted layers of the model, and we discuss the capabilities of different types of adversaries for each class of privacy attack.

Our approach aims to preserve the privacy of the training data, during both the training and prediction phases, by encrypting the most vulnerable layers of the model. In the semi-honest setting, we prove that no party, including the central server, can learn more information about the training data of any other party or the model parameters corresponding to any layer in \mathcal{L}_S , other than what can be deduced from their own data (including the model output, in case of predictions), and from the parameters and intermediate computations of the layers in \mathcal{L}_E . In the case of predictions requested by an external entity, we can make this claim stronger, as the querier should not learn anything, other than what can be deduced from only their own input data and the query output.

For the security proof, we proceed as in [62], but assuming the simulator is also given access to the parameters of the exposed layers at each iteration, and to the output of each decryption call. The idea is to see the overall scheme as a composition of the underlying FHE protocols, which are all simulatable. For the basic protocols like key generation and decryption, we rely on the proofs by Mouchet et al. [53], while for the distributed bootstrapping, we rely on the proof by Sav et al. [62]. Note that the security of our approach does not hold for a malicious adversary, which can exploit the decryption call in the protocol to decrypt the secret layers.

In the rest of the section, we discuss whether specific attacks can still run on the exposed layers. We consider three threat model configurations, based on the possible combinations of the ML and cryptographic adversary’s capabilities introduced in Section III:

- 1) ML-passive and crypto-passive, where the adversary follows the protocol and can only use inputs from the original dataset (no maliciously crafted inputs);
- 2) ML-active and crypto-passive, where the adversary follows the protocol but may craft malicious inputs for the training procedure;
- 3) ML-active and crypto-active, where the adversary can arbitrarily deviate from the protocol and may craft arbitrary malicious input.

Note that the distinction between the first two settings is important in real-world scenarios, since it has implications in terms of detectability and liability. If a client becomes corrupted during the training process, an ML-active attack is potentially more detectable than an ML-passive attack. Detection can occur by analyzing the intermediate updates or the final model, or by employing some form of commitment to the training dataset.

In Table II, we outline the capabilities of each attack discussed in Section IV for the threat model configurations described above. Note that, in contrast to FL in plaintext, the presence of encrypted layers restricts the adversary from freely conducting any inference on the model.

D. Differential Privacy and the Protection of Exposed Layers

We discuss the relation between our work and solutions based on differential privacy, and how the latter can be used to protect the exposed layers in our approach. The trade-offs offered by our solution and by DP are substantially different: privacy against efficiency and privacy against accuracy, respectively. In particular, our solution does not compromise the model’s accuracy, hence providing more utility than DP, while at the same time being more efficient than fully encrypted approaches, posing our work in between FHE-based and DP-based solutions.

DP can also be incorporated in our approach and used to mitigate the leakage from the exposed layers, providing in this way theoretical guarantees for our framework. We describe a possible implementation in Appendix F, where we show how to adapt the approach by Shokri and Shmatikov [65] to partially encrypted models. You can see our solution as a way to reduce the consumption of the privacy budget by encrypting some of the layers. The save in terms of privacy budget is proportional to the number of parameters that are encrypted over the total amount of parameters of the model.

E. Procedure to Choose \mathcal{L}_S

Since the privacy leakage of a model strongly depends on the training dataset, there is no general-purpose guideline on how many and which layers to encrypt. A practical approach we propose consists of getting a lower bound estimate on the privacy leakage through an assessment on the local training data. To do so, the parties can train a dummy model on their own private datasets and perform a privacy assessment locally.

TABLE II: Description of attack capabilities in different threat models, for different privacy attacks. For a description of the active variant of the attacks, we refer to the corresponding works.

Attack	Class	ML-passive & crypto-passive	ML-active & crypto-passive	ML-active & crypto-active
Nasr et al. [55]	Memb. inference	The attack is not possible, since the attacker cannot pass its target data point through the model.	The attack is possible, but limited to its passive variant on the exposed layers.	Also the active variant of the attack is possible, since the attacker can perform gradient ascent.
Fredrikson et al. [25]	Model inversion	The attack is not possible, since the attacker cannot pass the dummy input through the model.	The attack is not possible if the first layer is encrypted, since the attacker cannot backpropagate over the input layer.	The attack is always possible, the malicious adversary can decrypt the first layer if necessary.
Hitaj et al. [32]	Model inversion	The attack is not possible, since the attacker cannot pass the generator output through the model.	The attack is not possible if the first layer is encrypted, since the attacker cannot backpropagate to the generator.	The attack is always possible, the malicious adversary can decrypt the first layer if necessary.
Zhu et al. [77]	Model inversion	The attack is not possible, since the attacker cannot pass the dummy input through the model.	The attack is not possible, since the attacker cannot compute the derivative of the gradients with respect to the input.	The attack is always possible, the malicious adversary can decrypt any layer if necessary.
Melis et al. [50]	Property inference	The attack is not possible, since the attacker cannot pass the target batch through the model.	The attack is possible, but limited to the exposed layers. The active variant is not possible if the last layer is encrypted.	The attack is always possible, the malicious adversary can decrypt any layer if necessary.

Instead of exploring all possible combinations of private-exposed layers, the parties can rely on the insights discussed in Section IV to determine which configurations are the most meaningful to assess. Since each local dataset is a subset of the joint dataset, the privacy leakage assessed locally provides an empirical worst-case for the privacy leakage of the joint model. From the efficiency point of view, the parties can use the insights from Section V-B, by also taking into account their specific computation and communication constraints (e.g., bandwidth and network delay between the nodes). Finally, the parties can collectively agree on which layers to encrypt by leveraging MPC techniques, avoiding leaking potential information about their local dataset. Depending on the specific situation and requirements, the parties can perform a majority vote or compute the union of the local choices to reach a consensus on the layers to be encrypted.

We provide an example of how to agree on \mathcal{L}_S for the optimized version of our solution, in case membership inference attacks are considered. Each party P_i trains a model locally and assess it against the considered privacy attack across the training epochs, as in Figure 3. Then, they select a privacy threshold $\tau_i \in [0.5, 1]$, which fixes an upper bound on the model leakage they are willing to allow. The party then proceeds to compute their preferred choice for the layers to encrypt $\mathcal{L}_S^{i,g}$ for each epoch $g = 1, \dots, E_g$ as the minimum set of layers that keeps the attack accuracy under τ_i . For consistency reasons, if a layer L_j is included in $\mathcal{L}_S^{i,g}$, then all subsequent layers L_k for $k > j$ should be included as well. Moreover, the layer should be included in all the future epochs as well, that is $L_j \in \mathcal{L}_S^{i,g'}$ for all $g' > g$. Then, the parties use MPC to compute \mathcal{L}_S^g as the union of the $\mathcal{L}_S^{i,g}$ for each $g = 1, \dots, E_g$.

