**Olive**: Oblivious and Differentially Private Federated Learning on Trusted Execution Environment

Fumiyuki Kato  
Kyoto University  
fumiyuki@db.soc.i.kyoto-u.ac.jp

Yang Cao  
Kyoto University  
yang@i.kyoto-u.ac.jp

Masatoshi Yoshikawa  
Kyoto University  
yoshikawa@i.kyoto-u.ac.jp

**ABSTRACT**

Differentially private federated learning (DP-FL) has received increasing attention as it mitigates the privacy risk in federated learning. Despite the fact that various DP-FL schemes have been proposed, there are still utility gaps. The use of central differential privacy in FL (CDP-FL) can provide a good balance between privacy and model utility; however, it necessitates the use of a trusted server. Using local differential privacy for FL (LDP-FL) does not require a trusted server, but the trade-off between privacy and utility is poor. The recently proposed shuffle DP-based FL has the potential to bridge the utility gap between CDP-FL and LDP-FL for the absence of a trusted server; however, there is still a utility gap due to the limitation of privacy amplification. In this study, we propose Olive, a system that combines the merits of CDP-FL and LDP-FL by leveraging a Trusted Execution Environment (TEE). Our main technical contributions are the analysis of and countermeasures against the vulnerability of FL on TEE. To begin, we analyze memory access pattern leakage of FL on TEE and discover that there is a risk of sparsified gradients, which is common in FL. Second, we devise an inference attack to link the memory access pattern to the sensitive information of training data. Finally, we propose oblivious yet efficient aggregation algorithms to prevent memory access pattern leakage. Our experiments on real-world data demonstrate that our proposed method works efficiently even when training a model with hundreds of thousands of parameters while ensuring full obliviousness, which brings secure FL closer to realization.

**PVLDB Reference Format:**

Fumiyuki Kato, Yang Cao, and Masatoshi Yoshikawa. **Olive**: Oblivious and Differentially Private Federated Learning on Trusted Execution Environment. PVLDB, 14(1): XXX-XXX, 2020. doi:XX.XX/XXX.XX

**PVLDB Artifact Availability:**

The source code, data, and/or other artifacts have been made available at https://github.com/FumiyukiKato/FL-TEE.

1 INTRODUCTION

In today’s Big Data era, the challenge of privacy in machine learning is becoming more apparent. The strict enforcement of the multitude of regulations on privacy [26] are emblematic of this. Federated learning (FL) [3, 43] is an innovative paradigm for privacy-preserving machine learning. Typically, in FL, a server does not need to collect the raw data from the users (we exchangeably use participants or clients) but only collects gradients (or model delta), which are trained on the local data of participants at each round of model training; then, the server aggregates the collected gradients as a global model for the next round of local training. Thus, FL is expected to free data analyzers from the cost and privacy risk of managing training data containing sensitive information (e.g., medical data [53]). Major tech companies [52, 54] have deployed FL in production especially for mobile applications [10].

Many studies, however, highlight FL’s vulnerability to various types of attacks due to its decentralized and complicated scheme. When sharing the unprotected final or in-progress model, the users who participate in the training rounds may suffer membership [61] or label [68] inference attacks against the shared model. Moreover, recent studies [76, 78] have shown that a server could accurately reconstruct the raw data or infer sensitive attributes by observing the unprotected raw gradients sent by participants.

Differentially private FL (DP-FL) [24, 44] has received increased attention owing to its ability to mitigate privacy risks by guaranteeing Differential Privacy (DP) [19]. Researchers investigated various schemes for DP-FL in order to strike a good balance between the trust model and utility, as shown in Table 1. In CDP-FL [5, 24, 44, 71], a trusted server collects the raw participants’ data and takes the responsibility to privatize the global model, while in LDP-FL [41, 63, 69, 77], the clients perturb the gradients before sharing with an untrusted server. LDP-FL does not require a trusted server unlike CDP-FL but utility is limited due to the strict privacy criteria. To integrate both the model utility of CDP-FL and the trust model of LDP-FL, recent works [25, 40] proposed a scheme based on the shuffle model of DP [7, 21], which we call Shuffle DP-FL. This
scheme introduces a trusted shuffler sitting between the server and clients, who shuffles the locally-perturbed gradients and then sends them to the untrusted server. Through the mathematically-proven privacy amplification effect of such a scheme, Shuffle DP-FL can achieve good utility that is comparable with the CDP-FL setting but without the need of a trusted server. However, the privacy amplification effect for FL [25, 40] is limited when the number of model parameters is large, such as in deep learning models or when the participants are not sufficient. Hence, there is still a utility gap between CDP-FL and the state-of-the-art Shuffle DP-FL.

To fill this gap, we propose a new scheme for DP-FL by leveraging Trusted Execution Environment (TEE), which is called OLIVE (i.e., Oblivious Differentially PriVate Federated Learning on TEE) and illustrated in Figure 1. TEE [18, 55] is a secure hardware technique that enables secure computation in an untrusted environment without exposing sensitive data or processing to the host of the operating systems or hypervisor. OLIVE employs TEE to ensure secure model aggregation on an untrusted server so that only differentially private models are observable by the untrusted server or any third parties. Intuitively, OLIVE ensures that the transfer of gradients (or model delta) between TEE and participants are encrypted, that the model aggregation within TEE is invisible to the untrusted server, and that any data leaving TEE satisfies DP. The utility of OLIVE is exactly the same as the conventional CDP-FL as the computation inside TEE can be implemented for arbitrary algorithms.

However, there are many known serious vulnerabilities in TEE due to side-channel attacks [23, 50, 67, 72]. In particular, the attacks can expose data-dependent memory accesses patterns of confident execution and allow attackers to steal sensitive information such as RSA private key and human genome information [12]. What specific information an attacker can steal from these memory access patterns is a domain-specific question and is not yet known for FL, even though several studies attempt to use TEE for FL [16, 46, 47, 73, 75]. This means that it is still unclear whether the side-channel attack against FL on TEE is really dangerous and what type of attack is possible.

Oblivious algorithms [27, 62, 70] is known as an important technique that causes only data-independent memory access patterns and prevents these attacks. A general method is to make RAM itself oblivious, i.e., oblivious RAM (ORAM), and PathORAM [62] is known to be the most efficient one. However, PathORAM assumes a certain size of private memory space and is not applicable to practical TEE such as Intel SGX [57]. While Zerotrace [57] addresses this issue, the overhead is still significant. Therefore, we need to design an algorithm-specific method to obtain an efficient oblivious algorithm. [51] proposes efficient oblivious algorithm for specific machine learning algorithms. However, an efficient method for FL-specific aggregation algorithm, which can be a vulnerable part, are not yet known.

In this study, we address the aforementioned gaps: (1) we clarify the privacy risk by designing specific attacks on typical CDP-FL setting and demonstrating it on real-world scenario; (2) we design efficient oblivious algorithms to implement secure CDP-FL on TEE and evaluate it empirically. First, we find that the gradient position information is leaked in the FL aggregation algorithm in a sparsified setting. Sparsification is often used in FL [31, 38, 39, 56, 60] to reduce communication costs, which is a critical factor in practical scenario [3] (see Section 6). An attacker’s assumptions are as follows; the ability to observe memory access patterns; access to the i.i.d test dataset; and access to the trained model for each round, each of which is justified in detail in Section 3.2. The attacker’s goal is to infer a set of sensitive labels included in the target user’s training data, which is similar setting to [68]. After demonstrating the attack in real-world datasets, we propose efficient oblivious algorithms to completely prevent such an attack. We design our proposed method by carefully constructing existing oblivious building blocks such as oblivious sort [2, 8] and our designed oblivious components. In addition to fully-oblivious algorithms, we further investigate an optimization by adjusting data size in enclave and more efficient algorithms by relaxing the definition of obliviousness. Finally, we conducted extensive experiments on real-world data to show that our proposed method, designed for FL aggregation, is more efficient than the general-purpose PathORAM applied to SGX [57].

The following is a summary of our contributions.

- We propose OLIVE, an oblivious and differentially private federated learning system based on TEE. OLIVE takes the merits of good utility from CDP-FL and the advantage of a trust model the same as LDP-FL. We analyze how memory access patterns could be exposed to the untrusted server when TEE is used for model aggregation in FL. We found that there is a risk in the case of sparsified gradients.
- We designed a supervised learning-based sensitive label inference attack based on index information observed from side-channels of sparsified gradients. We demonstrated the attack on a real-world dataset. One of the results shows that when training with CNN on CIFAR100 with top-1.25% sparsification, the sensitive labels of training data (each participant has 2 out of 100 labels) leak with approximately 90% accuracy or better (Figure 6).
- We propose a novel oblivious algorithm that can efficiently execute model aggregation by carefully combining oblivious primitives such as oblivious sort or register-level oblivious updates. Experiments demonstrate the efficiency of our proposed method. In particular, it is much faster (more than

| Trust model | Utility |
|-------------|---------|
| CDP-FL [5, 24, 44, 71] | Trusted server | Good |
| LDP-FL [41, 63, 69, 77] | Untrusted server | Limited |
| Shuffle DP-FL [25, 40] | Untrusted server + Shuffler | Shuffle DP-FL ≤ CDP-FL |
| OLIVE (Ours) | Untrusted Server with TEE | OLIVE = CDP-FL |
where confidential computation. raw gradients [76, 78]. the server as the original data can be reconstructed even from the practical model scales [5, 44]. However, CDP-FL requires the server corresponds to the learning algorithm with perturbation (e.g., DP-SGD) in at most one client and any subset of outputs 's training data to learn. 's training data to learn. (Client-level) CDP-FL guarantees that it is probabilistically indistinguishable whether a client is participating in the process can be completed within a few seconds.

The remainder of this study is organized as follows. Section 2 presents the preliminaries of our work. Section 3 describes our system Olive and examines its security in terms of FL memory access pattern leakage. Section 4 demonstrates how the memory access pattern could be linked to users' training data. Section 5 gives the defenses against the attacks and the performance evaluations. Section 6 comprehensively discusses related works that motivate our work, and Section 7 concludes the study.

