SecureFL: Privacy Preserving Federated Learning with SGX and TrustZone

Eugene Kuznetsov*
Arizona State University
ekuznets@asu.edu

Yitao Chen*
Arizona State University
ychen404@asu.edu

Ming Zhao
Arizona State University
mingzhao@asu.edu

ABSTRACT

Federated learning allows a large group of edge workers to collaboratively train a shared model without revealing their local data. It has become a powerful tool for deep learning in heterogeneous environments. User privacy is preserved by keeping the training data local to each device. However, federated learning still requires workers to share their weights, which can leak private information during collaboration. This paper introduces *SecureFL*, a practical framework that provides end-to-end security of federated learning. SecureFL integrates widely available Trusted Execution Environments (TEE) to protect against privacy leaks. SecureFL also uses carefully designed partitioning and aggregation techniques to ensure TEE efficiency on both the cloud and edge workers. SecureFL is both practical and efficient in securing the end-to-end process of federated learning, providing reasonable overhead given the privacy benefits. The paper provides thorough security analysis and performance evaluation of SecureFL, which show that the overhead is reasonable considering the substantial privacy benefits that it provides.

CCS CONCEPTS

• **Security and privacy** → *Distributed systems security*; *Software security engineering*; • **Computing methodologies** → *Supervised learning by classification*; • **Computer systems organization** → *Neural networks.*

KEYWORDS

Federated Learning, Privacy, Edge Computing, Trusted Execution Environment

ACM Reference Format:
Eugene Kuznetsov, Yitao Chen, and Ming Zhao. 2021. SecureFL: Privacy Preserving Federated Learning with SGX and TrustZone. In *The Sixth ACM/IEEE Symposium on Edge Computing (SEC '21), December 14–17, 2021, San Jose, CA, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3453142.3491287

*Both authors contributed equally to this research.

SEC '21, December 14–17, 2021, San Jose, CA, USA

ACM ISBN 978-1-4503-8390-5/21/12…$15.00
https://doi.org/10.1145/3453142.3491287

1 INTRODUCTION

Federated learning allows a large number of edge workers to collaboratively train a shared model while keeping their private data local. Each device trains a local model with data that is kept local to the device. Periodically, the local models of the edge workers are sent to a server in the cloud. This server performs aggregation against all the local models and sends one shared model back to each edge worker. Examples include GBoard mobile keyboard which takes advantage of federated learning to train language models for tasks such as next-word prediction without exporting sensitive user data to servers [8, 15, 37, 45]; Android Messages can also provide personalized suggestions with the help of federated learning [13]. With the rapid technology advancement of smart edge devices (smartphones, wearables, and Internet of Things), more and more applications benefit from federated learning.

Even with keeping all of the training data locally, user privacy is still at risk [1, 4, 12, 28, 29, 49]. Recent works have demonstrated that federated learning may not always provide sufficient privacy guarantees, as communicating model updates throughout the training process can reveal sensitive information [4, 12, 29] and even incur deep leakage [49]. In order to further protect user privacy, federated learning systems use techniques such as secure multi-party computation (MPC) [24] to make sure that the server only has access to the aggregated version of the user's model updates. Secure MPC uses cryptographic primitives to ensure that individual user's data cannot be discovered. However, this approach does not consider the case where the shared model is hosted on an Honest-But-Curious (HBC) third-party hardware provider that has full access to the shared model. An HBC party is someone who is a legitimate participant that does not tamper with data but attempts to learn information from viewing the data [36].

Our approach, SecureFL, protects both the edge workers and the aggregator against common attacks such as membership inference attacks (MIA) [40] and information disclosure attacks [14, 41, 47, 48] using Trusted Execution Environments (TEEs). The design of our solution consists of two major components. In the cloud, our design provides secured aggregation and protects the shared model from an HBC resource provider using Intel SGX. On the edge, our method uses ARM TrustZone to provide secure model training and inference to protect the shared model from an HBC data provider or model user. On both cloud and edge, SecureFL carefully addresses the resource constraints of the available TEEs using techniques including model partitioning and partial aggregation. By combining SGX in the cloud server and TrustZone on the edge, we develop a new federated learning system that provides end-to-end protection of user privacy.

We prototype SecureFL on real TEE hardware/software stack, including SGX for the cloud aggregator and OP-TEE and Darknet

for the edge workers. We provide a comprehensive evaluation of SecureFL with a Raspberry Pi cluster that consists of ten Raspberry Pi 3B+ to understand the performance and overhead of SecureFL on commonly used IoT platforms. We also use 100 emulators on a 15-node cluster to evaluate the scalability of SecureFL. The most significant result of our study is the reasonable overhead to the federated learning process given the strong security benefits provided by SecureFL. To measure the overhead, we compare the total runtime of the SecureFL implementation and compare it to the baseline, no security implementation. The overhead for a single model is as small as 0.58%. When scaling up the system, our results show that the overhead is 14% with ten Raspberry Pi devices. In a more aggressive scalability test of 100 workers, our implementation has an overhead of 23.6%.

The contributions of this study are as follows: 1) design and implementation of a federated learning system for better user privacy using widely available SGX and TrustZone TEEs based on a commonly used deep learning framework, Darknet [38]; and 2) an in-depth examination of our federated learning system with a thorough security analysis and a comprehensive performance evaluation.

Federated learning raises unique privacy challenges due to the need of protecting each user's sensitive data while learning from many users. It also brings unique opportunities as models can be partitioned between secured and unsecured worlds to address the resource constraints. The approach taken by SecureFL can be generalized to other distributed learning systems and, more generally, distributed computing problems that share these characteristics.