F. Experimental Evaluation

In this section, we experimentally evaluate our partially encrypted model approach for different choices of \mathcal{L}_S . Our experiments show a similar run-time and communication performance of our framework for all datasets mentioned in Section IV-A. For space limitations, we report only the results for MNIST. The performance for a general-architecture MLP is discussed in Section V-B, and can be extrapolated using

the microbenchmark in Table IV. While for all datasets, the privacy evaluation is reported in Section IV.

1) *Experimental Setup*: We implement SMARTCRYPTNN³ in C++, building on top of the OpenFHE library⁴ for the multiparty CKKS functionalities. Our implementation uses CKKS with a 5-bit integral precision, 55-bit decimal precision (scaling factor), a moduli tower with 8 levels, and a cyclotomic ring degree of 2^{15} . To assess the performance of our prototype in a realistic scenario, we run the experiments within Mininet⁵, a network emulator that allows us to configure different network topologies and impose constraints on bandwidth and network delay. Different virtual hosts are spawned within a server with an Intel Xeon Platinum 8358 running at 2.60 GHz, with 64 threads on 32 cores, and 512 GB RAM. In particular, for the evaluation, we consider a setup with 3 training parties and a central server, communicating over TCP in a star topology network. The communication is constrained by 1Gbps bandwidth and 10ms network delay between the nodes.

Each party is provided with 30 examples from the MNIST dataset, and they jointly train an MLP model with two hidden layers of size 30, 20. Each layer uses sigmoid activation functions, which is approximated in $[-10, 10]$ by a polynomial of degree 13 for HE evaluation. To simplify the analysis of the trade-off given by the choice of \mathcal{L}_S , we decided to focus on the specific case of encrypting only one group of contiguous layers containing the output layer, which, from the investigation in Section IV, seems to be one of the most meaningful settings for our approach when considering inference attacks. That is, given a model f of depth L , we then have $\mathcal{L}_E = \{L_1, \dots, L_T\}$ and $\mathcal{L}_S = \{L_{T+1}, \dots, L_L\}$ for some $T \in \{0, \dots, L\}$. This way, the trade-off is controlled by the one-dimensional parameter T : when $T = 0$ we are in the extreme case of FL with full encryption of the model [62], while when $T = L$ we are in the extreme of FL in plaintext [48]. We report the results for the non-optimized version of our framework.

³<https://github.com/FedericoMazzone/SmartCryptNN>

⁴<https://github.com/openfheorg/openfhe-development>

⁵<https://github.com/mininet/mininet>

TABLE III: Execution time and communication size of our approach on the MNIST dataset for a 3-layer MLP, with varying levels of layer encryption: none ($T = 3$), last layer ($T = 2$), last two layers ($T = 1$), and full model encryption ($T = 0$).

T	Training		Inference	
	Run time (h)	Comm. size (GB)	Run time (s)	Comm. size (MB)
3	1.8e-2	3.9e-2	5.0e-3	0.0
2	726.7	980.8	23.5	10.3
1	1504.1	1988.8	50.0	23.3
0	2232.7	3083.4	76.8	36.3

2) *Performance Results*: As shown in Table III, the efficiency of our approach scales approximately linearly with T , both in terms of run time and communication size. The reported run time and communication size have been averaged among the training parties for consistency. In particular, the communication size refers to the total volume of messages received and sent per party over 300 epochs of training. The model achieves the same test accuracy as its plaintext counterpart (i.e., 48.7%). The noise from the FHE scheme and the approximation error of the activation functions do not have a significant impact on the training procedure.

In Figure 6, we observe that communication time is the dominant factor on the overall performance. Note that the

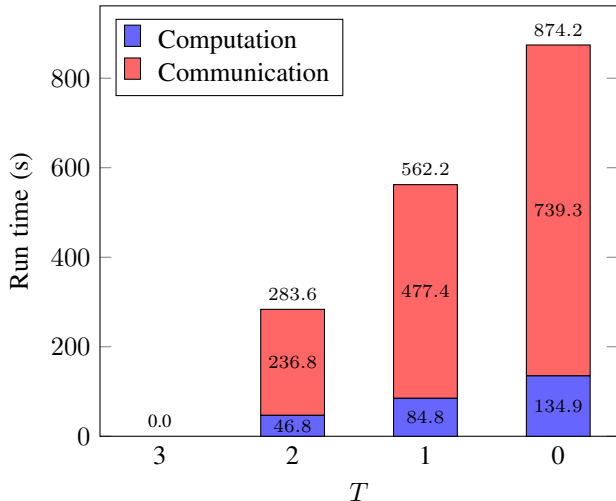


Fig. 6: Computation vs. communication time for one training pass in our approach on MNIST for a 3-layer MLP, with varying levels of encryption: none ($T = 3$), last layer ($T = 2$), last two layers ($T = 1$), and full model encryption ($T = 0$).

high communication and overall run time are partly due to our current implementation being still a prototype, and the use of relatively high-degree approximation for the activation functions. We acknowledge that there are large margins for optimization, starting from compressing the model updates before transmitting them. Also, using a tile-packing of the weight matrices as suggested in [3] may increase the performance of our approach. However, our main focus is on comparing the relative efficiency measures between different choices of encrypted layers, rather than their absolute values.

In Appendix E, we present the micro-benchmarks for most HE functionalities used in our approach.

3) *Privacy-Efficiency Trade-Off*: The more layers we encrypt, the higher the privacy, as less parameters are available to a potential adversary. However, encrypting more layers also leads to lower performance due to the overhead introduced by homomorphic evaluations and distributed bootstrapping. Thus, there is a trade-off between privacy and efficiency when deciding how many and which layers to encrypt in a model. The optimal choice depends on the specific use case, in particular on the types of attacks one wants to protect the model from, and the desired balance between privacy and performance.