2 PRELIMINARIES

2.1 Federated Learning

Federated learning (FL) [36, 43] is a new machine learning (ML) scheme with distributed optimization. In contrast to traditional centralized ML, many participants who provide training data train a global model on their own devices rather than sharing training data with a central server, which solves a variety of problems in the machine learning paradigm. For example, intuitively, there is no cost for the server to store and manage the sensitive data. The basic algorithm of FL, called FedAVG [43], trains models by repeating the steps of optimizing models in the local environment of the participants and updating the global model by aggregating the parameters of the locally trained models (e.g., taking a weighted average of gradient information over the number of training data). Additionally, FedSGD [43] exchanges locally updated gradients based on distributed stochastic gradient descent (SGD). To make our position clear, we describe various related FL studies in detail in Section 6.

Olive trains a global model that satisfies central DP (CDP). CDP-FL [5, 24, 44, 71] assumes a trusted server who accesses participants' raw data, privatizes the aggregated global model and publishes it to third parties. (Client-level) CDP-FL guarantees that it is probabilistically indistinguishable whether a client is participating in the training or not. It is defined as follows:

Definition 2.1 ((client-level) (ε, δ)-differential privacy [44]). A randomized mechanism \( M : \mathcal{D} \rightarrow Z \) satisfies (ε, δ)-DP if, for any two neighboring client sets \( D, D' \in \mathcal{D} \) such that \( D' \) differs from \( D \) in at most one client and any subset of outputs \( Z \subseteq Z \), it holds that

\[
Pr[M(D) \in Z] \leq \exp(\epsilon) Pr[M(D') \in Z] + \delta.
\]

where \( Z \) corresponds to the final trained model and \( M(D) \) corresponds to the learning algorithm with perturbation (e.g., DP-SGD) that uses input client \( D \)'s training data to learn.

In general, CDP-FL provides a good trade-off between privacy and utility (e.g., model accuracy) of differentially private models even at practical model scales [5, 44]. However, CDP-FL requires the server to access raw gradients, which leads to major privacy concerns on the server as the original data can be reconstructed even from the raw gradients [76, 78]. Olive addresses this issue by utilizing TEE’s confidential computation.

2.2 Trusted Execution Environment

TEE, as formally defined in [55], creates an isolated execution environment within untrusted computers (e.g., cloud VMs). Our security requirements are based on the well-known Intel SGX TEE implementation [18]. It is the extended instruction set of Intel x86 processors, and it enables the creation of an isolated memory region, called enclave. The enclave resides in the encrypted and protected memory region, called the Enclave Page Cache (EPC), in which all programs and data can be decrypted and quickly processed as well as transparently encrypted outside the CPU package by a memory encryption engine using a secret key that only the processor hardware can access. In other words, SGX follows a model in which the CPU package serves as a trust boundary; everything beyond it is considered untrusted. The processor prohibits access from any untrusted software, including the OS/hypervisor. Also, it provides a functionality that ensure the integrity of the program and data inside the enclave. Note that, for design reasons, the user-available size of the EPC is limited to about 96 MB on most of current SGX-equipped machines. Assuming that memory is allocated beyond this memory size constraint, SGX with Linux has paging mechanism which causes significant overhead for encryption and integrity checks. Several studies have shown that the performance is greatly degraded by the severe overhead [35, 64].

Attestation. SGX supports a remote attestation (RA), which can verify the correct initial state and genuineness of the trusted environment of the enclave. Requesting RA, we receive a report with measurements based on the hash of the initial enclave state, which is generated by the trusted processor. This can identify the program, the complete memory layout, and even builder’s key information. Intel Enhanced Privacy ID signs this measurement, and the Intel Attestation Service can verify the correctness of the signature. Intel. As a result, we realize verifiable and secure computations on the remote enclave. In addition to the verification, secure key exchange between the enclave and remote client is performed within this RA protocol. Therefore, after performing the protocol, we can begin communicating with a remote enclave over a secure channel using AES-GCM and so on.

2.3 Memory Access Pattern Leakage

Although the data itself are encrypted and cannot be seen, the memory access patterns may be exposed regardless of the use of TEE. Memory access pattern leakage occurs when an intrusive OS is allowed to observe cacheline accesses, for example, by injecting page faults [72] or by page table-based threats [50, 67]. It can be done by attaching probes directly to the memory bus if a physical machine is available. This attack has been demonstrated to be capable of obtaining sensitive information from enclaves [12].

To prevent this attack, a number of oblivious algorithms have been proposed to hide access patterns during execution. The definition of an oblivious algorithm is as follows:

Definition 2.2 (Oblivious algorithm [14]). An algorithm \( M \) is δ-statistical oblivious if, for any two input data \( l, l' \) of equal length, for any security parameter \( \lambda \), it holds that

\[
\text{Accesses}^M(\lambda, l) \stackrel{\Delta}{=} \text{Accesses}^M(\lambda, l')
\]
where $\text{Accesses}^M_M(\lambda, I)$ is a random variable denoting the ordered sequence of memory accesses the algorithm $M$ makes upon receiving the input $\lambda$ and $I$. $\delta(\lambda)$ denotes that the two distributions have at most $\delta(\lambda)$ statistical distance. The term $\delta$ is a function parameterized by parameter $\lambda$ that corresponds to the cryptographic security parameter. For the special case where $\delta$ is negligible, we say that $M$ is fully oblivious, and where $\delta$ is 1, not oblivious.

To construct an oblivious algorithm, the most typical approach is to use oblivious RAM (ORAM) [62]. However, as ORAM is constructed for a general use case as a key-value store, several task-specific methods (e.g., ML [51] or sampling [58]) are proposed from a performance perspective. Furthermore, ORAM generally assumes that there is trusted memory space as a client, which is incompatible with the SGX assumption of leaking memory access patterns in enclaves, and only the CPU registers should be considered trusted memory space. [51] implements oblivious machine learning algorithms using CMOV, which is an x86 instruction providing a conditional copy in the CPU registers. CMOV moves data from register to register based on a condition flag in the register, which is not observed by any memory access patterns. With this CMOV instruction, conditional branching can be implemented with a constant memory access pattern that does not depend on input. Moreover, using CMOV to implement a program with a single path and no conditional branching, we can also defend against timing-based side channel attacks [66] and branch prediction attacks [37]. We can construct oblivious algorithms based on primitives, such as oblivious move (o_move, Listing 1) and oblivious swap (o_swap, Listing 2). $o_{-}\text{move}(\text{flag}, x, y)$ is a function that takes a condition flag of type bool as its first argument and returns $x$ or $y$ depending on the flag; $o_{-}\text{swap}(\text{flag}, x, y)$ has a similar signature. Zerotrace [57] implements PathORAM on SGX by obliviously implementing client storage with these CMOV-based oblivious primitives. We compare this state-of-the-art with our proposed method in the experiment explained in Section 5.5.

3 PROPOSED SYSTEM

In this section, first, we clarify the threat model in our scenario; second, we present the system overview of Olive that trains the model guaranteeing CDP\(^1\) under untrusted server using TEE. Finally, we analyze the details of its potential privacy risk, followed by demonstrating a specific privacy attack in Section 4.

3.1 Scenario

Our scenario is a typical FL setting with a single server and clients. The server is responsible for orchestrating the training process, aggregating parameters, updating the global model, selecting clients for each training round, and validating the quality of the model. The server can be placed in a public or private environment; however, it is assumed to have a TEE capable of RA (e.g., Intel SGX). The clients have sensitive training dataset in their local edge devices.

**Threat model.** We assume an adversary who wants to leak information about the client’s training data as a semi-honest server. The server has (1) access to the trained model for each round of FL, and (2) the data i.i.d. with the client’s training data, and (3) the capability to observe the memory access patterns of TEE. These requirements can be respectively justified as follows. (1): The server can access the model parameters if there is no enclave-to-client encryption for the model delivery in the downlink. However, because the server is in charge of model validation, it makes sense for the server to have access to models during each round of training. Furthermore, attackers could easily blend in with massive clients. (2): Generally, the server has access to public datasets for model validation and/or for initial model training, which is essential in production-uses. (3): This follows the general threat model for TEE. Except for the trusted component (i.e., CPU package), all other components of the server, such as the OS/hypervisor, the main memory and all communication paths are untrusted. The server can observe memory access patterns by known and/or unknown side channel attacks as described in Section 2.3. To summarize more generally, we assume that the attacker is a semi-honest server that is simultaneously responsible for model aggregation and validation in FL.

3.2 System overview

Our proposed system, Olive, is based on the existing learning scheme of CDP-FL (i.e., DP-FedAVG [44] or similar algorithms that share locally optimized raw results and perform aggregation and perturbation on the server) equipping TEE on the server side as shown in Figure 1. As an initial setup, we provision an enclave, where each client exchanges shared keys (e.g., AES) and verifies the integrity of the processes running on the enclave via RA. If the attestation fails, the clients have to refuse to join the FL at this phase. We assume that communication between the client and server is performed over a secure channel (TLS), which the untrusted server terminates, and that sent gradients\(^2\) are doubly encrypted and can be decrypted only in the trusted enclave.

The overall algorithm of Olive is shown in Algorithm 1, where differences from the basic DP-FedAVG algorithm are shown in red. The first RA is omitted, and a different shared key $sk_i$ is stored in the enclave for each user $i$. For each round, participants are securely sampled in the enclave (Line 4). The selected users are memorized in the enclave and used for client verification when the encrypted gradient are loaded in the enclave (Line 7). The verification can be done by using the per-user shared key to check if the data sent can be decrypted correctly (i.e., authenticated encryption (AE) mode). This makes it impossible for malicious parties to inject users not securely sampled. As we will see in Section 3.3, the aggregation operation (Line 12) needs to be oblivious; we will present more detailed algorithms in Section 5. In this way, the gradients sent from all clients are not visible to the server; only the aggregated and updated parameters perturbed by Gaussian noise are observed. The perturbation allows us to guarantee the $(\varepsilon, \delta)$-DP of the model as quantified by moments accountant [1] as normal CDP-FL [31, 44].

