DeTA: Minimizing Data Leaks in Federated Learning via Decentralized and Trustworthy Aggregation

Pau-Chen Cheng
IBM Research
New York, USA

Kevin Eykholt
IBM Research
New York, USA

Zhongshu Gu*
IBM Research
New York, USA

Hani Jamjoom
IBM Research
New York, USA

K. R. Jayaram*
IBM Research
New York, USA

Enriquillo Valdez
IBM Research
New York, USA

Ashish Verma†
Amazon Inc.
New York, USA

Abstract

Federated learning (FL) relies on a central authority to oversee and aggregate model updates contributed by multiple participating parties in the training process. This centralization of sensitive model updates naturally raises concerns about the trustworthiness of the central aggregation server, as well as the potential risks associated with server failures or breaches, which could result in loss and leaks of model updates. Moreover, recent attacks have demonstrated that, by obtaining the leaked model updates, malicious actors can even reconstruct substantial amounts of private data belonging to training participants. This underscores the critical necessity to rethink the existing FL system architecture to mitigate emerging attacks in the evolving threat landscape. One straightforward approach is to fortify the central aggregator with confidential computing (CC), which offers hardware-assisted protection for runtime computation and can be remotely verified for execution integrity. However, a growing number of security vulnerabilities have surfaced in tandem with the adoption of CC, indicating that depending solely on this singular defense may not provide the requisite resilience to thwart data leaks.

To address the security challenges inherent in the centralized aggregation paradigm and enhance system resilience, we introduce DeTA, an FL system architecture that employs a decentralized and trustworthy aggregation strategy with a *defense-in-depth* design. In DeTA, FL parties locally divide and shuffle their model updates at the parameter level, creating random partitions designated for *multiple* aggregators, all of which are shielded within CC execution environments. Moreover, to accommodate the multi-aggregator FL ecosystem, we have implemented a two-phase authentication protocol that enables new parties to verify all CC-protected aggregators and establish secure channels to upstream their model updates. With DeTA, model aggregation algorithms can function without any alterations. However, each aggregator is now oblivious to model architectures, possessing only a fragmented and shuffled view of each model update. This approach effectively mitigates attacks aimed at tampering with the aggregation process or exploiting leaked model updates, while also preserving training accuracy and minimizing performance overheads.

CCS Concepts: • **Security and privacy → Systems security**; • **Computing methodologies → Machine learning approaches**.

Keywords: Federated Learning, Trusted Aggregation, Decentralized Aggregation, Parameter Shuffling

ACM Reference Format:
Pau-Chen Cheng, Kevin Eykholt, Zhongshu Gu, Hani Jamjoom, K. R. Jayaram, Enriquillo Valdez, and Ashish Verma. 2024. DeTA: Minimizing Data Leaks in Federated Learning via Decentralized and Trustworthy Aggregation. In *Nineteenth European Conference on Computer Systems (EuroSys '24), April 22–25, 2024, Athens, Greece.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3627703.3650082

*Corresponding authors: zgu@us.ibm.com, jayaramkr@us.ibm.com
†Affiliated with IBM Research during this work.

EuroSys '24, April 22–25, 2024, Athens, Greece

ACM ISBN 979-8-4007-0437-6/24/04.
https://doi.org/10.1145/3627703.3650082

1 Introduction

Federated learning (FL) [37, 49] is a collaborative training mechanism that enables multiple participating parties to collectively build a machine learning (ML) model. FL allows training participants to maintain their private data within their domains and share only model updates. The security advantages of FL make it an appealing choice for mutually distrusting or competing parties, as well as for those holding sensitive data, such as health or financial information, who aim to preserve the privacy of their data and models.

FL can be categorized into two types: *cross-device* and *cross-silo*. Cross-device FL is designed to scale to a vast number, often in the millions, of low-end mobile and Internet of Things (IoT) devices. However, only a fraction of them

may be online simultaneously as participant devices frequently and unpredictably join and leave the training process. Cross-device FL is typically initiated and managed by a well-established trusted orchestrator, such as Google [28] or Apple [15]. In contrast, cross-silo FL involves a small number (typically ranging from 2 to 100 [37]) of organizational or institutional participants, each of comparable size. These participants possess a substantial amount of isolated data and high-performance local training computing resources. Unlike cross-device FL, which operates under an undisputed central authority, all participants in cross-silo FL must collaboratively reach a consensus for the establishment of a central aggregator. This aggregator should be hosted within a secure and dependable environment. The inherent uncertainty surrounding aggregator deployment intensifies the concern about aggregator trustworthiness. This paper primarily focuses on the security aspects of model aggregation in cross-silo FL.

The dependence of FL on a central aggregator to coordinate and consolidate model updates naturally raises several security concerns. (1) FL participants are required to place blind trust in the central aggregator. In the presence of rogue host administrators or breached host systems, adversaries could gain access to the upstreamed model updates or manipulate the aggregation process. (2) The concentration of model updates in a single location may create a single point of failure. Adversaries only need to breach one system to obtain complete and intact model updates. (3) There was a widespread belief that exchanging model updates in FL communications was "privacy-preserving" and contained limited or no information about the raw training data. However, recent research works [8, 20, 22, 51, 82, 84, 88] have debunked this belief, demonstrating that private attributes can be inferred, and large fractions of training data can be reconstructed by exploiting model updates. This challenge to the privacy promises of FL becomes particularly significant in the presence of data breaches on aggregation servers.

Researchers have explored various methods to protect data privacy in FL aggregation, including differential privacy (DP) [5, 6, 23, 50, 67, 71], secure multi-party computation (SMC) [9, 53], and homomorphic encryption (HE) [1, 29]. However, these techniques still have their limitations in different aspects. DP-based aggregation protects data privacy by adding statistical noise to model updates. While effective, DP can significantly reduce the accuracy of the model and requires careful hyper-parameter tuning to minimize the loss [8, 16]. Moreover, FL systems relying solely on DP may still reveal individual participant models to the central aggregator, which may no longer be considered trustworthy, and leak the model to participants' business competitors. While SMC and HE can preserve data confidentiality in FL aggregation, they remain computationally expensive, with aggregation overheads significantly outpacing training time [9, 36].

In recent years, confidential computing (CC) [31, 34, 35, 39, 45, 48, 62], also known as trusted execution environment (TEE), has been widely integrated into modern CPUs. CC offers a practical and efficient means to protect data-in-use in isolated execution domains on untrusted machines (*e.g.,* hosted in public clouds) and allows users to remotely verify the execution integrity of workloads. CC has been extensively explored for protecting data privacy in ML applications [6, 25, 26, 32, 33, 57, 70]. To leverage CC in FL, one straightforward approach is to encapsulate the central aggregator within a CC execution environment [52, 54, 60]. This way, FL parties can upstream model updates into a verified and protected execution domain through secure channels with the root-of-trust on CC. However, it is important to note that CC is not a universal solution to all such challenges. We have witnessed the emergence of security vulnerabilities and attack vectors [10, 43, 44, 55, 72–75, 77, 78] along with the introduction of various CC technologies. Relying solely on a single line of defense may still leave the system vulnerable to data leaks in the event of any new security vulnerabilities disclosed in the future.

Our work is motivated by the following key insights:

(I) Concentrating model updates in a central aggregator discloses far more information than necessary for FL aggregation algorithms. Most aggregation algorithms involve coordinate-wise arithmetic operations across model updates. Thus, partitioning and permuting parameters within a model update do not affect aggregation results, having no impact on final model accuracy and convergence rate compared to traditional FL training. However, from the perspective of adversaries, obtaining fragmented and shuffled model updates offers no assistance for their malicious intentions, such as model theft, property inference, or data reconstruction. Consequently, our design advocates for decentralizing aggregation into multiple instances and internally shuffling each model update to mitigate the risk of data concentration.

(II) Recent attacks assume strong adversaries, whether *honest-but-curious* or with full control of the central aggregator. These adversaries can passively reconstruct training samples by solving specific optimization objectives [22, 84, 88], or actively manipulate model weights [8] and architectures [20] to accelerate reconstruction and scale to gradients computed on mini-batched training data. To address this issue, we advocate for FL participants to be able to verify the authenticity and integrity of aggregators before joining the training. All model updates must remain confidential from potential adversaries and should be computed with validated aggregation algorithms.

(III) While CC execution environments offer enhanced security, they may still be vulnerable to unforeseen exploits in the future. Our design aims to ensure that, even in the worst-case scenario where all CC-protected systems are breached, adversaries should not be able to piece together original

model updates of FL participants based on the data within the breached aggregators.

In this paper, we present DeTA, a cross-silo FL system architecture designed for trustworthy model aggregation and minimizing data leaks even under scenarios where aggregators might be breached or under the control of adversaries. It addresses crucial aspects such as participant model/data privacy, execution integrity, and computational efficiency. To achieve these goals, DeTA adopts a decentralized data aggregation strategy, incorporates parameter-level data shuffling, and employs a two-phase authentication protocol to verify aggregators running within CC execution environments. By combining these key techniques, DeTA provides a robust and resilient defense against data leakage threats.

Decentralized Data Aggregation. To break down information concentration, we establish multiple, rather than one, aggregators during training. At the party's side, each model update is disassembled at the parameter granularity and re-stitched to random partitions designated for different aggregators. Each aggregator only receives fragmentary fractions of model updates with no knowledge of their model architectures. Furthermore, users can deploy multiple aggregators to physical servers at different geo-locations. Thus, we can prevent an aggregator from becoming a *single point of failure* (*i.e.,* leaking entire and intact model updates) under security breaches.

Parameter-level Data Shuffling. As model aggregation algorithms only involve coordinate-wise operations, we allow parties to permute parameters within each already-partitioned model update to further obfuscate the information sent to each aggregator. The permutation changes dynamically at each training round. This mechanism ensures that adversaries cannot decipher the original internal data order of a model update without obtaining the permutation keys, which are kept in participant-controlled domains.

Trustworthy Multi-Aggregator Authentication. To support confidential and tamper-resistant model aggregation, we leverage hardware-assisted CC execution environments to isolate and shield all the decentralized aggregators. During the initial stage of FL aggregation bootstrapping, we employ a two-phase authentication protocol specifically for new parties to verify aggregators. Each party can verify the integrity of the aggregation process and its associated hardware root-of-trust before joining the training. Thus, we can prevent parties from sharing sensitive model updates with any compromised aggregators.

The key principle underlying DeTA is to establish a comprehensive *defense-in-depth* strategy that not only addresses known attacks but also anticipates adaptive attackers who may seek to breach the CC-protected aggregators by exploiting as-yet-unknown vulnerabilities. This multi-layered approach significantly elevates the barrier for potential attacks and bolsters the overall security posture of DeTA. In our security analysis, we demonstrated that the *defense-in-depth* design of DeTA can fundamentally prevent existing or adaptive attackers from reconstructing training data [22, 84, 88]. In addition, we evaluated DeTA's performance in training deep learning models of varying sizes across multiple datasets, including `MNIST`, `CIFAR-10`, and `VGG-16 (RVL-CDIP)`. We used a range of model aggregation algorithms and FL configurations and measured the accuracy/loss and latency. We demonstrated that DeTA can achieve the same level of accuracy/loss and converge at the same rate, with low-performance overheads compared to the traditional FL platform as the baseline.