In Figure 7, we represent this trade-off, using membership inference accuracy as the metric for privacy leakage. Each

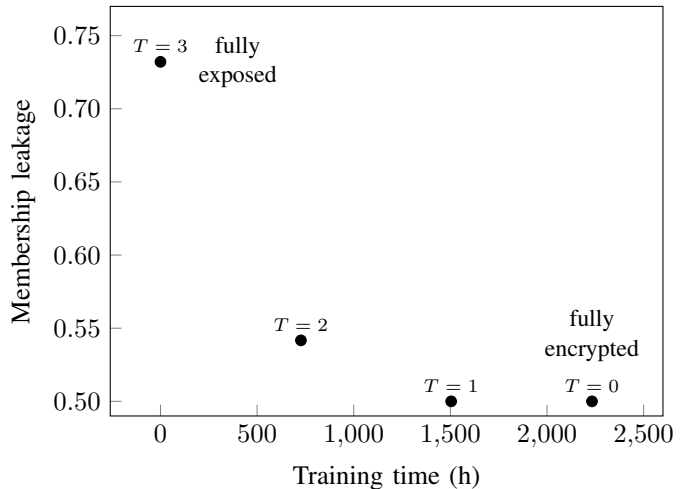


Fig. 7: Membership inference [55] on gradients, assuming attacker corrupted 2 out of 3 parties.

point on the plot represents a particular configuration of our approach, for different choices of \mathcal{L}_S . The optimum point of the trade-off occurs when both the leakage and the training time are minimized (i.e., at the origin point of the chart).

4) *Comparison with Prior Work*: The flexibility of our approach allows the user to sacrifice some privacy in order to gain training performance in terms of computation and communication time compared to fully encrypted solution like SPINDLE [26] or POSEIDON [62]. At the same time, it provides higher privacy levels than training entirely in plaintext, without compromising significant accuracy. Note that setting $T = 0$ corresponds to the original POSEIDON idea resulting in a fully encrypted model.⁶

For the very specific case presented in Figure 7, encrypting only the last layer ($T = 2$) provides a good trade-off. It offers a membership leakage very close to a random guess (54.17%, a 5.6 times reduction from the fully plaintext solution’s 73.21%, relative to the random guess baseline), while reducing the

⁶POSEIDON is not public accessible, hence we cannot directly compare their run time measures with ours. All potential optimizations are directly compatible with our construction and vice versa all our optimizations are compatible with POSEIDON.

training time with respect to the fully encrypted solution by a factor of 3.1. Assuming the adversary does not possess the target label, the leakage reduction factor increases to 17.8. While using the optimized version of our solution increases the run-time gain factor to 4.03 (by encrypting no layers till epoch 90, and only the last layer afterwards). The advantage provided by our solution may become even more evident for models with deeper architectures [42], [2], [20], particularly in settings with constrained communication networks.

VI. RELATED WORK

In addition to our overview of the literature on privacy attacks in Section IV, we provide a brief overview of related work on PPML in this section. In general, the PPML literature can be categorized into works that focus on ensuring privacy against adversaries during either the inference or training stage.

A. Privacy-Preserving Inference

In this scenario, an already trained model is typically sent to a cloud server, which provides predictions as a service on behalf of the model’s owner. The main goal is to ensure the confidentiality of the user’s query input and the corresponding output. This kind of oblivious prediction functionality can be achieved in multiple ways. One approach involves using leveled Homomorphic Encryption to encrypt the query input and perform the inference homomorphically (CryptoNets [28]). Another method involves using MPC among a cluster of non-colluding servers, possibly mixed with Garbled Circuits and HE (Gazelle [36], MiniONN [47], Chameleon [60], CryptFlow [43]). Some of those MPC solutions, by distributing the model across multiple servers, prevent the cloud servers from accessing or stealing the model. This aspect becomes particularly valuable when the model owner lacks trust in the service provider. Some line of work focuses on exploring inference on ML models in Trusted Execution Environments (TEEs), where the main challenges are ensuring performance and resiliency against side-channels attacks (Slalom [69]).

In addition, to protect against users attempting to retrieve information about the training data or reconstruct the model through a smart choice of queries (i.e., black-box attacks), various defense mechanisms have been proposed. General purpose solutions like using DP during model training (Song et al. [67], Abadi et al. [1]) have been widely studied, as well as specific defense strategies against certain attacks. For instance, possible strategies against inference attacks include perturbing the prediction vector with noise (Memguard [34]), masking the output confidence score by only revealing the top-k scores or just the prediction label (Shokri et al. [66], Choquette-Choo et al. [18], Li et al. [46]) employing adversarial regularization by jointly minimizing the classification loss and maximizing a theoretical attack model’s loss (Nasr et al. [54]), using knowledge distillation to put distance between the private dataset and the deployed model (PATE [56], Shejwalkar et al. [63]), or simply applying standard regularization techniques (Shokri et al. [66]). Furthermore, other approaches have been developed to prevent model stealing, such as PRADA [35], which checks for structured patterns in user queries, or like PrivDNN [59], which encrypts the individual neurons that contribute the most to the model’s utility. It is important to note, however, that

while the last two approaches enhance model privacy, they do not inherently safeguard the confidentiality of the training data.

B. Privacy-Preserving Training

To introduce adversaries during the training stage, we need to consider a collaborative learning setting. In this setting, adversaries may corrupt one or more training clients and potentially the supporting server, making it possible to run privacy attacks on the intermediate model updates. Differential privacy can be employed to inject noise during the federated training process, providing privacy guarantees at either the data level (Shokri and Shmatikov [65]) or the user level (McMahan et al.[49]). Since the supporting server could also be compromised, some lines of work have focused on concealing individual model updates by performing secure aggregation of such values. This is achieved using techniques such as secret sharing (SEPIA[12]), MHE (Shi et al.[64], Chan et al.[14]), or additive masking (Bonawitz et al.[9], Bell et al.[7]). Note that secure aggregation is orthogonal to our approach and can potentially be combined with it to conceal the individual updates corresponding to the exposed layers.

To completely prevent leakage from intermediate (aggregated) models, MPC can be employed, allowing the data owners to jointly execute the training mechanism in a secure manner, usually by exploiting secret sharing schemes. However, a major challenge arises when scaling to a large number of parties, as it leads to impractical communication complexity. To work around such overhead, the data-owners can delegate the computations to a small cluster of non-colluding servers, usually composed of 2 parties (SecureML [52]), 3 parties (ABY³ [51], Falcon [72], SecureNN [71]), or 4 parties (FLASH [13], Trident [15]). However, this delegation-based approach imposes strong assumptions on the non-collusion of the computing servers, strongly constraining the threat model.