3.3 Security Analysis

TEE enables us to learn a model that satisfies DP while protecting the raws gradients with encryption. However, it has certain limitations. The untrusted server can observe the memory access patterns\(^3\) in DP-FedSGD [44], we jointly refer to the model update data sent from participants as gradients in the later sections.

\(^1\)Strictly, our definition of DP is not information-theoretic but in the computational sense [45] due to relying on distributed and encrypted way.

\(^2\)In DP-FedAVG, the data shared from participants are not exactly gradients, but differences of model weights. However, the same argument can basically be made in DP-FedSGD. We jointly refer to the model update data sent from participants as gradients in the later sections.
through the side channels based on the known and/or unknown vulnerabilities as described in Section 2.3.

For formal modeling, let $g_i$ be the $k$-dimensional gradients sent by user $i$, and let $g^*$ be the $d$-dimensional global parameters after aggregation. In the typical case, $k = d$, where dense gradients are used. Let $G_i$ and $G^*$ be the memory for storing the values of $g_i$ and $g^*$, respectively, and let the number of clients participating in each round be $n$. The memory storing the entire gradients is denoted as $G = G_1 || ... || G_n$, where $||$ means concatenation. A memory access $a$ is represented as a triple $a = (A[i], op, val)$, where $A[i]$ denotes the $i$th address of memory $A$, $op$ denotes the operation for the memory, either read or write, and $val$ is the value to be written when $op$ is write and null otherwise. Therefore, the observed memory access pattern $\text{Accesses}$ can be represented as $\text{Accesses} = [a_1, a_2, ..., a_m]$ when the length of the memory access sequence is $m$.

In CDP-FL, the operations performed on the server side generally consist of summing the gradients from all users, averaging, and perturbing. We first notice that this procedure is oblivious to dense gradients. As shown in Figure 2, the summing operation involves updating the value of the corresponding index of $G^*$ while performing a linear scan on $G$, where the memory accesses are performed in a fixed order and at fixed addresses regardless of the contents of $G$. We refer this general summing part as Linear algorithm and show it in Appendix B. Additionally, the operation of averaging and perturbing just causes accesses on $G^*$ linearly.

**Proposition 3.1.** The Linear algorithm is fully oblivious for dense gradients. (The formal proof, which is almost obvious, is provided in the Appendix C.)

The operation is performed in $O(nd)$ since all elements of the gradient $G$ are accessed. Also, the operations of averaging and adding noise follow and are clearly fully oblivious; they are merely linear aggregated of the accessed gradients $G^*$ once in $O(d)$.

However, we find that when the gradients are in a sparsified format (which is often used in FL as shown in Section 6), the access pattern of the Linear algorithm is not oblivious, and it can leak sensitive information. A weight of sparse gradients generally consists of a tuple of index, which holds the location information of the parameter, and value, which holds the gradient value, regardless of its quantization and/or encoding methods to which the data are applied. Figure 3 shows the access pattern caused when the aggregation operation is used for sparsified gradients.

**Proposition 3.2.** The Linear algorithm is not oblivious for sparsified gradients.

**Proof.** The operation is to perform linear access on $G$ as well as dense gradients, but the access pattern occurred for sparsified gradients $\text{Accesses}^{\text{sparse}}$ is as follows:

$$\text{Accesses}^{\text{sparse}} = 
\{(G[i], \text{read}, +), (G^*[\text{id}_{ix}], \text{read}, +), (G^*[\text{id}_{ix}], \text{write}, +), ...
\}$$

where the indexes of sparsified gradients of user $i$ are $\text{id}_{ix1}, ..., \text{id}_{ixk}$ for all $i \in [n]$. The access pattern $\text{Accesses}^{\text{sparse}}$ is deterministic and corresponds one-to-one to the sequence of indexes of the input data. Considering two input data $I, I'$ that have different sequence of indexes, no overlap exists in the output distribution. Thus, the maximum statistical distance of two $\text{Accesses}^{\text{sparse}}$ is 1. □

We can find that the access pattern on aggregated gradients $G^*$ reveals, at least, a set of indices $\{\text{id}_{ixj} | j \in [d]\}$ for each user $i$, depending on the given gradients completely. Considering data-dependent sparsifications such as top-$k$ that are generally used in FL, the gradients index the sparsified gradients have may be sensitive to the training data. We demonstrate that they cause privacy leakage using a real-world dataset in the following section.
Algorithm 2 Attack on index: Jac or NN

Input: $i$: target user, $X_t$: test data with label $l$ ($l \in L$), round: $T$
1. $\text{index} \leftarrow \{\}$ → observed access patterns
2. /* Prepare teacher and target indices */
3. $\text{teacher} \leftarrow \{\} →$ teacher access patterns to train a classifier
4. for each round $t = 1, \ldots, T$
5. /* $T_t$: rounds participated in by user $i$ */
6. if $t \in T_t$ then
7. /* $A^t_l$: observed top-$k$ indices of user $i$ of round $t$ */
8. $\text{Store } A^t_l$ to index[$i, t$]
9. for each label $l \in L$, do
10. /* $\theta^t$: the global model after round $t$ */
11. /* $I_l^{(t)}$: top-$k$ indices training with $\theta^t$ and $X_t$ */
12. $\text{Store } I_l^{(t)}$ to teacher[$i, t$]
13. /* Calculate scores for each label $l$ */
14. $S \leftarrow []$ → form of [{label, similarity}]
15. /* If Jac: Jaccard similarity-based scoring (Sim) */
16. for each label $l \in L$, do
17. $\text{Store } (l, \text{SIM}(\|r \in T, \text{index}[i, r], \|r \in T, \text{teacher}[i, r])$ to $S$
18. /* If NN: neural network-based scoring */
19. Train the model $M_t$ with teacher[$i, t$] ($l \in L$) for each $t \in T$
20. for each label $l \in L$, do
21. $\text{Store } (l, \text{PREDICT}(M_t, ..., M_T, \|r \in T, \text{index}[i, r]))$ to $S$
22. /* If NN-SINGLE: using single neural network */
23. Train the model $M_b$ with $\|r \in T, \text{teacher}[i, r]$ ($l \in L$)
24. for each label $l \in L$, do
25. $\text{Store } (l, \text{PREDICT}(M_b, \|r \in T, \text{index}[i, r]))$ to $S$
26. /* 1D K-Means clustering KMEANS */
27. [labels, centroid] $\leftarrow$ KMEANS
28. return labels of the cluster with the largest centroid

Generality. Although our study focuses on the TEE, this problem generally occurs, as far as applying any data-dependent sparsification and encryption for gradients without any protection against aggregation operations. For example, SparseSecAgg [20] or threshold-based sparsification [56]. Regardless of the details of the encryption or sparsification implementation, the index information is restored and the aforementioned data-dependent accesses occur during aggregation.

4 ATTACK ON GRADIENT INDEX

4.1 Design

In this section, we design an attack by an aggregation server to demonstrate that the index information in the gradients is sufficient to cause a privacy leakage in the training data. We assume sparsified gradient setting by top-$k$ sparsification [39, 56, 60]. The attacker is based on the assumptions described in Section 3.1. The proposed attacks can be used to (1) raise awareness of the security/privacy risk of FL on TEE, which is a missing part in related works [16, 46, 47, 73], and (2) serve as an evaluation framework to assess the effectiveness of the defense methods.

The goal of the attack is to infer the target client’s sensitive label information of the training data. For example, when training federated learning on medical image data such as image data on breast cancer, the label of cancer or not is very sensitive, and participants may not want to reveal that information. We can see a similar attack goal in [68]. The overview of our designed attack is based on the intuition that the top-$k$ indices of the locally converged model parameters correlate with the labels of the local training data. This is consistent with the general idea of machine learning that labels and features are correlated. We train a classifier that takes the observed index information as input and the sensitive label set as output by supervised learning with public datasets. Access to such public datasets is justified, for example, by the need for model validation as described in Section 3.1. We design two basic methods: the Jaccard similarity-based nearest neighbor approach (Jac) and neural networks (NN). The detailed algorithms are shown in Algorithm 2. An overview of the methods is as follows:

1. First, the server has to prepare the test data $X_t$ with label $l$ for all $l \in L$, where $L$ is all possible labels.
2. In each round $t$ ($t \in T$), an untrusted server observes the memory access patterns through side channels, obtains the index information of the top-$k$ gradient indices $\text{index}[i, t]$ for each user $i$, and stores it (Lines 4-8).
3. The server computes the gradient of the global model with $\theta^t$ and $X_t$ without model updates for each round $t$ ($t \in T$), using the test data divided by labels, and obtains the top-$k$ indices $\text{teacher}[i, t]$ as teacher data for each label $l$ (Lines 9-12).
4. After all rounds $T$, in Jac, we calculate the Jaccard similarity between observed access patterns $\|r \in T, \text{index}[i, r]$ and $\|r \in T, \text{teacher}[i, r]$ for each label $l$ (Lines 15-17). We chose Jaccard similarity because, in the worst-case scenario, the index information sent by a participant is randomly shuffled, rendering the ordering meaningless.
5. In NN, the attacker trains neural networks using $\text{teacher}[i, t]$, the feature is indices and the target is a label (Line 19). The outputs of the model are the scores for each label. We use a trained model to predict what labels are included in the training data with input $\text{index}[i]$. For this task, we design following two different NN-based methods. One is to train a model $M_t$ for each round $t$ and average the output scores of the models to predict the labels (NN), and the other is to train a single model $M_b$ with the concatenated indices of the entire round as input and obtain one output (NN-SINGLE). In our experiment, both are multilayer perceptron with three layers. Note that for the model input, index information is represented as a multi-hot vector. As each client participates in only part of the rounds, in the case of NN-SINGLE, the indices of the rounds they do not participate in are all set to zero as input to the model. Although NN-SINGLE is expected to be able to capture the correlation over rounds better than NN, this zeroization may reduce the accuracy. Finally, as in Jac, we store the scores for each label obtained from the model prediction (Lines 20-21).
6. If the number of label types for the target client is known in advance, sort the scores in descending order and return labels by the known number. If the number of labels is unknown, K-means clustering is applied to the scores to classify them into 2 classes, and the labels with the highest centroid are returned (Lines 23-24).