The paper is organized as follows: Section 2 introduces the background and related works; Section 3 describes the threat model of the system; Section 4 discusses our system design; Section 5 provides a security analysis of our work; Section 6 presents the experimental results; and Section 7 concludes the paper.

2 BACKGROUND AND RELATED WORKS

2.1 Deep Learning and Neural Networks

Deep learning is the key to various mobile computing tasks, such as voice recognition, natural language processing, image classification, and object detection. DNNs have achieved remarkable accuracy in the above applications compared to other machine learning techniques. Image classification is one of the major applications of deep learning algorithms, which is computationally intensive and involves a large volume of data. Such tasks obtain images as input and use neural networks to label each image with a class that the image belongs to.

A neural network consists of multiple connected layers. A layer is an abstraction to help with the modular design of a network, where each layer performs a specific mathematical operation. Examples of common layers include fully-connected layers, which connect to all neurons from the previous layer and output a weighted sum, and convolutional layers, which use kernels with various sizes to extract the features from the overlapped area of the input.

Deep learning tasks include training a model and using the model to perform inference. Training follows the backward path of a network and updates the weights of each layer iteratively towards a target. Inference follows the forward path and uses the trained weights of each layer to make a prediction based on the input. In the context of image classification, training updates the weights of a model to reduce the difference between the prediction and the ground truth label, whereas inference uses images as input to predict the image class.

2.2 Federated learning

Federated learning is a learning paradigm that allows many edge workers to collaboratively train a shared model using their local data [27]. In particular, the learning task is solved by a large number of edge workers that participate in the learning. Workers tend to be distributed and trained on local data to preserve user privacy. After performing local training for a pre-defined number of iterations, each worker begins to contribute to the shared model. Each worker contributes to the training of the shared model by sending their local model updates to the aggregator. These updates are the weights of the model that each device has trained locally. The aggregator server receives the weights from all the workers and performs averaging on the received weights. The aggregator uses the federated averaging algorithm [27], which performs averaging of all the weights from the edge workers. This aggregation creates the shared model, combining the weights from all the workers. The weights are the information that each worker learns from its local dataset. The aggregator server then sends the shared model back to each of the edge workers. After receiving the aggregated weights from the aggregator, each worker then updates their local model with this shared model.

2.3 Privacy Risk of Federated Learning

Although federated learning protects privacy by not sharing the users' data, the model trained with the users' data is still shared. Recent works have shown that information disclosure attacks and membership inference attacks can be launched on a trained model to reveal sensitive information about the training data. These attacks are not unique to federated learning, but the distributed nature of a federated learning system makes it especially challenging to protect data privacy as it involves multiple actors and distributed attack surfaces in different cloud and edge environments.

2.3.1 Information Disclosure Attack. Information disclosure reveals sensitive information within an application. Sensitive information can be either data served as input to an application, intermediate data representations during the application execution process, or the final result of a process. Information disclosure attacks can happen if a vulnerability is exploited within an application or if data is incorrectly handled within an application. Information disclosure has been shown to be of concern with DNNs that run on confidential inputs [14]. Shallow layers of the DNN model correspond to low-level photographic information [41, 47, 48]. These layers are considered intermediate representations during the learning process, but reveal explicit information. An adversary that has access to these intermediate representations can view the input data with small loss of information.

2.3.2 Membership Inference Attack. In a membership inference attack, the attacker is trying to learn if particular data belongs to

the training dataset. There are two types of membership inference attack (MIA): black-box attack and white-box attack.

In a black-box MIA attack, the attacker can only access the model output. He can then use the model output to train a shadow model to identify if certain data is included in the training dataset [40]. There are three methods to generate training data for shadow models. The first method uses black-box access to the target model to synthesize this data. The second method uses the statistics about the population from which the target's training dataset was drawn. The third method extracts the training data from a noisy version, assuming the adversary has access. A typical example of a black-box MIA is to attack machine learning as service models provided by commercial providers.

In a white-box MIA attack, the attacker develops an attack model that uses internal knowledge of the target model (gradients and activation of layers) in addition to the target model's outputs to perform an attack. This is more effective compared to the same attack in a black-box setting. The attacker first applies a feature extraction component to each layer output of the target model. The output of all these feature extraction components is then combined using the encoder component, which is a fully-connected model with multiple hidden layers. The combined output is a score that predicts the membership probability of the input data.

2.4 Security Mechanisms

SGX. SGX is a hardware-based security mechanism to protect applications running on a remote server. The security feature is widely available on cloud servers and is a practical choice for our purpose. SGX provides a TEE to ring 3 (Intel user-space) applications running on Intel SGX processors [10]. The TEE that contains the application is known as an enclave. Enclaves, including metadata, are kept in physical memory known as the Enclave Page Cache (EPC). This EPC is a designated region of memory that is protected and encrypted using the Memory Encryption Engine (MEE) located inside the CPU [25]. The size of the EPC is currently limited to a maximum of 128 MB, which includes both enclaves and metadata. Whenever the application size exceeds this limit, a CPU-expensive SGX paging mechanism is triggered. An enclave communicates with the outside world through SGX specific function arguments and return values.

SGX TEE protects user-space applications from an untrusted operating system or an adversary that has full physical access to the machine. The encryption engine, located in the Intel processor, uses encryption to protect all SGX application data that leaves the CPU package. This prevents an adversary from executing bus snooping attacks or reading the contents of the EPC in main memory. Furthermore, SGX provides assurance to a remote user that the SGX application is running on the correct hardware, preventing an adversary from changing the underlying hardware [3]. SGX only assumes a trusted CPU package; all other system components and privileged software are considered untrusted.