To overcome the limitations of small cluster MPC solutions to the threat model, a promising research direction has emerged, leveraging FHE schemes to encrypt the model. By employing FHE, the federated learning process can be conducted entirely under encryption, enabling secure collaboration among a large number of parties (SPINDLE [26] for generalized linear models, POSEIDON [62] for neural networks). Our solution aligns with this trajectory and can be seen as a generalization of these approaches, with the primary aim of enhancing the efficiency of FL under FHE and making its use feasible in real-world scenarios.

VII. CONCLUSION AND FUTURE WORK

In this paper, we present SMARTCRYPTNN, a flexible solution for privacy-preserving training of neural networks in a federated setting. Our system allows users to trade-off little privacy for higher training performance, by selectively encrypting specific portions of the model using a multiparty FHE scheme. Through an investigation of various privacy attacks in the gray-box setting, where the adversary’s access is limited to the unencrypted layers of the model, we determine the layers that tend to leak more information and, consequently, identify which layers are advisable to encrypt. Our findings indicate that encrypting the last layers is particularly effective to mitigate membership inference attacks, while encrypting

the first layers helps preventing model inversion attacks. In the future, we plan to expand our investigation to include other classes of machine learning models. Additionally, we aim to enhance our framework to support various optimizers, loss functions, and layer types to make it more applicable to different datasets. Finally, we believe that our collaborative learning solution could be further optimized performance-wise by adding other MPC techniques to the toolset, for instance to compute the non-linear activation functions of the model.

ACKNOWLEDGMENT

This project has received funding from the European Union’s Horizon 2020 research and innovation programme under Grant Agreement No 965315. This result reflects only the author’s view and the European Commission is not responsible for any use that may be made of the information it contains.

REFERENCES

- [1] 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 SIGSAC conference on computer and communications security*, 2016.
- [2] A. F. Agarap, “Training deep neural networks for image classification in a homogenous distributed system,” 2019.
- [3] E. Aharoni, A. Adir, M. Baruch, N. Drucker, G. Ezov, A. Farkash, L. Greenberg, R. Masalha, G. Moshkovich, D. Murik *et al.*, “Helayers: A tile tensors framework for large neural networks on encrypted data,” *Proceedings on Privacy Enhancing Technologies*, vol. 1, no. 1, pp. 325–342, 2023.
- [4] A. Al Badawi, J. Bates, F. Bergamaschi, D. B. Cousins, S. Erabelli, N. Genise, S. Halevi, H. Hunt, A. Kim, Y. Lee, Z. Liu, D. Micciancio, I. Quah, Y. Polyakov, S. R.V., K. Rohloff, J. Saylor, D. Suponitsky, M. Triplett, V. Vaikuntanathan, and V. Zucca, “OpenFHE: Open-source fully homomorphic encryption library,” in *Proceedings of the 10th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, ser. WAHC’22. New York, NY, USA: Association for Computing Machinery, 2022, pp. 53–63. [Online]. Available: <https://doi.org/10.1145/3560827.3563379>
- [5] G. Asharov, A. Jain, A. López-Alt, E. Tromer, V. Vaikuntanathan, and D. Wichs, “Multiparty computation with low communication, computation and interaction via threshold fhe,” in *Advances in Cryptology–EUROCRYPT: 3Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2012.
- [6] A. Beimel, H. Brenner, S. P. Kasiviswanathan, and K. Nissim, “Bounds on the sample complexity for private learning and private data release,” *Machine learning*, vol. 94, pp. 401–437, 2014.
- [7] J. H. Bell, K. A. Bonawitz, A. Gascón, T. Lepoint, and M. Raykova, “Secure single-server aggregation with (poly) logarithmic overhead,” in *Proceedings of the SIGSAC Conference on Computer and Communications Security*, 2020.
- [8] M. Blatt, A. Gusev, Y. Polyakov, K. Rohloff, and V. Vaikuntanathan, “Optimized homomorphic encryption solution for secure genome-wide association studies,” *BMC Med Genomics*, vol. 13, no. Suppl 7, p. 83, Jul 2020.
- [9] 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,” in *proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1175–1191.
- [10] Z. Brakerski, “Fully homomorphic encryption without modulus switching from classical GapSVP,” in *Annual Cryptology Conference*. Springer, 2012, pp. 868–886.
- [11] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, “(leveled) fully homomorphic encryption without bootstrapping,” *ACM Trans. Comput. Theory*, vol. 6, no. 3, jul 2014. [Online]. Available: <https://doi.org/10.1145/2633600>
- [12] M. Burkhart, M. Strasser, D. Many, and X. Dimitropoulos, “SEPIA: Privacy-preserving aggregation of multi-domain network events and statistics,” in *USENIX Security Symposium*, 2010.
- [13] M. Byali, H. Chaudhari, A. Patra, and A. Suresh, “Flash: Fast and robust framework for privacy-preserving machine learning,” *Cryptology ePrint Archive*, 2019.
- [14] T. H. H. Chan, E. Shi, and D. Song, “Privacy-preserving stream aggregation with fault tolerance,” in *Financial Cryptography and Data Security: 16th International Conference, FC 2012, Kralendijk, Bonaire, February 27-March 2, 2012, Revised Selected Papers 16*. Springer, 2012, pp. 200–214.
- [15] H. Chaudhari, R. Rachuri, and A. Suresh, “Trident: Efficient 4pc framework for privacy preserving machine learning,” *arXiv preprint arXiv:1912.02631*, 2019.
- [16] H. Chen, I. Chillotti, and Y. Song, “Improved bootstrapping for approximate homomorphic encryption,” in *Advances in Cryptology – EUROCRYPT 2019*, Y. Ishai and V. Rijmen, Eds. Cham: Springer International Publishing, 2019, pp. 34–54.
- [17] J. H. Cheon, A. Kim, M. Kim, and Y. Song, “Homomorphic encryption for arithmetic of approximate numbers,” in *Advances in Cryptology–ASIACRYPT: International Conference on the Theory and Applications of Cryptology and Information Security*. Springer, 2017.
- [18] C. A. Choquette-Choo, F. Tramer, N. Carlini, and N. Papernot, “Label-only membership inference attacks,” in *International conference on machine learning*. PMLR, 2021, pp. 1964–1974.
- [19] G. Cohen, S. Afshar, J. Tapson, and A. Van Schaik, “Emnist: Extending mnist to handwritten letters,” in *2017 international joint conference on neural networks (IJCNN)*. IEEE, 2017, pp. 2921–2926.
- [20] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly *et al.*, “An image is worth 16x16 words: Transformers for image recognition at scale,” *arXiv preprint arXiv:2010.11929*, 2020.
- [21] J. Duchi, E. Hazan, and Y. Singer, “Adaptive subgradient methods for online learning and stochastic optimization,” *Journal of machine learning research*, vol. 12, no. 7, 2011.
- [22] C. Dwork, F. McSherry, K. Nissim, and A. Smith, “Calibrating noise to sensitivity in private data analysis,” in *Theory of Cryptography*. Springer, 2006.
- [23] C. Dwork, A. Roth *et al.*, “The algorithmic foundations of differential privacy,” *Foundations and Trends® in Theoretical Computer Science*, vol. 9, no. 3–4, pp. 211–407, 2014.
- [24] J. Fan and F. Vercauteren, “Somewhat practical fully homomorphic encryption,” *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, 2012.
- [25] 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*, 2015, pp. 1322–1333.
- [26] D. Froelicher, J. R. Troncoso-Pastoriza, A. Pyrgelis, S. Sav, J. S. Sousa, J.-P. Bossuat, and J.-P. Hubaux, “Scalable privacy-preserving distributed learning,” *Proceedings on Privacy Enhancing Technologies*, 2021.
- [27] K. Ganju, Q. Wang, W. Yang, C. A. Gunter, and N. Borisov, “Property inference attacks on fully connected neural networks using permutation invariant representations,” in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 619–633.
- [28] 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 *International conference on machine learning*. PMLR, 2016, pp. 201–210.
- [29] K. Han, S. Hong, J. H. Cheon, and D. Park, “Efficient logistic regression on large encrypted data,” *Cryptology ePrint Archive*, Paper 2018/662, 2018, <https://eprint.iacr.org/2018/662>. [Online]. Available: <https://eprint.iacr.org/2018/662>
- [30] M. Hardt and G. N. Rothblum, “A multiplicative weights mechanism for privacy-preserving data analysis,” in *2010 IEEE 51st annual symposium on foundations of computer science*. IEEE, 2010, pp. 61–70.
- [31] B. Hilprecht, M. Härterich, and D. Bernau, “Monte carlo and reconstruction membership inference attacks against generative models,” in *Proceedings on Privacy Enhancing Technologies*, 2019.