2 Threat Model

We adopt the same threat model as outlined in prior research on FL data leakage attacks [8, 20, 22, 82, 84, 88]. This model assumes that adversaries targeting aggregators seek to gain access to the model updates. Their objectives range from obtaining participant models for economic gains to extracting sensitive information, such as private training data. Adversaries could achieve this either by compromising aggregation servers as external attackers or by possessing the necessary privileges to access the data as system administrators. These adversaries may vary in their level of aggressiveness, ranging from being honest-but-curious [22, 82, 84, 88] to more actively attempting to manipulate the aggregation process [8, 20].

It is important to note that our perspective begins with the premise that FL participants are considered victims. Therefore, participants themselves are inherently *trusted* entities within our threat model. The focus of DeTA is to mitigate data leakage issues originating from the aggregation side. This paper does not delve into addressing the misbehavior of FL parties, such as local data or model poisoning [4, 69], backdoor attacks [2], collusion among parties, or vulnerabilities and breaches of parties' systems. The attacks originating from parties are beyond the scope of this paper, and their corresponding mitigation strategies have been explored in an orthogonal line of research [7, 21, 61, 79, 80].

3 Background

To begin, we provide an overview of the model aggregation algorithms commonly employed in FL, highlighting the key algorithmic structure that serves as the foundation for decentralization and shuffling of DeTA. Following that, we delve deeper into the utilization of CC techniques within DeTA.

3.1 Model Aggregation Algorithms

In FL, `Federated Stochastic Gradient Descent (FedSGD)` [67] and `Federated Averaging (FedAvg)` [49] are the most common model aggregation algorithms for deep neural network

(DNN) training, employing iterative merging and synchronizing model updates. Other DNN aggregation methods, such as `Coordinate Median` [81], `Krum` [7], and `Paillier-based Fusion` [46, 71], share the similar algorithmic structure with additional security enhancements. DeTA can support all of them. Here we describe the algorithms of `FedSGD` and `FedAvg` in detail.

We use θ to denote model parameters and L for the loss function. Each party has its training data/label pairs (x_i, y_i). In `FedSGD`, the parties choose to share the gradients $\nabla_\theta L_\theta(x_i, y_i)$ for a data batch to the aggregator. The aggregator computes the gradient sum of all parties and lets the parties synchronize their model parameters: $\theta \leftarrow \theta - \eta \sum_{i=1}^{N} \nabla_\theta L_\theta(x_i, y_i)$. Alternatively in `FedAvg`, the parties train for several epochs locally and upload the parameters: $\theta^i \leftarrow \theta^i - \eta \nabla_{\theta^i} L_{\theta^i}(x_i, y_i)$ to the aggregator. The aggregator computes the weighted average of model parameters $\theta \leftarrow \sum_{i=1}^{N} \frac{n_i}{n} \theta^i$, where n_i is the size of training data on party i and n is the sum of all n_i. Then, the aggregator sends the aggregated model parameters back to the parties for synchronization.

`FedAvg` and `FedSGD` are equivalent if we train only one batch of data in a single FL training round and synchronize model parameters, because gradients can be computed from the difference of two successive model parameter uploads. As `FedAvg` allows parties to batch multiple SGD iterations before synchronizing updates, it would be more challenging for data leakage attacks to succeed.

Algorithmic Structure of Model Aggregation. We observe that these model aggregation algorithms share a similar algorithmic structure, *i.e.,* they only involve coordinate-wise computation. If a model update is represented as a flattened vector `M`, these algorithms sum, average, or select the elements at `M[i]` from all parties — parameters at a given index `i` can be computed with no dependency on the parameters at any other indices.

This distinctive algorithmic structure of model aggregation algorithms allows us to perform aggregation on incomplete and (internally) out-of-order model updates. The only requirement is that all parties should contribute a fraction of each model update with deterministic transformation at each training round. In contrast, FL data leakage attacks, particularly their optimization procedures, require a global view of model updates from parties to progress. The completeness and data order of model updates play a pivotal role in uncovering training data. The absence of either of these factors leads to the failure of such attacks.

This feature enables us to decentralize the aggregation process. Each party can partition an entire model update into multiple fragments, distribute them to multiple aggregation servers, and execute the same model aggregation algorithms independently across all servers. Furthermore, before aggregation, each partitioned vector can also be shuffled at each training round, as long as all parties permute in the same order. Parties can reverse the permutation and merge the aggregated partitions locally when they receive the aggregated model updates for synchronization.

3.2 Confidential Computing

CC technologies share a common objective: protecting sensitive data and computations on untrusted third-party infrastructures. Their primary goal is to shield data-in-use, which refers to data loaded into main memory, while also minimizing the root-of-trust only to the processors. Major CPU vendors are competing to integrate CC capabilities into their processors [31, 34, 35, 39, 45, 48, 62]. Despite variances in implementation and terminology, these technologies adhere to fundamental security principles that align with similar system designs. These principles encompass the introduction of new execution modes or privilege levels, moving workload management functions to vendor-signed firmware or software, ensuring the secure or measured launch of trusted components, enforcing strict memory access controls, and providing memory encryption protection.

In this paper, we choose AMD Secure Encrypted Virtualization (SEV) [39], a virtual machine (VM) based CC solution, to isolate and shield aggregators. However, our implementation is not tied to any specific CC technologies and can be easily adapted with minimal changes to run on other VM-based CC [31, 34, 35, 45, 62]. SEV depends on AMD Secure Memory Encryption (SME) to enable runtime memory encryption. Along with the AMD Virtualization (AMD-V) architecture, SEV can enforce cryptographic isolation between confidential virtual machines (CVMs) and their hosting hypervisor. Therefore, SEV can prevent privileged system administrators, *e.g.,* at the hypervisor level, from accessing the data within the encrypted memory of CVMs.

When SEV is enabled, SEV hardware tags all code and data of a VM with an Address Space Identifier (ASID), which is associated with a distinct ephemeral Advanced Encryption Standard (AES) key, called VM Encryption Key (VEK). The keys are managed by the AMD Secure Processor (SP), which is a 32-bit ARM Cortex-A5 micro-controller integrated within the AMD System-on-Chip (SoC). Runtime memory encryption is performed via on-die memory controllers. Each memory controller has an AES engine that encrypts/decrypts data when it is written to the main memory or is read into the SoC. The control over memory page encryption is via page tables. Physical address bit 47, *i.e.,* `C-bit`, is used to mark whether the memory page is encrypted.

Similar to other CC technologies, SEV also provides a remote attestation mechanism for authenticating hardware platforms and attesting CVMs. The authenticity of the platform is proven with an identity key signed by AMD and the platform owner. Before provisioning any secrets, CVM owners should verify both the authenticity of SEV-enabled hardware and the measurement of Open Virtual Machine

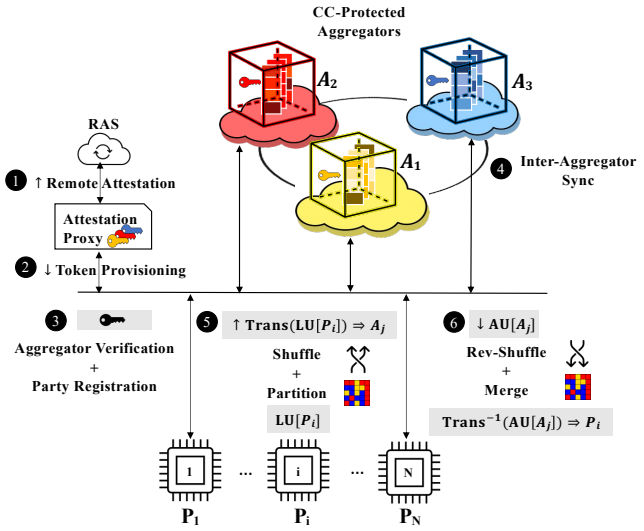


Figure 1. Federated Learning Life Cycle with DeTA

Firmware (OVMF), which enables Unified Extensible Firmware Interface (UEFI) support for booting CVMs.

4 System Design

We adhere to the security principle of *defense-in-depth* to strengthen the resilience against current and potential adaptive attackers. DeTA offers three key security features to achieve this: (1) partitioning a central aggregator into multiple decentralized instances, each with only a fragmentary view of the model updates, (2) dynamically shuffling parameters within each partitioned model update during each training round, and (3) authenticating tamper-resistant and trustworthy aggregators and protecting them via SEV. These security measures work in concert to minimize the attack surface and make our system resistant to data leakage attacks.

Federated Learning Life Cycle. In Figure 1, we present a concrete deployment of DeTA to explain the FL training life cycle step by step. We have N parties participating in the training and we denote the i-th party as P_i. Instead of establishing a central aggregator, we launch multiple SEV-protected aggregators. The j-th aggregator is denoted by A_j. An attestation proxy (AP) is deployed for checking the trustworthiness of aggregators with AMD's remote attestation service (RAS) (❶). After verifying the certificate chain and launching measurements of the aggregators, the AP provisions a secret token (❷) to each aggregator as a proof of trustworthiness. The tokens are injected into the encrypted memory of SEV CVMs. Each party needs to register with all aggregators after verifying the tokens to participate in the training (❸). The parties and the aggregators choose a model aggregation algorithm, *e.g.*, `FedAvg` [49] or `FedSGD` [67], and determine the number of training rounds. The training starts with synchronization among all aggregators (❹). In each training round, each party utilizes its local training data to generate a new local update, denoted as $LU[P_i]$. This update undergoes a series of transformations, including partitioning and shuffling. The resulting transformed update, labeled as $Trans(LU[P_i])$, is then uploaded to the corresponding A_j (❺). Each aggregator independently aggregates the model update fragments received from all parties and dispatches the aggregated update back. This aggregated update is designated as $AU[A_j]$ if it originates from A_j. Subsequently, each party reverses the transformation (referred to as $Trans^{-1}(AU[A_j])$) by shuffling the parameters back to their original order and merging the aggregated updates into its local model (❻). The global training ends once the pre-determined training criteria (like the number of training rounds or accuracy) are met.

4.1 Decentralized Data Aggregation

Decentralization has been a focal point in previous research on distributed learning [3, 41, 42, 76, 83], primarily serving as a load-balancing technique to mitigate the performance bottleneck associated with a central server and to enable scalability for a large number of parties. In these approaches, model updates are directly exchanged among parties in a peer-to-peer manner, eliminating the need for a central aggregator. However, it is important to note that these approaches were not designed to address the issue of data leaks, as a party could still observe the entire model update exchanged by another party. Moreover, these decentralized methods introduced additional complexities in establishing mutual trust among the parties.