SGX comes with inherent constraints that limit certain operations for secured SGX processes. The first mentioned constraint is the memory size. Another limitation of SGX is the restriction of privileged instructions. If the SGX application wants to run a system call, this requires a series of security checks within the hardware. The hardware security enforcement ensures that no information from the SGX enclave is leaked to the privileged OS or to any other process in the system.

Previous works have used SGX to perform inference and data processing, considering certain limitations in SGX. Gu et al. [14] proposed to partition deep neural networks (DNNs) and use SGX to secure the layers of a model in the EPC for model serving. The layers of a model contain intermediate representations. This work proposed the idea of securing intermediate representations to defend against input information disclosure. Yoo et al. [46] proposed to improve the performance of SGX by offloading computations to GPUs. The proposed approach places the first few layers of a deep neural network model into the EPC, while the remaining layers are exported outside the EPC for GPU processing. The previous works consider only model inference and do not consider a federated environment with distributed workers. Federated learning comes with additional constraints not considered in the previous works, such as system call overhead and memory constraints for the scalability of a federated system. However, the techniques applied in the previous works are orthogonal to our approach and may be considered for additional improvement.

TrustZone. ARM also provides a hardware security architecture to defend against a number of threats. ARM TrustZone is a widely available security feature that is specific to edge workers. Thus, it is chosen to protect the edge devices in this work. SGX provides a fixed hardware architecture from Intel, providing fixed memory partitions for main memory and limited to ring-3 instructions only. In contrast to SGX, ARM TrustZone allows a System on Chip (SoC) designer to make security specific decisions. These decisions apply to a range of components, such as memories, interrupts, peripheral interfaces, bus systems, and software [16]. All resources, both hardware and software, are partitioned to be in one of two worlds—secure world or normal world. The normal world is considered part of the untrusted operating system known as the Rich Execution Environment (REE). The secure world contains all trusted operations in a hardware-isolated TEE. All components that are security critical are placed in the secure world and are considered part of the TEE.

ARM TrustZone hardware guarantees that all resources allocated to the TEE are not accessible by any component in the normal world. This gives the freedom for a designer to make architecture specific decisions that will appropriately accommodate software requirements. For example, a designer can allocate an appropriate amount of main memory in the TrustZone TEE required for efficient training and inference of various models. In TrustZone, the designer is free to partition the main memory between secure world and normal world, as desired. Typical hardware designs allocate 3-5 MB of main memory to secure world [2]. A designated hardware bit (NS bit) is used to ensure that the the normal world cannot access the memory regions belonging to the secure world.

TrustZone can be leveraged to protect a variety of software, ranging from simple applications and libraries to complete operating systems [16]. Open Portable Trusted Execution Environment (OP-TEE) [33] is a TEE primarily designed for TrustZone hardware. It provides a secure operating system, known as its secure privileged layer. This secure operating system runs alongside a

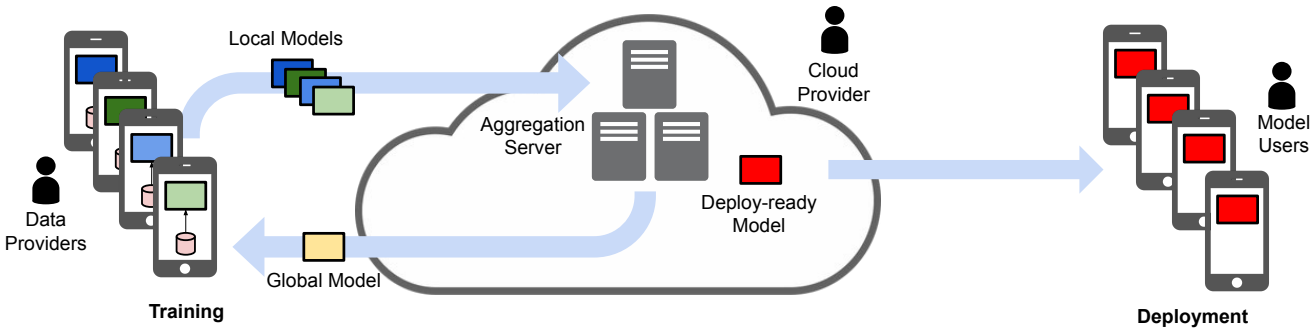


Figure 1: Actors in federated learning. Federated learning architecture consists of cloud resources and edge devices, and involves several actors in these environments. The data providers contribute private data by each training a local model using the local data on the local device. The cloud provider provides cloud resources for aggregating the models trained on the edge. The aggregation server performs the aggregation using algorithms such as FedAvg to produce a global model. The final model can be then deployed onto many edge devices and the model users use the deployed models to perform inference on the edge

non-secure Linux kernel running in the normal world [33]. OP-TEE also provides a set of libraries for user-space applications to run as trusted applications in the TEE. An API is provided to allow applications running in the normal world to communicate with the trusted applications using shared memory.

Previous works considered using TrustZone to protect DNN model training and inference. Mo et al. [30] used TrustZone to secure DNN models for both training and inference on the Darknet [38] framework. Due to the limited TEE size of TrustZone, model partitioning is used to protect only the last few layers. The layers that are closest to the raw data are considered the first few layers. The layers furthest away are considered the last few layers. Nasr et al. [31] showed that placing the last few layers of a DNN model significantly lowers the success of an MIA attack. Our proposed work uses this same observation and employs TrustZone protections to secure the DNN model against MIA attacks on the edge workers. Our work provides further contributions and insights to expand this attack model into a distributed federated learning environment.