- [32] 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.
- [33] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, "Labeled faces in the wild: A database for studying face recognition in unconstrained environments," in *Workshop on faces in 'Real-Life' Images: detection, alignment, and recognition*, 2008.
- [34] J. Jia, A. Salem, M. Backes, Y. Zhang, and N. Z. Gong, "Memguard: Defending against black-box membership inference attacks via adversarial examples," in *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, 2019, pp. 259–274.
- [35] M. Juuti, S. Szyller, S. Marchal, and N. Asokan, "Prada: protecting against dnn model stealing attacks," in *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2019, pp. 512–527.
- [36] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan, "{GAZELLE}: A low latency framework for secure neural network inference," in *USENIX Security Symposium*, 2018.
- [37] S. P. Kasiviswanathan, H. K. Lee, K. Nissim, S. Raskhodnikova, and A. Smith, "What can we learn privately?" *SIAM Journal on Computing*, vol. 40, no. 3, pp. 793–826, 2011.
- [38] A. Kim, A. Papadimitriou, and Y. Polyakov, "Approximate homomorphic encryption with reduced approximation error," in *Cryptographers' Track at the RSA Conference*. Springer, 2022.
- [39] A. Kim, Y. Song, M. Kim, K. Lee, and J. H. Cheon, "Logistic regression model training based on the approximate homomorphic encryption," *BMC medical genomics*, vol. 11, no. 4, pp. 23–31, 2018.
- [40] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.
- [41] K. Kluczniak and G. Santato, "On circuit private, multikey and threshold approximate homomorphic encryption," *Cryptology ePrint Archive*, Paper 2023/301, 2023, <https://eprint.iacr.org/2023/301>. [Online]. Available: <https://eprint.iacr.org/2023/301>
- [42] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in neural information processing systems*, vol. 25, 2012.
- [43] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma, "Cryptflow: Secure tensorflow inference," in *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2020, pp. 336–353.
- [44] B. Li, D. Micciancio, M. Schultz, and J. Sorrell, "Securing approximate homomorphic encryption using differential privacy," in *Advances in Cryptology – CRYPTO 2022*, Y. Dodis and T. Shrimpton, Eds. Cham: Springer Nature Switzerland, 2022, pp. 560–589.
- [45] W. Li, F. Milletari, D. Xu, N. Rieke, J. Hancox, W. Zhu, M. Baust, Y. Cheng, S. Ourselin, M. J. Cardoso *et al.*, "Privacy-preserving federated brain tumour segmentation," in *Machine Learning in Medical Imaging: 10th International Workshop, MLMI 2019, Held in Conjunction with MICCAI 2019, Shenzhen, China, October 13, 2019, Proceedings 10*. Springer, 2019, pp. 133–141.
- [46] Z. Li and Y. Zhang, "Membership leakage in label-only exposures," in *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 880–895.
- [47] J. Liu, M. Juuti, Y. Lu, and N. Asokan, "Oblivious neural network predictions via minionn transformations," in *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, 2017, pp. 619–631.
- [48] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas, "Communication-efficient learning of deep networks from decentralized data," in *Artificial intelligence and statistics*. PMLR, 2017, pp. 1273–1282.
- [49] H. B. McMahan, D. Ramage, K. Talwar, and L. Zhang, "Learning differentially private recurrent language models," *arXiv preprint arXiv:1710.06963*, 2017.
- [50] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *Symposium on security and privacy (SP)*. IEEE, 2019.
- [51] P. Mohassel and P. Rindal, "ABY3: A mixed protocol framework for machine learning," in *Proceedings of the SIGSAC conference on computer and communications security*, 2018.
- [52] P. Mohassel and Y. Zhang, "Secureml: A system for scalable privacy-preserving machine learning," in *Symposium on security and privacy (SP)*. IEEE, 2017.
- [53] C. Mouchet, J. Troncoso-Pastoriza, J.-P. Bossuat, and J.-P. Hubaux, "Multipart homomorphic encryption from ring-learning-with-errors," *Proceedings on Privacy Enhancing Technologies*, no. CONF, 2021.
- [54] M. Nasr, R. Shokri, and A. Houmansadr, "Machine learning with membership privacy using adversarial regularization," in *Proceedings of the 2018 ACM SIGSAC conference on computer and communications security*, 2018, pp. 634–646.
- [55] —, "Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning," in *Symposium on security and privacy (SP)*. IEEE, 2019, https://github.com/privacytrustlab/ml_privacy_meter, accessed 20-January-2022.
- [56] N. Papernot, M. Abadi, U. Erlingsson, I. Goodfellow, and K. Talwar, "Semi-supervised knowledge transfer for deep learning from private training data," *arXiv preprint arXiv:1610.05755*, 2016.
- [57] 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*, ser. CCS '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 2113–2129. [Online]. Available: <https://doi.org/10.1145/3460120.3485259>
- [58] M. G. Poirot, P. Vepakomma, K. Chang, J. Kalpathy-Cramer, R. Gupta, and R. Raskar, "Split learning for collaborative deep learning in healthcare," *arXiv preprint arXiv:1912.12115*, 2019.
- [59] L. Ren, Z. Liu, F. Li, K. Liang, Z. Li, and B. Luo, "Privdnn: A secure multi-party computation framework for deep learning using partial dnn encryption," *Proceedings on Privacy Enhancing Technologies*, vol. 3, pp. 1–18, 2024.
- [60] M. S. Riazi, C. Weinert, O. Tkachenko, E. M. Songhori, T. Schneider, and F. Koushanfar, "Chameleon: A hybrid secure computation framework for machine learning applications," in *Proceedings of the 2018 on Asia conference on computer and communications security*, 2018, pp. 707–721.
- [61] M. Rigaki and S. Garcia, "A survey of privacy attacks in machine learning," *arXiv preprint arXiv:2007.07646*, 2020.
- [62] S. Sav, A. Pyrgelis, J. R. Troncoso-Pastoriza, D. Froelicher, J.-P. Bossuat, J. S. Sousa, and J.-P. Hubaux, "Poseidon: Privacy-preserving federated neural network learning," in *Network And Distributed System Security Symposium*. The Internet Society, 2021.
- [63] V. Shejwalkar and A. Houmansadr, "Membership privacy for machine learning models through knowledge transfer," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 35, no. 11, 2021, pp. 9549–9557.
- [64] E. Shi, H. Chan, E. Rieffel, R. Chow, and D. Song, "Privacy-preserving aggregation of time-series data," in *Annual Network & Distributed System Security Symposium (NDSS)*. Internet Society., 2011.
- [65] R. Shokri and V. Shmatikov, "Privacy-preserving deep learning," in *Proceedings of the SIGSAC conference on computer and communications security*, 2015.
- [66] R. Shokri, M. Stronati, C. Song, and V. Shmatikov, "Membership inference attacks against machine learning models," in *Symposium on security and privacy (SP)*. IEEE, 2017.
- [67] S. Song, K. Chaudhuri, and A. D. Sarwate, "Stochastic gradient descent with differentially private updates," in *2013 IEEE global conference on signal and information processing*. IEEE, 2013, pp. 245–248.
- [68] T. Tieleman, G. Hinton *et al.*, "Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude," *COURSERA: Neural networks for machine learning*, vol. 4, no. 2, pp. 26–31, 2012.
- [69] F. Tramer and D. Boneh, "Slalom: Fast, verifiable and private execution of neural networks in trusted hardware," *arXiv preprint arXiv:1806.03287*, 2018.
- [70] S. Truex, L. Liu, K.-H. Chow, M. E. Gursoy, and W. Wei, "Ldp-fed: Federated learning with local differential privacy," in *Proceedings of the Third ACM International Workshop on Edge Systems, Analytics and Networking*, 2020, pp. 61–66.
- [71] S. Wagh, D. Gupta, and N. Chandran, "Securenn: 3-party secure