Note that, in the above attack, since we are observing index information before perturbation by central DP, the attack can mostly bypass...
the DP protection. In particular, the first round of observation is completely unaffected.

Lastly, the obtained information from side channels can also be used to design attacks for another purposes, for example, as additional features in reconstruction attack [30] or other inference attacks [49]. In our study, we aim to show that the top-k gradient indices that can be observed on untrusted servers in FL have sufficient information to cause privacy leakages; therefore, we leave the study of attacks for different purposes as a future directions.

4.2 Evaluation Task

In our evaluation of the attack method, the server performs an inference attack against any clients according to the scenario detailed in Section 3.1. The clients have a subset of labels and the attacker’s goal is to infer sensitive label set a target client holds in his training data. The attacker selects any subset or the entire set of users and performs an inference attack on each one. We introduce all and top-1 as accuracy metrics for evaluating the attack performance. We define all as the percentage of clients that are exactly matched with inferred labels, e.g., inferred label set is {1,3,5} and target client’s label set is {1,3,5}. And, we define top-1 as the percentage of clients that contain the highest score inferred label, e.g., the highest score inferred label is 5 and target client’s label set is {4,5}, which we consider a minimal privacy leak. In addition, to control the difficulty of the attack task, we adjust the distribution of the label set that the client has. The number of labels in the set and whether the number of labels is fixed or random are configurable. In the case of fixed label, all users have the same number of labels, and the attacker also knows the number. In the case of random label, we give the maximum number and all users have various numbers of labels. In general, random setting and larger number of labels is more difficult, although it depends on the metrics all and top-1.

4.3 Empirical Analysis

Here, we demonstrate how effectively our designed attack works in a real-world dataset. 

Setup. Table 2 shows the datasets and the global models we used in our experiments. The model details are found in Appendix D. In addition to the well-known image datasets MNIST and CIFAR 10 and 100, we also use Purchase100, which is tabular data and was used in [32] for membership inference attacks. We train global models with different numbers of parameters as shown in the table. The learning algorithm is based on DP-FedAVG [44] with top-k sparsification where we give the ratio (i.e., sparse ratio) instead of the k of top-k and select the ones with the highest absolute value in order. FL’s learning parameters include the number of users N, the participant sampling rate q, the number of rounds T, the sparse ratio α and noise multiplier σ (which is noise’s standard deviation divided by the clipping scale and commonly used in DP-SGD framework). The default setting is (N, q, T, α, σ) = (1000, 0.1, 3, 0.1, 1.12). The attack methods are evaluated for Jac, NN, and NN-single, as described in the previous section. T is smaller than normal FL scenarios, which means that our method requires only a few rounds to attack. Our all experimental source code and dataset is open3.

Results. Figure 4 shows the attack results by NN, NN-single and Jac on all datasets with a fixed number of labels and Figure 5 shows the results with a random number of labels. In CIFAR10, we use T = 1 because the model size is large and it takes long to train the attack model with multiple rounds in NN-single. The vertical axis indicates the success rate of the attacks and the horizontal axis indicates the number of labels each client possesses. If the number of labels is small, we can see that the attack has a very high probability of success with all three attacks. The success rate of top-i is high regardless of the number of labels while all drops with each additional label. In CIFAR10, MLP model maintains high success rate for large number of labels compared to CNN model. This indicates that the more complex the target model, the more significant the contribution of the index information to the attack. The NN-based method is more powerful on MNIST, but using the other dataset shows small differences. This indicates that the gradient index information is not complex and can be attacked using simple methods. This may be supported by the fact that the results of NN and NN-single are almost the same, so there is not much effective correlation across the rounds. In the case where the class label is 100 (Purchase100, CIFAR100), the success rate of the attack is reduced. In particular, the accuracy of CIFAR100 is low. We consider sparsity α of 0.1 is insufficient and will discuss more sparse case in the next paragraph. Overall, however, these results here are sufficient to highlight our proposed threat.

Figure 6 describes the relationships between the sparse ratio and attack performance. The number of client labels is fixed at 2. This shows the lower the sparse ratio is, the higher the success rate of the attack. This is because the index of label-correlated gradients becomes more distinguishable as the gradients become more sparse. CIFAR100 (right figure), in particular, demonstrates that the attack is only successful when the sparse ratio is low, and the sparse ratio is an important factor in assessing the attack. When sparse ratio is 0.3%, success rate reaches almost 1.0.

In Figure 7, we show the attack results on MLP of CIFAR10 for increasing noise scale. The horizontal axis indicates noise scale σ by DP and the left-side start points indicate no noise. Compared to the case with no noise, increasing the noise has almost no effect on the attack performance. This makes sense from our attack design, where the attacker observes the raw index information of gradients even though the global model satisfies DP, therefore, there is no direct noise on the index information. Also, at least for the first round, the index information can be observed with no noise on the global model. When we increase the noise multiplier extremely (σ is over 4), performance starts to downgrade, but such noise multiplier is over-strict in practical privacy degree.

Table 2: Datasets and global models in the experiments.

| Dataset    | Model (#Params) | #Label | #Record (Test) |
|------------|----------------|--------|----------------|
| MNIST      | MLP (50890)    | 10     | 70000 (10000)  |
| CIFAR10    | MLP (197320)   | 10     | 60000 (10000)  |
|            | CNN (62006)    |        |                |
| Purchase100| MLP (44964)    | 100    | 144000 (24000) |
| CIFAR100   | CNN (201588)   | 100    | 60000 (10000)  |

3https://github.com/FumiyukiKato/FL-TEE
Figure 4: Attack results on datasets with a fixed number of labels: Vulnerable, especially when there are few labels.

Figure 5: Attack results on datasets with a random number of labels (more difficult setting): When the number of labels is small, the attacker can attack the client without knowing the number of labels.

Figure 6: Attack results with variable sparse ratio: Higher the sparsity, the more successful the attack tends to be.

Figure 7: Attack results with variable noise multiplier $\sigma$. At realistic noise scales, the attack performance remains high.

Figure 8: Cacheline-level leakage on CNN of CIFAR10: Attacks are possible with slightly less accuracy.

Figure 8 shows the result with the index information observed at cacheline granularity (64B) that can be easily observed against SGX. We use CIFAR10, CNN and $\sigma = 0.1$. The accuracy is slightly lower than the case where all indices are observed, but the attack is still possible, which indicates that well-known vulnerability of Intel SGX can be sufficient to complete attack at least.

5 OBLIVIOUS ALGORITHMS

We have clarified the problems of applying a normal Linear algorithm in the sparsified gradients setting. In this section, we focus on the aggregation algorithm and discuss how the attacks can be prevented. The implementation details are based on Intel SGX. The notations used here are the same as those in Section 3.3.

First, we introduce the general ORAM-based method. We initialize the ORAM with $d$ zero values for aggregated parameters $g^*$, sequentially update the values with the received $nk$ gradients $g$, and finally retrieve the $d$ values from the ORAM. Since ORAM completely hides memory access to $g^*$, eventually, the algorithm is fully oblivious. However, as we will show in the experimental section, even the state-of-the-art PathORAM adapted to the TEE [57] has a large overhead, and a task-specific algorithm is desirable.

5.1 Baseline method

A simple way to accomplish the fully oblivious algorithm is to access all memory addresses to hide an access to a specific address.
Algorithm 3 Baseline

Input: \( g = g_1 \ldots g_n \): concatenated gradients, \( nk \) length
Output: \( g' \): aggregated parameters, \( d \) length
1: initialize aggregated gradients \( g' \)
2: for each \((idx, val) \in g\) do
3: \( c \) is the number of weights included in one cacheline */
4: \( offset \) indicates the position of idx in the cacheline */
5: for each \((idx, val) \in g'\) if idx \( \equiv \) offset (mod c) do
6: \( flag \leftarrow idx' = idx \)
7: \( val' \leftarrow o\_mov(flag, val', val + val) \)
8: write \( val' \) into \( idx' \) of \( g' \)
9: return \( g' \)

Algorithm 4 Advanced

Input: \( g = g_1 \ldots g_n \): concatenated gradients, \( nk \) length
Output: \( g' \): aggregated parameters, \( d \) length
1: /* initialization: prepare zero-valued gradients for each index */
2: \( g' \leftarrow (1, 0, \ldots, d, 0) \) \(\forall \) values are zero
3: \( g \leftarrow g \parallel g' \)
4: /* oblivious sort in \( O((nk + d) \log^2 (nk + d)) \) */
5: oblivious sort \( g \) by index
6: /* oblivious folding in \( O((nk + d)) \) */
7: \( idx \leftarrow \) index of the first weight of \( g \)
8: \( val \leftarrow \) value of the first weight of \( g \)
9: for each \((idx', val') \in g\) do
10: \( flag \leftarrow idx' = idx \)
11: \( M_0 \) is a dummy index and very large integer */
12: \( idx_{prior}, val_{prior} \leftarrow o\_mov(flag, idx, val, (M_0, 0)) \)
13: write \( (idx_{prior}, val_{prior}) \) into \( idx' - 1 \) of \( g \)
14: \( idx, val \leftarrow o\_mov(flag, idx', val'), (idx, val + val') \)
15: /* oblivious sort in \( O((nk + d)) \log^2 (nk + d)) \) */
16: oblivious sort \( g \) by index again
17: return the first \( d \) values as \( g' \)

In other words, when accessing \( G'[i] \), the dummy accesses are performed to \( G'[j] \) for each \( j \in [d] \). For each access, either of a dummy or updated true value is written to them, and the timing of writing of the true value is hidden by oblivious move \( o\_mov \). The baseline algorithm is described in Algorithm 3. It takes as input the concatenated gradients sent from all participants \( g(nk\text{-dimensional vector}) \), and returns the aggregated gradients \( g' \) \( (d\text{-dimensional vector}) \). We linearly access \( G' \) a number of times of length \( G \). In SGX, at least in the scope of the currently proposed attacks, the observed memory address is the granularity of the cacheline, so we optimize for cacheline size, where, for example, if the weight is \( 4 \) byte \((32\text{-bit floating point})\) and cacheline is \( 64 \) byte, we can get up to \( 16x \) speedup. Regardless of the constant times optimization, the complexities are \( O(nkd) \) in time and \( O(nk + d) \) in space.