3 THREAT MODEL

3.1 Federated Learning Architecture

As illustrated in Fig. 1, the federated learning architecture consists of cloud resources and edge devices, and involves several actors in these environments. The goal of federated learning is to train a model using the data collected on the edge devices while preserving the privacy of the data.

On the edge, the *data providers* contribute private data by each training a local model using the local data on the local device. After local training is performed for a pre-defined number of iterations on an edge device, the local model is uploaded to the aggregation server in the cloud.

In the cloud, the *cloud provider* provides cloud resources for aggregating the models trained on the edge. After receiving all of the local models from the edge devices, the aggregation server performs the aggregation using algorithms such as FedAvg, FedDF, FedBE, or FedMA [7, 23, 26, 43] and produces a global model. This global model is then sent back to each device which uses it to continue the next round of local training.

The above process is repeated across edge and cloud for many rounds until the global model converges. The final model can be then deployed onto many edge devices where the *model users* use the deployed models to perform inference on the edge. Note that a model user may or may not be a data provider in the training phase. Therefore, we discuss it as a separate actor.

3.2 Adversaries

The overarching goal of SecureFL is to protect the privacy of data while performing federated learning on the data efficiently. Even though a data provider does not share the data with other actors in the federated learning system, the model trained with the data is shared. The model contains sensitive information from all data providers in the federated learning system, as all data providers contribute their training to this model. As explained in Section 2.3.2, adversaries can launch information disclosure and MIA attacks on the shared model and learn sensitive information. Therefore, SecureFL needs to preserve the privacy of the shared model in federated learning.

3.2.1 Cloud Adversary. In the cloud, SecureFL protects the privacy of the shared model when it is being aggregated on cloud resources and when transferred from the cloud to the devices.

We assume an adversary on the cloud that is honestly running the SecureFL software, but is attempting to gain information regarding the private local data (provided by data providers) of each edge device. The adversary can be the cloud provider in the aforementioned system architecture (Fig. 1).

We consider that the adversary has access to the system and hardware located on the cloud. They can access and control the entire system software stack, including all privileged software and user-space code, operating system, and any other system management software on the cloud. An adversary with control of such system components can gain unprivileged access to all the uploaded local models and the aggregated global model. The adversary may be able to perform an information disclosure attack on the first few

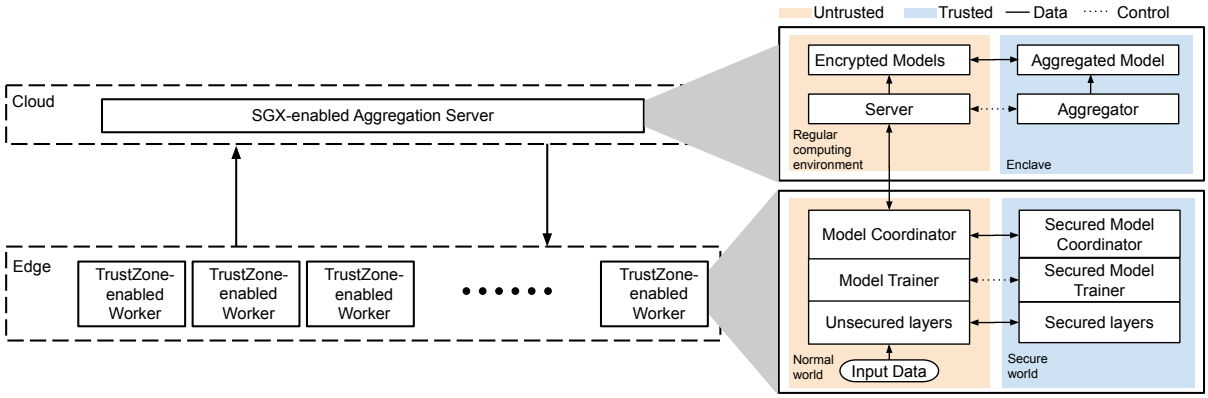


Figure 2: System diagram of SecureFL. The edge worker is protected by TrustZone and the cloud aggregator is protected by SGX. The untrusted environment is in orange and the trusted environment is in blue. The solid lines in the figure denote the data flow and the dotted lines denote the control flow.

layers of the models to reveal sensitive model inputs, as described in Section 2.3.1. The adversary can also attempt an MIA attack on the last few layers of the models to reveal information of the training data, as described in section 2.3.2.

3.2.2 Edge Adversary. On the edge, SecureFL protects the privacy of the shared model when it is being trained on the devices and when transferred from the devices to the cloud. It also protects the privacy of the model when deployed on the model users' devices.

We assume that the adversary on the edge is honestly running the SecureFL software, but is attempting to gain sensitive information regarding the data for training the shared model. The adversary can be a data provider or a model user. A data provider has access to its local training data, so there is no need to protect the local data from the owner of the data; the data provider may try to learn sensitive information from the data contributed by other data providers. A model user uses the deployed model to perform inference on local inputs, so there is no need to protect the local inputs from the owner of the inputs; but the model user may try to learn sensitive information from the training data contributed by the data providers.

We consider a local adversary that has full access to the normal world of the edge device. The adversary can access the entire software stack, including the normal world operating system and applications on the local device. The adversary has access to only the shared global model which is aggregated from all the data providers' local models. The aggregation process makes it difficult for an information disclosure attack to learn sensitive information about the local models' inputs from the aggregated shallow layers of the global model. But the adversary can still perform an MIA attack on the last few layers of the global model, revealing sensitive information of the training data on other edge devices.