computation for neural network training.” *Proceedings on Privacy Enhancing Technologies*, 2019.

- [72] S. Wagh, S. Tople, F. Benhamouda, E. Kushilevitz, P. Mittal, and T. Rabin, “Falcon: Honest-majority maliciously secure framework for private deep learning,” *arXiv preprint arXiv:2004.02229*, 2020.
- [73] Z. Wang, M. Song, Z. Zhang, Y. Song, Q. Wang, and H. Qi, “Beyond inferring class representatives: User-level privacy leakage from federated learning,” in *IEEE INFOCOM 2019-IEEE conference on computer communications*. IEEE, 2019, pp. 2512–2520.
- [74] S. Yeom, I. Giacomelli, M. Fredrikson, and S. Jha, “Privacy risk in machine learning: Analyzing the connection to overfitting,” in *Computer security foundations symposium (CSF)*. IEEE, 2018.
- [75] M. D. Zeiler and R. Fergus, “Visualizing and understanding convolutional networks,” in *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6–12, 2014, Proceedings, Part I 13*. Springer, 2014, pp. 818–833.
- [76] Y. Zhang, R. Jia, H. Pei, W. Wang, B. Li, and D. Song, “The secret revealer: Generative model-inversion attacks against deep neural networks,” in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 253–261.
- [77] L. Zhu, Z. Liu, and S. Han, “Deep leakage from gradients,” *Advances in neural information processing systems*, vol. 32, 2019.
- [78] X. Zhu, C. Vondrick, D. Ramanan, and C. C. Fowlkes, “Do we need more training data or better models for object detection?,” in *BMVC*, vol. 3, no. 5. Citeseer, 2012.

APPENDIX

A. MLPs and Gradient Descent

Multilayer Perceptrons (MLPs), also known as fully-connected or dense networks, are the simplest kind of feed-forward neural networks, where each neuron in one layer is connected to every neuron in the next layer. Due to their structure, the parameters of these models can be represented by matrices, and in some cases, an additive bias parameter is included for added flexibility. Given an MLP with L layers and $i \in \{1, \dots, L\}$, we denote by w_i and b_i the weight matrix and bias vector between layer $i - 1$ and layer i , respectively. We denote by l_i the output of layer i , that is $l_i = \phi_i(l_{i-1}w_i + b_i)$, where $l_0 := x$ and ϕ_i is the so-called activation function used to incorporate non-linearity in the model. And we denote the intermediate linear application output as $u_i = l_{i-1}w_i + b_i$. With a little abuse of notation, we will sometimes refer to the weight and bias w_i, b_i as to the parameters of the layer they allow to transition to, namely layer i . The model is then a parameterized function $f(x; w_i, b_i)$, whose output is l_L .

