Olive: Oblivious Federated Learning on Trusted Execution Environment Against the Risk of Sparsification

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

Yang Cao
Hokkaido University
yang@ist.hokudai.ac.jp

Masatoshi Yoshikawa
Osaka Seikei University
yoshikawa-mas@osaka-seikei.ac.jp

ABSTRACT

Combining Federated Learning (FL) with a Trusted Execution Environment (TEE) is a promising approach for realizing privacy-preserving FL, which has garnered significant academic attention in recent years. Implementing the TEE on the server side enables each round of FL to proceed without exposing the client's gradient information to untrusted servers. This addresses usability gaps in existing secure aggregation schemes as well as utility gaps in differentially private FL. However, to address the issue using a TEE, the vulnerabilities of server-side TEEs need to be considered—this has not been sufficiently investigated in the context of FL. The main technical contribution of this study is the analysis of the vulnerabilities of TEE in FL and the defense. First, we theoretically analyze the leakage of memory access patterns, revealing the risk of sparsified gradients, which are commonly used in FL to enhance communication efficiency and model accuracy. Second, we devise an inference attack to link memory access patterns to sensitive information in the training dataset. Finally, we propose an oblivious yet efficient aggregation algorithm to prevent memory access pattern leakage. Our experiments on real-world data demonstrate that the proposed method functions efficiently in practical scales.

PVLDB Reference Format:
Fumiyuki Kato, Yang Cao, and Masatoshi Yoshikawa. Olive: Oblivious Federated Learning on Trusted Execution Environment Against the Risk of Sparsification. PVLDB, 16(10): 2404 - 2417, 2023.
doi:10.14778/3603581.3603583

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 the current Big Data era, the challenge of preserving privacy in machine learning (ML) techniques has become increasingly apparent, as symbolized by the proposal of the GDPR [24]. Federated learning (FL) [43] is an innovative paradigm of privacy-preserving ML, which has been tested in production [8, 53, 55]. Typically, in FL, the server does not need to collect raw data from *users* (we use *participants* and *clients* interchangeably)—it only collects *gradients* (or model *parameters* delta) trained on the local data of users during each round of model training. The server then aggregates the collected gradients into a global model. Thus, FL is expected to enable data analyzers avoid the expenses and privacy risks of collecting and managing training data containing sensitive information.

However, multiple studies have highlighted the vulnerability of FL to various types of attacks owing to its decentralized scheme. One of its most extensively studied vulnerabilities is an inference attack on a client's sensitive training data during the aggregation phase by an untrusted server [21, 63, 72, 79, 84]. This attack arises from the requirement for each client to share raw gradient information with the central aggregation server in plain FL. This creates the risk of privacy leakage from the training data, making it a vulnerable attack surface. These attacks highlight the privacy/security problems of running FL on an untrusted server.

Enhancing FL using a Trusted Execution Environment (TEE) is a promising approach to achieve privacy-preserving FL, which has garnered significant attention in recent years [45, 50, 77, 78, 80]. TEE [18, 57] is a secure hardware technique that enables secure computation in an untrusted environment without exposing data or processing to the host (i.e., OS or hypervisor). TEE guarantees confidentiality, integrity, verifiability, and functionalities such as remote attestation, fully justifying its use on the untrusted server side in FL [29, 77, 78]. Gradients are transmitted to the TEE via a secure channel and computed securely in confidential memory, thereby eliminating the aforementioned attack surface.

Utilization of TEE is advantageous from several perspectives. Although similar functionality is provided by secure aggregation (SA)[1] based on pairwise masking, it sacrifices usability [10, 19, 32, 40]. This requires time-consuming synchronous distributed mask generation among multiple clients and lacks robustness with respect

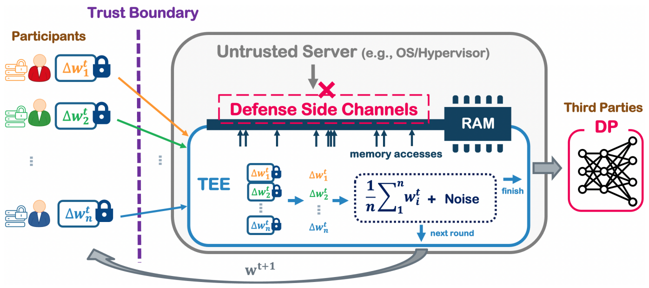

Figure 1: Olive, i.e., ObLIVious fEderated learning on TEE is the first method of its kind to prevent privacy risks caused by the leakage of memory access patterns during aggregation in FL rigorously. This allows, for example, to enjoy utility of CDP-FL without requiring a trusted server like LDP-FL.

Proceedings of the VLDB Endowment, Vol. 16, No. 10 ISSN 2150-8097.
doi:10.14778/3603581.3603583

[1]The recent paper [46] categorized TEE as a method of secure aggregation in FL.

to participant asynchronicity/dropouts [46], which is difficult to handle and can impede implementation by general practitioners. Further, SA is inflexible and makes it hard to do extensions, such as Byzantine resistance [80] and asynchrony [50]. In addition, application of gradient sparsification to FL with SA requires either random sparsification [19] or a common sparsified index among multiple clients [40] because of the pairwise constraints, impairing training quality. One simple and important solution to these problems is the use of a TEE, even though it requires additional special hardware.

In addition, FL with TEE addresses the utility gap in differentially private FLs [20, 22, 44], i.e., between local DP (LDP) and central DP (CDP). The recently studied Shuffle DP-FL [20, 23, 38], which aims to combine the best LDP-FL trust model [73, 82] with the model utility of the CDP-FL [3, 22, 44], exhibits a gap with respect to CDP-FL in terms of utility [20]. As depicted in Figure 1, TEE facilitates secure model aggregation on an untrusted server, which ensures only differentially private models are observable by the server. Without trust in the server, as in LDP-FL, model utility is equivalent to that of conventional CDP-FL because any DP mechanism can be implemented within the TEE, whereas the mechanism is restricted when using SA [32]. Since combining CDP-FL and TEE is one of the interesting use cases of our proposed method, we describe it in detail in Appendix D of the full paper [34] for completeness.

However, implementing a server-side TEE to achieve the aforementioned benefits requires careful analysis of the vulnerabilities of TEE. Several serious vulnerabilities are known to affect TEE owing to side-channel attacks [51, 71, 76], which can cause privacy leakage despite encryption. In particular, such attacks can expose data-dependent memory access patterns of confidential execution and enable attackers to steal sensitive information, such as RSA private keys and genome information [12]. The specific information that may be stolen from these memory access patterns is domain-specific and is not yet known for FL, although several studies have attempted to use TEE for FL [16, 45, 47, 77, 78]. Thus, the extent of the threat of side-channel attacks against FL on a TEE and the types of possible attacks remain critical open problems in this context.

Oblivious algorithms [25, 52, 65] are important leakage prevention techniques that generate only data-independent memory access patterns. A general approach involves making the RAM oblivious, e.g., oblivious RAM (ORAM). PathORAM [65] is known to be the most efficient technique. However, it assumes a private memory space of a certain size and is not applicable to practical TEE, such as Intel SGX [18]. Although Zerotrace [59] addresses this issue, its still incurs significant overhead. Therefore, the design of an algorithm-specific method to obtain an efficient algorithm is an important problem. In this context, [52] proposed an efficient oblivious algorithm for specific ML algorithms, and [83] studied SQL processing. However, an efficient method for FL-specific aggregation algorithm, which can be a vulnerable component of FL with a server-side TEE, has not yet been proposed.

In this study, we address the aforementioned gaps; (1) we clarify privacy risks associated with FL using server-side TEE by designing specific attacks and demonstrating them in real-world scenarios; (2) we propose a novel defense mechanism against these attacks by designing efficient oblivious algorithms and evaluating them empirically on a practical scale. Our analysis reveals that parameter position information is leaked during the execution of the FL aggregation algorithm in a *sparsified* environment. *Sparsification* is a common technique in FL [19, 37, 40, 58] to reduce communication costs and/or improve model accuracy [1]. We assume that the goal of an attacker is to infer a set of sensitive labels included in the target user's training data, which is a similar goal to previous works on privacy attacks in FL [21, 72]. We further assume that the attacker has access to the memory access patterns, the dataset for model validation, and the trained model for each round. Although sparsified index information in FL has been considered as somewhat private information in previous studies [38, 40], unlike in our study, no specific attacks have been investigated. After demonstrating the proposed attack on real-world datasets, we propose efficient oblivious algorithms to prevent such attacks completely. To this end, we carefully construct existing oblivious building blocks, such as the oblivious sort [6] and our designed components. Our proposed method Olive, an oblivious federated learning system based on server-side TEE, is resistant to side-channel attacks, enabling truly privacy-preserving FL. In addition to fully oblivious algorithms, we further investigate optimization by adjusting the data size in the enclave, and study more efficient algorithms by relaxing the definition of obliviousness. Finally, we conduct extensive experiments on real-world data to demonstrate that the proposed algorithm, designed for FL aggregation, is more efficient than the general-purpose PathORAM with SGX [59].

The contributions of this study are summarized below:

- We investigate the potential exposure of memory access patterns to an untrusted server when using TEE for model aggregation in FL. We discover a risk associated with sparsified gradients, which are frequently utilized in FL.
- We develop a supervised learning-based sensitive label inference attack that leverages index information obtained from the side-channels of sparsified gradients. We demonstrate the effectiveness of this attack on real-world datasets. Our results indicate that, when training a CNN on CIFAR100 with top-1.25% sparsification, the sensitive labels of training data (where each participant has 2 out of 100 labels) are leaked with 90% or better accuracy (Figure 6).
- We introduce a novel oblivious algorithm for efficient model aggregation by thoughtfully integrating oblivious primitives, such as oblivious sort and our custom-designed components. We conduct extensive experiments to showcase the efficiency of our proposed method. Notably, it outperforms a PathORAM-based method by more than 10 × and completes the aggregation process in just a few seconds for a model with a million parameters (Figure 9).

The remainder of this paper is organized as follows. Preliminary notions are presented in Section 2. The overview of the proposed system and the problem setting is described in Section 3. Sections 4 and 5 demonstrate the proposed attack and defense, respectively, with empirical evaluations. Section 6 discusses related works and Section 7 concludes.