4 SYSTEM DESIGN

4.1 Design Overview

We now introduce our framework for secure federated learning, SecureFL. The objective of SecureFL is to protect the privacy of data contributed by the data providers. Here we describe the security framework for each environment considering its respective adversary. Fig. 2 shows the complete workflow between the edge and cloud. The solid lines indicate the data flow of the trained model and the dotted lines represent control signals, indicating the interactions of various components. We will describe the main components in detail in the following subsections.

4.2 Attestation

SecureFL protects the privacy of data contributed by the data providers. Before a data provider contributes any data towards the federated learning process, attestation needs to be performed against the cloud provider and each data provider on the edge. Each data provider first performs attestation against the cloud server to verify the cloud TEE hardware and software. Once the cloud server is attested, the cloud server TEE software then performs attestation against each edge device to verify the device TEE hardware and software. This method allows each data provider to perform transitive attestation of all other data providers without having to attest each party directly. Each data provider needs only to perform attestation against the TEE environment of the cloud server which then continues attestation of all other data providers in the federated learning environment. Specifically, attestation of each cloud host or edge device verifies 1) the hardware of the host or device that implements the TEE, and 2) the SecureFL software that is initialized within the TEE.

4.2.1 SGX Attestation. To perform attestation against the cloud server TEE hardware, the TEE is required to contact a trusted provider (Intel) [3]. Each SGX CPU contains an embedded provisioning secret used for the secure assertion. This secret is burned into the hardware as part of the manufacturing process [10]. The trusted provider uses this embedded provisioning secret to establish a secure assertion that identifies the TEE hardware being used [3].

Verification of the initialized software in TEE begins with the enclave launch [3]. The SecureFL code is loaded into this enclave using special hardware instructions. During this process, a secure hardware log keeps track of how the enclave is built, also known

as the measurement. This measurement uses a cryptographic hash to prove the contents [10]. The measurement is compared against the expected cryptographic hash of the built enclave. If the hashes match, this proves to the data provider that the correct software has been built in the TEE.

4.2.2 TrustZone. Once attestation has been performed against the cloud server TEE, each data provider needs to perform attestation against all other data providers to verify TEE hardware and software. Having data providers attest each other directly would incur significant overhead with the increase in the number of devices participating in the federated learning process. The attestation of each edge device is instead performed by the TEE of cloud server. This provides transitive attestation of each data provider, allowing data providers to attest each other indirectly. Each data provider performs attestation of the cloud server TEE which then continues to attest all devices.

The verification of the edge device TEE hardware requires the hardware to be attested with the trusted manufacturer's attestation server, such as in Samsung Knox [39]. A trusted provider embeds unique keys within each device, allowing for a trusted boot process to derive device specific keys and the trusted manufacturer's endorsement of the hardware. These are used to verify the correct TEE hardware and are considered to be part of the root of trust.

Once the TEE hardware has been attested with the manufacturer, the software environment of the TEE can be verified. The immutable hardware root of trust allows the processor to boot from trusted code and provide measurements of the state of the environment and loaded software, including SecureFL. The cloud server can utilize these measurements to perform attestation with edge devices, contacting both the edge device and trusted manufacturer's attestation server to verify the TEE software of each edge device.

4.3 Edge Components

4.3.1 Model Trainers. The federated learning process starts with the distributed set of workers on the edge, bringing us to the first challenge: *how can local training be protected on each edge worker?* As discussed in Section 2.3.2, we need to protect against MIA attacks, but the limited size of the protected memory in TrustZone also imposes a constraint on the number of layers that TrustZone can protect. The TEE size is typically limited to 3-5 MB [2]. Fortunately, protecting only the last few layers can significantly reduce the success rate of such attacks. For example, related studies [30] have shown that placing only the last layer of a model into TrustZone can degrade adversary's success rate to random guess. Therefore, to protect local training, SecureFL splits the DNN model between the two worlds on the edge worker: the secure world and the normal world. The first set of layers are placed in normal world. The last few layers of the DNN model are placed in the secure world of TrustZone.

Once the DNN model layers are successfully partitioned between the two worlds, we encounter our next challenge: *how can training be performed on a single model in two separate worlds?* Because the normal world cannot access the resources of the secure world, we require separate training in the two worlds. To accomplish this, a separate secure Model Trainer is placed in the secure world. These two trainers communicate through an OPTEE TEE session which is a logical connection between the trusted and untrusted worlds [35]. Training starts with the normal world trainer. When the normal world trainer encounters the partition point, it sends the output of last layer before the partition point to the secure world trainer. The secure world trainer then continues to finish training the secured layers. Once the secure trainer finishes training the secure layers, it sends back the model output to the normal world to continue further training iterations.

4.3.2 Model Coordinator. In a federated learning environment, we need to upload and update our local model periodically. *How can the upload and the update be secured on a local model that is partitioned between two worlds?* We design Model Coordinator to facilitate this. The Model Coordinator has two roles: 1) copy the weights between the secure world and the normal world; 2) communicate with the server to send the local model and receive the aggregated model. Once the Model Trainer finishes several training steps, the Model Coordinator copies all the weights of the secured layers in the secure world to the normal world because the network interface is allocated to the normal world. The Model Coordinator uploads the local model to the server for aggregation. The Model Coordinator then waits for the server to send back the aggregated model. After receiving the aggregated model, the Model Coordinator copies all the secured weights from the aggregated model to the secure world and updates both the secured layers and non-secured layers of the local model and continue the training.