**Proposition 5.1.** Algorithm 3 is (cacheline-level) fully oblivious. (The formal proof is provided in the Appendix C.)

### 5.2 Advanced method

Here, we present a more advanced approach to FL aggregation. In cases where the number of model parameters is large, \( k \) and \( d \) are huge factors, and the computational complexity of the Baseline method becomes very large due to the product of \( k \) and \( d \). As described in Algorithm 4, we design a more efficient Advanced algorithm by carefully analyzing the operations on the gradients. Intuitively, our method is designed to compute \( g' \) directly from the operations on gradient data \( g \) to eliminate access to each memory address of the aggregated gradients \( g' \). This avoids the overhead of dummy accesses to \( g' \) as Baseline. The method is divided into four main components: initialization on gradients vector \( g \) (Line 1), oblivious sort (Line 4), oblivious folding (Line 6) and a second oblivious sort (Line 16). For oblivious sort, we use Batch’s Bitonic Sort [8], which we implemented in a register-level oblivious manner by oblivious swap \( o\_swap \) to obliviously compare and swap at all comparators in the sorting network.

We show a simple running example at \( n = 3 \) \((#user), k = 2 \) \((#sparsified dimension), d = 4 \) \((#dimension)\).

| Input | \( g_1 = \{1.0, 2.0, 4.0, 5.0\} \) |
|-------|----------------------------------|
| \( g_2 = \{2.0, 6.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
| Line 1-3 | \( g' = \{1.0, 1.0, 2.0, 6.0\} \) |
| \( g = \{1.0, 1.0, 2.0, 6.0\} \) | \( \parallel \) |
| \( g' = \{1.0, 2.0, 4.0, 5.0\} \) | \( g_3 = \{1.0, 1.4\} \) |
As mentioned in Section 2.2, accessing data outside the EPC in-
Although it can be improved to Advanced does not change its security charac-
participating in the round are controlled by enclave through se-
control the order of the groups, but it is secure because the users
been completed, the result is averaged and perturbed, then, loaded
carried over to the next group, and (4) only when all groups have
The access pattern generated here is also identical for any input data of equal length. Finally, Accesses$^{\text{advanced}}$ is identical independent of any input of equal length and this implies 0-statistical oblivious.

The complexities of the whole operation are $O((nk+d) \log^2 (nk+d))$ in time and $O(nk+d)$ in space. Our algorithm relies on oblivious sort, which dominates the asymptotic computational complexity. We use Batcher’s Bitonic Sort [8], which has $O(n \log^2 n)$ time complexity. Although it can be improved to $O(n \log n)$ by using AKS Sorting Network [2], they have a huge constant hidden in asymptotic analysis, therefore we use Batcher’s sort and denote computational complexity by it. The Advanced is asymptotically better than the Baseline due to the elimination of the $kd$ term. We also demonstrate empirically significant advantages in the experiment.

### 5.3 Optimization

Here, we describe an optimization that fit the basic SGX memory characteristics. Current SGX has two major levels of optimization for memory size. One is the size of the L3 cache (8 MB in our experiment). In SGX, the speedup is more significant because cache hit reduces not only the memory access time but also the data decrypting process. Second is the EPC size (96 MB in our experiment). As mentioned in Section 2.2, accessing data outside the EPC incurs serious paging overhead. Considering our methods, Baseline is computationally expensive, but most of the accesses to memory are linear. This is greatly accelerated by high cache hit rates and CPU’s prefetch functionality. On the other hand, in Advanced, because of the low locality of memory accesses in Batcher’s sort, the cache and EPC hit rate could be much lower.

We, therefore, implement an optimization by introducing a function to split users into appropriate groups before Advanced is executed. The procedure is as follows: (1) divide into groups of $h$ users each, (2) aggregate values for each group using Advanced, (3) the aggregated value is recorded in enclave and the result is carried over to the next group, and (4) only when all groups have been completed, the result is averaged and perturbed, then, loaded from enclave to untrusted area. It is important to note that the improvement to Advanced does not change its security characteristics. The untrusted server will be able to observe and even control the order of the groups, but it is secure because the users participating in the round are controlled by enclave through secure sampling, and it can abort if cheated. The key parameter is the number of people $h$ in each group. The overall computational complexity deteriorates slightly from $O((nk+d) \log^2 (nk+d))$ to $O(n/h((hk+d) \log^2 (hk+d)))$. However, this hides the speedup by cache hits and/or overhead of repeated data loading. Basically, lowering $h$ allows the benefit of cache hit, while lowering $h$ too much causes the huge overhead from repeated data loading. $h$ is independent of the data content and can be obtained experimentally offline and we confirm that there exists an optimal $h$ that achieves highest efficiency using real-world data.

### 5.4 Balancing Oblivious and Performance

We investigate further improvements by relaxing the full obliviousness to obtain more efficient aggregation algorithms. Note that the relaxation of obliviousness should not affect the training of the model and degrade the utility.

The relaxed security definition that has received attention recently, is Differentially Oblivious (DO) [4, 14, 17, 42]. Following the definition of Section 2.3, $(\epsilon, \delta)$-DO means that for any neighboring inputs $I, I'$, it holds that

$$\Pr[\text{Accesses}^M(\lambda, I)] \leq e^\epsilon \Pr[\text{Accesses}^M(\lambda, I')] + \delta.$$  

Basically, DO is DP applied to obliviousness, and shows an asymptotic improvement in computational complexity from full obliviousness. Not only in theory but also in practice, for example, improvements in efficiency have been reported for private set intersections [28]. However, DO is unlikely to work in our FL setting.

It is common for DO approaches to guarantee DP for the histogram of observed memory accesses. The methods proposed in [4, 42] are applicable to our scenario. The procedure is as follows: pad dummy data, perform an oblivious shuffle, and then update $g^*$ by performing linear access. The observed memory access pattern is equivalent to a histogram of the indices of all gradients. Dummy data should be padded as random noise enough to make this histogram DP. However, this can incur prohibitive costs in the FL setting. The first reason is that the randomization mechanism can only be implemented by padding dummy data [13], which means that only one-sided noise can be added, multiplying the noise scale by a constant, and the algorithms that can be represented by padding are very limited. The second reason is critical in our setting and different from previous works [4, 42]. Considering that the ML model dimension $d$ and even the sparsified dimension $k$ can be large, the noise easily becomes large. For example, consider DO guaranteed by Laplace noise, where $k$ is the sensitivity and $d$ is the dimension of the histogram, implying that the amount of noise is proportional to $kd$ and multiplied by some constant. It results in too large padded data to allocated in TEE’s strict memory resources. This ends up generating a larger amount of computation than the fully oblivious case by following oblivious shuffle or sorting.

### 5.5 Experimental results

In this section, we show how efficiently our designed defense works on a practical scale. Since it is obvious that the proposed algorithms provide a complete defense against our attack method, we do not evaluate its attack performance here.

**Setup:** We use an HP Z2 G4 Workstation with a 4-core 3.80 GHz Intel Xeon E-2174G CPU, 64 GB RAM, and 8 MB L3 cache, which supports the SGX instruction set and has 128 MB processor reserved memory (PRM) in which 96 MB EPC is available for user.

---

$^*$We have confirmed this prohibitive cost in our preliminary experiments.
We implement PathORAM based on the open source library when the number of parameters is very small is the This experimental code is open as well.\(\alpha\) various numbers of clients and low sparsity \(n\) = \(\alpha = 0.01\) and \(n\) (number of clients per round) = 100.

use. The datasets used for the evaluation are in Table 2 and also the synthetic dataset. Note that our method is fully oblivious, and the efficiency does not depend on the data content at all. The aggregation methods are Non Oblivious (Linear algorithm in Section 3.3), Baseline (Algorithm 3), Advanced (Algorithm 4), and PathORAM. We implement PathORAM based on the open source library\(^2\), which is a Rust implementation of Zerotrace [57]. The stash size is fixed at 20.

In this experiment, we use the execution times of the aggregation operation as an efficiency metric. We measure the time taken by an untrusted server to receive a request, pass the encrypted gradient to enclave, decrypt, aggregate, perturbate in the enclave, and return. This experimental code is open as well.

**Results:** Figure 10 shows the execution time plotted for the aggregation operation on synthetic data with different gradient sizes. \(\alpha\) is fixed at 0.01, and the horizontal axis indicates the original model parameter size \(d\). Our proposed Advanced method is approximately one order of magnitude faster than the Baseline method. Moreover, it is more robust to an increase in the number of parameters. Only when the number of parameters is very small is the Baseline method faster than the Advanced method, but this is because the overall execution time is instantaneous and the overhead of complicated operations such as oblivious sort is significant. PathORAM has a large overhead. The main factors are the refresh (randomization) for each update and the oblivious reading of the position maps along with other components. This result indicates that the aggregation process can be completed in a few seconds if the model scale is about 1M parameters, and our proposed method has practical performance on the real-world scale from parameter size of the global models in Table 2.