2 PRELIMINARIES

2.1 Federated Learning

Federated learning (FL) [35, 43] is based on a distributed optimization. The basic algorithm, called `FedAVG` [43], trains models by repeating model optimization steps in the local environment of the clients and updating the global model by aggregating the parameters in the server. FedSGD [43] exchanges locally updated gradients based on distributed stochastic gradient descent. Overall, users are not required to share their training data with the server, which represents a major advantage over traditional centralized ML.

Sparsification. To reduce communication costs and improve model accuracy, the sparsification of the model parameters before their transmission to the server has been extensively studied in FL [19, 28, 37, 40, 58, 61, 75]. All of the aforementioned methods sparsify parameters on the client side, apply an encoding that represents them as *value* and *index* information [75], transmit them to the server, and aggregate them into a dense global model on the server side. Exceptionally, [28, 40] used common sparsification among all clients using common sparsified indices and aggregated them into a sparse global model. However, as observed in [19], there is practically little overlap among the top-k indices for each client in real-world data, especially in the non-i.i.d. environment, which is common in FL. This highlights the one of limitations of pairwise masking-based SA [19, 40] (see Section 6). In general, top-k sparsification is the standard method. By transmitting only the top-k parameters with large absolute gradients to the aggregation server, communication cost is reduced by more than 1 ~3 orders of magnitude [58]. This technique outperforms the random selection of k indices (random-k) [19], particularly when the compression ratio is smaller than 1% [28, 40, 58, 75]. Other sparsification methods, such as threshold-based [58], top-k under LDP [39] and the recently proposed convolutional kernel [75], also exist. However, these sparsified gradients can lead to privacy leakages through the index. In [38, 40], the set of user-specific top-k indices was treated as private information; however, no specific attacks were investigated.

2.2 Trusted Execution Environment

The TEE, as defined formally in [57], creates an isolated execution environment within untrusted computers (e.g., cloud VMs). We focus on a well-known TEE implementation—Intel SGX [18]. It is an extended instruction set for Intel x86 processors, which enables the creation of an isolated memory region called an *enclave*. The enclave resides in an encrypted and protected memory region called an *EPC*. The data and programs in the EPC are transparently encrypted outside the CPU package by the Memory Encryption Engine, enabling performance comparable to native performance. SGX assumes the CPU package to be the trust boundary—everything beyond it is considered untrusted—and prohibits access to the enclave by any untrusted software, including the OS/hypervisor. Note that for design reasons, the user-available size of the EPC is limited to approximately 96 MB for most current machines. When memory is allocated beyond this limit, SGX with Linux provides a special paging mechanism. This incurs significant overhead for encryption and integrity checks, resulting in poor performance [33, 41, 68].

Attestation. SGX supports remote attestation (RA), which can verify the correct initial state and genuineness of an enclave. On requesting the RA, a report with measurements based on the hash of the initial enclave state generated by the trusted processor is received. This facilitates the identification of the program and completes the memory layout. Intel EPID signs this measurement and the Intel Attestation Service verifies the correctness of the signature as a trusted third party. Consequently, verifiable and secure computations are performed in a remote enclave. Simultaneously, a secure key exchange is performed between the enclave and the remote client within this RA protocol. Therefore, after performing RA, communication with a remote enclave can be initiated over a secure channel using AES-GCM and so on.

2.3 Memory Access Pattern Leakage

Although the data are encrypted and cannot be viewed in enclaves, memory/page access patterns or instruction traces can be exposed irrespective of the use of a TEE [12, 36, 51, 71, 76]. This may lead to sensitive information being stolen from enclaves [12]. For example, cacheline-level access pattern leakage occurs when a malicious OS injects page faults [76] or uses page-table-based threats [51, 71]. Moreover, if a physical machine is accessible, probes may be attached to the memory bus directly.

To prevent such attacks, oblivious algorithms have been proposed to hide access patterns during the secure execution of the process. An oblivious algorithm is defined as follows.

Definition 2.1 (Oblivious algorithm [14]). An algorithm $\mathcal{M}$ is δ-statistically oblivious if, for any two input data I and I' of equal length and any security parameter λ, the following relation holds:

$$\mathbf{Accesses}^{\mathcal{M}}(\lambda, I) \overset{\delta(\lambda)}{\equiv} \mathbf{Accesses}^{\mathcal{M}}(\lambda, I')$$

where $\mathbf{Accesses}^{\mathcal{M}}(\lambda, I)$ denotes a random variable representing the ordered sequence of memory accesses. The algorithm $\mathcal{M}$ is generated upon receiving the inputs, λ and I. $\overset{\delta(\lambda)}{\equiv}$ indicates that the statistical distance between the two distributions is at most $\delta(\lambda)$. The term δ is a function of λ which corresponds to a cryptographic security parameter. When δ is negligible, we say that $\mathcal{M}$ is *fully oblivious*, and when δ is 1, it is *not oblivious*.

A typical approach for constructing an oblivious algorithm utilizes an ORAM, such as PathORAM [65]. Although ORAMs are designed for general use as key-value stores, several oblivious task-specific algorithms, such as ML [52] and SQL processing [83] (see Section 6 for details), have been proposed from a performance perspective. They are constructed based on oblivious sort [6] and/or access to all memory (i.e., linear scan), and are distinct from ORAM at the algorithmic level. Further, ORAM generally assumes that the existence of a trusted memory space such as *client storage* [65], which is incompatible with the SGX assumption of leaking access patterns in enclaves. Thus, only CPU registers should be considered to be trusted memory spaces [59]. [52] implemented oblivious ML 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. Using the CMOV instruction, conditional branching can be implemented with a constant memory access pattern that does not depend on the input, thereby removing the leakage of subsequent code addresses. For example, Zerotrace [59] implements PathORAM on SGX by obliviously implementing client storage based on CMOV. We can construct and use low-level oblivious primitives, such as *oblivious move* (o_mov, Listing 1 in our full paper [34]) and *oblivious swap* (o_swap, Listing 2 in [34]).

`o_mov(flag,x,y)` is a function that accepts a Boolean condition flag as its first argument and returns `x` or `y` depending on the flag. Therefore, designing an appropriate oblivious algorithm for SGX requires a combination of high-level algorithm designs, such as the oblivious sort and low-level primitives.

3 PROPOSED SYSTEM

In this section, we first clarify our scenario and threat model, and then present a system overview of the Olive. Finally, we analyze the details of the potential privacy risk, followed by discussion of a specific privacy attack and evaluation in Section 4.

3.1 Scenario

We target a typical FL scenario with a single server and clients using identical format data (i.e., horizontal FL). The server is responsible for training orchestration, aggregating parameters, updating the global model, selecting clients for each training round, and validating model quality. The server-side machine is assumed to be placed in a public or private environment [29, 77] and is equipped with a TEE capable of RA (e.g., Intel SGX).

Threat model. We assume an adversary to be a *semi-honest* server that allows FL algorithms to run as intended, while trying to infer the sensitive information of clients based on shared parameters. This is a compatible threat model with those in existing studies on FL with SA [10] and even with server-side TEE [45, 77, 78]. The semi-honest threat model is selected despite using TEE, because the assumed attack in this work does not diverge from the established FL protocol. The goal of the adversary is not to damage the availability (e.g., DoS attacks) or undermine the utility of the model (e.g., data-poisoning attacks) [5, 66, 80] as *malicious* attackers in FL context. Note that several side-channel attacks against TEE require malicious (i.e., privileged) system software, which we distinguish from an attacker and categorize as *malicious* in FL. Nevertheless, [9] reported that malicious servers improve inference attacks in FL. In Section 5.6, we discuss the relationship between such malicious servers and the privacy and security of the proposed system.

We assume that the server has (1) access to the trained model during each round of FL, (2) access to the global test dataset, and (3) the capability to observe the memory access patterns of the TEE. These requirements can be justified as follows. (1): Because the server is in charge of model validation, it makes sense for the server to have access to the global models during all rounds. Alternatively, attackers can easily blend in with clients to access global models. (2): Generally, the semi-honest server that has access to public datasets for model validation covers the overall dataset distribution, which is essential in production uses. Similar assumptions have been made in previous studies on inference attacks [28, 74]. Subsequently, we experimentally evaluate the required dataset volume (Figure 8). (3): This follows the general threat assumption for TEE. The SGX excludes side-channel attacks from the scope of protection [18, 51]. Except for the trusted hardware component (i.e., the CPU package), all other components of the server, e.g., the system software (i.e., OS/hypervisor), main memory, and all communication paths, are considered to be untrusted. The server can observe memory access patterns through known or unknown side-channel attacks, as described in Section 2.3.

3.2 System overview

The proposed system, namely the Olive (Figure 1), follows basic `FedAVG` algorihtm with standard top-k sparsification; however, the TEE is placed on the server side with a server-side algorithm resistant to side-channel attacks. As an initial configuration, we provide an enclave in which each client verifies the integrity of the processes running on the enclave via RA and exchanges shared keys (AES-GCM). If attestation fails, the client must refuse to join the FL in 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 the transmitted gradients[2] are doubly encrypted and can only be decrypted in the trusted enclave.

The overall algorithm of the Olive is presented in Algorithm 1, where the differences with respect to the basic `FedAVG` algorithm are highlighted in red. The initial provisioning is omitted and a different shared key, sk_i, is stored in the enclave for each user, i ($\in [N]$) (line 1). In each round, the participants are securely sampled in the enclave (line 4). The selected users are memorized in the enclave and used for client verification (line 9) after the encrypted data are loaded into the enclave (line 8). On the client side, locally trained parameters are top-k sparsified (line 21), and then encoded and encrypted (line 22). The encrypted data loaded into the enclave are decrypted and verified (line 11). Verification (lines 9, 11) is not essential to our work; however, it prevents man-in-the-middle attacks and biased client selection. As discussed in Section 3.3, the aggregation operation (line 12) is required to be oblivious, and we present lower-level and detailed algorithms in Section 5 to this end. In accordance with the principle that the Trusted Computing Base (TCB) should be minimized, only the aggregation operation is performed in the enclave. Finally, the aggregated parameters are loaded outward from the enclave (line 13). Thus, the parameters transmitted by all clients remain completely invisible to the server,—only the aggregated parameters are observable.

3.3 Security Analysis

Although TEE enables model training while protecting raw gradients, an untrusted server can observe the memory access patterns, as described in Section 2.3. Here, we analyze the threats that exist based on memory access patterns.