The model is trained by minimizing the empirical risk with respect to a given loss function. For supervised learning, we assume to have a training dataset D of labeled examples (x, y) , where x is the feature vector and y is the ground-truth label. Given a model $f(x; w_i, b_i)$, the goal is to optimize the model’s parameters by minimizing some loss function $L(f(x), y)$. To estimate the gradient of L with respect to the model parameters w_i, b_i , feedforward and backpropagation are used. During feedforward, an input data x is propagated layer by layer through the network, computing all the u_i and l_i . The output prediction l_L is then compared to the actual label y using the chosen loss function L to compute the loss value. The backpropagation algorithm then calculates the gradients of the loss function with respect to the model’s parameters. When the loss function L , the model f , and the example (x, y) are clear from the context, we will write ∇w_i and ∇b_i in place of $\nabla_{w_i} L(f(x), y)$ and $\nabla_{b_i} L(f(x), y)$, respectively.

This step is repeated for a batch of examples B , and the resulting gradients are averaged to get a better approximation of the actual loss gradient on the real population. The parameters are then updated by following the negative direction of the gradient, by a step size proportional to a learning rate $\eta > 0$:

$$w_i \leftarrow w_i - \frac{\eta}{|B|} \sum_{(x,y) \in B} \nabla_{w_i} L(f(x), y),$$

$$b_i \leftarrow b_i - \frac{\eta}{|B|} \sum_{(x,y) \in B} \nabla_{b_i} L(f(x), y).$$

This iterative process of feeding the data forward, computing the loss, and updating the model’s parameters continues until convergence or for a fixed number of iterations.

B. Dataset Description

In the following, we give a more detailed description of the datasets we used for our experiments.

AT&T Database of Faces: This face dataset ⁷ was created at the AT&T Laboratories Cambridge. It consists of 400 gray-scale images of size 112x92, depicting the faces of 40 individuals in various lighting conditions and facial expressions.

EMNIST Letters: This letter dataset ⁸ is part of the extended version of the MNIST dataset by NIST [19]. It consists of 145,600 gray-scale images, representing both upper- and lower-case handwritten letters, which has been centered and resized to 28x28. The dataset contains 26 classes, one for each letter from ‘a’ to ‘z’.

Labeled Faces In the Wild (LFW): This face dataset ⁹ was developed by researchers at the University of Massachusetts, Amherst [33]. It consists of 13,233 RGB images, depicting the faces of 5,749 individuals. The dataset has been further labeled with attributes such as gender, race, age group, hair style, and eyewear.

Locations: This location dataset ¹⁰ was created by the authors of [66] from Foursquare check-in data for the city of Bangkok. The processed dataset contains 5010 examples, each corresponding to a unique user. Each record comprises 446 binary features, indicating whether a user visited a specific region or location type. The data is clustered into 30 classes, representing different geosocial types. Following [66], we use 1200 examples for training, and the remaining data for validation.

MNIST: Standard handwritten digits dataset by NIST ¹¹. It consists of 70,000 gray-scale images, centered, and resized to 28x28. The dataset contains 10 classes, one for each digit from ‘0’ to ‘9’. Due to the small number of classes and the low feature variability within the same class, this dataset has been observed to be particularly resilient against membership inference attacks [66]. For evaluation purposes, we want to start from a situation in which the target model is vulnerable. Thus, we drastically reduce the training set to a mere 100 examples. For compatibility with our current implementation of SMARTCRYPTNN, we resize the images to 8x8.

⁷<https://cam-orl.co.uk/facedatabase.html>

⁸<https://www.nist.gov/itl/products-and-services/emnist-dataset>

⁹<http://vis-www.cs.umass.edu/lfw/>

¹⁰<https://github.com/privacytrustlab/datasets>

¹¹<http://yann.lecun.com/exdb/mnist>

Purchase-100: This purchase dataset¹⁰ was created by the authors of [66] starting from Kaggle’s “acquire valued shoppers” challenge dataset, containing the shopping history data of several users. The processed dataset contains 197,324 examples, each corresponding to a unique user. Each record comprises 600 binary features, indicating whether a user purchased a given product. The data is clustered into 100 classes, representing different purchase styles. For our experiments, we use 1000 examples for training and the remaining data for validation.

Texas-100: This hospital dataset¹⁰ was created by the authors of [66] starting from the Hospital Discharge Data records released by the Texas Department of State Health Services. The processed dataset contains 67,330 examples, each corresponding to a unique patient. Each record comprises 6,169 binary features, containing information about the patient, the causes of injury, the diagnosis, and the procedures the patient underwent. The data is clustered into 100 classes, representing the 100 most frequent medical procedures present. For our experiments, we use 1000 examples for training and the remaining data for validation.

C. Threshold CKKS

Threshold CKKS supports the same operations as the regular single-key CKKS (see Section 2 of [38] for details). The operations that are different in the threshold CKKS are public key generation, evaluation key generation (for rotations and/or multiplication), decryption, and bootstrapping.

The public key generation is done in a distributed manner, where each party generates a public key share (a public key for its secret share) using a common random polynomial, and then all public key shares are summed up to generate the collective public key. The generation of a collective automorphism (rotation) key is done in the same manner because automorphism is a linear operation. The only difference is that multiple common random polynomials may be needed (one for each digit of each automorphism key). For more details, see [53].

The generation of the relinearization (multiplication) key is more involved: it requires extra rounds as the encrypted key is a square of the original secret key, and the evaluation of quadratic function has to be done in two steps, as illustrated in Protocol 2 of [53].

The decryption is also done in a distributed manner. Each party obtains a partial decryption by evaluating the inner product with respect to their secret share. The partial decryption results are summed up to yield the decryption results. To hide the secret share used for partial decryption from other parties that may see the partial decryption, smudging/flooding noise has to be added during the evaluation of partial decryption [5]. This noise has to be significantly larger than the current approximation noise in CKKS to erase any traces of the secret share. Klucznik and Santato [41] suggested tight flooding noise estimates for threshold FHE based on the prior estimates for the flooding noise required in the context of approximate homomorphic encryption [44]. Note that these estimates are higher than those for achieving IND-CPA^D security for CKKS [44]; hence it is sufficient to only consider threshold FHE flooding noise in the case of threshold CKKS. OpenFHE

provides a way to configure the flooding noise based on the statistical security parameters specified by the user.

The distributed CKKS bootstrapping procedure performs masked partial decryptions using secret shares and then adds an encryption of the negated mask using large parameters, i.e., Q_L , to generate a refreshed encryption of the message. The mask requires additional 2-3 RNS limbs to achieve desired statistical security. The procedure is described in more detail in [62].