Figure 11 shows the performance results on MNIST (MLP) for various numbers of clients and low sparsity \(\alpha = 0.1\). \(q\) is fixed at 0.3 for all case later. When the number of clients \(N\) is large \(10^4\), the Baseline method is more efficient. The reason is that lower sparsity and more clients increase \(nk\), which has a larger overhead for both Baseline and Advanced, but affects Advanced more, which follows the analysis on cache hits in Section 5.3. At \(N = 10^4\), the memory size majorly required by Advanced is (vector to be obliviously sorted) = \(5089 \times 8 + 3000 + 50890 \times 8 \approx 122\) MB.

\(^2\)https://github.com/mobilecoinofficial/mc-oblivious

Figure 12 shows the effects of optimization for Advanced algorithm on MLP models for MNIST (left) and CIFAR100 (right). (\(> 96\) MB of EPC size) since each cell of gradient is 8 bytes (32-bit unsigned integer for index and 32-bit floating point for value). Batcher’s sort requires repeated accesses between two very distant points on the vector, which could cause a large number of paging operation until Advanced finishes, but in Baseline it hardly occurs. Moreover, since the model size \(d\) is fairly small (i.e., MNIST (MLP) has only 50K), the overhead of the dummy accesses of Baseline is not significant. However, our optimization introduced in Section 5.3 successfully utilizes these observations.

Figure 12 shows the effects of the optimization on Advanced. The left figure shows the results for the same settings as the rightmost graph (\(N = 10^4\)) in Figure 11, showing that Advanced is dramatically faster with optimal client size. First, when the number of clients \(h\) is extremely finely divided, iterative loading and other minor costs become non-negligible, and the overhead conversely increases. However, if \(h\) is gradually increased, execution time becomes several times more efficient. Considering that the size of the L3 cache is 8 MB and that \(dx = 0.04\) MB of gradient data per client, the L3 cache can hold up to about 200 clients data to be sorted. In fact, the cost per plot is the lowest, roughly 10 seconds at \(h = 100\) and it is about the same up to around 200, a significant improvement from 290 seconds of original Advanced. The waviness of the plot seems to be related to the data size and the L2 cache (1 MB), which...
does not seem to have as large an impact as the L3 cache. However, the efficiency drops significantly around approximately $h = 2000$, which is due to the overhead of EPC paging and might be a major consideration of the non-linear memory accesses in Advanced. The right figure shows the results with MLP on CIFAR100 at $\alpha = 0.01$, $N = 10^4$. In this case, Advanced is much faster originally, but there is optimal $h$ that can be further accelerated by optimization. While the execution time is 16 seconds without optimization, it reduces to 5.7 seconds at around 150 clients. Note that, in practice, the optimal $h$ can be determined through offline evaluation.

6 RELATED WORKS

Security and Privacy threats in FL. The third parties who obtain machine learning models may infer sensitive information about participants’ training data by inference attacks [49, 68] and/or reconstruction attacks [9, 30]. Inference attacks estimate whether a specific sample is included training data, or estimate sensitive properties for the training data. Reconstruction attacks estimate the training data itself (e.g., image data). These attacks have been studied for an extended period of time, not only in the context of FL, but also in the context of ML. However, in the case of FL, the importance of risk assessment has increased because white-box model access is easily available to all participants during the training process. To address these attacks against malicious third parties, perturbing the model training with differential privacy has been shown to be effective [48, 65], and a typical method is DP-SGD [1], which injects Gaussian noise into the SGD steps to guarantee DP. Furthermore, [15, 46, 74] proposes using TEE on the client side to disallow malicious behavior.

The aggregation server in FL also leads to privacy concerns. FedAVG [43] requires participants to share the raw gradients with the server, which has been shown to be a potential privacy violation [76, 78]. Using a technique known as Deep Leakage from Gradient, the raw gradient information can be used to reconstruct important information about the training data [78]. [49] reports a large scale membership inference attack on FL. To protect against such attacks, [63, 77] propose perturbing the gradients guaranteeing LDP. Another direction is to hide gradients using secure computation techniques such as homomorphic encryption [6, 22, 29, 33] and secure aggregation [11, 20, 34]. Although the server cannot see the raw gradient, these methods may not always be acceptable to practitioners because of their high computational costs and/or complex initialization and synchronization requirements, which also restrict the available DP mechanisms. In addition, these defenses themselves are not resistant to side channels leaking gradient index information in sparsified settings.

Olive can protect against both the untrusted server and any malicious third parties. Olive employs TEE to address the privacy concerns caused by an untrusted server and uses differential privacy to provide formal privacy guarantee against third parties. More importantly, Olive hardens the security guarantee on TEE with oblivious algorithms for DP-FL.

FL with TEE. FL with TEE is a promising direction for efficient and secure FL. In addition to confidentiality on gradients, it can provide remote program integrity and verifiability by remote attestation. Recently, multiple proposals have been made to use TEE in FL [16, 46, 47, 73, 75]. PPFL [46] uses TEE to hide the parameters to prevent semi-honest client and server attacks on the global model. However, side channel attacks are not covered. In [75] and [16], using multiple servers, the gradient aggregation step is hierarchical and/or partitioned so that the gradient information can only be partially observed by each server. Their assumption is the reconstruction attack based on raw gradient information, where gradient leakage of less than 80% is acceptable. Our attack is based on gradient index information, and the goal is label inference [68]; hence, the analysis and solution are completely different. Furthermore, our proposed method is more practical since we require only one server and one TEE, compared to their method of distributed processing, which assumes multiple non-colluding servers with TEEs. Flattee [47] is similar to Olive in that it uses TEE and DP to learn FL. [47] mentions server-side obliviousness, but does not provide any solutions for side channel resistance to gradient index leakage. Our study includes the analysis of access patterns in FL’s aggregation procedure and the design and demonstration of attack methods to thoroughly motivate our defenses in addition to the specific solutions which leads to stronger security than any other methods in FL running on a single central TEE.

Differential Privacy in FL. The combination of FL and DP has been well studied [5, 24, 40, 41, 44, 63, 69, 71, 77]. LDP and shuffle DP can be applied to improve the trust model of CDP-FL that requires trusted server, as introduced in Section 2.1.

In LDP-FL [41, 63, 69, 77], the clients perturb the gradients before sharing with an untrusted server, guaranteeing formal privacy against both malicious third parties and the untrusted server. LDP-FL does not require a trustful server unlike CDP-FL. However, LDP-FL suffers from lousy privacy-utility trade-off, especially when the number of users is not sufficient (i.e., the signal is drowned in noise) or the number of the model parameters is large (i.e., more noises are needed for achieving the same level of DP). Unfortunately, it is limited to models with an extremely small number of parameters or companies with a huge user base (e.g., 10 million).

To overcome the weakness of the utility of LDP by privacy amplification, a method using the shuffler model [7, 21], has been proposed [40], i.e., Shuffle DP-FL. This method introduces a trusted shuffler instead of trusting the server and achieves some level of utility. However, clearly, it cannot outperform CDP in utility because we can simulate the shuffler mechanism on a trusted server. The privacy amplification of the shuffler also has weaknesses, such as the need for a large number of participants and small parameter size due to the underlying LDP limitation. In Olive, the operations applied in CDP-FL can be performed in TEE, and the utility is basically the same as the CDP case, without trusting the server using secure computation with TEE. These are summarised in Table 1.

Sparsification for communication cost. The models to be trained in FL are expected to include deep models such as image recognition and language models, and the parameter size can be large. Since the participants and the aggregation server need to exchange these parameters in each training round, one of the major problems inherent in FL is bandwidth [59]. To solve this issue, sparsification of the model parameters before sending and then aggregating at the server has attracted attention [31, 38, 39, 56, 60]. By sending only the top-$k$ parameters with large absolute gradients to the aggregation server, we can reduce the communication cost.
by more than 1–3 orders of magnitude. Moreover, interestingly, this could reduce the sensitivity to the aggregated gradients, which contributes to reducing the amount of noise [31, 40]. Specifically, in the case of CDP-FL, when a constant clipping size is used, the (top-k) sparsification leaves larger values after clipping, which increases the utility at the same privacy. In our study, we focus on demonstrating the privacy risk to side channels posed by sparsification.

7 CONCLUSION

In this study, we proposed OLIVE, a federated learning system that can benefit from the utility of CDP-FL and the trust model of LDP-FL, which is accomplished using TEE. We analyzed the security/privacy risks of using TEE on the server side in FL and demonstrated these risks, we proposed a fully-oblivious but efficient algorithm. Our experimental results showed that our proposed algorithm was more efficient than state-of-the-art general-purpose ORAM, and can be a practical method on the real-world scale. We believe our method can be a practical and powerful technique for secure FL on cloud servers with Intel SGX.

REFERENCES

[1] Martin Abadi, Andy Chu, Ian Goodfellow, H Brendan McMahan, Ilya Mironov, Kunal Talwar, and Li Zhang. 2016. Deep learning with differential privacy. In Proceedings of the 2016 ACM SIGSAC conference on computer and communications security. 308–318.

[2] Miklos Ajtai, Janos Komlos, and Endre Szemeredi. 1983. An O (n log n) sorting network. In Proceedings of the fifteenth annual ACM symposium on Theory of computing. 1–9.

[3] Mohammed Aledhari, Rehma Us Sami, and Basak Gulder. 2021. Sparsified secure aggregation for privacy-preserving federated learning. arXiv preprint arXiv:2112.12872 (2021).

[4] Joshua Allen, Bolin Ding, Janardhan Kulkarni, Harsha Nori, Olga Ohrimenko, and Sergey Yekhanin. 2019. An algorithmic framework for differentially private data analysis on trusted processors. Advances in Neural Information Processing Systems 32 (2019).

[5] Gulen Andrew, Om Thakkar, H Brendan McMahan, and Swaroop Ramaswamy. 2021. Differentially Private Learning with Adaptive Clipping. Advances in Neural Information Processing Systems (NeurIPS 2021) (2021).

[6] Yoshinori Aono, Takuya Hayashi, Lihua Wang, Shiho Moriai, et al. 2017. Privacy-preserving deep learning via additively homomorphic encryption. IEEE Transactions on Information Forensics and Security 13, 5 (2017), 1333–1345.