For formal modeling, let g_i denote the k-dimensional gradient transmitted by user i and let g^* be the d-dimensional global parameter after aggregation. In the typical case, $k = d$, when dense gradients are used. Let $\mathbf{G}_i$ and $\mathbf{G}^*$ denote the memories required to store the gradients of g_i and g^*, respectively, and let the number of clients participating in each round be n. The memory that stores the entire gradient is denoted by $\mathbf{G} = \mathbf{G}_1 \parallel ... \parallel \mathbf{G}_n$, where $\parallel$ denotes concatenation. A memory access, a, is represented as a triple $a = (\mathbf{A}[i], \texttt{op}, \texttt{val})$, where $\mathbf{A}[i]$ denotes the i-th address of the memory, $\mathbf{A}$; `op` denotes the operation for the memory—either `read` or `write`; and `val` denotes the value to be written when `op` is `write`, and `null` otherwise. Therefore, the observed memory access pattern, **Accesses**, can be represented as $\textbf{Accesses} = [a_1, a_2, ..., a_m]$ when the length of the memory access sequence is m.

[2]In `FedAVG`, the data shared by users are not exactly gradients—rather, they are the delta of model weights. However, in the context of compatibility with `FedSGD`, we jointly refer to model update data transmitted by users as *gradients* or *parameters*.

Algorithm 1 Olive: Oblivious FL on TEE

```
Input: $N$: # participants, $\eta_c$, $\eta_s$: learning rate
1: KeyStore ← Remote Attestation with all user $i$    ▹ key-value store in
   enclave that stores $sk_i$: user $i$'s shared key from RA in provisioning
2: procedure Train($q$, $\eta_c$, $\eta_s$)
3:     Initialize model $\theta^0$
4:     for each round $t = 0, 1, 2, \ldots$ do
5:         $Q^t$ ← (sample users from $N$ for round $t$)    ▹ securely in enclave
6:         for each user $i \in Q^t$ in parallel do
7:             Enc($\Delta_i^t$) ← EncClient($i, \theta^t, \eta_c$)
8:             LoadToEnclave(Enc($\Delta_i^t$))
9:             check if user $i$ is in $Q^t$
10:            $sk_i$ ← KeyStore[$i$]    ▹ retrieve user $i$'s shared key
11:            $\Delta_i^t$ ← Decrypt($Enc(\Delta_i^t)$, $sk_i$)
12:        /* Obliviously performed, such as Algorithm 3 or 4 */
           $\tilde{\Delta}^t = \frac{1}{|Q^t|}\sum_{i \in Q^t} \Delta_i^t$    ▹ oblivious algorithm
13:        LoadFromEnclave($\tilde{\Delta}^t$)
14:        $\theta^{t+1} \leftarrow \theta^t + \eta_s \bar{\Delta}^t$
15: procedure EncClient($i$, $\theta^t$, $\eta$, $C$)
16:     $\theta \leftarrow \theta^t$
17:     $\mathcal{G}$ ← (user $i$'s local data split into batches)
18:     for batch $g \in \mathcal{G}$ do
19:         $\theta \leftarrow \theta - \eta \nabla \ell(\theta; g)$
20:     $\Delta \leftarrow \theta - \theta^t$
21:     $\Delta$ ← TopkSparse($\Delta$)    ▹ top-$k$ sparsification on gradients
22:     Enc($\Delta'$) ← Encrypt($\Delta, sk_i$)    ▹ Authenticated Encryption (AE)
   mode, such as AES-GCM, with shared key, $sk_i$, from RA
23:     return Enc($\Delta$)
```

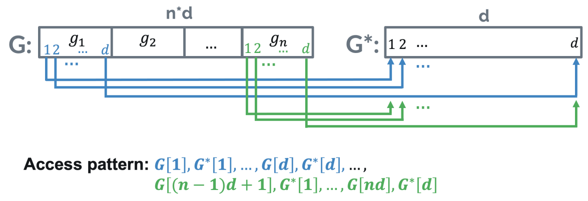
n*d
d
G:
g_1
g_2
...
g_n
G*:
1 2 ... d
Access pattern: $G[1], G^*[1], \ldots, G[d], G^*[d], \ldots,$
$G[(n-1)d+1], G^*[1], \ldots, G[nd], G^*[d]$

Figure 2: Dense gradients induce uniform access patterns.

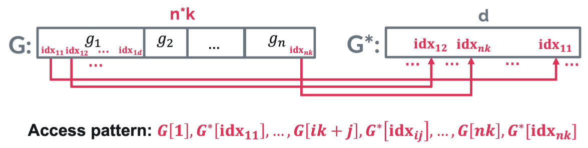
n*k
d
G:
g_1 idx_{11} idx_{12} ... idx_{1d}
g_2
...
g_n idx_{nk}
G*:
idx_{12} idx_{nk} idx_{11}
Access pattern: $G[1], G^*[\text{idx}_{11}], \ldots, G[ik+j], G^*[\text{idx}_{ij}], \ldots, G[nk], G^*[\text{idx}_{nk}]$

Figure 3: Sparse gradients induce biased access patterns.

In FL, operations performed on the server side generally consist of summing and averaging the gradients obtained from all users. We first note that this procedure is oblivious to *dense gradients*. As depicted in Figure 2, the summing operation involves updating the value of the corresponding index of $\mathbf{G}^*$ while performing a linear scan on $\mathbf{G}$, where memory accesses are performed in a fixed order and at fixed addresses, irrespective of the content of $\mathbf{G}$. We refer to this general summing part as the *linear* algorithm and present it in Appendix B of [34] for completeness.

Proposition 3.1. *The linear algorithm is fully oblivious to dense gradients.* (An formal proof is presented in [34]).

The linear algorithm is executed in $O(nd)$ because all the elements of the gradient $\mathbf{G}$ are accessed. In addition, the averaging operation only accesses $\mathbf{G}^*$ linearly in $O(d)$, which is obviously fully oblivious.

However, when the gradients are *sparsified*, which is often an important scenario in FL, the access pattern of the *linear* algorithm is not oblivious, and sensitive information may be leaked. The weights of sparse gradients are generally given by tuples of `index`, which hold the location information of the parameter, and a `value`, which holds the gradient value. This is irrespective of its quantization and/or encoding because it requires calculating the sum of the original dense gradients. Figure 3 depicts the access pattern when an aggregation operation is used for sparsified gradients.

Proposition 3.2. *The linear algorithm is* **not** *oblivious to sparsified gradients.*

Proof. Linear access to $\mathbf{G}$ for sparsified gradients occurs when the access pattern, $\mathbf{Accesses}^{\mathbf{sparse}}$, satisfies

$$\mathbf{Accesses}^{\mathbf{sparse}} =$$
$$[(\mathbf{G}[1], \texttt{read}, *), (\mathbf{G}^*[\text{idx}_{11}], \texttt{read}, *), (\mathbf{G}^*[\text{idx}_{11}], \texttt{write}, *), \ldots,$$
$$(\mathbf{G}[nk], \texttt{read}, *), (\mathbf{G}^*[\text{idx}_{nk}], \texttt{read}, *), (\mathbf{G}^*[\text{idx}_{nk}], \texttt{write}, *)]$$

where the `index`es of sparsified gradients of user i are $\text{idx}_{i1}, .., \text{idx}_{ik}$ for all $i \in [n]$. The access pattern, $\mathbf{Accesses}^{\mathbf{sparse}}$, is deterministic and corresponds in a one-to-one fashion with the sequence of the `index`es of the input data. Considering two input data, I and I', with different sequences of `index`es, no overlap exists in the output distribution. Then, the statistical distance between them is 1. □

The access pattern on the aggregated gradients, $\mathbf{G}^*$, reveals at least one set of indices $\{\text{idx}_{ij} \mid j \in [d]\}$ for each user i, depending on the given gradients. Considering data-dependent sparsifications, such as top-k, which are generally used in FL, the gradient indices of the sparsified gradients may be sensitive to the training data. In the next section, we demonstrate that privacy leakage can be caused on a real-world dataset.

Generality and Limitation. Let us now clarify the format and method of sparsified gradients. Although various quantization and/or encoding methods in FL have been studied(e.g., [60]), quantization is irrelevant to the problem of leakage considered in this study because it affects only the values and not the index, and encoding is irrelevant because it is eventually decoded on the server side. For example, in [19, 40], the index location information was encoded in d-dimensional one-bit array, but the same problem occurred during aggregation. As aggregation is performed on the original dense gradients, each update requires access to a specific index of the dense gradients ($\mathbf{G}^*$), resulting in identical access patterns. It should also be noted that risk is sparsification-dependent. If the client's training data and observed indices are uncorrelated, then index leakage is not considered to be a risk. For example, when random-k is adopted, as in [19], no risk is involved. While threshold-based sparsification [58] is almost identical to top-k, LDP-guaranteed index [39] and the recently proposed convolution-kernel-based index [75] are still unclear. These index information can correlate to some extent with the client's training data, but not as much as top-k. The scope of our study is limited to the demonstration that attacks are possible with the standard top-k—the investigation of various other sparsifications are left for future research.

Algorithm 2 Attack on index: JAC or NN

```
Input: $i$: target user, $X_l$: test data with label $l$ ($l \in L$), round: $T$
index ← {}                                        ▷ observed access patterns
/* Prepare teacher and target indices */
teacher ← {}              ▷ teacher access patterns to train a classifier
for each round $t = 1, \dots, T$ do
    /* $T_i$: rounds participated in by user $i$ */
    if $t \in T_i$ then
        /* $A_i^{(t)}$: observed top-$k$ indices of user $i$ of round $t$ */
        Store $A_i^{(t)}$ to index[$i, t$]
        for each label $l \in L$ do
            /* $\theta^t$: the global model after round $t$ */
            /* $I_l^{(t)}$: top-$k$ indexes training with $\theta^t$ and $X_l$ */
            Store $I_l^{(t)}$ to teacher[$l, t$]
/* Calculate scores for each label $l$ */
S ← []                                  ▷ form of [(label, similarity)]
/* If JAC: Jaccard similarity-based scoring (SIM) */
for each label $l \in L$ do
    Store ($l$, SIM($\|_{\tau \in T_i}$index[$i, \tau$], $\|_{\tau \in T_i}$teacher[$l, \tau$]) to S
/* If NN: neural network-based scoring */
Train the model $M_t$ with teacher[$l, t$] ($l \in L$) for each $t$ ($\in T$)
for each label $l \in L$ do
    Store ($l$, PREDICT($M_1, ..., M_T$, $\|_{\tau \in T_i}$index[$i, \tau$])) to S
/* If NN-SINGLE: using single neural network */
Train the model $M_0$ with $\|_{\tau \in T}$teacher[$l, \tau$] ($l \in L$)
for each label $l \in L$ do
    Store ($l$, PREDICT($M_0$, $\|_{\tau \in T}$index[$i, \tau$])) to S
/* 1D K-Means clustering KMEANS */
[labels, centroid] ← KMEANS(S)
return labels of the cluster with the largest centroid
```

4 ATTACK ON GRADIENT INDEX

4.1 Design

In this section, we design a server-side attack to demonstrate that privacy leakage of the training data can occur based on the index information in the gradients. We assume a sparsified gradient based on top-k [37, 58, 62]. The attacker is assumed to satisfy the assumptions listed in Section 3.1. The proposed attacks can be used to raise awareness of the security/privacy risks of FL on TEE, which have not been reported in related works [16, 45, 47, 77], and also serve as an evaluation framework for defenses.

