

A Privacy-Preserving Federated Learning Protocol for Multilayer Perceptrons

Ceren Yıldırım(✉), Kamer Kaya, and Erkay Savaş

Sabancı University, Istanbul, Turkey
{cerenyildirim,kaya,erkays}@sabanciuniv.edu

Abstract. Federated learning is a machine learning technique that allows multiple distributed clients to collaboratively train an ML model without sharing their private data with any of the parties involved. However, ensuring the privacy of client data during the FL process remains an ongoing concern. In this study, we propose a homomorphic-encryption-based privacy-preserving FL protocol for multilayer perceptrons, which is shown to be secure under the presence of colluding honest-but-curious clients. The possibility of client collusion attacks is eliminated by utilizing the inherent permutability of neural networks. Our results indicate that our protocol does not incur any considerable loss in accuracy during the training process. Furthermore, it offers minimal computation costs by utilizing the batching technique of homomorphic operation and employing only the inexpensive homomorphic addition operation for the aggregation process.

Keywords: Federated Learning · Multilayer Perceptron · Homomorphic Encryption

1 Introduction

With the increase of distributed data sources and the fragmented nature of data, federated learning (FL) has attracted considerable attention in recent years. FL is a distributed approach for developing machine learning (ML) models across multiple parties while ensuring each party's confidential training data remains decentralized. For this reason, it has drawn significant attention across various fields, including healthcare, transportation, and finance [10,13,22]. This technique enables access to abundant data sources and utilizes them to improve and enhance ML models. However, during the process of FL, ensuring the privacy and security of the data involved in training remains a challenge. Even if the data itself is not directly shared with anyone, the model parameters shared with the involved parties might inadvertently leak sensitive information about clients' confidential data. Therefore, securing the parameters of the clients' local models becomes a crucial concern. In order to overcome such privacy concerns, several

This chapter was originally published under a non-OA model but is now published under the Open Access license CC BY-NC-ND 4.0. The copyright holder has been updated.

© The Author(s) 2025
M. Mahmud et al. (Eds.): AII 2024, CCIS 2607, pp. 504–519, 2025.
https://doi.org/10.1007/978-3-032-04657-4_34

privacy-preserving measures have been proposed, which include using differential privacy (DP), homomorphic encryption (HE), secure multi-party computation (SMPC), or trusted execution environment (TEE) [16].

In this work, we propose a privacy-preserving FL application for multilayer perceptrons, using HE to ensure the privacy of the local information. Our protocol assumes a single-server setting, where multiple clients who own local ML models, trained on their local data, are involved. The clients aim to generate a more generalizable global model with better classification performance by aggregating their local models, which is achieved by the server. For this purpose, our protocol uses the Federated Averaging (FedAvg) [18] algorithm. During the process, neither the clients nor the server should be able to learn any information about the other clients' local models or data.

By leveraging threshold fully HE (t-FHE) [2] on top of the Cheon-Kim-Kim-Song (CKKS) encryption scheme [4], any information coming from the local model of the clients is encrypted, and the aggregation is achieved homomorphically. The aggregated model parameters can only be decrypted in the presence of a coalition of clients. However, while using HE alone offers a degree of security, in the existence of colluding clients, the local model parameters of honest clients can be disclosed to semi-honest clients. Similarly, in a setting where only two clients are involved, each client can easily reverse the linear averaging operation to disclose the other client's model information. To eliminate the probability of these collusion attacks, we propose a remediation technique that exploits the inherent permutability of neural networks. After the aggregation, the server creates an equivalent model whose output is the same as the global model, by permuting the weights of the global model. This equivalent model is sent back to the clients to be decrypted and used again in the subsequent round of the protocol.

2 Background

In this section, we provide background information regarding the key building blocks used in our work.

2.1 Homomorphic Encryption

Homomorphic encryption (HE) is an encryption technique that enables computation on encrypted data without requiring the decryption key [9]. After the necessary computations are performed on the ciphertext, the owner of the decryption key can decrypt it to view the transformed message.

In this work, we use the Cheon-Kim-Kim-Song (CKKS) encryption scheme, which is a scheme that supports approximate floating point arithmetic operations [4]. CKKS supports a property called *batching*, which allows the packing of multiple plaintext data in the slots of a ciphertext and enables us to work in a single instruction, multiple data (SIMD) setting.

2.2 Threshold Fully Homomorphic Encryption

Threshold Fully Homomorphic Encryption scheme (t-FHE) is a variant of the classical single-key HE scheme that extends it to a multi-party setting [2]. In t-FHE, key generation and decryption are conducted in a distributed manner,

while the encryption operation and homomorphic computations are performed the same way as in the single-key version. Each client i first generates a pair of public and secret keys, (pk_i, sk_i), and broadcasts their public key pk_i to generate a common public key pk as well as the evaluation key evk. To decrypt a ciphertext, each party i partially decrypts the encrypted message $E(m)$ using their share of the secret key, sk_i. These partial decryptions $D_i(E(m))$ are summed up to retrieve the decrypted message. In this work, we use the t-FHE implementation provided in the OpenFHE library, which supports a variety of popular FHE schemes, including CKKS [3].