Within DeTA, we embrace an alternative decentralization strategy with a strong emphasis on security. Our approach involves separating the central aggregator into multiple functional instances, each of which possesses restricted and obfuscated access to the model updates. Achieving this decentralization relies on the utilization of techniques such as *randomized model partitioning* and *inter-aggregator training synchronization*.

Randomized Model Partitioning. We exploit the coordinate-wise computation of model aggregation algorithms to support randomized model update partitioning. We split each model update into disjoint partitions based on the number of deployed aggregators. Before training starts, we randomly generate a *model mapper* for each to-be-trained DNN model. We allow the parties to choose the proportion of model parameters for each aggregator. The *model mapper* is agreed upon and shared by all the parties that participate in the FL training.

In Figure 2, the k parameters of a local model are mapped to three aggregators. The colors denote which aggregator a parameter is mapped to. The model update is disassembled

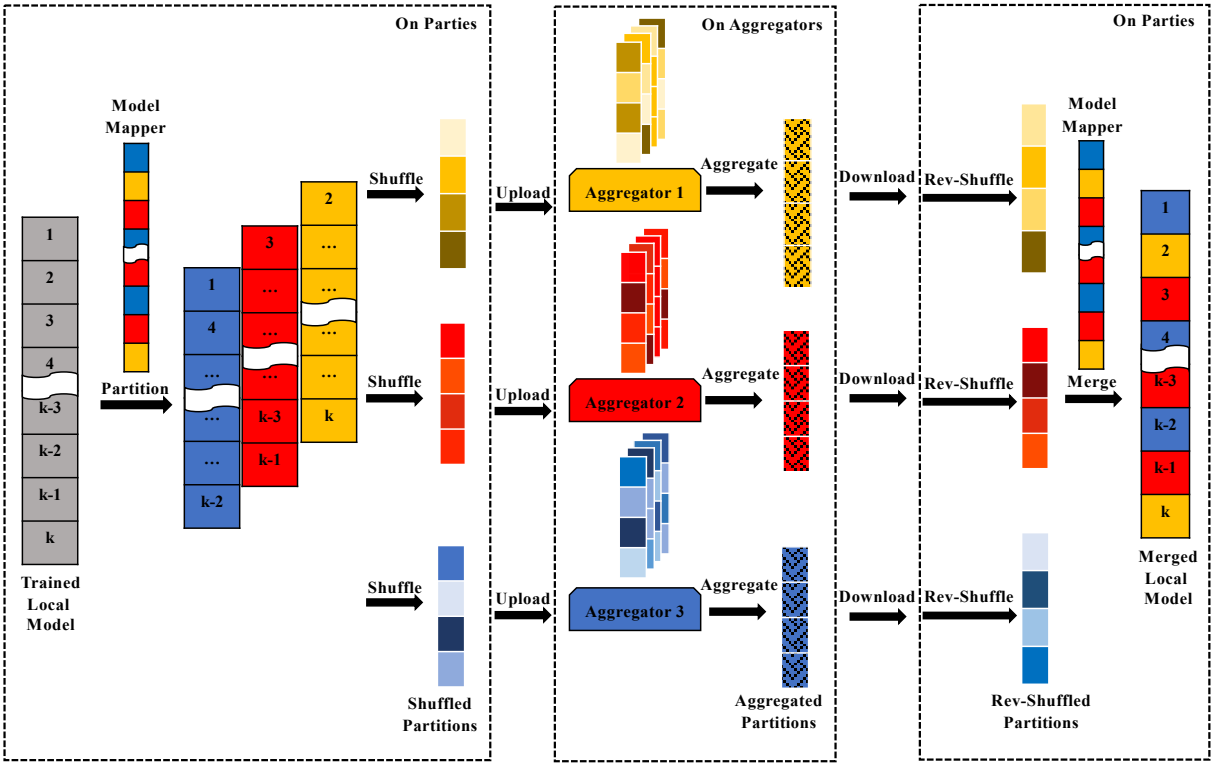


Figure 2. Model Partitioning and Parameter Shuffling

and regrouped at the parameter granularity for different aggregators. Once parties download aggregated model updates from different aggregators, they query the *model mapper* again to merge model updates to their original positions within the local model.

Now each aggregator can only view a fragment of each model update as a flattened vector. Such fragmented model updates no longer keep the model architecture information because unassociated parameters have been removed and remaining parameters have been squeezed to occupy all empty slots in sequence.

Inter-Aggregator Training Synchronization. As we deploy multiple decentralized aggregators in DeTA, we need to maintain communication channels between aggregators for training synchronization. DeTA randomly selects one of the aggregators as the *initiator* node. All the other aggregators become *follower* nodes and wait for the commands from the *initiator*. At each training round, the *initiator* first notifies all parties to start local training and retrieves the model updates for aggregation. Thereafter, it notifies all the *follower* aggregator nodes to pull their corresponding model updates, aggregate them together, and distribute the aggregated updates back to the parties.

4.2 Parameter-level Data Shuffling

To further obfuscate the information transferred from the parties to the aggregators, we employ a dynamic shuffling scheme to permute the parameters of each partitioned model update at every training round. Each permutation is seeded by the *combination* of a permutation key (*e.g.,* dispatched from a trusted key broker service) agreed among all parties and a dynamically generated *training identifier* dispatched at the start of each training round. Thus, the permutation changes across training rounds, but is deterministic for all parties. As shown in Figure 2, after receiving an aggregated model update, each party can reverse shuffle the parameters within the model update back to their original order.

The dynamic shuffling scheme is also based on the insight that the data order of parameters in each model update is irrelevant for model aggregation algorithms, but it is crucial for optimization procedures used in data leakage attacks. With dynamic shuffling enabled, even adversaries who have potentially breached the CC-protected aggregators can only obtain permuted model updates and the data order dynamically changes at each training round.

Let us assume that each data leakage attack takes time T. The attack cost would be of the order $O(2^{|key\_size|} \cdot T)$ with an exhaustive search for the permutation key. It is worth noting that the specific numerical values and their statistical distribution in model weights are irrelevant to the cost of this

order-recovering attack. The cost is determined by the effort required to recover the original order of the parameters, hence the need to exhaust the key space for permutation. The key size is configurable based on the user's security requirement. If the key space is sufficiently large, it becomes computationally infeasible for adversaries to recover the original order of shuffled data, even in the event of compromising all shielded aggregators.

In particular, we want to emphasize that our *parameter-level* shuffling scheme is fundamentally different from the shuffling scheme of the `Encode-Shuffle-Analyze (ESA)` [6] architecture. These two schemes serve different security purposes and operate at different granularity levels. The `ESA` shuffler acts as an intermediary between the encoder and analyzer. It reorders an array of model updates collected from all parties to break data linkage for anonymity. The granularity of `ESA`'s shuffling is at the level of each model update. Thus, an aggregator cannot associate model updates with their owners. However, our data shuffling scheme allows parties to internally permute parameters within each individual (partitioned) model update. Therefore, our shuffling scheme operates at the granularity of a single parameter (*i.e.,* a floating number) within each model update. Our scheme intends to prevent aggregators from obtaining pristine model updates.

Applicable Aggregation Algorithms. DeTA's partitioning and shuffling mechanisms can apply to a wide range of aggregation algorithms, including most popular algorithms like `FedSGD` and `FedAvg`, as well as Byzantine-robust algorithms such as `Krum` [7], `Coordinate Median` [81], and `FLAME` [56], which are specifically designed to mitigate poisoning and backdoor attacks. For example, `FLAME` works by detecting anomalous model updates from malicious parties and clustering model updates to eliminate outliers that indicate data or model poisoning in parties. It is important to note that shuffling (permutation of a vector) does not affect the distance between vectors. With partitioning and shuffling in DeTA, the original N-dimensional vector is partitioned into multiple sub-vectors, and shuffling still preserves the distance used for clustering. The difference is that the original `FLAME` clustering is based on the entire vector, whereas with partitioning enabled, the clustering is conducted independently on partitioned vectors within multiple aggregators. The outliers injected by malicious clients can still be eliminated.

But DeTA's partitioning and shuffling may not be compatible with aggregation algorithms that require global model access. For example, `FLTrust` [11] requires the aggregator to maintain a reference model to detect deviated model updates. In these cases, exposing the complete model to the aggregator may not be desirable if the parties have limited trust in the aggregation server. However, users can still configure DeTA to run a single aggregator in a CVM without turning on partitioning and shuffling, allowing them to strike a balance between security and usability based on their trust levels on different components. Ultimately, the appropriate level of security and usability will vary depending on the specific needs and circumstances of each user.

4.3 Trustworthy Multi-Aggregator Authentication

The existing remote attestation protocol of SEV is tailored for the verification of a single CVM. However, the challenge escalates when aiming to establish trustworthy FL aggregation with multiple decentralized aggregators. In DeTA, we have devised a new two-phase authentication protocol that empowers FL parties to verify all CC-protected aggregators and build secure channels for protecting their upstreamed model updates during transit. The chain of trust across these two phases is securely connected through the use of authentication tokens.

Phase I: Launching Trustworthy Aggregators. In our setup, each aggregator is containerized and deployed within an SEV CVM using the Kata Container [40]. We first pause the CVM launching process and instruct the AMD Secure Processor to generate an attestation report. This report includes the certificate chain and the measurement of the OVMF. The attestation report is then transmitted to an AP, which retrieves the AMD root certificates from AMD's RAS and assumes the responsibility of report verification. It is worth noting that the AP is established and controlled by the participating parties, not by the aggregators themselves. The AP proceeds to verify the certificate chain to authenticate the hardware platform and assess the integrity of the OVMF firmware. Subsequently, the AP generates a launch blob with a packaged secret, which encompasses an ECDSA prime251v1 key. This key acts as an authentication token representing a trusted aggregator and is employed in Phase II for the authentication of aggregators. The hypervisor then injects this secret into the CVM's physical memory space, and the paused launching process is resumed to establish a trustworthy aggregator.

Phase II: Multi-Aggregator Authentication. Parties participating in FL must ensure that they are interacting with trustworthy aggregators with SEV protection. To enable aggregator authentication, in Phase I, the AP provisions an ECDSA key as an authentication token during CVM deployment. This token is used for signing challenge requests and thus serves to identify a trustworthy aggregator. Before participating in FL training, a party first verifies an aggregator by engaging in a challenge-response protocol. The party sends a randomly generated nonce to the aggregator. The aggregator digitally signs the nonce using its corresponding ECDSA key and then returns the signed nonce to the requesting party. The party verifies that the nonce is signed with the corresponding ECDSA key. If the verification is successful,

the party then proceeds to register with the aggregator to participate in FL training. This process is repeated for all aggregators. After registration, end-to-end secure channels can be established to protect communications between aggregators and parties for exchanging model updates. We enable Transport Layer Security (TLS) to protect the communication between a party and an aggregator. Thus, all model updates are protected both within CVMs and in transit.

This two-phase authentication protocol can effectively prevent the collusion between aggregators and parties. First, malicious or tampered aggregators (*e.g.,* with the collusion code) cannot be launched as they will fail the attestation for integrity verification. Second, all parties must undergo authentication and registration to join the training, further minimizing the risk of deploying impersonated parties controlled by aggregators.