The goal of the attack is to infer the target client's sensitive label information based on the training data. For example, when training FL on medical image data, such as image data on breast cancer, the label of the cancer is very sensitive, and participants may not want to reveal this information. A similar attack goal was considered in [21, 72]. Our designed attack is based on the intuition that the top-k indices of the locally converged model parameters are correlated with the labels of the local training data. We train a classifier that accepts the observed index information as the input by supervised learning using a public test dataset and the output is the sensitive label set. Access to the dataset is justified, for example, by the need for model validation, as described in Section 3.1 and in previous studies on inference attacks [28, 74]. We design two basic methods—the Jaccard similarity-based nearest neighbor approach (JAC) and a neural network (NN). The detailed algorithm is presented in Algorithm 2. An overview of these methods is provided below:

(1) First, the server prepares the test data X_l with label l for all $l \in L$, where L denotes the set of all possible labels.
(2) In each round t ($\in T$), an untrusted server observes the memory access patterns through side-channels, obtains the index information of the top-k gradient indices **index**[i, t] for each user i, and stores it (lines 4–8).
(3) The server computes the gradient of the global model with θ^t and X_l, without model updates for each round t ($\in T$), using the test data categorized by labels, and obtains the top-k indices **teacher**[l, t] as teacher data for each label (lines 9–12).
(4) After the completion of all rounds T, in JAC, we calculate the Jaccard similarity between $\|_{\tau \in T_i}$**teacher**[l, τ] and observed access patterns, $\|_{\tau \in T_i}$**index**[i, τ] for each label l (lines 15–17). Jaccard similarity is selected because, in the worst-case scenario, the index information transmitted by a participant is randomly shuffled, rendering the sequence meaningless.
(5) In NN, the attacker trains neural networks using **teacher**[l, t], with indices as the features and labels as the target (line 19). The outputs of the model are the scores of the label. Subsequently, we use a trained model to predict the labels included in the training data corresponding to the input, **index**[i]. For this task, we design the two following NN-based methods. In the first method, a model, M_t, is trained during each round, t, and the output scores of the models are averaged to predict the labels (NN). In the second method, a single model, M_0, is trained using the concatenated indices of the entire round as input and a single output is obtained (NN-SINGLE). In our experiment, both cases involve a multilayer perceptron with three layers (described in Appendix F of [34]). Note that as the model input, index information is represented as a multi-hot vector. In the case of NN-SINGLE, each client participates in only a proportion of the rounds—the indices of the rounds they do not participate in are set to zero as the 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 via model prediction (lines 20–21).
(6) If the number of labels of the target client is known, the scores are sorted in descending order and the highest labels are returned. 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).

Finally, the information obtained from the side-channels can also be used to design attacks for other purposes, such as additional features in reconstruction [27] or other inference attacks [49]. The aim of this study is simply to demonstrate that the top-k gradient indices that can be observed on untrusted servers contain sufficient information to cause privacy leakages; therefore, we leave the study of attacks for different purposes to future research.

4.2 Evaluation Task

In our evaluation of attacks, the server performs an inference attack on any client in the scenario detailed in Section 3.1. The clients have a subset of labels, and the attacker's goal is to infer the sensitive label

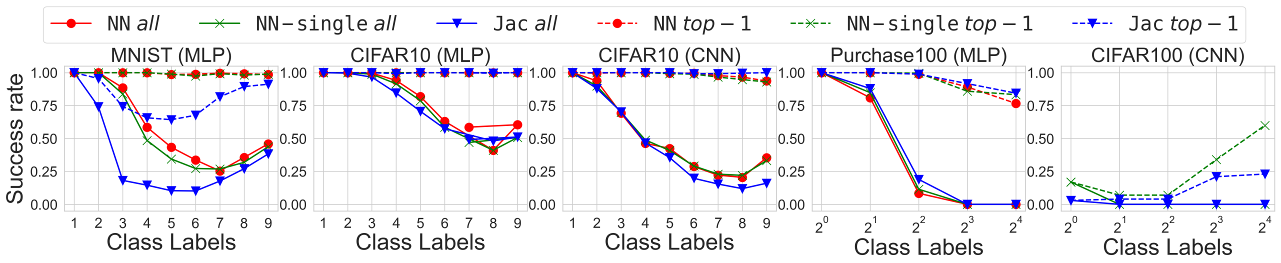

Figure 4: Attack results on datasets with a fixed number of labels: Vulnerable, especially when there are few labels.

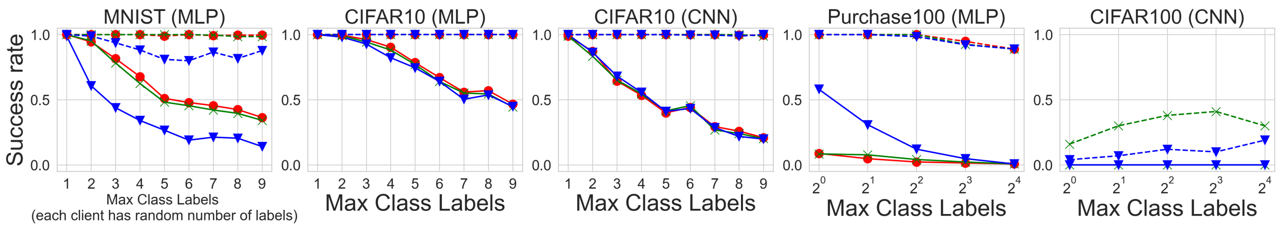

Figure 5: Attack results on datasets with a random number of labels (more difficult setting): When the number of labels is low, the attacker can attack the client without knowing the exact number of labels.

Table 1: Datasets and global models in the experiments.

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

set of a target client based on their training data. The attacker selects any subset or the entire set of users and performs an inference attack on each user. We utilize *all* and *top-1* as accuracy metrics for evaluating attack performance. We define *all* as the percentage of clients that match the inferred labels exactly, e.g., the inferred label set is {1,3,5}, and the target client's label set is {1,3,5}. We define *top-1* as the percentage of clients that contain the highest scored inferred label, e.g., the highest scored inferred label is five, and the target client's label set is {4,5}, which we consider to be a minimal privacy leak. In addition, we adjust the distribution of the label set such that the client is able to control the difficulty of the attack. The number of labels in the set and the number of labels that are *fixed* or *random* are configurable. In the case of a *fixed* label, all users exhibit the same number of labels, which is known to the attacker. In the case of the *random* label, the maximum number is assigned, and all users exhibit various numbers of labels. Generally, *random* label and larger numbers of labels are more difficult to infer.

4.3 Empirical Analysis

Here, we demonstrate the effectiveness of the designed attack.
Setup. Table 1 lists the datasets and global models used in the experiments. Details of the model, including the attacker's NN, are

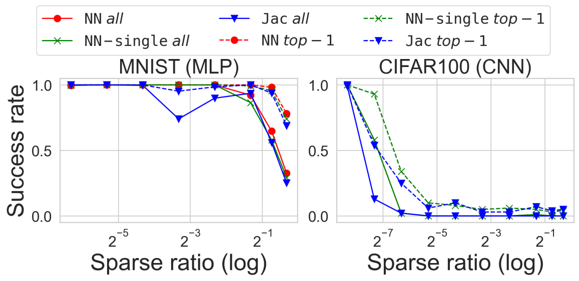

Figure 6: Attack results w.r.t. sparse ratios: Higher the sparsity, the more successful the attack tends to be.

provided in Appendix F of [34]. In addition to the well-known image datasets, MNIST and CIFAR 10 and 100, we also use Purchase100, which comprises tabular data used in [31] for membership inference attacks. We train the global models using different numbers of parameters, as listed in Table 1. The learning algorithm is based on Algorithm 1, in which we provide the sparse ratio, α, instead of k in top-k. FL's learning parameters include the number of users, N; the participant sampling rate, q; the number of rounds, T. The default values are given by $(N, q, T, \alpha) = (1000, 0.1, 3, 0.1)$. The attack methods are evaluated for Jac, NN, and NN-single, as described in the previous section. T is smaller than that in normal FL scenarios, which implies that our method requires only a few rounds of attacks. All experimental source codes and datasets are open[3].
Results. Figure 4 depicts the attack results for NN, NN-single, and Jac on all datasets with a *fixed* number of labels, and Figure 5 presents the results with a *random* number of labels. In CIFAR100, $T = 1$ is used because the model size is large. The y-axis represents

[3]https://github.com/FumiyukiKato/FL-TEE

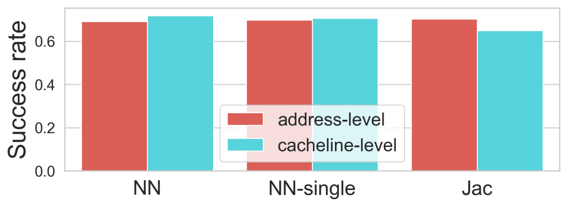

Figure 7: Cacheline-level leakage on CNN of CIFAR10: Attacks are possible with at least slightly less accuracy.