2.3 Multilayer Perceptron

Multilayer perceptron (MLP) refers to feedforward neural networks with fully connected layers and an activation function. MLPs consist of an input layer, hidden layers, and an output layer. Each layer consists of multiple units (neurons). The units of the input layer are initialized with the value of the input. Each unit in the hidden and output layers performs a weighted sum of its inputs and passes the result through an activation function to receive its activation value. These activation values are fed to the next layer as input. MLPs are trained using *backpropogation*, which is a method that estimates the gradients of a loss function and updates the weights starting from the last layer, moving to previous layers [21].

2.4 Federated Learning and Federated Averaging

Federated learning (FL) is an ML technique that allows multiple distributed clients to collaboratively train an ML model without having to disclose their private local data, under a single centralized server [18]. Federated Averaging (FedAvg), an FL algorithm based on iterative model averaging, was introduced by [18]. In this algorithm, each client calculates the gradient of the loss function using their own local data and updates their model, as seen in Eq. (1).

$$W_i \leftarrow W_i - \eta \nabla F_i \left(W_i\right) \tag{1}$$

Here, W_i refers to the i-th client's model, $\nabla F_i \left(W_i\right)$ is the average gradient on the client's local data, and η is the learning rate. To reduce the number of communication rounds, this step is repeated multiple times locally before the model is sent to the central server for aggregation.

Once the local models are received by the centralized server, it calculates their weighted average to generate a global model, as shown in Eq. (2).

$$W_g \leftarrow \sum_{i=1}^{N} \frac{n_i}{n} W_i \tag{2}$$

Here, W_g refers to the global model at the given iteration, N is the number of clients, n_i is the size of the client's local dataset, and n is the size of the total client data. This process repeats iteratively for a number of communication rounds.

3 Related Work

Numerous studies in the literature focus on the privacy of aggregation-based FL. As there is an abundance of literature on the subject, we cover only a small and relevant portion of it and recommend the readers to refer to [16,20] for more information.

During the FL process, even though the clients' local datasets are kept private, the shared model information can inadvertently leak information regarding the private datasets of clients to unauthorized parties. Model inversion and membership inference attacks can potentially leak information regarding the data the models have been trained on [8,14,24]. In order to overcome these privacy concerns, techniques such as SMPC, DP, HE, and TEE have been employed.

In our work, we use an HE-based approach to overcome privacy concerns pertaining to FL. Other works in the literature that employ HE include [25], which proposes a privacy-preserving FL approach for convolutional neural networks (CNN) for the COVID-19 detection use case. In addition, [7] proposes a protocol that uses the Paillier HE scheme to perform aggregation-based FL over MLPs.

Furthermore, there are HE-based approaches that adapt HE to a multi-party setting to improve the security definition over the client models. In these approaches, there typically exists a global public that is key used for encryption, and secret key shares that are distributed among clients and used for decryption. [17] proposes a multikey HE-based protocol for the FedAvg algorithm.

Another example is PriCell, which is a solution that uses multi-party homomorphic encryption (MHE) to achieve privacy-preserving FL of CNN models for disease-associated cell classification [23].

The protocols mentioned in this section are compared with our work in greater detail in Sect. 5.5.

4 System Description

Our privacy-preserving FL protocol works under the assumption of a single-server setting. A general overview of the protocol in a 3-client setting is demonstrated in Fig. 1.

The protocol executes the following steps:

1. Each client C_i generates a pair of public/secret key shares (pk_i, sk_i), then broadcasts pk_i to generate a common public key, pk. The clients also agree on an initial set of weights for the global model, W_g. As the initialization step of the protocol, this step is only performed once and is not displayed in Fig. 1.
2. Each C_i initializes their local model with the weights of the new global model, W_g, then, trains their model using their own local dataset.
3. After training, C_i extracts the weights W_i of their local model. These values are prepared for the weighted averaging by multiplying them by the size of C_i's local dataset, then dividing it by the number of total data points shared amongst all clients (this is assumed to be public knowledge). The normalized values W_i are encrypted, as $E(W_i)$, and sent to the server S.

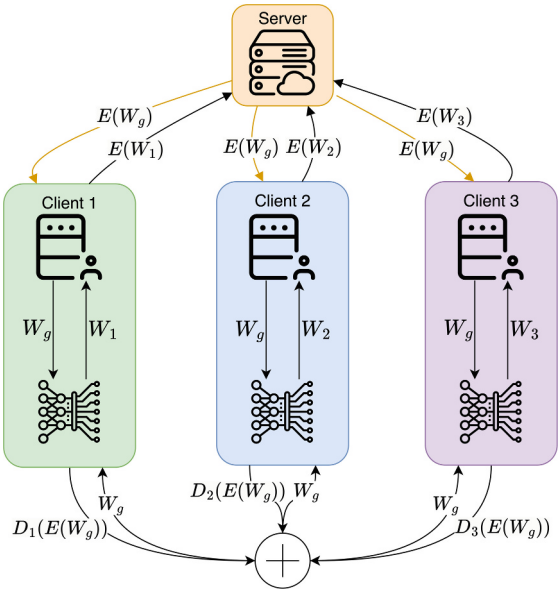


Fig. 1. A general diagram displaying the privacy-preserving FL protocol.