The secured layers must be protected when they are copied from the secure world to the normal world, and the whole model needs to be protected when it is sent from the client to the server. To meet these requirements, the secure world and the normal world have their own AES engines. Allowing separate AES engines solves two problems: 1) the secure world AES engine can protect the secure layers from MIA attacks when sending the weights to the normal world, and 2) the normal world AES engine can encrypt all remaining layers, allowing the entire model to be protected from MIA attacks when being sent to the server. With this protection in place, the Model Coordinators can send and receive weights to update the local layers in their respective world.

4.4 Cloud Components

4.4.1 Server. The Server is the communication interface to all the distributed workers. It sends and receives all of the data between the Aggregator and the individual workers. A naive approach would be to simply place this interface in SGX along with the Aggregator. As discussed in Section 2.4, SGX imposes a limitation on applications. Applications placed in the enclave cannot simply execute system calls. Every system call triggers a series of security checks that incurs overhead to the application [10]. Placing the communication interface in SGX would incur a significant amount of overhead due to the amount of communication required to transfer large models between the server and each worker. To avoid such overhead, this interface is placed outside of SGX. Placing the interface outside of the SGX TEE does not introduce any security implications. The role of the network interface is to simply transfer data between the network and memory. In SecureFL, this data is encrypted and does not leak any information to the unsecured network interface.

4.4.2 Aggregator. The Aggregator is the component that aggregates the local models uploaded by the edge workers and creates a shared global model. This component must be carefully designed to perform aggregation. SGX provides isolation between a regular computing environment and the SGX enclave. The enclave is given a dedicated region of main memory, known as the EPC [25]. This region of memory can only be accessed by the enclave process and is restricted access by any other parties, including other user-space processes, privileged software and the OS. However, the enclave program is allowed access to non-EPC memory regions. The communication interface is placed outside of the enclave to avoid the high overhead of system calls. This means that the interface cannot access the EPC memory to store the local models received from each edge worker. To solve this problem, SecureFL allocates memory outside of the enclave, in the regular computing environment, to buffer the local models. The location of this memory is shared with the enclave using the SGX function call arguments. The Aggregator running in the enclave can then copy the local models from the shared memory in the regular computing environment to a secure memory region located inside the enclave to perform secured aggregation. The models buffered in the shared memory are considered secured from malicious parties as they are received and stored in encrypted format. Once these models are copied into Secure Memory within the enclave, they are protected by SGX hardware from unauthorized access.

The next problem to consider is *how to copy the local models from the non-EPC memory region into the EPC*. The Aggregator must consider that the size of the EPC is limited to 128 MB [10]. It needs to determine if the DNN models from all the edge workers can fit inside the Secure Memory in the enclave. As mentioned in Section 2.4, an application that exceeds the size of the EPC triggers the SGX paging mechanism. This paging mechanism is expensive and adds additional overhead to the runtime of a program. Therefore, the Aggregator first determines how much data to copy to avoid triggering SGX paging. When the model size exceeds the EPC capacity, the Aggregator cannot even load a single model into the encalve; instead, it will copy only part of a model into EPC and perform aggregation part by part to avoid the paging overhead. Once the weights are copied into secure memory, the Aggregator will send these weights to its AES engine to be decrypted, and then perform aggregation on these layers. Once aggregation is completed, the Aggregator frees the Secure Memory and begins copying the next subset of the models from the non-EPC shared memory for the next round of aggregation. After aggregating all the local models, the Aggregator encrypts the final global model and copy it back to the shared memory, allowing the communication interface to send this updated global model back to the edge workers.

The last problem to consider is how to perform aggregation when only a subset of the worker models are given to the Aggregator. The Aggregator only performs a summation on the subset of the DNN models copied into Secure Memory. This summation is performed for each subset that is copied in and stores the intermediate summation result. This intermediate summation result adds to the summation result of the next subset of DNN models. Once there are no more DNN models to be copied into Secure Memory, the Aggregator divides the summation result by the number of workers to obtain the average.

5 SECURITY ANALYSIS

In this section, we provide a security analysis of SecureFL. We analyze the components on both the cloud and edge workers and the key functions that need to be secured to protect the privacy of training data in the context of federated learning.

5.1 Broadcast

The broadcast step consists of the edge workers (clients) downloading the shared model from the aggregator (server). A malicious user on the cloud or the edge can gain unauthorized access to this information and manipulate data. To protect the broadcast in SecureFL, the shared model is protected on both ends, cloud and edge. On the cloud, the shared model is first encrypted in a TEE. This protects the model from being seen or manipulated by a malicious user or administrator on the server. The encrypted shared model is then downloaded by the worker. On the edge worker, the shared model layers are decrypted. For sensitive layers, the decryption is done inside a TEE on the edge worker. This protects the layers from unauthorized access to the sensitive shared model layers. The remaining layers are decrypted outside of the TEE on the edge worker.

5.2 Client Computation

In federated learning, the training data is kept local on the edge workers (clients). Each worker performs computation to update the local model. This computation is typically a training program that updates the local model. Both the local data and the training program on the edge worker are susceptible to attacks by a malicious user. The storage and computation on the edge worker require careful planning to avoid exposing sensitive data to a malicious user.

The attack surface considered in SecureFL is the MIA attack discussed in Section 2.3.2. In this attack, an HBC adversary obtains sensitive information regarding the model and its training data by viewing the last few layers of the model and performing an MIA attack against these layers. To mitigate such an attack, SecureFL places the last layers of a model and the training program for these layers in the TEE to restrict access. The TEE on the edge worker provides full isolation from the user environment and protects the execution of the training program and the model weights during runtime.