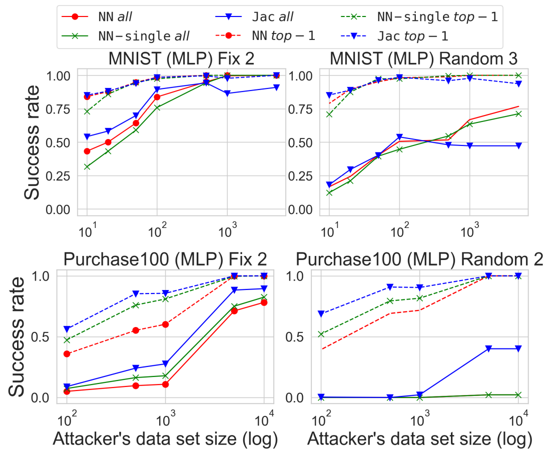

Figure 8: The size of data that an attacker needs to access to achieve high success rate can be very small.

the success rate of the attacks, and the x-axis represents the number of labels possessed by each client. When the number of labels is small, all three attacks exhibit a high probability of success. The success rate of *top-1* is high irrespective of the number of labels, whereas *all* decreases with each additional label. On CIFAR10, the MLP model maintains a higher success rate for a large number of labels compared to the CNN model. This indicates that the complexity of the target model is directly related to the contribution of the index information to the attack. The NN-based method is more powerful on MNIST, but it performs similarly to the other methods on the other datasets. This indicates that the gradient index information is not complex and can be attacked using simple methods, such as Jac. The results of NN and NN-single are almost identical; therefore, there is not much effective correlation across the rounds. When the number of class label is 100 (Purchase100, CIFAR100), the success rate of the attack is reduced. In particular, the accuracy of CIFAR100 is low in this case. However, as shown in later, this is surprisingly improved by using a smaller sparse rate.

Figure 6 depicts the relationship between the sparse ratio and attack performance. The number of client labels is fixed to two. The results indicate that the sparse ratio is inversely related to the success rate of the attack. This is because the indices of label-correlated gradients become more distinguishable as the sparsity increases. In particular, the case of CIFAR100 demonstrates that the attack is successful only when the sparsity ratio is low. For instance, when the sparsity ratio is 0.3%, the success rate is almost 1.0. Thus, sparsity ratio is an important factor in an attack.

Figure 7 depicts a comparison of attack performance based only on index information observed at the cacheline granularity (64 B), which can be easily observed against SGX [76] with CIFAR10 and CNN. The accuracies are almost identical. The NN-based method exhibits slightly higher accuracy, whereas Jac exhibits slightly poorer accuracy. Therefore, the attack is still possible despite observations at the granularity of the cacheline, which indicates that the well-known vulnerability of SGX is sufficient to complete an attack.

Figure 8 depicts the evaluation of the size of a dataset required by an attacker to succeed in an attack. The default test dataset accessible to the attacker is presented in Table 1—we randomly reduce it on this basis while maintaining the same number of samples for each label. We evaluate the number of labels in the fixed and random labels using the MNIST and Purchase100 datasets, respectively. In MNIST, performance can be preserved even when the amount of data is reduced, which weakens the assumption on dataset size. For example, it is surprisingly noted that, even with 100 samples (i.e., 10 samples per label and 1% of the original evaluation), performance is not affected significantly. On Purchase100, the impact is small, but a meaningful attack is possible with some reduction in data size.

5 OBLIVIOUS ALGORITHMS

In this section, we focus on an aggregation algorithm that can cause privacy leakage, as described in the previous section, and discuss potential avenues of attack prevention. The notation used here is identical to that in Section 3.3.

First, we introduce the general ORAM-based method. We initialize ORAM with d zero values for the aggregated parameters, g^*; update the values with the received nk gradients, g, sequentially; and finally retrieve the d values from the ORAM. Because ORAM completely hides memory access to g^*, the algorithm is fully oblivious. However, as established in the experimental section, even the state-of-the-art PathORAM adapted to TEE [59] incurs a significant overhead—thus, a task-specific algorithm is preferable.

5.1 Baseline method

Full obliviousness can be simply achieved by accessing all memory addresses to hide access to a specific address. When accessing $\mathbf{G}^*[i]$, a dummy access is performed on $\mathbf{G}^*[j]$ for each $j \in [d]$. For each access, either a dummy or an updated true value is written, and the timing of writing the true value is hidden by an oblivious move (`o_mov`). The Baseline algorithm is described in Algorithm 3. It accepts the concatenated gradients transmitted by all participants, g (nk-dimensional vector), as input and returns the aggregated gradients, g^* (d-dimensional vector) as output. We make linear accesses to $\mathbf{G}^*$ for a number of times equal to the length of $\mathbf{G}$. Assuming that the memory address is observable at the granularity of the cacheline, as in a traditional attack against the SGX [76], some optimization may be performed. When the weight is four bytes (32-bit floating point) and cacheline is 64 bytes, a 16× acceleration can be achieved. Irrespective of this optimization, the computational and spatial complexities are $O(nkd)$ and $O(nk + d)$, respectively.

Proposition 5.1. *Algorithm 3 is (cacheline-level) fully oblivious.* (A formal proof is provided in Appendix C of [34].)