4. Upon receiving $E(W_i)$ from all clients, S performs the secure aggregation of the models.
5. S broadcasts the encrypted aggregated model $E(W_g)$ to all clients. Each C_i partially decrypts the global model and produces a share $D_i(E(W_g))$. These shares are communicated amongst the participants and aggregated to get the decrypted result, W_g.
6. This process repeats starting from Step 2, as many times as the number of rounds, R.

The protocol is a privacy-preserving version of the FedAvg algorithm. However, in the presence of colluding clients, the simple weighted averaging of weights could reveal the model of an honest client to the other colluding clients involved, which in turn could leak information regarding the honest client's local dataset. Even though encryption prevents the server S from inferring any information about the local/global models, it does not prevent colluding clients from collaborating to learn an honest client's local model parameters.

To illustrate the case, consider an N-client setting. The global model update, W_g, is calculated by taking the weighted average of all local models, $W_g = \frac{n_1 W_1 + \ldots + n_N W_N}{n}$. Now, assume that $N-1$ clients form a coalition to learn the local model of the remaining client, C_N. In order to do so, they share their local models with one other, and given W_g, they can extract W_N by reversing the linear aggregation operation. Furthermore, in a 2-client setting, both clients can infer the local model of one another directly without any extra communication, as W_g and C_1's local model W_1 are sufficient for C_1 to directly infer W_2, and vice versa.

To overcome the possibility of collusion attacks, we propose to take advantage of the *permutability* of feedforward neural networks [11]. This property states that the units of a hidden layer can be arbitrarily permuted without changing the output of the network. Given a model W, to permute the units in layer i, the rows of the weight matrix $W^{(i)}$ should be permuted according to some permutation order O, and the same permutation order should be applied to the columns of the subsequent weight matrix $W^{(i+1)}$.

Figure 2 shows the weight matrices of two subsequent densely connected layers, $W^{(i)}$ and $W^{(i+1)}$. Each dense layer computes the operation $Wx+b$ to produce an output, where x is the input and b is the bias of the given layer. By applying a permutation on the units of $W^{(i)}$ (i.e., rows), then applying the same permutation on the weights that these units are assigned to in the next layer $W^{(i+1)}$ (i.e., columns), we guarantee that the output of the network does not change.

$$W^{(i)} = \begin{pmatrix} w_{11}^{(i)} & w_{12}^{(i)} & w_{13}^{(i)} & w_{14}^{(i)} \\ w_{21}^{(i)} & w_{22}^{(i)} & w_{23}^{(i)} & w_{24}^{(i)} \\ w_{31}^{(i)} & w_{32}^{(i)} & w_{33}^{(i)} & w_{34}^{(i)} \\ w_{41}^{(i)} & w_{42}^{(i)} & w_{43}^{(i)} & w_{44}^{(i)} \end{pmatrix} \qquad W^{(i+1)} = \begin{pmatrix} w_{11}^{(i+1)} & w_{12}^{(i+1)} & w_{13}^{(i+1)} & w_{14}^{(i+1)} \\ w_{21}^{(i+1)} & w_{22}^{(i+1)} & w_{23}^{(i+1)} & w_{24}^{(i+1)} \\ w_{31}^{(i+1)} & w_{32}^{(i+1)} & w_{33}^{(i+1)} & w_{34}^{(i+1)} \\ w_{41}^{(i+1)} & w_{42}^{(i+1)} & w_{43}^{(i+1)} & w_{44}^{(i+1)} \end{pmatrix}$$

Fig. 2. Interchanging the units of $W^{(i)}$ and the corresponding weights of the next layer $W^{(i+1)}$ to demonstrate permutability.

In order to permute the encrypted rows and columns efficiently, each layer's row or column values are encrypted in a single ciphertext using the batching method. As each ciphertext holds data from a single row or column, shuffling the ciphertexts without changing their internal order can generate a new model equivalent to the original one. Furthermore, the SIMD nature of the batching method significantly reduces the time taken for aggregation by scaling the complexity of aggregation operations to the number of units instead of the number of parameters.

The permutation order O is only known by the server S and not by any of the clients. After the server S aggregates the corresponding weights, S permutes the encrypted rows/columns of the layers. Furthermore, the bias values b are permuted accordingly to ensure that the output of the network stays the same.

As the clients do not know the permutation order O that has been applied to the W_g, they cannot directly infer the local parameters of the remaining clients by reversing the aggregation operation, even in the existence of colluding parties, or in a 2-party setting. Therefore, our approach does not reveal any information about the local models in the existence of $N-1$ or fewer colluding semi-honest parties.

