Poster: FLATEE: Federated Learning Across Trusted Execution Environments

Arup Mondal
Department of Computer Science
Ashoka University
arup.mondal_phd19@ashoka.edu.in

Yash More
Department of Computer Science
Ashoka University
yashrajmore29@gmail.com

Ruthu Hulikal Rooparaghunath
Department of Computer Science
Ashoka University
hulikalruthu@gmail.com

Debayan Gupta
Department of Computer Science
Ashoka University
debayan.gupta@ashoka.edu.in

2021 IEEE European Symposium on Security and Privacy (EuroS&P) | 978-1-6654-1491-3/21/$31.00 ©2021 IEEE | DOI: 10.1109/EUROSP51992.2021.00054

***Abstract*—Federated learning allows us to distributively train a machine learning model where multiple parties share local model parameters without sharing private data. However, parameter exchange may still leak information. Several approaches have been proposed to overcome this, based on multi-party computation, fully homomorphic encryption, etc.; many of these protocols are slow and impractical for real-world use as they involve a large number of cryptographic operations. In this paper, we propose the use of Trusted Execution Environments (TEE), which provide a platform for isolated execution of code and handling of data, for this purpose. We describe FLATEE, an *efficient* privacy-preserving federated learning framework across TEEs, which considerably reduces training and communication time. Our framework can handle malicious parties (we do not natively solve adversarial data poisoning, though we describe a preliminary approach to handle this).**

***Index Terms*—Federated Learning, Trusted Execution Environment, Secure Multi-Party Computation, Homomorphic Encryption, Differential Privacy**

1. Introduction

While traditional machine learning approaches depend on a central training data set, privacy considerations have driven interest in decentralized learning frameworks, where parties collaborate to train an ML model without sharing their respective training datasets. Federated learning (FL) [1] is a powerful approach for collaborative and privacy-preserving learning: here, parties collectively train a model by training locally and then exchanging model parameters (instead of actual training data), which keeps their data private. However, recent work [2] has demonstrated that parameter interaction and the final model may leak information about the training dataset.

Multi-party computation has been used for privacy-preserving ML [3]. Made possible by a range of cryptographic primitives, MPC [4], [5] allows multiple parties to compute a function without revealing the inputs of any individual party (beyond what is implied by the output). Several schemes have already been proposed for privacy-preserving FL using MPC [6], but these often take a very long time to train, and may also incur high data-transmission costs. Further, they cannot usually deal with participants dropping out during the FL process [6].

Trusted Execution Environments (TEE) are an emerging hardware primitive: Intel's Software Guard Extensions (SGX) [7] provide a module within chipsets that enable the creation of secure containers called *enclaves*. These hardware-enforced "reverse sandboxes" allow data and code to be processed without the influence of code running in the traditional registers of the processor. An SGX system can use hardware-based attestation to prove that an enclave executes exactly the functions promised and nothing else (assuming one trusts Intel). TEEs incur lower overheads compared to traditional software protections.

Our Contributions: We propose an efficient, privacy-preserving federated learning framework using a TEE (Intel SGX). We have an FL server $\mathcal{S}$ and a set of parties $\mathcal{P} = P_1, P_2, \ldots, P_n$ where each P_i has a private dataset $\mathbb{D}_i$. Critically, unlike recent MPC or Fully Homomorphic Encryption (FHE) based solutions, we do **not** exchange data – however obfuscated – with the server. Instead, we use traditional FL techniques modified for privacy, training models separately within each party/client TEE, and then combining them securely within the server. Specifically:

- FLATEE, a privacy-preserving federated learning framework based on TEEs, enables the parties to efficiently train a distributed model. Additionally, FLATEE provides strict privacy guarantees of training and is also resistant to *data-poisoning* and *model-poisoning* attacks.
- We use Differential Privacy (DP) based techniques with low clipping bounds and high noise variance to prevent backdoor attacks. We also use Multi-KRUM [8] to guarantee resiliency from k malicious updates out of n total updates, where $2k + 2 < n$.

This paper is organized as follows: Sec 2 provides background on FL and TEEs. Sec 3 describes our framework and its trust model. Sec 4 discusses future work.