5 Implementation

We developed DeTA by extending the IBM Framework for Federated Learning (FFL) [47] to support trustworthy aggregation, decentralized multi-aggregators with model partitioning, and dynamic internal permutation of model updates. We used the AMD EPYC 7642 (ROME) processor running SEV API Version 0.22 [64]. We extended `QEMU` with the patch [18] to support AMD SEV `LAUNCH_SECRET` and extended Kata Runtime to provide remote attestation via client-side `gRPC` [24] communication with the AP server. Finally, we implemented our AP server as a `gRPC` service using a modified version of the AMD SEV-Tool [66] to support CVM owners' tasks, e.g., attesting the AMD SEV-enabled hardware platform, verifying the OVMF launch measurement, and generating the launch blob. We added 2631 LOC to the FFL, 3043 LOC to the AP server, and 1037 LOC to the Kata Runtime.

Our current prototype is constructed using IBM FFL, but DeTA's design can be adopted by any FL framework employing model update aggregation, such as `Flower` [19] and `FedML` [17]. Moreover, our prototype can readily integrate with other CC solutions, such as Intel Trust Domain Extensions (TDX) [12, 35], and extend to leverage NVIDIA Confidential Computing [14] in H100 Tensor Core GPU. The only necessary adjustment is to modify the AP server to accommodate additional CC attestation.

Open Source Status. The shuffling component of DeTA has already been integrated as an aggregation algorithm in the IBM FFL[1]. The decentralization and TEE components are in the process of being released as an independent open-source project.

[1] https://github.com/IBM/federated-learning-lib

6 Security Analysis

In our security analysis, we subject DeTA to a worst-case scenario, wherein we assume that attackers have successfully breached the CC-protected aggregators and gained access to all the model updates upstreamed by the FL parties. Our objective is to assess whether adversaries can reconstruct the training data from these model updates, which have undergone transformations involving model partitioning and parameter shuffling.

We evaluated the effectiveness of DeTA against three FL data leakage attacks that can reconstruct training data based on model updates: `Deep Leakage from Gradients (DLG)` [88], `Improved DLG (iDLG)` [84], and `Inverting Gradients (IG)` [22]. We used the implementations from their public repositories for our experiments. We evaluated each attack in the following two configurations:

In the first configuration, we evaluated each attack with only model partitioning enabled, varying the *partition factor* by 40%, thus lowering the percentage of model updates accessible to the attack. For example, a *partition factor* of 0.6 means the attack has access to 60% of parameters in a model update.

In the second configuration, we enabled parameter shuffling together with model partitioning and re-evaluated the model, again varying the partition factor by 40%. For example, a *partition factor* of 0.6 means the attack has access to 60% of parameters in a model update, in addition, the parameters within this 60% partition are shuffled.

The current design of DeTA does not allow aggregators to maintain a global model, thus adversaries do not know model architectures. Adversaries can neither retrieve the unmodified model update associated with an input sample nor query the model for the dummy input's current loss gradients. As such, in a real deployment of DeTA, these attacks would not succeed as they lack both these two critical components. However, to analyze the effects of the security measures of DeTA, we relaxed the constraints and allowed adversaries to query the complete, unperturbed model as a black box. Therefore, we allowed the attacks to compute the dummy inputs' loss gradients, but the original inputs' loss gradients were still transformed by DeTA. In this stronger attack scenario, we demonstrate that DeTA remains effective and prevents the attacks from leaking information by reconstructing data from model updates.

6.1 Data Reconstruction Attacks

Here we briefly describe the algorithms of these reconstruction attacks. In `DLG` [88], the attack reconstructs a training sample x based on the shared gradient updates. It randomly initializes a dummy input x' and a label y', which are fed into the model to compute the loss gradients $\nabla_\theta \mathcal{L}_\theta(x', y')$ with regard to the model weight θ. Then, the attack uses an L-BGFS solver to minimize the following cost function to

reconstruct x:

$$\underset{x',y'}{\text{argmin}}||\nabla_\theta \mathcal{L}_\theta(x', y') - \nabla_\theta \mathcal{L}_\theta(x, y)||^2$$

As the differentiation requires second-order derivatives, the attack only works on models that are twice differentiable. Although the DLG attack was proven effective, the reconstructions and labels generated after optimization were sometimes of low quality and incorrect respectively. In the following iDLG [84] attack, the authors demonstrated that the signs of the loss gradients with respect to the correct label are always opposite to the signs of the other labels. Thus, the ground truth labels can be inferred based on the model updates to improve reconstruction quality.

IG [22] makes two major modifications to DLG and iDLG. First, the authors asserted that it is not the magnitudes of the gradients that are important, but rather the directions of the gradients. Based on this reasoning, they used a cosine distance cost function, which encouraged the attack to find reconstructions that resulted in the same changes in gradients' directions. Their new cost function is:

$$\underset{x' \in [0,1]^n}{\text{argmin}}\ 1 - \frac{\langle \nabla_\theta \mathcal{L}_\theta(x', y), \nabla_\theta \mathcal{L}_\theta(x, y) \rangle}{||\nabla_\theta \mathcal{L}_\theta(x', y)||||\nabla_\theta \mathcal{L}_\theta(x, y)||} + \alpha TV(x')$$

In this cost function, they (1) constrained their search space to $[0, 1]$, (2) added total variation (TV) as an image prior, and (3) minimized their cost function based on the signs of the loss gradients and the ADAM optimizer. This modification was inspired by adversarial attacks on DNNs, which used a similar technique to generate adversarial inputs [68].

We can see that DLG, iDLG, and IG attacks all require a *complete, in-order* view of either the model or the loss gradients to minimize their respective cost functions. Otherwise, $\nabla_\theta \mathcal{L}_\theta(x, y)$ cannot be accurately measured or aligned with respect to the reconstructed input x'.

6.2 DLG and iDLG Results

We used a randomly initialized LeNet model for evaluation as done in DLG and iDLG and evaluated both attacks using 1000 randomly selected inputs from the *CIFAR-100* dataset. We ran each attack for 300 iterations. The results of DeTA against DLG and iDLG are reported in Tables 1 and 2.

We partitioned the results into four ranges based on the mean squared error (MSE) between the original and the reconstructed images. MSE is the metric adopted in DLG and iDLG for measuring the quality of reconstructed images in *CIFAR-100*. Through visual inspection, an MSE below 1.0×10^{-3} compared to the original images resulted in recognizable reconstructions. We highlight this threshold in red in Tables 1 and 2. Without DeTA in place, DLG and iDLG, resulted in generating 66.6% and 83.7% recognizable reconstructions respectively without model partitioning. These results are used as the baseline and highlighted in the *underlined* columns. For these columns, the *partition factor* is 1 (denoted as "Full"). It means the attack had access to an *entire* model update.

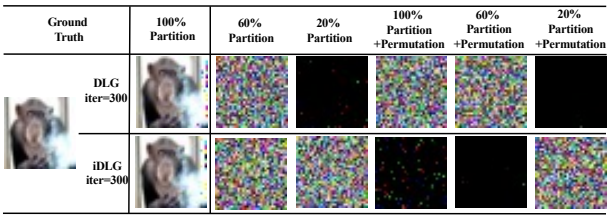


Figure 3. Reconstruction Examples of DLG and iDLG with Model Partitioning and Parameter Shuffling

Table 1. Comparison of Fidelity Threshold (MSE) for DLG with Model Partitioning and Parameter Shuffling

DLG	Partition			Partition+Shuffle		
MSE	**Full***	$0.6^\dagger$	$0.2^\dagger$	Full‡	$0.6^\ddagger$	$0.2^\ddagger$
$[0, 1 \times 10^{-3})$	**66.6%**	0%	0%	0%	0%	0%
$[1 \times 10^{-3}, 1)$	**1.3%**	0%	0%	0%	0%	0%
$[1, 1 \times 10^{2})$	**8.1%**	38.9%	20.5%	0%	0%	0.2%
$\geq 1 \times 10^{2}$	**24.0%**	61.1%	79.5%	100%	100%	99.8%

$\star$ Without DeTA (with access to each full model update), 66.6% of reconstructions generated by DLG are recognizable ($MSE < 1 \times 10^{-3}$).
† After enabling partitioning (with access to 60% and 20% of each model update), all reconstructions are not recognizable ($MSE > 1$).
‡ After combining shuffling with partitioning (with access to 100%, 60%, and 20% of each model update), all reconstructions are not recognizable ($MSE > 1 \times 10^2$).

However, as soon as model partitioning is enabled, the reconstruction quality drops significantly. Due to model partitioning, irrespective of whether the attacked aggregator got a 0.6 or 0.2 partition (60% or 20% of the model update), both attacks' estimates of the original gradients are increasingly inaccurate as fewer of the original model's updates are available. In turn, both attacks cannot correctly minimize the cost function, which we observed during the attack process. With partitioning, neither attack can generate any recognizable reconstructions.

Enabling parameter shuffling in addition to model partitioning adds an additional layer of protection against reconstruction attacks as evidenced by the increased MSE of the reconstructions in Tables 1 and 2 — almost 100% of reconstructed images fall within the highest MSE bucket. Even with all of the model updates, neither attack can generate a recognizable reconstruction as shuffling of the model parameters prevents the attacks from correctly aligning their gradient estimations.

We present the reconstructions generated by DLG and iDLG attacks in the first and second rows of Figure 3. The first column shows the original images. The second column presents the reconstructed images (after 300 iterations) when the entire and in-order model updates (100% partition) were provided to the DLG and iDLG attacks. These results are used

Table 2. Comparison of Fidelity Threshold (MSE) for iDLG with Model Partitioning and Parameter Shuffling

iDLG	Partition			Partition+Shuffle		
MSE	**Full***	0.6†	0.2†	Full‡	0.6‡	0.2‡
$[0, 1\times10^{-3})$	**83.7%**	0%	0%	0%	0%	0%
$[1\times10^{-3}, 1)$	**1.5%**	0%	0%	0%	0%	0%
$[1, 1\times10^{2})$	**6.6%**	66.5%	18.3%	0.2%	0.2%	0.7%
$\geq 1\times10^{2}$	**8.2%**	33.5%	81.7%	99.8%	99.8%	99.3%

⋆ Without DeTA (with access to each full model update), 83.7% of reconstructions generated by iDLG are recognizable ($MSE < 1\times10^{-3}$).
† After enabling partitioning (with access to 60% and 20% of each model update), all reconstructions are not recognizable ($MSE > 1$).
‡ After combining shuffling with partitioning (with access to 100%, 60%, and 20% of each model update), all reconstructions are not recognizable ($MSE > 1\times10^{2}$).