In this protocol, it is possible to work with any number of layers in a given MLP. However, if a model consists of an odd number of layers, that would require us to permute both the rows and the columns of a single inner hidden layer to ensure that all layers are permuted. Thus, this would require us to apply the permutation on the slots of encrypted ciphertexts also, which can be a highly

expensive operation. For this reason, we choose to work with models with an even number of layers.

5 Results and Discussion

In this section, we present an evaluation and analysis of our protocol in terms of its classification accuracy, runtime performance, communication cost, and provided security. Furthermore, we present a comparison of our protocol with other recent works in the literature that use HE schemes to ensure privacy in an FL setting. Moreover, the implementation of our algorithm is publicly available on https://github.com/cerenyildirim/PPMLP.

5.1 Experimental Setting

We use the MNIST hand-written digit images dataset to evaluate the performance of our proposed protocol [6]. We partition the MNIST training set among the clients in both IID and non-IID fashion. In the IID case, the training set is divided uniformly randomly among the involved clients. For the non-IID case, we consider two different partitioning strategies. In the first non-IID strategy (Non-IID), the training set is divided based on the class of data items, where each client gets the entire subset of different classes. In the second non-IID strategy (Non-IID-Dirichlet), the class distribution skewed partitioning described in [26] is used. The Dirichlet distribution with a concentration parameter of 0.5 is used to generate partitionings with unbalanced class distributions. In our experimental setting, we test the accuracy and runtime performance of the protocol in the presence of $N \in [3, 5, 10]$ clients.

The MNIST dataset consists of black and white 2D images of digits. The model we use consists of 4 dense layers. Each hidden layer consists of 128, 64, and 32 units, and the output layer contains 10 units (number of classes in the MNIST dataset). Between each dense layer, a dropout layer exists with dropout rates of 0.5, 0.25, and 0.25 in the presented order. Each local model and the global model consist of a total of $111,146$ parameters. As we use the batching method to fit multiple data in a single ciphertext, each encrypted model is made up of 554 ciphertexts. The models' batch size B is set to 100, and the local epoch E is chosen as 1, and all clients participate in the training in all rounds.

We use the OpenFHE library's t-FHE implementation of the CKKS scheme. We use the Python programming language and the official Python wrapper for the OpenFHE library. In the encryption setting, the ring dimension is selected as 8192 (a total of 4096 slots exists in a given ciphertext) and the scaling modulus size is set to 50 bits. The default security level of 128 bits is used. To train and test the models, we use the Keras and TensorFlow libraries [1,5]. The timings are taken on a machine with an AMD Ryzen Threadripper PRO 3955WX CPU (16 cores, 4.3 GHz Max Boost Frequency, 504 GB RAM).

5.2 Classification Performance

In this work, we use the CKKS HE scheme, which performs approximate computations over encrypted data. Hence, the decrypted result may differ in precision from the actual result to some extent. For this reason, we tested our same protocol in both plaintext and encrypted format to compare their accuracies. The results indicated that any precision loss coming from the CKKS encryption scheme was negligibly low, and did not affect the performance of the models.

Figures 3, 4 and 5 display the accuracies for different number of clients N in the Non-IID, Non-IID-Dirichlet, and IID settings, respectively. Furthermore, we present the ground truth accuracy in each figure for comparison. The parameters for the ground truth model are selected to be $E = 20, B = 100$. We keep the batch size the same as the FL models, but the number of epochs is set to 20, which is a reasonable point at which the accuracy starts to slow down its improvement on the validation set, providing a realistic estimate for a ground truth model.

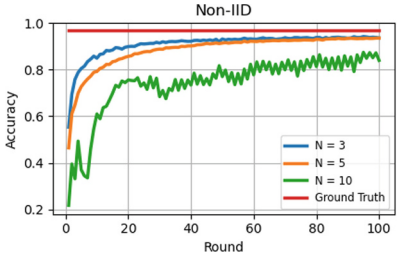


Fig. 3. Classification accuracy comparison for different N in Non-IID setting.

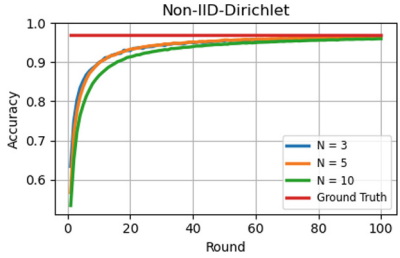


Fig. 4. Classification accuracy comparison for different N in Non-IID-Dirichlet setting.

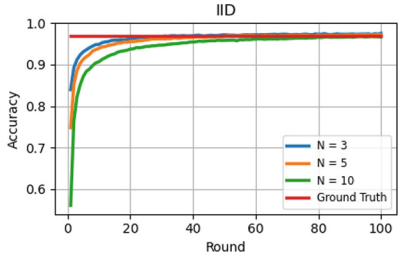


Fig. 5. Classification accuracy comparison for different N in IID setting.