Motivation: We address two main drawbacks of existing frameworks. Popular FL frameworks such as [6], [9] often incur large amounts of training time and communication latencies (due to the computationally intensive cryptographic operations involved). Privacy-preserving

DOI 10.1109/EuroSP51992.2021.00054

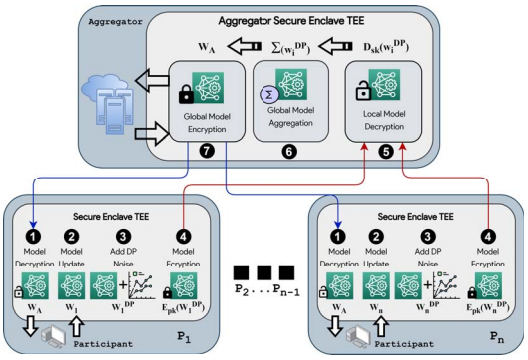

Figure 1. Schematic overview of FLATEE

federated learning frameworks are vulnerable to post-quantum attacks, as they only involve traditional encryption techniques. Our proposed framework relies on quantum-resilient cryptographic schemes which prevent such potential attacks and protects user-sensitive data. Furthermore, our TEE-based approach offers a significant time improvement over others, as shown in Table 1.

2. Technical Background and Preliminaries

Federated Learning. Federated learning (FL) [1] is a distributed approach to Machine Learning which allows models to be trained on a large body of decentralized data with many participants. FL is an example of the technique of "bringing code to data, not data to code", and is suitable for use cases with sensitive data (health care, financial services, etc.). In FL, each party trains a model locally and exchanges only model parameters with an FL *server* or *aggregator*, instead of the private training data.

Trusted Execution Environment. A Trusted execution environment (TEE) is a hardware extension that aims to provide *integrity* and *confidentiality* guarantees to security-sensitive computation performed on a computer where all the privileged software (kernel, hypervisor, etc) is potentially malicious. Specifically [7]: (1) *Authenticity & Confidentiality* of the code running on a TEE is ensured. (2) The *State Integrity* of run-time states is also ensured including memory, CPU registers, and I/O; states are stored in persistent memory. (3) The content of a TEE is *dynamic* and can be updated during execution. (4) An "ideal" TEE is *secure* against all software and hardware attacks. (5) A TEE is *trustworthy* and can provide proof of correctness of the executed computation to a third-party. (6) Provides proof that users are interacting with software hosted inside the TEE (the *attestation* functionality).

A major aim of TEEs is to solve the problem of secure remote computation – execution on an untrusted machine while having integrity and trust guarantees. One example of such a system is Intel SGX [7], which provides a secure container using trusted hardware to give a remote user the ability to upload the code and data to this container. Several measures ensure the confidentiality of the executed computation and intermediate data.

3. FLATEE Framework

Let $\mathcal{S}$ be the FL server and $\mathcal{P}$ be a set of n parties, where i^{th} party P_i holds its own private dataset $\mathbb{D}_i$, and $\mathcal{M}_{FL}$ is the DL model to be trained by the parties' private data. Each party agrees on $\mathcal{M}_{FL}$ before starting the training process and authenticates to the FL server. We assume an honest-but-curious, non-colluding FL server, which runs the protocol honestly but may try to glean information from the trained models. *Curious, colluding participants* may inspect messages exchanged between the FL server or final model to glean the private data of other participants (we discuss the malicious case later).

3.1. FLATEE Detailed Operations

Setup. $\mathcal{S}$ and all $\mathcal{P}$ have SGX-enabled machines, and agree to train a model $\mathcal{M}_{FL}$. $\mathcal{S}$ uses its SGX module to authenticate all $\mathcal{P}$ and aggregate the parties' trained models using the *federated average function* [1] (*e.g.,* weighted mean, geometric median etc.) to generate the global trained model. Each P_i checks the model then trains it locally on private data, then sends the *encrypted model parameters* to $\mathcal{S}$. E and D are *post-quantum secure* en/decryption; "pk" and "sk" denote public and private keys (private keys never leave SGX).