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:         $flag \leftarrow idx^* == idx$                    ▷ target index or not
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: $g' \leftarrow \{(1,0), \ldots, (d,0)\}$                ▷ all values are zero
3: $g \leftarrow g \| g'$                                   ▷ concatenation
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 ▷ Note: start from the second weight of $g$
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 take the first $d$ values as $g^*$
```

5.2 Advanced method

Here, we present a more advanced approach to FL aggregation. In cases with large numbers of model parameters, k and d are significant factors and the computational complexity of the Baseline method becomes extremely high because of 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 the gradient data, g, to eliminate access to each memory address of the aggregated gradients, g^*. This avoids the overhead incurred by dummy access to g^*, as in the Baseline. The method is divided into four main steps: *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 Batcher's Bitonic Sort [6], which is implemented in a register-level oblivious manner using oblivious swap (`o_swap`) to compare and swap at all comparators in the sorting network obliviously. Appendix E in [34] illustrates a running example for better understanding.

As given by Algorithm 4, we first apply an initialization to g, where we prepare zero-valued gradients for each index between 1 and d (declared g') and concatenate them with g (lines 1–3). Thus, g has length $nk + d$. This process guarantees that g has at least one weight indexed for each value between 1 and d; however, aggregation of the concatenated g yields exactly the same result as the original g because the added values are all zero. We then apply an oblivious sort to g using the parameter's index (lines 4–5). Rather than eliminating the connection between the client and gradient, this serves as a preparation for subsequent operations to compute the per-index aggregate values. Next, the *oblivious folding* routine is executed (lines 6–14). It linearly accesses the values of g and cumulatively writes the sum of the values for each index in g. Starting from the first place, it adds each value to the subsequent value if the neighboring indices are identical, and writes a zero-valued dummy index, M_0, in place of the original one. M_0 is a large integer. Otherwise, if the neighboring indices are different, we stop adding values, and the summation of the new index is initiated anew. Thus, we finally obtain g such that only the last weight of each index bears the correct index and aggregated value, and all the remaining ones bear dummy indices. In addition, the initialization process described above guarantees that d distinct indices always exist. In this phase, the index change-points on g during folding are carefully hidden. If the index change-points are exposed, the number corresponding to each index (i.e., the histogram of the indices) is leaked, which can cause catastrophic results. Therefore, oblivious folding employs `o_mov` to make conditional updates oblivious and hide not only the memory access of the data, but also low-level instructions. Finally, we apply an oblivious sort to g (lines 15–16). After sorting, in g, weights with indices between 1 and d are arranged individually, followed by weights with dummy indices. Finally, taking the `values` of the first d weights of the sorted g, we return this as the final aggregated gradient, g^* (line 17).

PROPOSITION 5.2. *Algorithm 4 is fully oblivious.*

PROOF. The access pattern, $\textbf{Accesses}^{\textbf{advanced}}$, is somewhat complicated, but obliviousness can be considered using a modular approach. Our oblivious sort relies on Batcher's Bitonic Sort, in which sorting is completed by comparing and swapping the data in a deterministic order, irrespective of the input data. Therefore, access patterns generated using this method are always identical. In oblivious folding, the gradient is linearly accessed once; thus, the generated access pattern is identical for all input data of equal length. Finally, $\textbf{Accesses}^{\textbf{advanced}}$ are identical and independent of inputs of equal length, this implies 0-statistical obliviousness. □

The complexity of the entire operation is $O((nk+d)\log^2(nk+d))$ in time and $O(nk+d)$ in space. The proposed algorithm relies on an oblivious sort, which dominates the asymptotic computational complexity. We use Batcher's Bitonic Sort [6], which has $O(n \log^2 n)$ time complexity. The Advanced is asymptotically better than the Baseline because of the elimination of the kd term.

5.3 Optimization

In this subsection, we describe an optimization method that fits the basic SGX memory characteristics. The current SGX comprises two major levels of memory size optimization. The first factor is the size of the L3 cache (e.g., 8 MB). In SGX, the acceleration is significant because the cache hit reduces not only the memory access time but also the data-decrypting process. The second factor is the EPC size (e.g., 96 MB). As mentioned in Section 2.2, accessing data outside the EPC incurs serious paging overhead. Compared to the proposed methods, the Baseline is computationally expensive; however, most memory accesses are linear. Thus, it is greatly accelerated by the high cache hit rates and the prefetch functionality of the CPU. However, in Advanced, the low locality of memory accesses in Batcher's sort reduces the cache and EPC hit rates.

Therefore, optimization is performed by introducing a function to split users into appropriate groups before executing Advanced to keep the data processed at one time within the EPC size. This procedure involves the following steps: (1) divide into groups of h users each; (2) aggregate values for each group using Advanced; (3) record the aggregated value in the enclave, and carry over the result to the next group; and (4) only average the result when all groups have been completed and then load them from the enclave to the untrusted area. Note that the improvement to Advanced does not change its security characteristics. An external attacker can only see the encrypted data, and any irregularities in the order or content of the grouped data can be detected and aborted by enclave. The key parameter is the number of people, h, in each group. The overall computational complexity increases slightly to $O(n/h((hk+d)\log^2(hk+d)))$. However, this hides the acceleration induced by cache hits and/or the overhead incurred by repeated data loading. Basically, although lowering h improves the benefit of cache hits, lowering it too much results in a large amount of data loading. The optimal value of h is independent of data and can be explored offline. Our results indicate that there exists an optimal h that achieves the highest efficiency in the experiment.

5.4 Relaxation of Obliviousness

We investigate further improvements by relaxing the condition of full obliviousness to achieve better efficiency. A relaxed security definition that has recently garnered attention is that of *differentially oblivious* (DO) [2, 14, 17, 42, 54]. DO is DP applied to obliviousness. This relaxation can theoretically improves the efficiency from full obliviousness. In practice, improvements have been reported for RDB queries [54] whose security model, in which access pattern leakage within the enclave is out of the scope, differs from ours.

However, DO is unlikely to work in the FL setting. DO approaches commonly guarantee DP for the histogram of observed memory accesses. We construct a DO algorithm based on [2, 42]. The procedure involves the following steps: pad dummy data, perform an obvious shuffle (or sorting), and update g^* by performing linear access on $\mathbf{G}$. The observed memory access pattern is equivalent to a histogram of the indices corresponding to all gradients, and the dummy data are required to be padded with sufficient random noise to make this histogram DP. However, this inevitably incurs prohibitive costs in the FL setting. The first reason for this is that the randomization mechanism can only be implemented by padding dummy data [13], which implies that only positive noise can be added, and the algorithms covered by padding are limited (e.g., the shifted Laplace mechanism). The second reason is critical in our case and differs from previous studies [2, 42]. Considering that the ML model dimension, d, and even the sparsified dimension, k, can be large, noise easily becomes significant. For example, considering the DO guaranteed by Laplace noise, where k denotes the sensitivity and d is the dimension of the histogram, the amount of noise is proportional to kd and multiplied by a non-negligible constant, owing to the first reason [2]. This produces huge array data to which oblivious operations must be applied, resulting in a larger overhead than in the fully oblivious case.

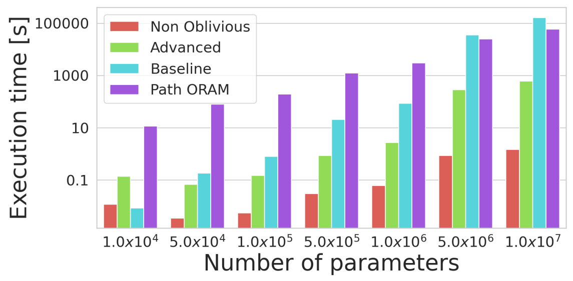

Figure 9: Performance results on a synthetic dataset w.r.t. models of various sizes: *Advanced* functions efficiently. α **(sparse ratio)** $= 0.01$ **and** n **(number of clients per round)** $= 100$.

5.5 Experimental results

In this section, we demonstrate the efficiency of the designed defense method on a practical scale. Because it is obvious that the proposed algorithms provide complete defense against our attack method, their attack performances are not evaluated here. In addition, our previous algorithms do not degrade utility—the only trade-off for enhanced security is computational efficiency.

Setup: We use an HP Z2 SFF G4 Workstation with a 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, of which 96 MB EPC is available for user use. We use the same datasets as those in Table 1 and synthetic data. Note that the proposed method is fully oblivious and its efficiency depends only on the model size. The aggregation methods are the *Non Oblivious* (*linear* algorithm in Section 3.3), the *Baseline* (Algorithm 3), the *Advanced*(Algorithm 4), and *PathORAM*. We implement PathORAM based on an open-source library[4] that involves a Rust implementation of Zerotrace [59]. The stash size is fixed to 20. In the experiments, we use *execution time* as an efficiency metric. We measure the time required by an untrusted server from loading the encrypted data to the enclave to completion of aggregation.

Results: Figure 9 depicts the execution time for the aggregation operation on the synthetic dataset with respect to model size. α is fixed to 0.01, and the x-axis represents the original model parameter size, d. The proposed *Advanced* is approximately one order of magnitude faster than *Baseline*. Moreover, it is more robust with respect to an increase in the number of parameters. Only when the number of parameters is very small is *Baseline* faster than *Advanced*, because when the model is extremely small, *Baseline*'s simplicity becomes dominant. *PathORAM* also incurs a large overhead. The theoretical asymptotic complexity of the original PathORAM-based algorithm is $O((nk)\log(d))$ because a single update on ORAM can be performed in $O(\log(d))$. However, this is an ideal case and the overhead of the constant factor is large when PathORAM is adapted to the SGX security model (i.e., ZeroTrace [59]). The overhead is

[4]https://github.com/mobilecoinofficial/mc-oblivious

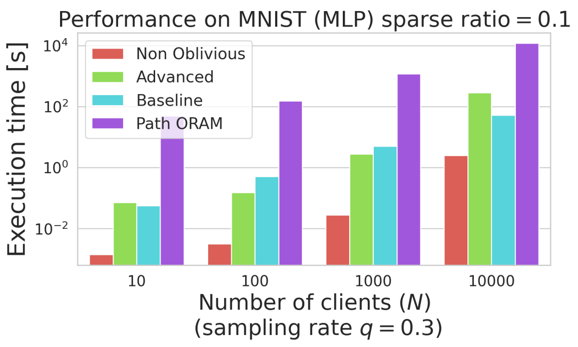

Figure 10: Performance results w.r.t. various numbers of clients (N) at low sparsity ($\alpha = 0.1$): the *Advanced* gradually worsens with increasing number of clients.

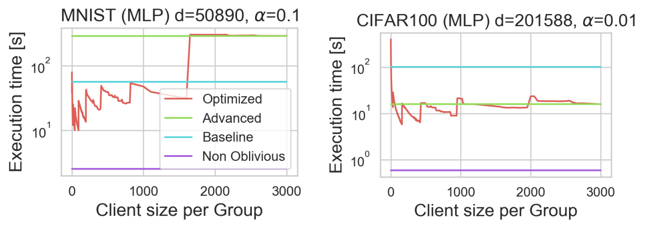

Figure 11: The effects of optimizing the *Advanced* on MLP models on MNIST (left) and CIFAR100 (right).

primarily induced by the *refresh* operation corresponding to each update and the oblivious reading of the position maps. The result suggests that *PathORAM*'s superiority does not appear until the data size increases hugely. Overall, the results indicate that the aggregation process can be completed in a few seconds, even if the model scale involves approximately 1M parameters.

Figure 10 depicts the performances on MNIST (MLP) corresponding to various numbers of clients and low sparsity ($\alpha = 0.1$). The *Baseline* method is more efficient when the number of clients, N, is large (10^4). Firstly, the model size d is fairly small (i.e., MNIST (MLP) consists of only 50K parameters). Hence, the overhead of the dummy access operations of *Baseline* is not significant. The second reason is that the lower sparsity and higher number of clients increases nk, which increases the overhead for both *Baseline* and *Advanced*, but affects *Advanced* more, as explained by the analysis of cache hits in Section 5.3. At $N = 10^4$, the memory size required by *Advanced* is given by (vector to be obliviously sorted) $= 5089 * 8 * 3000 + 50890 * 8 \approx 122$ MB (> 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 require a large number of pagings until *Advanced* finishes; however, in *Baseline*, this hardly occurs. However, the optimization introduced in Section 5.3 successfully addresses this problem.