For all settings, it can be inferred that it becomes increasingly challenging for accuracy to converge to a better result as the number of clients grows. The situation becomes more prominent in the Non-IID setting since the classes are strictly divided among clients. We observe that the ground truth accuracy is not reached even after a total of 100 rounds. This is expected since FedAvg is known to have convergence issues in statistically heterogeneous settings.

In the Non-IID-Dirichlet setting, the number of clients is observed to be less effective when it comes to influencing the accuracy of the resulting models. As the classes are not strictly partitioned between the clients and each client likely holds data from different classes even if in small amounts, the overall accuracy of the global model is observed to be not as poor as in the Non-IID case.

In the IID, setting, the global model converges to the accuracy of the ground truth model more quickly. Furthermore, we can reach the highest accuracy performance in the IID case as each client has roughly the same amount of data coming from each class. However, it should be reminded that the ground truth model is not fixed, but rather an estimate of a reasonable model trained on the provided data. Hence, its accuracy could potentially be improved by increasing the count of epochs, in line with the number of rounds R executed in the FL setting.

5.3 Runtime Performance

In Fig. 6, we present the timings for a single round of training considering various numbers of clients N. The timings are considered for three main operations: encoding and encryption of the local models, the server evaluation time, and decryption of the resulting global model. We find it sufficient to only consider the non-IID partitioned data for the timing analysis, as we do not expect to see any meaningful difference in runtime based on the data partitioning methods In our protocol, the encoding/encryption of the local models, as well as the partial decryption of the resulting global model can be performed concurrently and independently by each client. To account for such parallelizable operations, the mentioned timing results are normalized by the N involved.

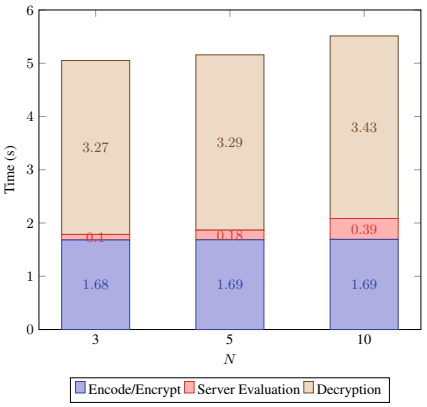


Fig. 6. Timings of a single round for different number of clients N.

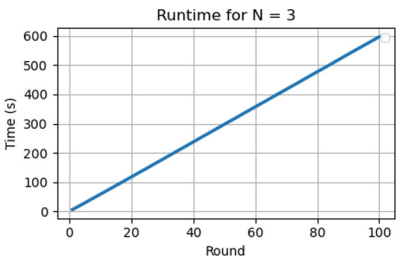


Fig. 7. Runtime across different number of rounds R.

From the results, it can be observed that most of the runtime is allocated for the decryption of the ciphertexts, while the actual aggregation and permuta-

tion operations performed by the server make up an insignificant portion of the total runtime. Even though the server evaluation time increases with N due to the number of homomorphic addition operations increasing with it, the server evaluation time remains relatively negligible and is not observed to surpass the overhead coming from the encryption and decryption operations. The time that encryption and decryption operations take directly relates to the number of ciphertexts making up the global model.

Figure 7 shows the total runtime of the protocol in a 3-client setting for the first 100 rounds. Aside from the encoding/encryption, server evaluation, and decryption time, the total runtime also includes the key generation, local model training, and local model update times. It can be observed that the runtime increases linearly with the round number, as one would expect to see.

5.4 Security Evalution

We assume that all parties involved in the protocol (including the server) are honest-but-curious (semi-honest). In our protocol, as all the values received and processed by the server are in encrypted format, the security of these relies solely on the CKKS encryption scheme, whose security depends on the hardness assumption of the quantum-resistant ring learning with errors (RLWE) problem.

The clients do not have access to the local models of other clients. However, they can view the global model updates received at each round. Our proposed permutation technique prohibits the clients from possibly colluding and learning the local model of any other client. The security of the algorithm against this sort of collusive inference is directly tied to the number of units in layers, and the number of layers in the given model.

The weights of each pair of successive layers are permuted based on the number of units in the foremost layer. In our implementation, each pair of layers is shuffled using a unique permutation order. This means that for each round, we use 2 different permutation orders for the total of 4 dense layers in our model (i.e., two unique permutation vectors of sizes 128 and 32). The number of possible permutation orders for the given layers determines the number of equivalent models we can generate to replace the original global model. Thus, in total, there exists $128! \times 32! \approx 2^{834}$ different equivalent models for a single global model update, making it computationally infeasible to directly infer a client's local model at a given round.

It should be noted that even a single hidden layer consisting of 64 units proves to be sufficient to deliver more than 256-bit of security against brute-force collusion attacks. The security bound determines the maximum number of attempts for colluding clients to reverse the weighted averaging process and uncover the local model. The colluding clients can utilize the data they own to evaluate and attack the generated models in order to select a model that aligns with the expected performance of the honest client's local model. However, the sheer amount of possibilities and the computational effort to perform such an analysis make this collusive inference computationally infeasible. Furthermore, even after such an attack, there is no guarantee that the selected model would

be the one belonging to the honest client, as the colluding clients are unable to directly infer which of these models is actually owned by the honest client.