[7] Borja Balle, James Ball, Adrià Gascón, and Kobbi Nissim. 2019. The privacy blanket of the shuffle model. In Annual International Cryptology Conference. Springer, 638–667.

[8] Kenneth E Batcher. 1968. Sorting networks and their applications. In Proceedings of the 33rd annual ACM symposium on Theory of computing. 1–9.

[9] Irem Ergun, Hasin Us Sami, and Fatih S. Akyol. 2021. An Algorithmic Framework for Differentially Private Federated Learning. In Proceedings of the 33rd International Conference on Distributed Computing Systems (ICDCS 2021), IEEE, 1–10.

[10] Yuan Fang, Yu Chen, Tao Xiang, Zheli Liu, and Jin Li. 2020. A training-inference privacy-preserving federated learning scheme with trusted execution environment. Information Sciences 522 (2020), 69–79.

[11] Pau-Chen Cheng, Kevin Eykholt, Zhongshu Gu, Hani Jamjoom, K. R. Jayaraman, Danica K. Ziegler, and Ashley Verma. 2021. Separation of Powers in Federated Learning (Poster Paper). In Proceedings of the First Workshop on Systems Challenges in Reliable and Secure Federated Learning (Virtual Event, Germany) (ResilientFL ’21). Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3477114.3488765

[12] Shumu Chen, Danyang Zhuo, Elaine Shi, and THHubert Chan. 2021. Differentially-Oblivious Database Joins: Overcoming the Worst-Case Curve of Fully Oblivious Algorithms. Cryptology ePrint Archive (2021).

[13] Victor Costan and Srinivas Devadas. 2016. Intel sgx explained. IACR Cryptol. ePrint Arch. 2016, 86 (2016), 1–18.

[14] Cynthia Dwork. 2006. Differential privacy. In Proceedings of the 33rd international conference on Automata, Languages and Programming-Volume Part II. Springer-Verlag, 1–12.

[15] Yu Chen, Fang Luo, Tong Li, Tao Xiang, Zheli Liu, and Jin Li. 2020. A training-integrity privacy-preserving federated learning scheme with trusted execution environment. Information Sciences 522 (2020), 69–79.

[16] Victor Costan and Srinivas Devadas. 2016. Intel sgx explained. IACR Cryptol. ePrint Arch. 2016, 86 (2016), 1–18.

[17] Shumu Chen, Danyang Zhuo, Elaine Shi, and THHubert Chan. 2021. Differentially-Oblivious Database Joins: Overcoming the Worst-Case Curve of Fully Oblivious Algorithms. Cryptology ePrint Archive (2021).

[18] Victor Costan and Srinivas Devadas. 2016. Intel sgx explained. IACR Cryptol. ePrint Arch. 2016, 86 (2016), 1–18.

[19] Cynthia Dwork. 2006. Differential privacy. In Proceedings of the 33rd international conference on Automata, Languages and Programming-Volume Part II. Springer-Verlag, 1–12.

[20] Irem Ergun, Hasin Us Sami, and Basak Gulder. 2021. Sparsified secure aggregation for privacy-preserving federated learning. arXiv preprint arXiv:2112.12872 (2021).

[21] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Kunal Talwar, and Abhradeep Thakurta. 2019. Amplification by shielding: From local to central differential privacy via anonymity. In Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms. SIAM, 2468–2479.

[22] Haokun Fang and Quan Qian. 2021. Privacy Preserving Machine Learning with Homomorphic Encryption and Federated Learning. Future Internet 13, 4 (2021), 94.

[23] Shufan Fei, Zheng Yan, Wenshuang Li, and Haoming Xie. 2021. Security vulnerabilities of SGX and countermeasures: A survey. ACM Computing Surveys (CSUR) 54, 6 (2021), 1–36.

[24] Robin C Geyer, Tassilo Klein, and MoNa Nahi. 2017. Differentially private federated learning: A client level perspective. NDSS Workshop: Machine Learning on the Phone and other Consumer Devices (2017).

[25] Antonious Girgis, Deepesh Data, Suhas Diggavi, Peter Kairouz, and Ananda Theertha Suresh. 2021. Shuffled Model of Differential Privacy in Federated Learning. In International Conference on Artificial Intelligence and Statistics. PMLR, 2521–2529.

[26] Michelle Goddard. 2017. The EU General Data Protection Regulation (GDPR): European regulation that has a global impact. International Journal of Market Research 59, 6 (2017), 703–705.

[27] Oded Goldreich. 1987. Towards a theory of software protection and simulation by oblivious RAMs. In Proceedings of the nineteenth annual ACM symposium on Theory of computing. 182–194.

[28] Adel I. Ghorbal, Ralf Kuester, and Mike Rosulek. 2018. Cheaper private set intersection via differentially private leakage. Proceedings on Privacy Enhancing Technologies 2019, 3 (2019).

[29] Adam Groce, Jure Leskovec, and Mike Rosulek. 2019. Private set intersection via differentially private leakage. Proceedings on Privacy Enhancing Technologies (2019).

[30] Meng Han, Hongwei Li, Guowen Xu, Sen Liu, and Haomiao Yang. 2019. Towards efficient and privacy-preserving federated deep learning. In ICC 2019-2019 IEEE International Conference on Communications (ICC). IEEE, 1–6.

[31] Briland Hitaj, Giuseppe Ateniese, and Fernando Perez-Cruz. 2017. Deep models under the GAN: Information leakage through collaborative deep learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. 603–618.

[32] Fui Hu, Yamun Gong, and Yuanxiao Guo. 2021. Federated Learning with Sparseification-Amplified Privacy and Adaptive Optimization. In JFCAI.

[33] Bavgar Jayaraman and David Evans. 2019. Evaluating Differentially Private Machine Learning in Practice. In 28th USENIX Security Symposium (USENIX Security 19). USENIX Association, Santa Clara, CA, 1895–1912. https://www.usenix.org/conference/usenixsecurity19/presentation/jayaraman

[34] Zhifeng Jiang, Wei Wang, and Yang Liu. 2021. Flash: Additively symmetric homomorphic encryption for cross-silo federated learning. arXiv preprint arXiv:2109.06075 (2021).

[35] Peter Kairouz, Ziyu Liu, and Thomas Steinke. 2021. The Distributed Discrete Gaussian Mechanism for Federated Learning with Secure Aggregation. In Proceedings of the 38th International Conference on Machine Learning, ICML 2021, 18–24 July 2021, Virtual Event (Proceedings of Machine Learning Research), Marina Meila and Tong Zhang (Eds.). Vol. 139. PMLR, 5201–5212. http://proceedings.mlr.press/v139/kairouz21a.html

[36] Fumiuki Kato, Yang Cao, and Masatoshi Yoshikawa. 2021. PCT-TEE: Trajectory-based Private Contact Tracing System with Trusted Execution Environment. ACM Transactions on Spatial Algorithms and Systems (TOSAS) 8, 2 (2021), 1–35.

[37] Jakub Konečný, H Brendan McMahan, Felix X Yu, Peter Richtárik, Ananda Theertha Suresh, and Dave Bacon. 2016. Federated learning: Strategies for improving communication efficiency. arXiv preprint arXiv:1603.05400 (2016).

[38] Sangho Lee, Ming-Wei Shih, Prasun Gera, Tae soo Kim, Hyesson Kim, and Marcus Peinado. 2017. Inferring fine-grained control flow inside SGX enclaves with branch shadowing. In 26th USENIX security symposium (USENIX security 17).
KIMBERLY Powell. 2019. Nvidia clara federated learning to deliver ai to hospitals.

Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. 2020. A survey.

Sajin Sasy, Sergey Gorbunov, and Christopher W. Fletcher. 2018. ZeroTrace: Oblivious Memory Primitives from Intel SGX. In 35th USENIX Security Symposium (USENIX Security 16). USENIX Association, 1–16. https://doi.org/10.5440/1.5893150

Mohammad Naseri, Jamie Hayes, and Emiliano De Cristofaro. 2020. Toward Leaky Page Invariants and SGX. In Proceedings of the 2020 IEEE Symposium on Security & Privacy. IEEE, 429–445. https://doi.org/10.1109/SP424.2020.9155414

Osama Shahid, Seyyedamin Pouriyehe, Reza Meimandi Parizi, Quan Z. Sheng, Gautam Srivastava, and Liang Zhao. 2021. Communication Efficiency in Federated Learning: Achievements and Challenges. ArXiv abs/2107.10996 (2021).

Shaoxuai Shi, Kaiyong Zhao, Qiang Wang, Zhenhong Tang, and Xiaowen Chu. 2020. A Convergence Analysis of Distributed SGD with Communication-Efficient Gradient Sparsification. In ICML '20: Proceedings of the 37th International Conference on Machine Learning. PMLR, 3411–3419.

Reza Shokri, Mario Stronati, Congzheng Song, and Vitaly Shmatikov. 2017. Membership inference attacks against machine learning models. In 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 3–18.

Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyu Yu, and Srinivas Devadas. 2013. Path ORAM: An Extremely Simple Oblivious RAM Protocol. In Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security (CCS ’13). Association for Computing Machinery, New York, NY, USA, 299–310. https://doi.org/10.1145/2508859.2516660

Lichao Sun, Jianwei Qian, and Xun Chen. 2021. LDP-FL: Practical Private Aggregation in Federated Learning with Local Differential Privacy. In Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence, IJCAI-21, Zhu-Hua Zhou (Ed.). International Joint Conferences on Artificial Intelligence Organization, 1571–1578. https://doi.org/10.24963/ijcai.2021/217. Main Track.

Mesya Tsantrizos, Ali Shafiei, and Rajeev Balaubaramonian. 2018. VAIL: Reducing paging overheads in SGX with efficient integrity verification structures. In Proceedings of the Twenty-Third International Conference on Architectural Support for Programming Languages and Operating Systems. 665–678.

Alexander M Turner et al. 2019. Exploring the landscape of backdoor attacks on deep neural network models. Ph.D. Dissertation. Massachusetts Institute of Technology.