Choosing where to partition a model between the secure world and normal world presents a trade-off between privacy and performance. Previous work [30] has shown that in some cases, such as the AlexNet model with CIFAR-100 dataset, placing only the last layer of the model can reduce the success rate of an MIA attack from 85% to 50% (the equivalent of random guess) when measuring standard precision and recall metrics [30, 31, 40]. Our results show that placing more layers in the TEE incurs more runtime overhead. Our results also confirm that the size of TrustZone memory imposes a practical limit on the number layers that can fit in TrustZone.

5.3 Aggregation

The server collects the model updates from each worker and performs aggregation. Aggregation combines all of the knowledge the workers learn from its local data. The security considerations of the

aggregator are critical. It is the central location in which all worker data is received. A malicious server administrator can view the worker updates or manipulate the aggregation process that affects all workers.

In SecureFL, the aggregation process is protected entirely inside a TEE. The worker updates are encrypted when received by the server. The decryption, aggregation, and re-encryption of all worker weights are done inside the TEE. A malicious or HBC cloud administrator does not have access to any of the worker model data or the aggregation process.

5.4 Model Update

The Model Update takes place after aggregation on the server has completed. Each worker downloads the shared global model from the aggregator. This global shared model is then used to replace the local model of each worker. An unsecure model update would allow a malicious party to view or alter the local model of the worker.

In SecureFL, the Model Update is secured inside of the TEE. After the secured aggregation process is completed, the aggregated model on the server is encrypted in the SGX TEE. This encrypted model is then sent to each worker securely. The workers download this encrypted global model and partition the model between normal and secure world. The secure world model layers are decrypted in the TrustZone TEE and the unsecured layers are decrypted in normal world. This process protects the worker's model update process and local model from being viewed or accessed by an unauthorized party.

5.5 Deployment

Once a model has completed the federated learning process, it is ready to be deployed to the model users' edge devices. We consider the model user to be an HBC adversary as described in Section 3.2.2. The adversary attempts to perform an MIA attack against the deployed model to gain sensitive information regarding the training data. To protect against such MIA attacks, similarly, we place the last few layers of the deployed model in the TEE of the edge device. The security analysis done in Section 5.2 also applies here.

5.6 Other Attacks

There are a number of attacks that can be exploited against the federated learning environment. SecureFL focuses on protecting the privacy of federated learning and considers only a subset of these attacks, such as the information disclosure and MIA attacks. Mitigation for other various attacks may also be incorporated as part of SecureFL. For example, side channel attacks [5, 6, 18] are not considered in SecureFL. In a Sybil attack, an adversary can control multiple malicious edge workers in order to influence the shared model and have it leak private information of the honest workers [11]. Mitigation strategies for such attacks are not considered as part of SecureFL, since we assume the edge users to be honest-but-curious. However, solutions [17, 32, 42] to these attacks are orthogonal to our work and can be implemented along with SecureFL. We leave these mitigation strategies for future work.

6 EVALUATION

6.1 Methodology

We designed our experiments to answer the following questions: *1) How much runtime overhead is incurred by the security mechanisms? 2) What contributes the most to the runtime overhead? 3) What is the workload on the workers? 4) What is the workload on the aggregation server? 5) How scalable is our approach?*

6.1.1 Models. We consider network architectures that are suitable for edge workers, since commonly used DNNs, such as ResNet-34, are computationally expensive for TrustZone on the edge workers. Specfically, we evaluate our framework with three commonly used models, LeNet [21], VGG-7 and VGG-16. LeNet has five convolutional layers and two fully connected layers. Each convolutional layer is followed by a max pooling layer. The activation is a rectified linear activation unit (ReLU). VGG-7 has five convolutional layers and two fully connected layers. Each convolutional layer is followed by a max pooling layer. The activation is a rectified linear activation linear unit (ReLU). The last few layers of each model are placed into TrustZone. To create a baseline, each model is tested with no layers placed in TrustZone. We test each model with up to 5 layers placed into TrustZone.

6.1.2 Dataset. We choose commonly used datasets, MNIST [22], CIFAR-10 [19], and Tiny ImageNet [20] for edge workers to evaluate our framework. The MNIST dataset consists of 60,000 28x28 grayscale training images and 10,000 validation images. CIFAR-10 consists of 60,000 32x32 RGB images, belonging to ten categories. There are 50,000 training images and 10,000 validation images.

6.1.3 Test platforms. Table 1 lists the hardware specifications of our test platforms. In order to evaluate the performance of our framework and understand the overhead of securing federated learning, we conducted various experiments on a real testbed. The testbed consists of two components: aggregator and workers. The aggregator is SGX-enabled and runs on an Ubuntu 18.04 LTS server which has a 3.6 GHz quad-cord Intel i7-7700 and 16 GB main memory. The size of the EPC for SGX enclaves is 128 MB. The workers run on Raspberry Pi 3B+, a widely used IoT platform. The OS is OP-TEE, version 3.9. OP-TEE is a TEE for a non-secure Linux kernel running on ARM processors, relying on ARM TrustZone technology as the underlying hardware isolation mechanism. We built a cluster using ten Raspberry Pi's for our experiment. For larger scale experiments with more workers, we used OP-TEE with Quick Emulator (QEMU) version 7. Although the emulators cannot represent the real performance of the workers, we can run many instances to evaluate the scalability of the aggregator. We set up a total of 100 QEMU workers on four servers.