The clients could possibly try to learn some information regarding the local data of other clients through advanced model inference attacks performed on the global updates. However, the acquired information through these attacks cannot be attributed to a particular client, and attacks on global models are often weaker compared to attacks on local models, especially for a higher number of clients [19].

However, in this work, we do not take into account the likelihood that colluding participants might attempt to recover information about the permutation and the honest client's model by leveraging the information available to them both at the beginning and at the end of each round. This information may enable the colluding participants to identify patterns and correlations between the data/models they possess and what they receive. This presents a significant problem that could weaken the security assumption provided by the permutation. As a result, the actual security level for the honest client's model may be lower than the theoretical bit security discussed in this section, particularly if the model weights are distributed in a highly non-uniform fashion.

To address this problem and limit the information that the colluding participants possess (which includes the global model updates at the end of each round), one could leverage techniques such as differential privacy to perturb the global model updates. By doing so, one could limit the capabilities of the colluding participants to identify any correlations between their prior knowledge and the information they later receive, thereby preventing the discovery of confidential information. While we recognize the significance of this issue, it falls outside the scope of this paper. For future work, a more in-depth investigation of such advanced attacks could be performed in greater detail.

Finally, we examine the scenario in which the server may also participate in a collusion attack. Until now, we have only considered client collusions. However, if the server is involved in the collusion attack, our protocol can handle collusion between less than $N-1$ clients and the server. The server is the only entity with the permutation information. Hence, the local models of $N-1$ clients combined with the permutation information from the server, are enough to reconstruct the local model of an honest party. Therefore, we can conclude that our protocol is resistant to $N-1$ colluding clients, as well as $N-2$ colluding clients and the server.

5.5 Comparison with Other Works

In the literature, there are other works that utilize HE to evaluate aggregation-based privacy-preserving FL protocols for neural network models. In this section, we compare our work with four other approaches in the literature. We present a qualitative analysis of our works in terms of security and performance. We consider the metrics used in [16] to provide a basis for our comparative analysis. A summary of the security comparison of all works discussed in this section is presented in Table 1.

Table 1. Security comparison of our work and others in the literature (LM Vis.: Local model visibility, GM Vis.: Global model visibility, SH: Semi-honest).

	Method	Threat Model		LM Vis.		GM Vis.	
		S	C	S	C	S	C
Ours	t-FHE; CKKS	SH coll. $N-2$ Cs	$N-1$ coll. SH	✗	✗	✗	✓
Wibawa et al. [25]	HE; BFV	Non-coll. SH	$N-2$ coll. SH	✗	✗	✗	✓
Fang et al. [7]	Paillier	Non-coll. SH	$N-2$ coll. SH	✗	✗	✗	✓
Ma et al. [17]	xMK-CKKS	SH coll. $N-2$ Cs	$N-2$ coll. SH	✗	✗	✓	✓
Sav et al. [23]	MHE; CKKS	SH	$N-1$ coll. SH	✗	✗	✗	✗

In [25], the protocol operates under the single-key HE scheme. All clients share the same secret key, which is unknown to the server. The clients encrypt their local models using the secret key, the server performs the aggregation, and the decryption is performed locally by each client. The clients' view consists of its local model and the global model. The server's view is only limited to the encrypted local and global models, whose security depends on the BFV encryption scheme (with the hardness assumption of RLWE). In this protocol, the clients and the server are assumed to be semi-honest. However, the following attacks are possible under the description of the given protocol:

- A malicious client might intercept the communication between the server and an honest client to learn the honest client's local model. The malicious client can decrypt the honest client's model as it is in possession of the secret key.
- $N-1$ semi-honest and colluding clients could come together to discover the local model of the honest client, using the attack described in Sect. 4.
- The server might collude with one or more clients to learn the secret key, thus, being able to decrypt any client's local model.

[7] uses an improved version of the Paillier cryptosystem to achieve FL on MLPs. At each round, a new public/secret key pair is generated by the key management center and shared with the clients. Thus, at each round, the clients are in possession of a single secret key which is used for decryption. The clients encrypt their local gradients using the public key. The server performs aggregation of the encrypted gradients and sends the result back to the clients who locally decrypt them. The clients update their local model with the new gradients and the same process repeats for several rounds. Since the secure aggregation portion of the protocol follows the same general approach as in [25], all the attack scenarios that apply to it also apply to this protocol.