Figure 11 illustrates the effects of the optimization method on *Advanced*. The left figure shows the results under the same conditions as the rightmost bars in Figure 10 ($N = 10^4$), indicating that *Advanced* is dramatically faster with an optimal client size. When the number of clients per group, h (represented along the x-axis), is small, the costs of iterative loading to the enclave become dominant, and the overhead conversely increases. However, if h is gradually increased, the execution time decreases. Considering that the size of the L3 cache is 8 MB and data size per user is $d\alpha = 0.04$ MB, the L3 cache can accommodate up to approximately 200 clients. The results of MNIST (MLP) indicate that the lowest is, approximately 10 s, at around $h = 100$, which is a significant improvement compared to 290 s in the original *Advanced*. The small waviness of the plot appears to be related to the L2 cache (1 MB), which does not have an impact as large as that of the L3 cache. The efficiency decreases significantly around $h = 2000$, owing to the EPC paging. The figure on the right depicts the results on CIFAR100 (MLP) at $\alpha = 0.01$ and $N = 10^4$. In this case, *Advanced* is initially much faster, but there is an optimal h that can be further improved. The pre-optimization execution time of 16 s is reduced to 5.7 s at around 150 clients.

5.6 Discussion

Threat assumption. Boenisch et al. [9] reported that *malicious* servers improve inference attack performance beyond *semi-honest*. This type of attack involves crafting global model parameters (called *trap weights* in [9]) and controlling client selection in rounds to highlight the updates of the target user by a malicious server. To prevent parameter tampering, [11] proposed a defense strategy using a cryptographic commitment scheme. The Olive can adopt a similar strategy based on a cryptographic signature. Aggregation is performed within the enclave, and the aggregated global model is signed with the private key in the enclave. This ensures that the model is not tampered with outside the enclave, i.e., malicious server. Any client can verify this using a public key which can be easily distributed after RA. In addition, TEE prevents malicious client selection by securely running in the enclave. Therefore, privacy is not violated at least such type of the attack. Other possible malicious server behaviors can influence the security of the Olive, including denial-of-service (DoS) attacks [30], which are outside the threat model of the Olive, as well as TEE and are difficult to prevent.

Security of SGX. Finally, we discuss the use of SGX as a security primitive against known attacks. According to [51], the objectives of attacks against SGX can be classified into the following three: (1) stealing memory/page access patterns or instruction traces [12, 36, 71, 76], (2) reading out memory content [15, 69], and (3) fault injection [48]. (1) is the target of our defense. The speculative execution attacks of (2) are mostly handled by microcode patches. Hence, the protection is usually not required in the application. However, if the microcode is not updated, the gradient information of the enclave may be stolen by a malicious attacker, which is beyond the scope of this study. The fault injection of (3) is covered within the scope of microcode/hardware [48, 51] and lies outside our security. This may cause DoS even using TEE [30].

In addition, another risk exists if malicious code is embedded in the code executed in the enclave. This can be prevented by verifying the enclave state using RA; however, this requires the source code to be publicly available and assessed. Further, as discussed in [70], the SDK may involve unintended vulnerabilities. To benefit from the security of SGX, the code of TCB must be written properly.

6 RELATED WORKS

Security and Privacy threats in FL. FL contains many attack surfaces because of its decentralized and collaborative scheme. These can be broadly classified into inference attacks by semi-honest parties [21, 49, 72] and attacks that degrade or control the quality of the model by malicious parties [5, 66, 80]. However, [9] demonstrated that malicious servers may enable effective inference attacks by crafting aggregated parameters. Our target is taken to be an inference attack by a semi-honest server. Inference attacks include reconstruction [7, 27], membership [49], and label inferences [21, 72]. In particular, it has been reported that shared parameters observed by a server contain large amounts of private information [79, 84]. Our work targets gradient-based label inference attacks, [21, 72] use the gradients themselves, focusing on the values, and not only on the indices leaking from the side-channel, as in our method. To the best of our knowledge, this is the first study to demonstrate label inference using only sparsified index information.

Secure aggregation (SA) [46] is a popular FL method for concealing individual parameters from the server and it is based on the lightweight pairwise-masking method [10, 19, 32], homomorphic encryption [4, 26] or TEE [78, 80]. Another approach is to ensure (local) DP for the parameter to privatize the shared data; however, this sacrifices the utility of the model [67, 81, 82]. In this study, we study SA using TEE—further details are provided in the next paragraph. Recent studies have investigated combinations of SA and sparsification, such as random-k [19] and top-k [40]. However, these are not in harmony because they require the same sparsified indices among clients for mask cancellation. [40] proposed generation of common masks by taking a union set of top-k indexes among clients, which incurs extra communication costs and strong constraints. This can be serious for the top-k because, in fact, Ergun et al. [19] showed that the top-k indices exhibits little overlap between clients, which is especially noticeable in the non-i.i.d. as in FL. In [19], only a pair of users exhibited a common index; however, this was applicable only to random-k sparsification. In the case of TEE, a common index or random-k is not required; but, individual indices can still be leaked through side-channels. Therefore, our work focuses on attacks and defense strategies at this point.

FL with TEE. Using TEE in FL is a promising approach [16, 45, 47, 77, 78] in this context. In addition to the confidentiality of gradients (i.e., SA functionality), TEE provides remote program integrity and verifiability via *remote attestation*. The major difference from centralized ML using TEE [29, 52] is that the training data are not shared to the server and they are not centralized in the latter case, which can be critical because of privacy or contractual/regulatory reasons or for practical reasons, i.e., big and fast data at multiple edges. It is also important to outsource heavy computations required for ML training from TEE's limited computational resources to external clients. PPFL [45] uses a TEE to hide parameters to prevent semi-honest client and server attacks on a global model. Citadel [77] addressed the important goal of making the design of models confidential in collaborative ML using TEE. However, side-channel attacks were not covered. In [78] and [16], the gradient aggregation step was taken to be hierarchical and/or partitioned using multiple servers such that the gradient information could only be partially observed by each server. The authors assumed reconstruction attack and that a gradient leakage of less than 80% was acceptable, which differs from our assumption completely. In this study, the attack is based only on the gradient index information, and the goal is label inference. Further, our proposed defense is more practical since we require only one server and one TEE, compared to the aforementioned method of distributed processing, which assumes multiple non-colluding servers with TEEs. Flatee [47] used TEE and DP in FL. [47] mentioned server-side obliviousness, but did not provide any analysis and solution for the leakages via side-channels. Our study includes an analysis of access patterns in the aggregation procedure of FL and the design and demonstration of attack methods to motivate our defenses thoroughly in addition to specific solutions that lead to stronger security than any other method in FL on a single central TEE.

Oblivious techniques. The oblivious algorithm [25, 52, 65] is known to induce only independent memory access patterns for the input data. Although PathORAM [65] is the most efficient ORAM implementation, it assumes a private memory space of a certain size (called as *client storage*) and is not applicable to Intel SGX [59]. Zerotrace [59] adapted PathORAM to the SGX security model, in which the register is only private memory. The authors used the oblivious primitive proposed in [52], in which the program did not leak instruction sequences from the CPU register, using x86 conditional instructions. Our proposed algorithm also uses the low-level primitives; however, high-level algorithms are considerably different. [83] studied oblivious SQL processing. Their proposal included a *group-by* query, which is similar to our proposed algorithm in concept. Our aggregation algorithm computes the summed dense gradients based on multiple sparse gradients, which can be viewed as a special case of the *group-by* query. But, our method is more specialized, for instance, we first prepare the zero-initialized dense gradients to hide the all of index set that are included and then obliviously aggregated, which is impossible in the case of *group-by*. In addition, the aforementioned algorithms are fundamentally different because they focus on the data distributed across nodes. Further, [83] did not consider the technique proposed by [52] for linear access, which can induce additional information leaks in the conditional code [76]. [56, 64] studied compiling and transforming approaches from high-level source code to low-level oblivious code. They proposed a compiler that automatically identifies non-oblivious parts of the original source code and fixes them. But, the authors did not provide customized high-level algorithms for specific purposes, unlike our method. The Differentially Obliviousness (DO) [2, 14, 54] is described in detail in Section 5.4.

7 CONCLUSIONS

In this study, we analyzed the risks of FL with server-side TEE in a sparsified gradient setting, and designed and demonstrated a novel inference attack using gradient index information that is observable from side-channels. To mitigate these risks, we proposed an oblivious federated learning system, called the Olive, by designing fully oblivious but efficient algorithms. Our experimental results demonstrated that the proposed algorithm is more efficient than the state-of-the-art general-purpose ORAM and can serve as a practical method on a real-world scale. We believe that our study is useful for realizing privacy-preserving FL using a TEE.

REFERENCES

[1] Mohammed Aledhari, Rehma Razzak, Reza M Parizi, and Fahad Saeed. 2020. Federated learning: A survey on enabling technologies, protocols, and applications. *IEEE Access* 8 (2020), 140699–140725.

[2] 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).

[3] Galen 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).

[4] 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.

[5] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How To Backdoor Federated Learning. In *Proceedings of the Twenty Third International Conference on Artificial Intelligence and Statistics (Proceedings of Machine Learning Research)*, Silvia Chiappa and Roberto Calandra (Eds.), Vol. 108. PMLR, 2938–2948. https://proceedings.mlr.press/v108/bagdasaryan20a.html

[6] Kenneth E Batcher. 1968. Sorting networks and their applications. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*. 307–314.

[7] Abhishek Bhowmick, John Duchi, Julien Freudiger, Gaurav Kapoor, and Ryan Rogers. 2018. Protection against reconstruction and its applications in private federated learning. *arXiv preprint arXiv:1812.00984* (2018).

[8] Google AI Blog. 2017. Federated Learning: Collaborative Machine Learning without Centralized Training Data. https://ai.googleblog.com/2017/04/federated-learning-collaborative.html

[9] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. 2021. When the curious abandon honesty: Federated learning is not private. *arXiv preprint arXiv:2112.02918* (2021).

[10] Keith Bonawitz, Vladimir Ivanov, Ben Kreuter, Antonio Marcedone, H. Brendan McMahan, Sarvar Patel, Daniel Ramage, Aaron Segal, and Karn Seth. 2017. Practical Secure Aggregation for Privacy-Preserving Machine Learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security* (Dallas, Texas, USA) *(CCS '17)*. Association for Computing Machinery, New York, NY, USA, 1175–1191. https://doi.org/10.1145/3133956.3133982

[11] Simone Bottoni, Giulio Zizzo, Stefano Braghin, and Alberto Trombetta. 2022. Verifiable Federated Learning. In *Workshop on Federated Learning: Recent Advances and New Challenges (in Conjunction with NeurIPS 2022)*. https://openreview.net/forum?id=0HIa3HIyIHN

[12] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. 2017. Software grand exposure:SGX cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies*.

[13] Benjamin M Case, James Honaker, and Mahnush Movahedi. 2021. The Privacy-preserving Padding Problem: Non-negative Mechanisms for Conservative Answers with Differential Privacy. *arXiv preprint arXiv:2110.08177* (2021).