Table 3. Comparison of Final Cosine Distance for IG with Model Partitioning and Parameter Shuffling

IG	Partition			Partition+Shuffle		
Cosine Distance	**Full***	0.6†	0.2†	Full‡	0.6‡	0.2‡
[0, 0.01)	**100%**	0%	0%	0%	0%	0%
[0.01, 0.2)	**0%**	0%	0%	0%	0%	0%
[0.2, 0.4)	**0%**	100%	0%	0%	0%	0%
[0.4, 0.6)	**0%**	0%	98%	0%	0%	0%
[0.6, 0.8)	**0%**	0%	2%	0%	0%	0%
[0.8, 1]	**0%**	0%	0%	100%	100%	100%

⋆ Without DeTA (with access to each full model update), all the cosine distances of IG reconstructions fall into the range of $[0, 0.01)$, indicating that IG's optimization can converge.
† After enabling partitioning (with access to 60% and 20% of each model update), all the cosine distances > 0.2, indicating that IG's optimization cannot converge.
‡ After combining shuffling with partitioning (with access to 100%, 60%, and 20% of each model update), all the cosine distances > 0.8, indicating that IG's optimization cannot converge.

as the baseline for comparison. In columns 3-7, we present the reconstructions with different combinations of partitioning and shuffling enabled. The visual results are consistent with the MSE data in Tables 1 and 2. No recognizable reconstructions can be generated with partitioning and shuffling enabled.

6.3 IG Results

We used a randomly initialized ResNet-18 model for evaluation as done in IG and evaluated using 50 randomly selected inputs from the ImageNet dataset. We ran the attack for 24,000 iterations with two random restarts.

MSE is no longer an accurate metric for measuring image similarity for large-sized data examples of ImageNet. Instead, we measured the cosine distance, *i.e.,* the cost function of IG, to show that DeTA effectively hinders the optimization procedure. We partitioned the cosine distance (bounded in $[0, 1]$) into six ranges and present the results in Table 3. Without DeTA in place, the cosine distance of IG's cost function is always smaller than 0.01 with a *partition factor* of 1 (*i.e.,* no model partitioning and denoted as "Full"). These results are used as the baseline and highlighted in the column with the underlined values. With DeTA enabled, IG can no longer correctly minimize the cost function. The cosine distance values in the optimization procedures are stuck at a level significantly larger than 0.01. For example, with a 0.6 *partition factor* and no shuffling, all cosine distance values are in the range of $[0.2, 0.4)$. With shuffling enabled, the cosine distance further increased to the range of $[0.8, 1]$.

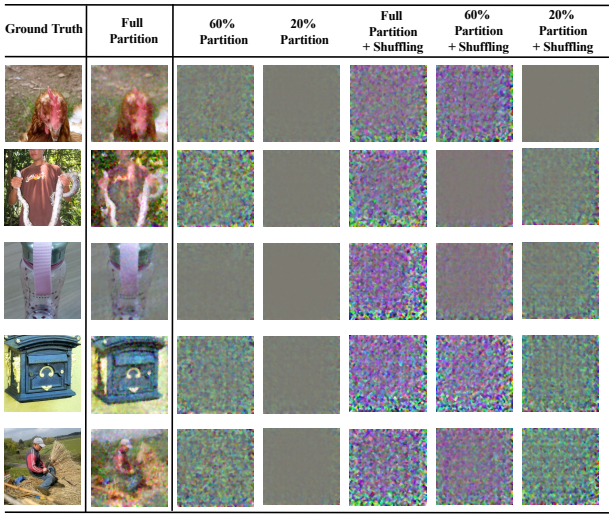


Figure 4. Reconstruction Examples of IG with Model Partitioning and Parameter Shuffling

In Figure 4, we display five ImageNet examples used in our IG reconstruction experiments. The first column shows the original images. The second column presents the reconstructed images (after 24,000 iterations) when the entire model updates were provided to the IG attack. These results are used as the baseline for comparison. In the third through seventh columns, we present the reconstructions with different combinations of partitioning and shuffling enabled. Compared to the baseline, none of the reconstructions contain recognizable information related to the ground truth examples. Without the complete, unperturbed model updates, IG is unable to minimize its cost function.

7 Performance Evaluation

We evaluated the performance of DeTA with two metrics. First, we measured the loss/accuracy of the models generated at each training round. It demonstrates that the convergence rate of DeTA is aligned with the baseline system and DeTA does not lead to model accuracy degradation. Second, we measured the latency of FL training. The latency of model training refers to the total time to finish a specified number

of training rounds. We recorded the time after finishing each training round at the aggregator. The latency results reflect the performance overhead incurred by the security features (*i.e.,* decentralized aggregation, parameter shuffling, and CC protection) added in DeTA. Our evaluation covers a spectrum of *cross-silo* FL training applications from three aspects: (1) adaptability to different FL model aggregation algorithms, (2) performance comparisons with different numbers of participating parties, and (3) support for larger DNN models with non-IID training data distribution.

In our evaluation, we set up three SEV-protected aggregators. Each aggregator ran on a machine with AMD EPYC 7642 CPU. Each party ran on a machine with an Intel Xeon E5-2690 CPU, one NVIDIA Tesla P100 GPU, and 120 GB DRAM. The baseline for comparison is IBM FFL with one central aggregator. The party's configurations, model architectures, and hyper-parameter settings are the same for DeTA and FFL.

7.1 Performance with Different Model Aggregation Algorithms

As indicated in the DeTA's design, we support model aggregation algorithms with coordinate-wise arithmetic operations. We evaluated DeTA respectively with three model aggregation algorithms, *i.e.,* `Iterative Averaging`, `Coordinate Median` [81], and `Paillier` [46, 71], with the `MNIST` dataset. `Iterative Averaging` is the core algorithm supporting `FedAvg` and `FedSGD`. It sends queries to all registered parties at each training round to collect information, *e.g.,* model updates or gradients, averages the updates, and broadcasts the fused results to all parties. `Coordinate Median` is a model aggregation algorithm that selects a coordinate-wise median from collected responses to tolerate Byzantine failures of adversarial parties. The `Paillier-based Fusion` algorithm supports aggregation with Additively Homomorphic Encryption [58]. We trained deep learning models on the `MNIST` dataset with ten training rounds for `Iterative Averaging`, `Coordinate Median`, and three rounds for `Paillier`. Each round has three local epochs.

`MNIST` contains 60,000 examples in the training set. We randomly partitioned the training set into four equal sets for four parties. Each party has 15,000 examples for local training. The trained model is a convolutional neural network (ConvNet) with eight layers.

Accuracy/Loss and Convergence Rate. We present the model loss and accuracy at each training round in Figures 5a, 5b, and 5c. The horizontal axes are the number of training rounds. The left vertical axes show the loss and the right vertical axes present the model accuracy. The loss/accuracy results of DeTA and FFL have the same patterns for all three model aggregation algorithms. DeTA and FFL converge at the same rate on `MNIST` after one training round. The final models achieve the same accuracy level (above 98%) for both DeTA and FFL.

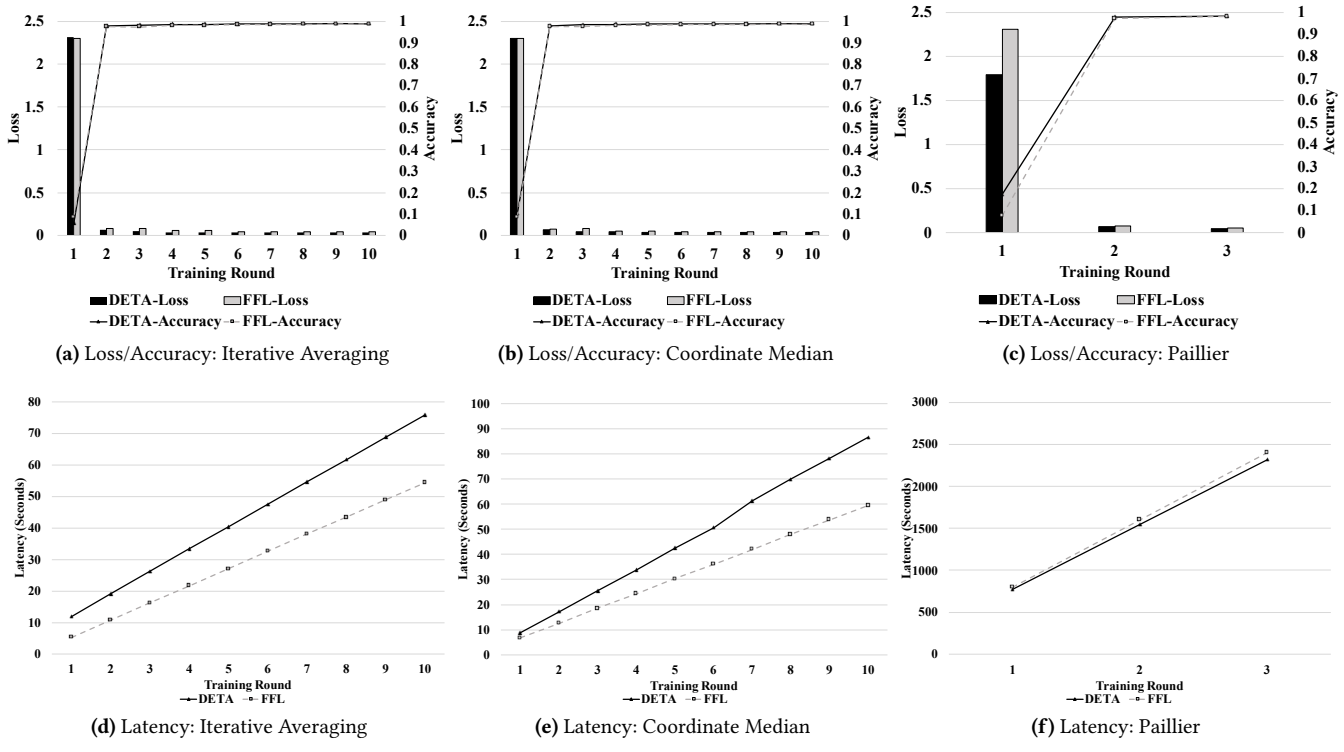
Training Latency. We present the training latency data of DeTA and FFL in Figures 5d, 5e, and 5f. The vertical axes are the accumulated time spent to finish that training round. We observed that for `Iterative Averaging`, DeTA used 75.83 seconds to finish the 10-round training and FFL used 54.32 seconds. Compared to the baseline FFL system, the added security features in DeTA incurred additional 0.40× latency for training the `MNIST` model. Similarly for `Coordinate Median`, DeTA incurred additional 0.45× in latency. Due to the heavyweight additively homomorphic encryption operations, training a model on `MNIST` with `Paillier-based Fusion` itself (without DeTA) for only three rounds took 2000+ seconds (~100× slower compared to `Iterative Averaging` and `Coordinate Median`). However, DeTA finished training with 0.04× improvement in latency compared to FFL. The reason is that the dominant performance factors of `Paillier` aggregation are the encryption/decryption operations. However, as the models are partitioned in DeTA for multiple decentralized aggregators, the `Paillier` encryption/decryption and aggregation are accelerated — computed in parallel by operating on smaller model partitions both on the aggregators and on the parties.