In [17], a federated aggregation solution using multi-key homomorphic encryption over the CKKS encryption scheme (xMK-CKKS) is proposed. In the protocol, both the server and the clients are assumed to be honest-but-curious. In the multi-key HE approach, there exists a global public key, but the shares of the private keys are distributed among the clients. At each training round, the clients get the global model from the server and train it using their private

dataset. The local model weights are encrypted and sent to the server. The server performs aggregation of the weights and sends the encrypted global model back to the clients. Clients perform partial decryptions of the encrypted model and send these results to the server. The server then aggregates the partial decryptions to obtain the global model update. This protocol ensures the security over local model weights, such that no outsider, server, or any other client can see the local models. However, the global model itself is visible to the server. Moreover, the protocol is shown to be collusion-resistant only in the presence of $N-2$ colluding clients or less. Thus, it is assumed that for the protocol to be secure, there must be at least 2 honest clients involved. If this condition is not met, collusion attacks can occur, compromising the honest client's local model.

In [23], MHE is used to achieve privacy-preserving FL. In this scenario, each party holds a share of the secret key. The local models are in encrypted form, and the local trainings are performed homomorphically by each party. The trained models are shared with one of the parties for aggregation, and the resulting global model is sent to the remaining parties. After a certain number of training rounds, the model can be decrypted for further analysis or kept confidential for encrypted model inference. During the entire FL process, no decryption is required. Thus, the protocol ensures end-to-end model protection and offers resistance against inference attacks. However, it suffers from a significant computational overhead due to the training of local models being performed on encrypted data, thus requiring the expensive bootstrapping operation to be applied to the ciphertexts. This limits the protocol's applicability for deeper and more complex neural network models.

In our work, we rely on t-FHE for privacy-preserving FL. As in the case of [17] and [23], the shares of the secret key are distributed among the clients. Our protocol offers security under the assumption of colluding semi-honest participants and presents improvements upon the security of [7,25], and [17]. In our protocol, the permutation of the global model weights prevents up to $N-1$ colluding clients from inferring information regarding the local model of honest clients. If the server collaborates with the clients in a collusion attack, our protocol remains resistant to collusion if fewer than $N-1$ clients are involved, aligning with the security assumption of [17]. Furthermore, a malicious client intercepting the communication between the server and an honest client is unable to decrypt the honest client's local model due to the use of t-FHE.

However, our protocol does not have the strong end-to-end security assumption of [23] as ours requires the global model to be decrypted by the clients after each training round. This has the risk of potentially exposing information about one client's dataset to other clients, as discussed in Sect. 5.4. However, our approach offers a significantly more robust solution in terms of computational overhead, as the only operation we perform homomorphically is the inexpensive addition operation used for aggregation. Thus, we offer a highly scalable approach that could more easily support FL over deeper and more complex models. Hence, in settings where computational overhead is of a bigger concern than end-to-end security, our protocol could be preferred.

6 Conclusion

In this work, we present a privacy-preserving FL protocol for MLPs using t-FHE. In our protocol, the server gains no information regarding the local or global models, while the clients can only view the aggregated models. The possibility of a collusion attack between clients is removed by utilizing the permutability of neural networks. We show that our protocol works without losing accuracy in the encrypted setting while remaining computationally efficient. By only using the inexpensive addition operations and taking advantage of the batching property of FHE, we minimize the number of operations that are required to be performed on encrypted data.

For future work, our approach could be adapted to more advanced federated aggregation techniques, such as FedProx [15] or Scaffold [12], which aim to mitigate issues related to the non-IID division of datasets. Furthermore, our protocol could be adapted to more involved types of feedforward neural networks, such as CNNs. Additionally, our protocol can be applied to other popular datasets, such as CIFAR-10, as well as real-world datasets, to evaluate its applicability for various settings and use cases.

Acknowledgments. This work was supported by the European Union's Horizon Europe research and innovation program through the Twinning Project (enCRYPTON) under Grant 101079319 and the PHASE IV AI Project under Grant 101095384. This work is funded by the European Union. Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

References

1. Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous systems (2015). https://www.tensorflow.org/, software available from tensorflow.org
2. Asharov, G., Jain, A., López-Alt, A., Tromer, E., Vaikuntanathan, V., Wichs, D.: Multiparty computation with low communication, computation and interaction via threshold FHE. In: Pointcheval, D., Johansson, T. (eds.) EUROCRYPT 2012. LNCS, vol. 7237, pp. 483–501. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29011-4_29
3. Badawi, A.A., et al.: OpenFHE: open-source fully homomorphic encryption library. Cryptology ePrint Archive, Paper 2022/915 (2022). https://eprint.iacr.org/2022/915
4. Cheon, J.H., Kim, A., Kim, M., Song, Y.: Homomorphic encryption for arithmetic of approximate numbers. In: Takagi, T., Peyrin, T. (eds.) ASIACRYPT 2017. LNCS, vol. 10624, pp. 409–437. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-70694-8_15
5. Chollet, F., et al.: Keras (2015). https://keras.io
6. Deng, L.: The mnist database of handwritten digit images for machine learning research. IEEE Signal Process. Mag. **29**(6), 141–142 (2012)