Jo Van Bulcke, Frank Piessens, and Raul Strackx. 2018. Nemesis: Studying microarchitectural timing leaks in rudimentary CPU interrupt logic. In Proceedings of the 2018 ACM Conference on Computer and Communications Security. 178–195.

Jo Van Bulcke, Nico Weischedel, Rüdiger Kapitzka, Frank Piessens, and Raul Strackx. 2017. Telling your secrets without page faults: Stealthy page table-based attacks on enclave execution. In 26th USENIX Security Symposium (USENIX Security 17). 1041–1056.

Aydin Wainahk, Fahriyo Ventola, Till Müllß, Jens Keim, Carlos Garcia Cordero, Efpraim Zimmer, Tim Grube, Kristian Kersting, and Max Mühlhäuser. 2022. User-Level Label Leakage from Gradients in Federated Learning. Proceedings on Privacy Enhancing Technologies 2022, 2 (2022), 227–244.

Teng Wang, Xuelfeng Zhang, Jingyu Feng, and Xinyu Yang. 2020. A Comprehensive Survey on Local Differential Privacy toward Data Statistics and Analysis. Sensors 20, 24 (Dec 2020), 7030. https://doi.org/10.3390/s20247030

Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. 2014. Oblivious data structures. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 215–226.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard Y Huang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. 2020. Federated learning with differential privacy: Algorithms and performance analysis. IEEE Transactions on Information Forensics and Security 15 (2020), 3454–3469.

Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In 2015 IEEE Symposium on Security and Privacy. IEEE, 640–656.

Chengliang Zhang, Ran Chen, Shengwei Wang, and Xianmin Liu. 2015. Inference attacks on enclaved execution. In Proceedings of the 2015 Conference on Computer and Communications Security (CCS ’15). ACM, 885–896.

Yang Zhao, Jun Zhao, Mengmeng Yang, Teng Wang, Ning Wang, Lingjuan Lyu, Yafei Zhou, Tony Qin, Jinjian Zheng, and Ling Fan. 2022. User-Level Label Leakage from Gradients in Federated Learning. Proceedings on Privacy Enhancing Technologies, 2 (2022), 227–244.

Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. 2014. Oblivious data structures. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 215–226.

Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In 2015 IEEE Symposium on Security and Privacy. IEEE, 640–656.

Chengliang Zhang, Ran Chen, Shengwei Wang, and Xianmin Liu. 2015. Inference attacks on enclaved execution. In Proceedings of the 2015 Conference on Computer and Communications Security (CCS ’15). ACM, 885–896.

Xiao Shaun Wang, Kartik Nayak, Chang Liu, TH Hubert Chan, Elaine Shi, Emil Stefanov, and Yan Huang. 2014. Oblivious data structures. In Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security. 215–226.

Kang Wei, Jun Li, Ming Ding, Chuan Ma, Howard Y Huang, Farhad Farokhi, Shi Jin, Tony QS Quek, and H Vincent Poor. 2020. Federated learning with differential privacy: Algorithms and performance analysis. IEEE Transactions on Information Forensics and Security 15 (2020), 3454–3469.

Yuanzhong Xu, Weidong Cui, and Marcus Peinado. 2015. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In 2015 IEEE Symposium on Security and Privacy. IEEE, 640–656.

Chengliang Zhang, Ran Chen, Shengwei Wang, and Xianmin Liu. 2015. Inference attacks on enclaved execution. In Proceedings of the 2015 Conference on Computer and Communications Security (CCS ’15). ACM, 885–896.
A OBLIVIOUS PRIMITIVES

Here we describe the detailed implementation of the oblivious primitive we used. The C inline assembler-like pseudo-code is shown here. However, the Rust implementation we actually used is available in the public repository.

```c
int o_swap(bool flag, uint64 x, uint64 y) {
    /* inline assembly */
    /* register mapping: */
    flag => rax, x => rdx, y => r8 */
    mov rax, rdx
    test rax, -1
    cmovz rax, r8
    return r8
}
```

**Listing 1:** Oblivious move based on CMOV

```c
int o_mov(bool flag, uint64 x, uint64 y) {
    /* inline assembly */
    /* register mapping: */
    flag => rax, x => rdx, y => r8 */
    mov rax, rdx
    test ecx, -1
    cmovnz r10, r11
    cmovnz r9, r10
    mov r10, r8
    return rax
}
```

**Listing 2:** Oblivious swap based on CMOV

B GENERAL FL AGGREGATION ALGORITHM

We show a general FL aggregation algorithm. The main focus here is on which memory addresses are accessed in the operation.

**Algorithm 5** Linear algorithm (and averaging and perturbing)

**Input:** $G = G_1 \parallel \ldots \parallel G_n$ where $G_p$ $(p \in [n])$ is gradient from user $p$ and $k$ length vector, $G$ is $nk$ length vector and $G$'s element $g_{pq}$ $(q \in [nk])$ is composed of (index, value)

**Output:** $G'$: Aggregated gradient and $d$ length vector

1. procedure Aggregation($G$)
2. /* Linear algorithm */
3. Initialize gradients $G'$
4. for $i = 1, \ldots, n$ do
5.   for $j = 1, \ldots, k$ do
6.     $G'[G[k \ast (i-1) + j].index] += G[k \ast (i-1) + j].value$
7. /* Averaging and Perturbing with linear access */
8.   for $i = 1, \ldots, d$ do
9.     $G'[i] /= n$
10.  for $i = 1, \ldots, d$ do
11.    $z \leftarrow$ Random noise (e.g., Gaussian distribution)
12.   $G'[i] += z$
13. return $G'$

C PROOFS OF OBLIVIOUSNESS

Proof of Proposition 3.1.

Proof. Let the access pattern of Linear algorithm for dense gradients be $Accesses_{\text{dense}}$, then, the pattern is represented as follows:

$Accesses_{\text{dense}} = \{[G[1], \text{read}, +], (G'[1], \text{write}, +), \ldots, (G[nd], \text{read}, +), (G'[d], \text{read}, +), (G'[d], \text{write}, +)\}$

This means reading the sent gradients $G[i + j]$, reading the corresponding aggregated gradients $G'[j]$, adding them together, and then writing them to aggregated gradient $G'[j]$ again, for any $i \in [n]$ and $j \in [d]$. For any two input data $I, I'$ of equal length, for any security parameter $\lambda$, $Accesses_{\text{dense}}$ is identical and the statistical distance $\delta = 0$. Finally, Linear algorithm is $0$-statistical oblivious. □

Proof of Proposition 5.1.

Proof. Let the access pattern observed through algorithm 3 be $Accesses_{\text{baseline}}$, and it is as follows:

$Accesses_{\text{baseline}} = \{[G[1], \text{read}, +], (G'[1], \text{write}, +), \ldots, (G'[d/c], \text{write}, +), \ldots, (G[k], \text{read}, +), (G'[1], \text{write}, +), \ldots, (G'[d/c], \text{write}, +)\}$

where $c$ is the number of gradients included in one cacheline and $G'$ is an array with $d/c$ cells where $G'$ is divided at the granularity of a cacheline. Since $Accesses_{\text{baseline}}$ is the identical sequence for any inputs of the same length, algorithm 3 is $0$-statistical oblivious. □
Table 4: Architectures of the neural networks used in Section 4. \( d \) is the number of parameters of the global model trained in FL and \(|L|\) is the number of labels of inference target.

| Name   | Layers                  | Details                  |
|--------|-------------------------|--------------------------|
| NN     | 2 Fully Connected Layers | Input: \( d \)          |
|        |                         | Hidden: 1000             |
|        |                         | Dropout: 0.5             |
|        |                         | Activation: ReLU         |
|        |                         | Output: \(|L|\)          |
| NN-SINGLE | 2 Fully Connected Layers | Input: \( d \)          |
|        |                         | Hidden: 2000             |
|        |                         | Dropout: 0.5             |
|        |                         | Activation: ReLU         |
|        |                         | Output: \(|L|\)          |

**D MODEL ARCHITECTURES**

Here are some details about the neural network model we used in our experiments. The code for all models is available from our public repository.

Table 3 shows the model used as the FL’s global model throughout all experiments. Table 4 describes the detailed design of the model used in the neural network-based attack in section 4.3.

Table 3: Architectures of the neural networks used as global models in all FL experiments in Sections 4.3 and 5.5. Readers can find the details of ResNet-18 at https://github.com/weiaicunzai/pytorch-cifar100/blob/master/models/resnet.py.

| Name               | Layers                  | Details                  |
|--------------------|-------------------------|--------------------------|
| MNIST MLP          | 2 Fully Connected Layers | Input: 28 * 28           |
|                    |                         | Hidden: 64               |
|                    |                         | Dropout: 0.5             |
|                    |                         | Activation: ReLU         |
|                    |                         | Output: 10               |
| CIFAR10 MLP        | 2 Fully Connected Layers | Input: 3 * 32 * 32       |
|                    |                         | Hidden: 64               |
|                    |                         | Dropout: 0.5             |
|                    |                         | Activation: ReLU         |
|                    |                         | Output: 10               |
| CIFAR10 CNN        | Convolutional 1         | Input: 6 * 14 * 14       |
|                    |                         | Activation: ReLU         |
|                    |                         | Maxpooling:              |
|                    |                         | kernel size: 2           |
|                    |                         | stride: 2                |
|                    | Convolutional 2         | Input: 16 * 5 * 5        |
|                    |                         | Activation: ReLU         |
|                    |                         | Hidden1: 120             |
|                    |                         | Hidden2: 84              |
|                    |                         | activation: ReLU         |
|                    |                         | Output: 10               |
| Purchase100 MLP    | 2 Fully Connected Layers | Input: 600               |
|                    |                         | Hidden: 64               |
|                    |                         | Dropout: 0.5             |
|                    |                         | Activation: ReLU         |
|                    |                         | Output: 100              |
| CIFAR100 CNN       | ResNet-18               |                          |