7.2 Performance with Different Numbers of Parties

In this experiment, we aimed to understand the performance effects of involving different numbers of parties. We trained a ConvNet with 23 layers on `CIFAR-10` with four and eight parties. We randomly partition the training set into equal sets for the parties. Each party has 10,000 examples for FL training. We trained this model with 30 training rounds, with each round consisting of one local epoch.

Accuracy/Loss and Convergence Rate. We present the model accuracy and loss at each training round in Figure 6a. The patterns for the loss and accuracy are similar for DeTA and FFL with both four parties and eight parties. It indicates that the models converge at a similar rate with different numbers of parties. The accuracy results of the final model trained with FFL are 76.99% (four parties) and 76.43% (eight parties). The final model accuracy results with DeTA are 79.93% (four parties) and 81.41% (eight parties).

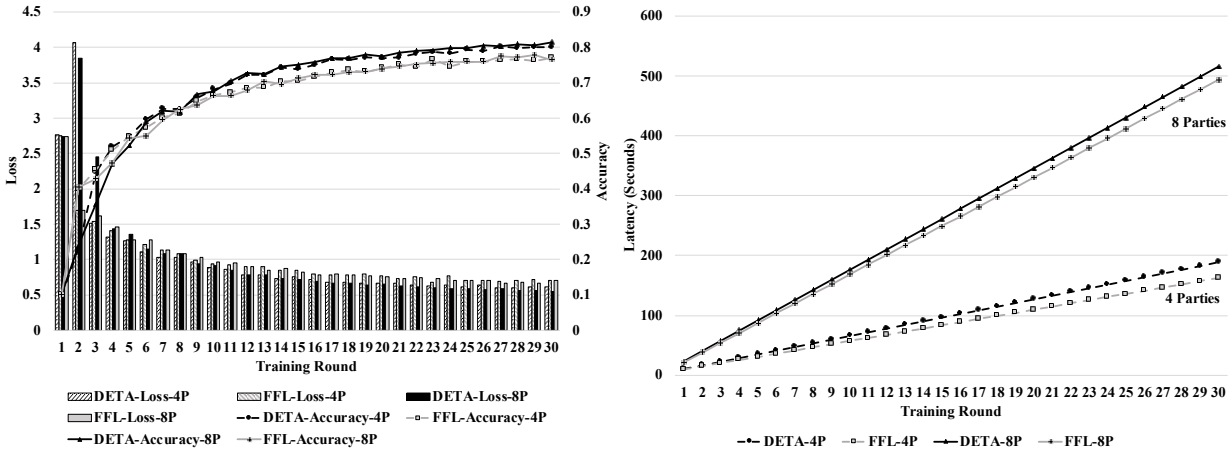
Training Latency. We present the training latency data of DeTA and FFL in Figure 6b. In the four parties scenario, it took DeTA 182.91 seconds to finish 30 rounds of training and FFL 157.41 seconds. Our added features incurred additional 0.16× in latency for training the `CIFAR-10` model. In the eight parties scenario, it took DeTA 498.68 seconds to finish 30 rounds of training and FFL 477.34 seconds. The latency only increased by 0.04×. We also find that adding more parties increases the latency for both FFL and DeTA at the same



(a) Loss/Accuracy: Iterative Averaging **(b)** Loss/Accuracy: Coordinate Median **(c)** Loss/Accuracy: Paillier

(d) Latency: Iterative Averaging **(e)** Latency: Coordinate Median **(f)** Latency: Paillier

Figure 5. MNIST Loss/Accuracy/Latency Comparison Between DeTA and FFL (IID with Four Parties)



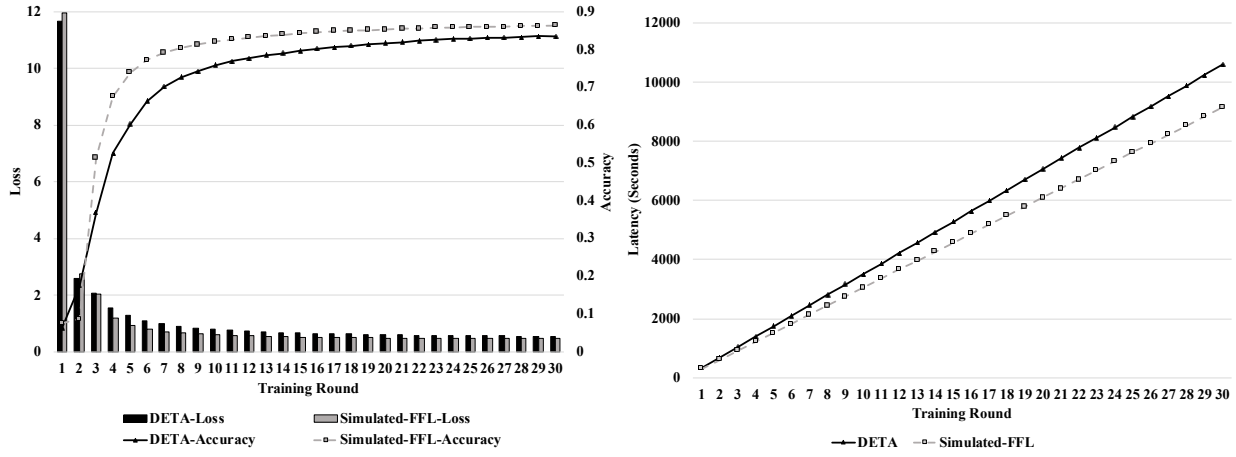
(a) Loss/Accuracy (4 Parties vs. 8 Parties) **(b)** Latency (4 Parties vs. 8 Parties)

Figure 6. CIFAR10 Loss/Accuracy/Latency Comparison Between DeTA and FFL (IID with Four/Eight Parties)

pace. The security features of DeTA do not lead to additional latency with regard to more parties.

7.3 Training with Non-IID Training Data

In this experiment, we measured the performance of training a larger, more complex deep learning model with non-IID training data distribution. We used a pre-trained `VGG-16` model on the `ImageNet` to train a document classifier on the `RVL-CDIP` [30] dataset with 16 classes. For transfer learning with `RVL-CDIP` classification, we replaced the last three fully connected layers of `VGG-16` due to differences in the number of prediction classes. The `RVL-CDIP` dataset has 320,000 training images and 40,000 test images. We partitioned the training data based on the non-IID 90-10 skew data split for eight parties. Each party has approximately 40,000 training examples with skew distribution among different classes,



(a) Loss/Accuracy Comparison

(b) Latency Comparison

Figure 7. RVL-CDIP Loss/Accuracy/Latency Comparison Between DeTA and FFL (non-IID with Eight Parties)

i.e., the two dominant classes contain 90% of training data, while the remaining 14 classes have 10%. We trained deep learning models with 30 training rounds. Each round has one epoch. As `RVL-CDIP` dataset is not officially supported in FFL, we simulated the FFL implementation for performance comparison.

Accuracy/Loss and Convergence Rate. We present the model accuracy and loss at each training round in Figure 7a. Similarly, the models converge at a similar rate with DeTA and FFL. The accuracy result of the final model trained with DeTA is 83.50% and 86.19% with simulated FFL.

Training Latency. We present the training latency data of DeTA and simulated FFL in Figure 7b. It took DeTA 2.85 hours and FFL 2.46 hours to finish 30 rounds of training. Our added security features in DeTA only incurred additional 0.16× in latency for training the `RVL-CDIP` model.

8 Related Work

We give an overview of the security defenses against FL data leakage attacks and analyze their pros and cons from the perspectives of security and utility trade-offs. We compare DeTA with them to demonstrate our contributions.

8.1 Differential Privacy

In the ML setting, DP can be used to apply perturbations for mitigating information leakage. Compared to cryptographic schemes, DP is more computationally efficient at the cost of a certain utility loss due to the added noise. Centralized differential privacy (CDP) is typically conducted under the assumption of a trusted central server. CDP [23, 50] can achieve an acceptable balance between privacy and accuracy, but it does not fit a threat model where the aggregation server might be honest-but-curious, malicious, or compromised. To remove the trusted central server assumption in the threat model, local differential privacy (LDP) [5, 67] lets each client conduct differentially private transformations of their private data before sharing them with the aggregator. However, achieving LDP comes at the cost of utility loss as every participant must add enough noise to ensure DP in isolation. All forms of DP require fresh hyper-parameter tuning, *i.e.,* batch size and learning rate. With DP, hyper-parameters are different for various values of the privacy parameter ϵ. This requires several FL jobs to be run just to find good hyper-parameters, which can be problematic as one is trying to coordinate different competing entities.

Due to conflicting threat models, the decentralized data aggregation model in DeTA is incompatible with CDP, which requires a central aggregator in FL training. However, we can still protect the aggregation within a CC environment to strengthen CDP. DeTA can be seamlessly integrated with LDP as the LDP's perturbations only apply to model updates on the parties' devices.

8.2 Cryptographic Schemes

Homomorphic encryption allows arithmetic operations on ciphertexts without decryption. Aono *et al.* [1] used additively HE to protect the privacy of gradients to prevent information leakage. Hardy *et al.* [29] encrypted vertically partitioned data with an additively HE and learned a linear classifier in the FL setting. One downside of HE-based schemes is that since all parties have to encrypt the model with the same public key for aggregation over encrypted models to work, one needs a trusted third party to generate that key pair. Further, aggregation over HE-encrypted models has high computational and communication costs, since homomorphically encrypting a model increases its size around 32 times [27].

Secure multi-party computation allows different parties to compute a joint function without revealing their inputs to each other and has been widely researched in collaborative analytics and multi-party learning [9, 9, 53, 59, 71, 86, 87]. SecureML [53] conducted privacy-preserving learning via SMC. It required data owners to encrypt and secretly share their data among two non-colluding servers in the initial setup. Helen [87] is a coopetitive system using maliciously SMC for training linear models with a strong adversarial setting where each party could only trust itself. Bonawitz *et al.* [9] proposed a secure aggregation protocol for aggregating individual model updates via SMC. The servers can only learn the information from the aggregated results. It is also robust when clients frequently drop in the FL's cross-device setting. However, SMC is also computationally expensive and has scalability issues. Bonawitz *et al.* [9] illustrated this theoretically with several computation and communication costs that grow quadratically with the number of parties and empirically in their experiments.

Our work in DeTA primarily uses model partition and parameter permutation — inexpensive when compared to SMC protocols. Further, many SMC protocols do not permit asynchronous training because they require cohort formation and active transmission over multiple rounds. Asynchronous training is prevalent in FL because participants often have other competing private workloads, variations in data availability, and differences in compute hardware available for training.