7. Fang, H., Qian, Q.: Privacy preserving machine learning with homomorphic encryption and federated learning. Future Internet **13**(4), 94 (2021)
8. Fredrikson, M., Jha, S., Ristenpart, T.: Model inversion attacks that exploit confidence information and basic countermeasures. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1322–1333 (2015)
9. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the Forty-First Annual ACM Symposium on Theory of Computing, pp. 169–178 (2009)
10. Guo, Z., You, L., Liu, S., He, J., Zuo, B.: Icmfed: an incremental and cost-efficient mechanism of federated meta-learning for driver distraction detection. Mathematics **11**(8), 1867 (2023)
11. Hecht-Nielsen, R.: On the algebraic structure of feedforward network weight spaces. In: Advanced Neural Computers, pp. 129–135. Elsevier (1990)
12. Karimireddy, S.P., Kale, S., Mohri, M., Reddi, S., Stich, S., Suresh, A.T.: Scaffold: stochastic controlled averaging for federated learning. In: International Conference on Machine Learning, pp. 5132–5143. PMLR (2020)
13. Lee, C.M., Fernández, J.D., Menci, S.P., Rieger, A., Fridgen, G.: Federated learning for credit risk assessment. In: HICSS, pp. 386–395 (2023)
14. Li, Q., et al.: A survey on federated learning systems: vision, hype and reality for data privacy and protection. IEEE Trans. Knowl. Data Eng. **35**(4), 3347–3366 (2021)
15. Li, T., Sahu, A.K., Zaheer, M., Sanjabi, M., Talwalkar, A., Smith, V.: Federated optimization in heterogeneous networks. Proc. Mach. Learn. Syst. **2**, 429–450 (2020)
16. Liu, Z., Guo, J., Yang, W., Fan, J., Lam, K.Y., Zhao, J.: Privacy-preserving aggregation in federated learning: a survey. IEEE Trans. Big Data (2022)
17. Ma, J., Naas, S.A., Sigg, S., Lyu, X.: Privacy-preserving federated learning based on multi-key homomorphic encryption. Int. J. Intell. Syst. **37**(9), 5880–5901 (2022)
18. McMahan, B., Moore, E., Ramage, D., Hampson, S., y Arcas, B.A.: Communication-efficient learning of deep networks from decentralized data. In: Artificial Intelligence and Statistics, pp. 1273–1282. PMLR (2017)
19. Nasr, M., Shokri, R., Houmansadr, A.: Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In: 2019 IEEE Symposium on Security and Privacy (SP), pp. 739–753. IEEE (2019)
20. Qi, P., Chiaro, D., Guzzo, A., Ianni, M., Fortino, G., Piccialli, F.: Model aggregation techniques in federated learning: a comprehensive survey. Futur. Gener. Comput. Syst. (2023)
21. Rumelhart, D.E., Hinton, G.E., Williams, R.J.: Learning internal representations by error propagation, parallel distributed processing, explorations in the microstructure of cognition, ed. de rumelhart and j. mcclelland. vol. 1. 1986. Biometrika **71**(599-607), 6 (1986)
22. Sarkar, S., Agrawal, S., Gadekallu, T.R., Mahmud, M., Brown, D.J.: Privacy-preserving federated learning for pneumonia diagnosis. In: Tanveer, M., Agarwal, S., Ozawa, S., Ekbal, A., Jatowt, A. (eds.) Neural Information Processing. ICONIP 2022. CCIS, vol. 1794, pp. 345–356. Springer, Singapore (2023). https://doi.org/10.1007/978-981-99-1648-1_29
23. Sav, S., Bossuat, J.P., Troncoso-Pastoriza, J.R., Claassen, M., Hubaux, J.P.: Privacy-preserving federated neural network learning for disease-associated cell classification. Patterns **3**(5) (2022)

24. Shokri, R., Stronati, M., Song, C., Shmatikov, V.: Membership inference attacks against machine learning models. In: 2017 IEEE Symposium on Security and Privacy (SP), pp. 3–18. IEEE (2017)
25. Wibawa, F., Catak, F.O., Kuzlu, M., Sarp, S., Cali, U.: Homomorphic encryption and federated learning based privacy-preserving cnn training: covid-19 detection use-case. In: Proceedings of the 2022 European Interdisciplinary Cybersecurity Conference, pp. 85–90 (2022)
26. Yurochkin, M., Agarwal, M., Ghosh, S., Greenewald, K., Hoang, N., Khazaeni, Y.: Bayesian nonparametric federated learning of neural networks. In: Chaudhuri, K., Salakhutdinov, R. (eds.) Proceedings of the 36th International Conference on Machine Learning. Proceedings of Machine Learning Research, vol. 97, pp. 7252–7261. PMLR, Long Beach, California, USA (09–15 June 2019). http://proceedings.mlr.press/v97/yurochkin19a.html

Open Access This chapter is licensed under the terms of the Creative Commons Attribution-NonCommercial-NoDerivatives 4.0 International License (http://creativecommons.org/licenses/by-nc-nd/4.0/), which permits any noncommercial use, sharing, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if you modified the licensed material. You do not have permission under this license to share adapted material derived from this chapter or parts of it.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