[14] TH Hubert Chan, Kai-Min Chung, Bruce M Maggs, and Elaine Shi. 2019. Foundations of differentially oblivious algorithms. In *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2448–2467.

[15] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. 2019. Sgxpectre: Stealing intel secrets from sgx enclaves via speculative execution. In *2019 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 142–157.

[16] Pau-Chen Cheng, Kevin Eykholt, Zhongshu Gu, Hani Jamjoom, K. R. Jayaram, Enriquillo Valdez, and Ashish 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, 16–18. https://doi.org/10.1145/3477114.3488765

[17] Shumo Chu, Danyang Zhuo, Elaine Shi, and TH Hubert Chan. 2021. Differentially Oblivious Database Joins: Overcoming the Worst-Case Curse 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), 1–118.

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

[20] Úlfar Erlingsson, Vitaly Feldman, Ilya Mironov, Ananth Raghunathan, Shuang Song, Kunal Talwar, and Abhradeep Thakurta. 2020. Encode, shuffle, analyze privacy revisited: Formalizations and empirical evaluation. *arXiv preprint arXiv:2001.03618* (2020).

[21] Chong Fu, Xuhong Zhang, Shouling Ji, Jinyin Chen, Jingzheng Wu, Shanqing Guo, Jun Zhou, Alex X Liu, and Ting Wang. 2022. Label inference attacks against vertical federated learning. In *31st USENIX Security Symposium (USENIX Security 22)*. 1397–1414.

[22] Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *NIPS 2017 Workshop: Machine Learning on the Phone and other Consumer Devices* (2017).

[23] 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.

[24] 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.

[25] 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.

[26] Meng Hao, 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.

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

[28] Rui Hu, Yanmin Gong, and Yuanxiong Guo. 2022. Federated learning with sparsified model perturbation: Improving accuracy under client-level differential privacy. *arXiv preprint arXiv:2202.07178* (2022).

[29] Tyler Hunt, Congzheng Song, Reza Shokri, Vitaly Shmatikov, and Emmett Witchel. 2018. Chiron: Privacy-preserving machine learning as a service. *arXiv preprint arXiv:1803.05961* (2018).

[30] Yeongjin Jang, Jaehyuk Lee, Sangho Lee, and Taesoo Kim. 2017. SGX-Bomb: Locking down the processor via Rowhammer attack. In *Proceedings of the 2nd Workshop on System Software for Trusted Execution*. 1–6.

[31] Bargav 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

[32] 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

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

[34] Fumiyuki Kato, Yang Cao, and Masatoshi Yoshikawa. 2023. Olive: Oblivious Federated Learning on Trusted Execution Environment against the risk of sparsification. *arXiv preprint arXiv:2202.07165* (2023).

[35] Jakub Konečnỳ, 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:1610.05492* (2016).

[36] Sangho Lee, Ming-Wei Shih, Prasun Gera, Taesoo Kim, Hyesoon Kim, and Marcus Peinado. 2017. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing.. In *USENIX Security Symposium*, Vol. 19. 16–18.

[37] Yujun Lin, Song Han, Huizi Mao, Yu Wang, and William J Dally. 2018. Deep Gradient Compression: Reducing the communication bandwidth for distributed training. In *The International Conference on Learning Representations*.

[38] Ruixuan Liu, Yang Cao, Hong Chen, Ruoyang Guo, and Masatoshi Yoshikawa. 2021. Flame: Differentially private federated learning in the shuffle model. In *AAAI*.

[39] Ruixuan Liu, Yang Cao, Masatoshi Yoshikawa, and Hong Chen. 2020. Fedsel: Federated sgd under local differential privacy with top-k dimension selection. In *International Conference on Database Systems for Advanced Applications*. Springer, 485–501.

[40] Shiwei Lu, Ruihu Li, Wenbin Liu, Chaofeng Guan, and Xiaopeng Yang. 2023. Top-k sparsification with secure aggregation for privacy-preserving federated learning. *Computers & Security* 124 (2023), 102993.

[41] Kajetan Maliszewski, Jorge-Arnulfo Quiané-Ruiz, Jonas Traub, and Volker Markl. 2021. What is the price for joining securely? benchmarking equi-joins in trusted execution environments. *Proceedings of the VLDB Endowment* 15, 3 (2021), 659–672.

[42] Sahar Mazloom and S. Dov Gordon. 2018. Secure Computation with Differentially Private Access Patterns. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security* (Toronto, Canada) *(CCS '18)*. Association for Computing Machinery, New York, NY, USA, 490–507. https://doi.org/10.1145/3243734.3243851

[43] H Brendan McMahan, Eider Moore, Daniel Ramage, and Blaise Agüera y Arcas. 2016. Federated learning of deep networks using model averaging. *arXiv preprint arXiv:1602.05629* (2016).

[44] H. Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2018. Learning Differentially Private Recurrent Language Models. In *6th International Conference on Learning Representations, ICLR 2018, Vancouver, BC, Canada, April 30 - May 3, 2018, Conference Track Proceedings*. OpenReview.net. https://openreview.net/forum?id=BJ0hF1Z0b

[45] Fan Mo, Hamed Haddadi, Kleomenis Katevas, Eduard Marin, Diego Perino, and Nicolas Kourtellis. 2021. PPFL: Privacy-Preserving Federated Learning with Trusted Execution Environments. In *Proceedings of the 19th Annual International Conference on Mobile Systems, Applications, and Services* (Virtual Event, Wisconsin) *(MobiSys '21)*. Association for Computing Machinery, New York, NY, USA, 94–108. https://doi.org/10.1145/3458864.3466628

[46] Mansouri Mohamad, Malek Onen, Wafa Ben Jaballah, and Mauro Contu. 2023. SoK: Secure Aggregation based on cryptographic schemes for Federated Learning. In *Proceedings of Privacy Enhancing Technologies Symposium*, Vol. 1.

[47] A. Mondal, Y. More, R. Rooparaghunath, and D. Gupta. 2021. Poster: FLATEE: Federated Learning Across Trusted Execution Environments. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE Computer Society, Los Alamitos, CA, USA, 707–709. https://doi.org/10.1109/EuroSP51992.2021.00054

[48] Kit Murdock, David Oswald, Flavio D Garcia, Jo Van Bulck, Daniel Gruss, and Frank Piessens. 2020. Plundervolt: Software-based fault injection attacks against Intel SGX. In *2020 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1466–1482.

[49] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2019. Comprehensive privacy analysis of deep learning: Passive and active white-box inference attacks against centralized and federated learning. In *2019 IEEE symposium on security and privacy (SP)*. IEEE, 739–753.

[50] John Nguyen, Kshitiz Malik, Hongyuan Zhan, Ashkan Yousefpour, Mike Rabbat, Mani Malek, and Dzmitry Huba. 2022. Federated learning with buffered asynchronous aggregation. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 3581–3607.

[51] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. 2020. A survey of published attacks on Intel SGX. *arXiv preprint arXiv:2006.13598* (2020).

[52] Olga Ohrimenko, Felix Schuster, Cédric Fournet, Aastha Mehta, Sebastian Nowozin, Kapil Vaswani, and Manuel Costa. 2016. Oblivious multi-party machine learning on trusted processors. In *25th USENIX Security Symposium (USENIX Security 16)*. 619–636.

[53] Matthias Paulik, Matt Seigel, Henry Mason, Dominic Telaar, Joris Kluivers, Rogier van Dalen, Chi Wai Lau, Luke Carlson, Filip Granqvist, Chris Vandevelde, et al. 2021. Federated Evaluation and Tuning for On-Device Personalization: System Design & Applications. *arXiv preprint arXiv:2102.08503* (2021).

[54] Lianke Qin, Rajesh Jayaram, Elaine Shi, Zhao Song, Danyang Zhuo, and Shumo Chu. 2022. Adore: Differentially Oblivious Relational Database Operators. *Proc. VLDB Endow.* 16, 4 (dec 2022), 842–855. https://doi.org/10.14778/3574245.3574267

[55] Swaroop Ramaswamy, Rajiv Mathews, Kanishka Rao, and Françoise Beaufays. 2019. Federated learning for emoji prediction in a mobile keyboard. *arXiv preprint arXiv:1906.04329* (2019).

[56] Ashay Rane, Calvin Lin, and Mohit Tiwari. 2015. Raccoon: Closing digital side-channels through obfuscated execution. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*. 431–446.

[57] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 57–64.

[58] Atal Sahu, Aritra Dutta, Ahmed M Abdelmoniem, Trambak Banerjee, Marco Canini, and Panos Kalnis. 2021. Rethinking gradient sparsification as total error minimization. *Advances in Neural Information Processing Systems* 34 (2021), 8133–8146.

[59] Sajin Sasy, Sergey Gorbunov, and Christopher W. Fletcher. 2018. ZeroTrace: Oblivious Memory Primitives from Intel SGX. In *25th Annual Network and Distributed System Security Symposium, NDSS 2018, San Diego, California, USA, February 18-21, 2018*. The Internet Society.

[60] Felix Sattler, Simon Wiedemann, Klaus-Robert Müller, and Wojciech Samek. 2019. Robust and communication-efficient federated learning from non-iid data. *IEEE transactions on neural networks and learning systems* 31, 9 (2019), 3400–3413.

[61] Osama Shahid, Seyedamin Pouriyeh, 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).

[62] Shaohuai Shi, Kaiyong Zhao, Qiang Wang, Zhenheng Tang, and Xiaowen Chu. 2019. A Convergence Analysis of Distributed SGD with Communication-Efficient Gradient Sparsification.. In *IJCAI*. 3411–3417.

[63] Reza Shokri, Marco 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.

[64] Rohit Sinha, Sriram Rajamani, and Sanjit A Seshia. 2017. A compiler and verifier for page access oblivious computation. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 649–660.

[65] Emil Stefanov, Marten van Dijk, Elaine Shi, Christopher Fletcher, Ling Ren, Xiangyao 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* (Berlin, Germany) *(CCS '13)*. Association for Computing Machinery, New York, NY, USA, 299–310. https://doi.org/10.1145/2508859.2516660

[66] Lili Su and Jiaming Xu. 2019. Securing Distributed Gradient Descent in High Dimensional Statistical Learning. *Proc. ACM Meas. Anal. Comput. Syst.* 3, 1, Article 12 (mar 2019), 41 pages. https://doi.org/10.1145/3322205.3311083

[67] 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*, Zhi-Hua Zhou (Ed.). International Joint Conferences on Artificial Intelligence Organization, 1571–1578. https://doi.org/10.24963/ijcai.2021/217 Main Track.

[68] Meysam Taassori, Ali Shafiee, and Rajeev Balasubramonian. 2018. VAULT: 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.

[69] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings fo the 27th USENIX Security Symposium*. USENIX Association.

[70] Jo Van Bulck, David Oswald, Eduard Marin, Abdulla Aldoseri, Flavio D Garcia, and Frank Piessens. 2019. A tale of two worlds: Assessing the vulnerability of enclave shielding runtimes. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. 1741–1758.

[71] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. 2017. Telling your secrets without page faults: Stealthy page table-based attacks on enclaved execution. In *26th USENIX Security Symposium (USENIX Security 17)*. 1041–1056.

[72] Aidmar Wainakh, Fabrizio Ventola, Till Müßig, Jens Keim, Carlos Garcia Cordero, Ephraim 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.

[73] Teng Wang, Xuefeng 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

[74] Zhibo Wang, Mengkai Song, Zhifei Zhang, Yang Song, Qian Wang, and Hairong Qi. 2019. Beyond inferring class representatives: User-level privacy leakage from federated learning. In *IEEE INFOCOM 2019-IEEE conference on computer communications*. IEEE, 2512–2520.

[75] Donglei Wu, Xiangyu Zou, Shuyu Zhang, Haoyu Jin, Wen Xia, and Binxing Fang. 2022. SmartIdx: Reducing Communication Cost in Federated Learning by Exploiting the CNNs Structures. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 4254–4262.

[76] 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.

[77] Chengliang Zhang, Junzhe Xia, Baichen Yang, Huancheng Puyang, Wei Wang, Ruichuan Chen, Istemi Ekin Akkus, Paarijaat Aditya, and Feng Yan. 2021. Citadel: Protecting Data Privacy and Model Confidentiality for Collaborative Learning with SGX. *arXiv preprint arXiv:2105.01281* (2021).

[78] Yuhui Zhang, Zhiwei Wang, Jiangfeng Cao, Rui Hou, and Dan Meng. 2021. ShuffleFL: gradient-preserving federated learning using trusted execution environment. In *Proceedings of the 18th ACM International Conference on Computing Frontiers*. 161–168.

[79] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2020. idlg: Improved deep leakage from gradients. *arXiv preprint arXiv:2001.02610* (2020).

[80] Lingchen Zhao, Jianlin Jiang, Bo Feng, Qian Wang, Chao Shen, and Qi Li. 2021. Sear: Secure and efficient aggregation for byzantine-robust federated learning. *IEEE Transactions on Dependable and Secure Computing* 19, 5 (2021), 3329–3342.

[81] Qi Zhao, Chuan Zhao, Shujie Cui, Shan Jing, and Zhenxiang Chen. 2020. PrivateDL: privacy-preserving collaborative deep learning against leakage from gradient sharing. *International Journal of Intelligent Systems* 35, 8 (2020), 1262–1279.

[82] Yang Zhao, Jun Zhao, Mengmeng Yang, Teng Wang, Ning Wang, Lingjuan Lyu, Dusit Niyato, and Kwok-Yan Lam. 2020. Local differential privacy-based federated learning for internet of things. *IEEE Internet of Things Journal* 8, 11 (2020), 8836–8853.

[83] Wenting Zheng, Ankur Dave, Jethro G Beekman, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2017. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *NSDI*, Vol. 17. 283–298.

[84] Ligeng Zhu and Song Han. 2020. Deep leakage from gradients. In *Federated learning*. Springer, 17–31.