8.3 Confidential Computing

Confidential computing techniques offer users the ability to delegate their computations to remote third-party servers that have root-of-trust established at the CPU package level. These techniques hold significant appeal for collaborative machine learning scenarios, which often involve vast quantities of privacy-sensitive training data, multiple parties with varying levels of trust, and stringent data protection regulations. CC serves as a dependable execution environment that can isolate and manage machine learning processes while also reducing the reliance on costly cryptographic primitives. For example, CC has been employed to support secure model inference [25, 70], privacy-preserving multi-party machine learning [26, 32, 33, 52, 57], and analytics on sensitive data [6, 13, 63, 85]. More Recently, VM-based CC solutions, *e.g.,* AMD SEV [38, 39, 65], Intel TDX [12, 35], IBM Secure Execution [34], Power PEF [31], Arm CCA [45], RISC-V CoVE [62], have been developed to facilitate the deployment of unmodified applications within protected confidential virtual machines.

While CC provides valuable security measures for collaborative machine learning, it is not a one-size-fits-all solution to address all trust issues. Different CC technologies come with their own functional and security constraints. Moreover, CC systems may still be vulnerable to emerging security threats [10, 43, 44, 55, 72–75, 77, 78]. A single flaw could potentially compromise the entire foundation of a supposedly trustworthy system. Therefore, when evaluating a CC-protected system, it is imperative to discern the distinct security properties of various CC implementations and also consider the worst-case scenario in the event of CC failures.

Much like other research endeavors, our work also leverages CC technologies to protect aggregation as a first line of defense against data leakage. However, by integrating CC protection with techniques like model partitioning and parameter permutation, we can ensure that even in the event of a breach in CC environments, attackers remain unable to exploit leaked model updates for malicious purposes.

9 Conclusion

Confidential computing has emerged as a practical and efficient approach for establishing trustworthiness in federated learning aggregation. However, the adoption of CC has also brought to light a growing number of security vulnerabilities. Relying solely on this single layer of defense is no longer sufficient to ensure system resilience against data leaks. In response to these emerging challenges, we have reexamined the security model for FL aggregation. This reassessment has led to the development of DeTA, a new cross-silo FL architecture that incorporates decentralized aggregation, parameter shuffling, and robust multi-aggregator authentication. The primary objectives of DeTA are to break data concentration and minimize data leakage surface within FL aggregation. DeTA enables automated model partitioning and parameter shuffling for FL participants and facilitates the deployment of CC-protected aggregators in a decentralized environment with no utility loss and low-performance overheads. Most importantly, DeTA has proven to be highly effective in mitigating FL data leaks, even when confronted with the worst-case scenario involving breaches of CC execution environments.

References

[1] 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.

[2] Eugene Bagdasaryan, Andreas Veit, Yiqing Hua, Deborah Estrin, and Vitaly Shmatikov. 2020. How to backdoor federated learning. In *International conference on artificial intelligence and statistics*. PMLR, 2938–2948.

[3] Aurélien Bellet, Rachid Guerraoui, Mahsa Taziki, and Marc Tommasi. 2018. Personalized and private peer-to-peer machine learning. In *International Conference on Artificial Intelligence and Statistics*. PMLR, 473–481.

[4] Arjun Nitin Bhagoji, Supriyo Chakraborty, Prateek Mittal, and Seraphin Calo. 2019. Analyzing federated learning through an adversarial lens. In *International Conference on Machine Learning*. PMLR, 634–643.

[5] 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).

[6] Andrea Bittau, Úlfar Erlingsson, Petros Maniatis, Ilya Mironov, Ananth Raghunathan, David Lie, Mitch Rudominer, Ushasree Kode, Julien Tinnes, and Bernhard Seefeld. 2017. Prochlo: Strong privacy for analytics in the crowd. In *Proceedings of the 26th Symposium on Operating Systems Principles*. ACM, 441–459.

[7] Peva Blanchard, El Mahdi El Mhamdi, Rachid Guerraoui, and Julien Stainer. 2017. Machine learning with adversaries: Byzantine tolerant gradient descent. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*. 118–128.

[8] Franziska Boenisch, Adam Dziedzic, Roei Schuster, Ali Shahin Shamsabadi, Ilia Shumailov, and Nicolas Papernot. 2023. When the curious abandon honesty: Federated learning is not private. In *2023 IEEE 8th European Symposium on Security and Privacy (EuroS&P)*. IEEE, 175–199.

[9] 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*. ACM, 1175–1191.

[10] Robert Buhren, Christian Werling, and Jean-Pierre Seifert. 2019. Insecure Until Proven Updated: Analyzing AMD SEV's Remote Attestation. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1087–1099.

[11] Xiaoyu Cao, Minghong Fang, Jia Liu, and Neil Gong. 2021. FLTrust: Byzantine-robust Federated Learning via Trust Bootstrapping. In *Proceedings of NDSS*.

[12] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2023. Intel TDX Demystified: A Top-Down Approach. *arXiv preprint arXiv:2303.15540* (2023).

[13] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2020. Oblivious coopetitive analytics using hardware enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems*. ACM, 1–17.

[14] Gobikrishna Dhanuskodi, Sudeshna Guha, Vidhya Krishnan, Aruna Manjunatha, Rob Nertney, Michael O'Connor, and Phil Rogers. 2023. Creating the First Confidential GPUs. *Commun. ACM* 67, 1 (2023), 60–67.

[15] Apple Differential Privacy Team. 2017. Learning with Privacy at Scale. (2017). https://docs-assets.developer.apple.com/ml-research/papers/learning-with-privacy-at-scale.pdf

[16] Josep Domingo-Ferrer, David Sánchez, and Alberto Blanco-Justicia. 2021. The limits of differential privacy (and its misuse in data release and machine learning). *Commun. ACM* 64, 7 (2021), 33–35.

[17] fedml 2024. Federated learning for Cross-Silo. https://www.fedml.ai/federate/octopus/index.

[18] Tobin Feldman-Fitzthum. 2020. sev: add sev-inject-launch-secret. https://lists.gnu.org/archive/html/qemu-devel/2020-10/msg04361.html.

[19] flower 2024. Flower A Friendly Federated Learning Framework. https://flower.ai/.

[20] Liam Fowl, Jonas Geiping, Wojtek Czaja, Micah Goldblum, and Tom Goldstein. 2021. Robbing the fed: Directly obtaining private data in federated learning with modified models. *arXiv preprint arXiv:2110.13057* (2021).

[21] Clement Fung, Chris JM Yoon, and Ivan Beschastnikh. 2018. Mitigating sybils in federated learning poisoning. *arXiv preprint arXiv:1808.04866* (2018).

[22] Jonas Geiping, Hartmut Bauermeister, Hannah Dröge, and Michael Moeller. 2020. Inverting gradients-how easy is it to break privacy in federated learning? *Advances in Neural Information Processing Systems* 33 (2020), 16937–16947.

[23] Robin C Geyer, Tassilo Klein, and Moin Nabi. 2017. Differentially private federated learning: A client level perspective. *arXiv preprint arXiv:1712.07557* (2017).

[24] gRPC 2021. A high performance, open source universal RPC framework. https://grpc.io/.

[25] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Hani Jamjoom, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. Confidential Inference via Ternary Model Partitioning. *arXiv preprint arXiv:1807.00969* (2018).

[26] Zhongshu Gu, Hani Jamjoom, Dong Su, Heqing Huang, Jialong Zhang, Tengfei Ma, Dimitrios Pendarakis, and Ian Molloy. 2019. Reaching data confidentiality and model accountability on the caltrain. In *49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks*. IEEE, 336–348.

[27] Saransh Gupta, Rosario Cammarota, and Tajana Šimunić Rosing. 2022. Memfhe: End-to-end computing with fully homomorphic encryption in memory. *ACM Transactions on Embedded Computing Systems* (2022).

[28] Andrew Hard, Kanishka Rao, Rajiv Mathews, Swaroop Ramaswamy, Françoise Beaufays, Sean Augenstein, Hubert Eichner, Chloé Kiddon, and Daniel Ramage. 2018. Federated learning for mobile keyboard prediction. *arXiv preprint arXiv:1811.03604* (2018).

[29] Stephen Hardy, Wilko Henecka, Hamish Ivey-Law, Richard Nock, Giorgio Patrini, Guillaume Smith, and Brian Thorne. 2017. Private federated learning on vertically partitioned data via entity resolution and additively homomorphic encryption. *arXiv preprint arXiv:1711.10677* (2017).

[30] Adam W Harley, Alex Ufkes, and Konstantinos G Derpanis. 2015. Evaluation of Deep Convolutional Nets for Document Image Classification and Retrieval. In *International Conference on Document Analysis and Recognition*. IEEE, 991–995.

[31] Guerney D. H. Hunt, Ramachandra Pai, Michael Le, Hani Jamjoom, Sukadev Bhattiprolu, Rick Boivie, Laurent Dufour, Brad Frey, Mohit Kapur, Kenneth A. Goldman, Ryan Grimm, Janani Janakirman, John M. Ludden, Paul Mackerras, Cathy May, Elaine R. Palmer, Bharata Bhasker Rao, Lance Roy, William A. Starke, Jeff Stuecheli, Ray Valdez, and Wendel Voigt. 2021. Confidential Computing for OpenPOWER. In *Proceedings of the Sixteenth European Conference on Computer Systems*. ACM, 294–310.

[32] 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).

[33] Nick Hynes, Raymond Cheng, and Dawn Song. 2018. Efficient Deep Learning on Multi-Source Private Data. *arXiv preprint arXiv:1807.06689* (2018).

[34] IBM. 2022. Introducing IBM Secure Execution for Linux 1.3.0. https://www.ibm.com/docs/en/linuxonibm/pdf/l130se03.pdf. (2022).

[35] Intel. 2021. Intel® Trust Domain Extensions. https://cdrdv2.intel.com/v1/dl/getContent/690419. (2021).

[36] K. R. Jayaram, Archit Verma, Ashish Verma, Gegi Thomas, and Colin Sutcher-Shepard. 2020. MYSTIKO: Cloud-Mediated, Private, Federated Gradient Descent. In *2020 IEEE 13th International Conference on Cloud Computing (CLOUD)*. IEEE, 201–210.

[37] Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, K. A. Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, Rafael G.L. D'Oliveira, Salim El Rouayheb, David Evans, Josh Gardner, Zachary Garrett, Adrià Gascón, Badih Ghazi, Phillip B. Gibbons, Marco Gruteser, Zaid Harchaoui, Chaoyang He, Lie He, Zhouyuan Huo, Ben Hutchinson, Justin Hsu, Martin Jaggi, Tara Javidi, Gauri Joshi, Mikhail Khodak, Jakub Konečný, Aleksandra Korolova, Farinaz Koushanfar, Sanmi Koyejo, Tancrède Lepoint, Yang Liu, Prateek Mittal, Mehryar Mohri, Richard Nock, Ayfer Özgür, Rasmus Pagh, Mariana Raykova, Hang Qi, Daniel Ramage, Ramesh Raskar, Dawn Song, Weikang Song, Sebastian U. Stich, Ziteng Sun, Ananda Theertha Suresh, Florian Tramèr, Praneeth Vepakomma, Jianyu Wang, Li Xiong, Zheng Xu, Qiang Yang, Felix X. Yu, Han Yu, and Sen Zhao. 2019. Advances and Open Problems in Federated Learning. https://arxiv.org/abs/1912.04977

[38] David Kaplan. 2017. Protecting vm register state with sev-es. *White paper* (2017).