FLATEE Protocol

1) Each P_i agrees on a model $\mathcal{M}_{FL}$ and $\mathcal{S}$ publishes a hash of the model $\mathcal{H}(M_{FL})$ by which everyone can verify their local version of the model.
2) To ensure P_i trains $\mathcal{M}_{FL}$, we check the sign measurement $TEE^{P_i}_{measure}$ signed by the the P_i TEE's private key $TEE^{sk}_{key_{P_i}}$. P_i also authenticates its TEE with TEE of $\mathcal{S}$.
3) P_i trains the local model and adds DP noise to the model parameters, and then sends the encrypted model parameters (using $TEE^{pk}_{key_{FLserver}}$) to $\mathcal{S}$.
4) Encrypted model parameters from all P_i, $\{E(\mathcal{M}^{\mathbb{D}_{P_1}}_{FL}), E(\mathcal{M}^{\mathbb{D}_{P_2}}_{FL}), \ldots, E(\mathcal{M}^{\mathbb{D}_{P_n}}_{FL})\}$ are decrypted inside of the FL server TEE's using $TEE^{sk}_{key_{FLserver}}$, *i.e.*, we perform $\{D(E(\mathcal{M}^{\mathbb{D}_{P_1}}_{FL})), D(E(\mathcal{M}^{\mathbb{D}_{P_2}}_{FL})), \ldots, D(E(\mathcal{M}^{\mathbb{D}_{P_n}}_{FL}))\}$.
5) $\mathcal{S}$ runs the *federated average function* in its TEE to aggregate all $\mathcal{P}$'s trained model parameters and get a global trained model. $\mathcal{S}$ should use *data- obliviousness* (*e.g.,* Oblivious RAM) to hide the actual memory reference sequence.
6) $\mathcal{S}$ calculates the loss function over the global model. If it satisfies error constraints, $\mathcal{S}$ sends the encrypted global model $E(\mathcal{M}_G)$ to $\mathcal{P}$, else we do another round – steps 3 to 5. A party can drop out at any time of the training process but can join only after a training round.

Threat Model and Poisoning Attacks. We assume *curious and colluding participants*. We separately consider adversarial participants who contribute poisoned updates to introduce backdoors into the shared model: here, assume that the adversary intends to harm the performance, or introduce backdoors, into the shared *global* model, or leak private information about the used training dataset. For this work, we limit the adversaries to label flipping attacks, pixel-pattern backdoor attacks, and deep leakage from gradients using reconstruction attacks.

We use Multi-KRUM [8], a byzantine-resilient gradient aggregation algorithm, to address poisoning attacks.

TABLE 1. COMPARISON BETWEEN VARIOUS PRIVACY-PRESERVING FEDERATED LEARNING FRAMEWORK

Framework	Comms†	Privacy Capability		Threat Model			Privacy Guarantees		Security Guarantees		Techniques Used					Features				
		Inference	Training	Participants	Aggregator	Required TPA	Computation	Output	Data Poisoning	Model Poisoning	HE	TP	SS+AE	FE	TEE	Scalability	Dynamic Participants	Post-Quantum Security	Statistical Heterogeneity
PySyft [10]	$2mn + n$	○	●	□	☒	✓	●	●	○	○	●	○	○	○	○	◐	○	○	○
Truex et al. [6]	$2mt + mn + n$	○	●	■	☒	✓	●	●	○	○	○	●	○	○	○	●	○	○	○
Bonawitz et al. [9]	$2mn + n$	○	●	■	☒	✓	●	●	○	○	○	○	●	○	○	●	◐	○	○
HybridAlpha [11]	$mn + m + n$	◐	●	■	☒	✓	●	●	◐	○	○	○	○	●	○	●	●	○	○
FLATEE (This Work)	mn	●	●	■	☒	×	●	●	●	●	○	○	○	○	●	●	●	●	●

"HE" is homomorphic encryption'; "TP" is Threshold-Paillier system; "SS+AE" secret sharing with key agreement protocol and authenticated encryption scheme; "FE" is functional encryption; and "TEE" is Trusted Execution Environment. "TPA" is *trusted third party* is used to set up a master private key and a master public key that will be used to derive multiple public keys to one or more parties who intend to encrypt their data. □ denotes honest party; ☒ denotes semi-honest party; ■ denotes dishonest party; ○ denotes does not provides property; ◐ denotes partially provides property; ● denotes provides property. Communications† – The number of crypto-related operations required in each training round, where n is the number of participants and m is the number of aggregators, and t is the threshold for decryption of Threshold-Paillier cryptosystem.