D. Homomorphic Operations

To perform homomorphic computations efficiently, we can pack an entire vector in a single ciphertext and exploit the SIMD capabilities of CKKS to perform vector addition and component-wise multiplication in constant time.¹² However, in a neural network, we also need to multiply by weight matrices and evaluate non-linear activation functions. Here, we describe the approaches we adopt to perform efficient vector-matrix multiplication under encryption and homomorphically evaluate non-polynomial functions.

1) *Matrix Multiplication*: To efficiently perform matrix multiplication under encryption, the idea is to encode the matrix as a vector in such a way that only one homomorphic multiplication is required. There exist multiple encoding schemes in the literature for matrices. For instance, in the column-based approach [39], [29], one encodes a matrix by concatenating its columns one after the other. The vector-matrix multiplication is then performed by first replicating the vector to match the number of columns of the matrix, then performing a SIMD multiplication between the two, and finally by performing cumulative addition of the result. All those operations can be realized by combining homomorphic additions, rotations, and multiplications by a masking vector. Moreover, the vector replication and the cumulative addition can be made more efficient by recursion, though requiring padding the inputs to a suitable power of two. Similarly to this column-based approach, a row-based encoding can be employed as well [8].

To perform subsequent matrix multiplications, we use the alternating packing approach proposed in [62]. It is based on the observation that the result of a vector-matrix multiplication in column-based encoding requires extra rotations to prepare it for a multiplication with another column-based encoded matrix, while it is perfectly ready for a multiplication with another row-based encoded matrix. The idea is then to encode the matrices that are in consecutive secret layers by alternating between column- and row-based encodings. Note that multiplying by the transpose of a matrix is equivalent to multiplying by the matrix in the opposite encoding. We exploit this property to efficiently compute gradients in the back-propagation step.

2) *Evaluating Non-Polynomial Functions*: We use the Chebyshev interpolation to homomorphically evaluate non-linear functions for a given input range, which is implemented in OpenFHE for the CKKS scheme [4]. The evaluation of Chebyshev series in OpenFHE is performed using

¹²If the vector is too long to fit the number of slots dictated by given FHE parameters, one can split the vector among multiple ciphertexts.

the Paterson-Stockmeyer algorithm adapted to Chebyshev basis [16], which requires only $O(\sqrt{d})$ homomorphic multiplications to evaluate degree- d polynomials.

layers. This error would propagate to the exposed layers during backpropagation, producing a DP-like effect.

E. FHE Micro-benchmark

In this section, we provide measurements for various FHE functionalities, enabling estimation of our approach’s scalability for different model architectures. The execution run time and communication size per training party, averaged over multiple runs, are presented in Table IV. For each functionality, we have divided the execution time into computation and communication time. Note that the missing time from the total execution time reflects the idle time when parties are waiting for other parties to complete their computations in order to proceed.

F. Differential Privacy for Exposed Layers

To mitigate the leakage from the exposed layers additional privacy-enhancing techniques can be employed, such as differential privacy [22]. Applying DP to the parameters or gradients of exposed layers would provide a theoretical privacy guarantee to our solution, albeit introducing an additional trade-off between privacy and accuracy. By applying noise only on the exposed layers, rather than the entire model, our approach can achieve a higher level of privacy for the same privacy budget compared to standard FL solutions with differential privacy [65], [49], [45], [70].

We adapt the approach of Shokri and Shmatikov [65], which uses the sparse vector technique [23], [30] to privately upload a small, perturbed subset of the gradients to the global model. Given a privacy budget ϵ per epoch allocated to each training party P_i , we split this budget among the exposed parameters, and use Laplacian mechanism to add noise to the corresponding gradient value. The sensitivity of the training mechanism is estimated by clipping the gradient values within the range $[-\gamma, \gamma]$, resulting in a sensitivity of 2γ . The clipping range value should be independent of the specific training dataset, to avoid leaking sensitive information. We suggest setting it by calculating the median of the unclipped gradients over the course of training, as proposed in [1].

For each value g in ∇w_j for $L_j \in \mathcal{L}_E$, random Laplacian noise $r_g \sim \text{Lap}(2c\gamma_g/\epsilon)$ is generated, where γ_g is the estimated clipping bound for g , and c is the total number of exposed gradients $c = \sum_{L_j \in \mathcal{L}_E} (|\nabla w_j| + |\nabla b_j|)$. The gradient g is then clipped within $[-\gamma_g, \gamma_g]$, and the noise r_g is added before uploading to the central server. A similar process can be followed for the exposed bias gradients. These operations are performed on the aggregated gradient obtained after several local iterations, each computed over a randomly sampled batch. Applying noise to each computed gradients could be done as well, and the overall effect over batches can be analyzed using the privacy amplification theorem [37], [6]. More advanced techniques, such as privacy accountants [49], [1], can also be potentially adapted to work in our partially encrypted model solution.

As additional remark, differential privacy could also be achieved by reducing the precision of the underlying FHE scheme, thus allowing for larger encryption error on the secret

TABLE IV: Microbenchmarks of different FHE functionalities, for $N = 3$ parties, 5-bit integral precision, 55-bit decimal precision, 8-level moduli tower, and 2^{15} cyclotomic ring degree. The *one layer* functionality refers to a fully connected layer, while the *one pass* functionality refers to an entire pass (forward or backward) of the model under encryption.

Functionality	Execution time (s)			Comm. size (MB)
	Comp.	Comm.	Total	
Vector-Matrix mult. (64x32)	6.289	-	6.289	-
Vector-Matrix mult. (32x16)	3.291	-	3.291	-
Sigmoid evaluation (deg. 13)	11.046	-	11.046	-
FHE setup	0.043	0.020	0.143	0.001
Pub./Priv. key gen.	0.151	0.347	0.611	12.005
Relin. key gen.	0.900	0.788	2.336	81.020
Rot. key gen. (x23)	4.378	86.458	101.662	1296.182
Model generation	0.989	-	0.989	-
Decryption	0.376	34.090	34.999	15.013
Bootstrapping	1.265	49.287	51.160	22.767
One layer forward	20.294	147.737	170.125	68.302
One layer backward	24.617	98.699	124.262	45.535
One pass forward	60.944	443.211	510.438	204.907
One pass backward	73.913	296.096	372.849	136.605
One training pass	134.857	739.307	883.287	341.512
Send model	0.157	17.674	17.831	27.015
Update params.	2.039	-	2.039	-
Aggregation	1.697	56.454	60.265	37.532