[39] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016).

[40] Kata Containers 2021. The speed of containers, the security of VMs. https://katacontainers.io.

[41] Anastasia Koloskova, Sebastian Stich, and Martin Jaggi. 2019. Decentralized stochastic optimization and gossip algorithms with compressed communication. In *International Conference on Machine Learning*. PMLR, 3478–3487.

[42] Anusha Lalitha, Osman Cihan Kilinc, Tara Javidi, and Farinaz Koushanfar. 2019. Peer-to-peer federated learning on graphs. *arXiv preprint arXiv:1901.11173* (2019).

[43] Mengyuan Li, Yinqian Zhang, and Zhiqiang Lin. 2020. CROSSLINE: Breaking "Security-by-Crash" based Memory Isolation in AMD SEV. *arXiv preprint arXiv:2008.00146* (2020).

[44] Mengyuan Li, Yinqian Zhang, Zhiqiang Lin, and Yan Solihin. 2019. Exploiting unprotected I/O operations in AMD's Secure Encrypted Virtualization. In *28th USENIX Security Symposium*. USENIX, 1257–1272.

[45] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. 2022. Design and Verification of the Arm Confidential Compute Architecture. In *16th USENIX Symposium on Operating Systems Design and Implementation (OSDI 22)*. 465–484.

[46] Changchang Liu, Supriyo Chakraborty, and Dinesh Verma. 2019. Secure model fusion for distributed learning using partial homomorphic encryption. In *Policy-Based Autonomic Data Governance*. Springer, 154–179.

[47] Heiko Ludwig, Nathalie Baracaldo, Gegi Thomas, Yi Zhou, Ali Anwar, Shashank Rajamoni, Yuya Ong, Jayaram Radhakrishnan, Ashish Verma, Mathieu Sinn, Mark Purcell, Ambrish Rawat, Tran Minh, Naoise Holohan, Supriyo Chakraborty, Shalisha Whitherspoon, Dean Steuer, Laura Wynter, Hifaz Hassan, Sean Laguna, Mikhail Yurochkin, Mayank Agarwal, Ebube Chuba, and Annie Abay. 2020. IBM Federated Learning: an Enterprise Framework White Paper V0.1. *arXiv preprint arXiv:2007.10987* (2020).

[48] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution. *The Second Workshop on Hardware and Architectural Support for Security and Privacy* 10, 1 (2013).

[49] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. 2017. Communication-efficient learning of deep networks from decentralized data. In *Artificial Intelligence and Statistics*. PMLR, 1273–1282.

[50] H Brendan McMahan, Daniel Ramage, Kunal Talwar, and Li Zhang. 2017. Learning differentially private recurrent language models. *arXiv preprint arXiv:1710.06963* (2017).

[51] Luca Melis, Congzheng Song, Emiliano De Cristofaro, and Vitaly Shmatikov. 2019. Exploiting unintended feature leakage in collaborative learning. In *2019 IEEE Symposium on Security and Privacy*. IEEE, 691–706.

[52] 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*. 94–108.

[53] Payman Mohassel and Yupeng Zhang. 2017. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy*. IEEE, 19–38.

[54] Arup Mondal, Yash More, Ruthu Hulikal Rooparaghunath, and Debayan Gupta. 2021. Poster: Flatee: Federated learning across trusted execution environments. In *2021 IEEE European Symposium on Security and Privacy (EuroS&P)*. IEEE, 707–709.

[55] 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*. IEEE, 1466–1482.

[56] Thien Duc Nguyen, Phillip Rieger, Roberta De Viti, Huili Chen, Björn B Brandenburg, Hossein Yalame, Helen Möllering, Hossein Fereidooni, Samuel Marchal, Markus Miettinen, et al. 2022. {FLAME}: Taming backdoors in federated learning. In *31st USENIX Security Symposium (USENIX Security 22)*. 1415–1432.

[57] 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, 619–636.

[58] Pascal Paillier. 1999. Public-key cryptosystems based on composite degree residuosity classes. In *International conference on the theory and applications of cryptographic techniques*. Springer, 223–238.

[59] Rishabh Poddar, Sukrit Kalra, Avishay Yanai, Ryan Deng, Raluca Ada Popa, and Joseph M Hellerstein. 2021. Senate: A Maliciously-Secure {MPC} Platform for Collaborative Analytics. In *30th USENIX Security Symposium*. USENIX.

[60] Do Le Quoc and Christof Fetzer. 2021. Secfl: Confidential federated learning using tees. *arXiv preprint arXiv:2110.00981* (2021).

[61] Phillip Rieger, Thien Duc Nguyen, Markus Miettinen, and Ahmad-Reza Sadeghi. 2022. Deepsight: Mitigating backdoor attacks in federated learning through deep model inspection. *arXiv preprint arXiv:2201.00763* (2022).

[62] Ravi Sahita, Vedvyas Shanbhogue, Andrew Bresticker, Atul Khare, Atish Patra, Samuel Ortiz, Dylan Reid, and Rajnesh Kanwal. 2023. CoVE: Towards Confidential Computing on RISC-V Platforms. In *Proceedings of the 20th ACM International Conference on Computing Frontiers*. 315–321.

[63] Felix Schuster, Manuel Costa, Cédric Fournet, Christos Gkantsidis, Marcus Peinado, Gloria Mainar-Ruiz, and Mark Russinovich. 2015. VC3: Trustworthy data analytics in the cloud using SGX. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 38–54.

[64] SEV API 2019. Secure Encrypted Virtualization API Version 0.22. https://developer.amd.com/wp-content/resources/55766.PDF.

[65] AMD SEV-SNP. 2020. Strengthening VM isolation with integrity protection and more. *White Paper* (2020).

[66] SEV-Tool 2019. SEV-Tool. https://github.com/AMDESE/sev-tool.

[67] Reza Shokri and Vitaly Shmatikov. 2015. Privacy-preserving deep learning. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. ACM, 1310–1321.

[68] Christian Szegedy, Wojciech Zaremba, Ilya Sutskever, Joan Bruna, Dumitru Erhan, Ian Goodfellow, and Rob Fergus. 2013. Intriguing properties of neural networks. *arXiv preprint arXiv:1312.6199* (2013).

[69] Vale Tolpegin, Stacey Truex, Mehmet Emre Gursoy, and Ling Liu. 2020. Data poisoning attacks against federated learning systems. In *Computer Security–ESORICS 2020: 25th European Symposium on Research in Computer Security, ESORICS 2020, Guildford, UK, September 14–18, 2020, Proceedings, Part I 25*. Springer, 480–501.

[70] Florian Tramer and Dan Boneh. 2018. Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. *arXiv preprint arXiv:1806.03287* (2018).

[71] Stacey Truex, Nathalie Baracaldo, Ali Anwar, Thomas Steinke, Heiko Ludwig, Rui Zhang, and Yi Zhou. 2019. A hybrid approach to privacy-preserving federated learning. In *Proceedings of the 12th ACM Workshop on Artificial Intelligence and Security*. ACM, 1–11.

[72] 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 *27th USENIX Security Symposium*. USENIX, 991–1008.

[73] Jo Van Bulck, Daniel Moghimi, Michael Schwarz, Moritz Lipp, Marina Minkin, Daniel Genkin, Yarom Yuval, Berk Sunar, Daniel Gruss, and

Frank Piessens. 2020. LVI: Hijacking Transient Execution through Microarchitectural Load Value Injection. In *2020 IEEE Symposium on Security and Privacy*. IEEE, 54–72.

[74] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2020. SGAxe: How SGX fails in practice.

[75] Stephan van Schaik, Marina Minkin, Andrew Kwong, Daniel Genkin, and Yuval Yarom. 2020. CacheOut: Leaking data on Intel CPUs via cache evictions. *arXiv preprint arXiv:2006.13353* (2020).

[76] Paul Vanhaesebrouck, Aurélien Bellet, and Marc Tommasi. 2017. Decentralized collaborative learning of personalized models over networks. In *Artificial Intelligence and Statistics*. PMLR, 509–517.

[77] Jan Werner, Joshua Mason, Manos Antonakakis, Michalis Polychronakis, and Fabian Monrose. 2019. The SEVerESt Of Them All: Inference Attacks Against Secure Virtual Enclaves. In *Proceedings of the 2019 ACM Asia Conference on Computer and Communications Security*. ACM, 73–85.

[78] Luca Wilke, Jan Wichelmann, Mathias Morbitzer, and Thomas Eisenbarth. 2020. SEVurity: No Security Without Integrity–Breaking Integrity-Free Memory Encryption with Minimal Assumptions. *arXiv preprint arXiv:2004.11071* (2020).

[79] Chen Wu, Xian Yang, Sencun Zhu, and Prasenjit Mitra. 2020. Mitigating backdoor attacks in federated learning. *arXiv preprint arXiv:2011.01767* (2020).

[80] Chulin Xie, Minghao Chen, Pin-Yu Chen, and Bo Li. 2021. Crfl: Certifiably robust federated learning against backdoor attacks. In *International Conference on Machine Learning*. PMLR, 11372–11382.

[81] Dong Yin, Yudong Chen, Ramchandran Kannan, and Peter Bartlett. 2018. Byzantine-robust distributed learning: Towards optimal statistical rates. In *International Conference on Machine Learning*. PMLR, 5650–5659.

[82] Hongxu Yin, Arun Mallya, Arash Vahdat, Jose M Alvarez, Jan Kautz, and Pavlo Molchanov. 2021. See through Gradients: Image Batch Recovery via GradInversion. *arXiv preprint arXiv:2104.07586* (2021).

[83] Liangqi Yuan, Lichao Sun, Philip S Yu, and Ziran Wang. 2023. Decentralized Federated Learning: A Survey and Perspective. *arXiv preprint arXiv:2306.01603* (2023).

[84] Bo Zhao, Konda Reddy Mopuri, and Hakan Bilen. 2020. iDLG: Improved Deep Leakage from Gradients. *arXiv preprint arXiv:2001.02610* (2020).

[85] 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 *14th USENIX Symposium on Networked Systems Design and Implementation*. USENIX, 283–298.

[86] Wenting Zheng, Ryan Deng, Weikeng Chen, Raluca Ada Popa, Aurojit Panda, and Ion Stoica. 2021. Cerebro: A Platform for Multi-Party Cryptographic Collaborative Learning. In *30th USENIX Security Symposium*. USENIX.

[87] Wenting Zheng, Raluca Ada Popa, Joseph E Gonzalez, and Ion Stoica. 2019. Helen: Maliciously secure coopetitive learning for linear models. In *2019 IEEE Symposium on Security and Privacy*. IEEE, 724–738.

[88] Ligeng Zhu, Zhijian Liu, and Song Han. 2019. Deep leakage from gradients. In *Advances in Neural Information Processing Systems*. 14774–14784.