Instead of using a publicly available validation dataset, it scores each local model parameter based on its deviation from every other submitted local model parameters in every federated round. Multi-KRUM [8] guarantees resiliency from k malicious updates out of n total updates, when $2k + 2 < n$. For update $V_i \ \forall \ i \in [1, n]$, $Score(V_i)$ is calculated as the sum of euclidean distances between V_i and V_j, where V_j denotes the $n-k-2$ closest vectors to V_i as follows: $Score(V_i) = \sum_{i \to j} \mid\mid V_i - V_j \mid\mid^2$. Here, $i \to j$ denotes the fact that V_j belongs to $n-k-2$ closest vectors to V_i. The $n-k$ updates with the lowest scores are selected for aggregation, and the rest are discarded. Multi-KRUM offers ease of implementation and provides defense against model replacement attacks. In the future, we plan to compare aggregation techniques that defend against poisoning attacks, such as FoolsGold, median, and trimmed mean. We refer the reader to [8] for security guarantees and convergence analysis of Multi-KRUM.

DP [12] based techniques with low clipping bounds and high noise variance can render backdoor attacks ineffective, but can have a slight impact on the accuracy of the *global* model. Hence, to minimize the efficacy of model poisoning attacks with DP, but without impacting model performance, we can use gradient pruning where gradients with small magnitudes are pruned to zero.

4. Future Work

In this work, we assume existence of ideal TEEs which are not affected by any micro-architectural attacks like *Spectre*, *Meltdown*, *Foreshadow*, *Plundervolt*, etc. We continue to work towards including cryptographic techniques to present a micro-architectural attack resistant protocol for the same. We also plan to present a complete implementation of our proposed framework whilst considering all known micro-architectural attack vectors. Finally, we also plan to compare the efficiency and cost of FLATEE with existing protocols achieving similar results.

References

[1] H. B. McMahan, E. Moore, D. Ramage, S. Hampson *et al.*, "Communication-efficient learning of deep networks from decentralized data," *arXiv preprint arXiv:1602.05629*, 2016.

[2] L. Melis, C. Song, E. De Cristofaro, and V. Shmatikov, "Exploiting unintended feature leakage in collaborative learning," in *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2019, pp. 691–706.

[3] P. Ramachandran, S. Agarwal, A. Mondal, A. Shah, and D. Gupta, "S++: A fast and deployable secure-computation framework for privacy-preserving neural network training," *arXiv preprint arXiv:2101.12078*, 2021.

[4] B. Mood, D. Gupta, H. Carter, K. Butler, and P. Traynor, "Frigate: A validated, extensible, and efficient compiler and interpreter for secure computation," in *2016 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 2016, pp. 112–127.

[5] J. Perry, D. Gupta, J. Feigenbaum, and R. N. Wright, "Systematizing secure computation for research and decision support," in *International Conference on Security and Cryptography for Networks*. Springer, 2014, pp. 380–397.

[6] S. Truex, N. Baracaldo, A. Anwar, T. Steinke, H. Ludwig, R. Zhang, and Y. Zhou, "A hybrid approach to privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 1–11.

[7] D. Gupta, B. Mood, J. Feigenbaum, K. Butler, and P. Traynor, "Using intel software guard extensions for efficient two-party secure function evaluation," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 302–318.

[8] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer, "Machine learning with adversaries: Byzantine tolerant gradient descent," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, 2017, pp. 118–128.

[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] T. Ryffel, A. Trask, M. Dahl, B. Wagner, J. Mancuso, D. Rueckert, and J. Passerat-Palmbach, "A generic framework for privacy preserving deep learning," *arXiv preprint arXiv:1811.04017*, 2018.

[11] R. Xu, N. Baracaldo, Y. Zhou, A. Anwar, and H. Ludwig, "Hybridalpha: An efficient approach for privacy-preserving federated learning," in *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*, 2019, pp. 13–23.

[12] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy." 2014.