The workers are located on a 15-node cluster at *site-A*. The aggregator server is located at *site-B*. A VPN is used to establish a connection between the worker and server networks, with a ping of 56 ms. The Raspberry Pi workers are co-located on a local network and use a VPN to establish a connection with the aggregator server. The Pi network and the server have a ping of 32 ms. The specifications of each device are listed in Table 1.

Table 1: Test Platform Specifications

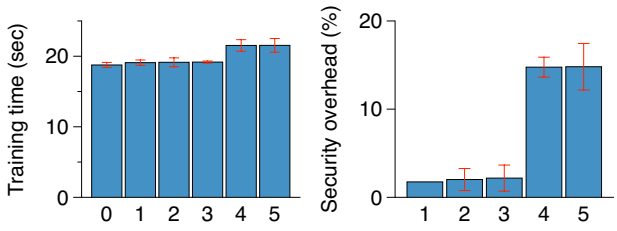
Specifications		
Raspberry Pi 3B+	OS	Op-tee OS
	CPU	1.4 GHz quad-core Cortex A53
	Memory	1 GB
	Network	SMSC 100-Megabit LAN9514-JZX
QEMU Worker	OS	Op-tee OS
	CPU	Dual-core vCPU
	Memory	8 GB
	Network	Intel 10-Gigabit X540-AT2
QEMU Server	OS	Ubuntu 16.04 LTS
	CPU	2.4 GHz 32-core Intel E5-2630 v3
	Memory	64 GB
	Network	Intel 10-Gigabit X540-AT2
Aggregator Server	OS	Ubuntu 18.04 LTS
	CPU	3.6 GHz quad-core Intel i7-7700
	Memory	16 GB
	Network	Intel 1-Gigabit I219-LM

Note that although edge devices can typically run larger models for inference, they are generally insufficient to train larger models [9]. We ran SecureFL with VGG-16 on Tiny ImageNet [20] and found it too large to run due to memory constraint. VGG-16 has sixteen convolutional layers and three fully connected layers. Each convolutional layer is followed by a max pooling layer. The activation is a rectified linear activation linear unit (ReLU). The Tiny ImageNet dataset is a subset of the ImageNet dataset used in the ILSVRC 2012 competition [20]. It consists of 110,000 64x64 RGB images, belonging to 200 categories. There are 100,000 training images and 10,000 validation images. To place the last layer in TrustZone, we kept the depth of the model and reduced the width by reducing the number of output channels of the convolutional layers. With a batch size of 50, it takes 15 minutes to train 10 iterations. Therefore, in the following evaluation, we focus on the two models, LeNet and VGG-7, which are more suitable for training on edge devices.

6.2 Runtime Overhead

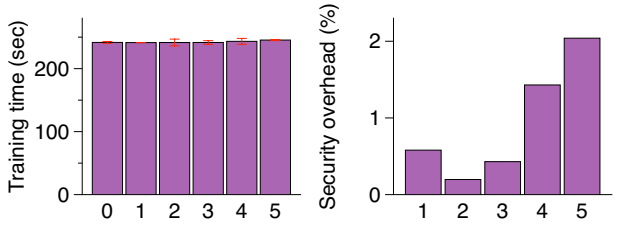
To answer the first question, *How much runtime overhead is incurred by the security mechanisms?*, we measured the runtime performance of edge workers on training neural networks. Specifically, we changed the number of layers in TrustZone to understand how the runtime overhead changes. We first evaluated LeNet [21] on the MNIST dataset and VGG-7 on the CIFAR-10 dataset. We measured the runtime of one round of federated learning, including ten training iterations on the edge and one aggregation step on the server. For comparison, we used a baseline of federated learning without adding any security mechanisms (without using SGX on the server, without encrypting the communication, and without TrustZone). We provide a breakdown of these mechanisms in Figure 5. For both models, we used a batch size of 50.

Fig. 3a illustrates the training time of LeNet with a varying number of layers executed in TrustZone, ranging from 1 to 5. Our baseline is the training time when no layers are executed in TrustZone. Fig. 3b shows the overhead in percentages when varying



(a) LeNet training. **(b) LeNet security overhead.**

Figure 3: LeNet training time overhead with a varying number of layers executed in TrustZone, ranging from 1 to 5. The baseline is the training time when no layers are executed in TrustZone.



(a) VGG-7 training. **(b) VGG-7 security overhead.**

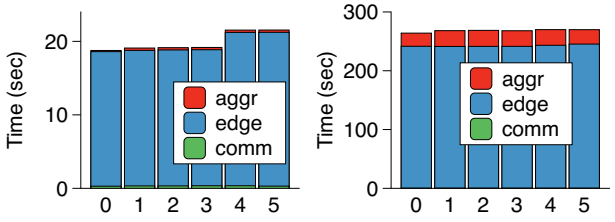
Figure 4: VGG-7 training time overhead when varying the model partitioning point. The VGG-7 model takes about 7X longer than LeNet due to its complexity.

the number of layers in TrustZone. The overhead increases from 1.6% (with only one layer in TrustZone) to 12.7% (with five layers in TrustZone). The security overhead increases with the number of layers in TrustZone. The majority of the overhead on the edge worker comes from updating the trusted layers of the local model with the received shared model. This includes copying to and from TrustZone, along with encrypting and decrypting the secured model layers. When only one layer is in TrustZone, the security overhead is negligible. The overhead from each part of the secured federated learning architecture is further investigated in the next section.

Fig. 4a shows the training time of VGG-7 when varying the model partitioning point. The complexity of VGG-7 manifests in the training time. The VGG-7 model takes, on average, 300 seconds to complete the ten iterations, which is about 7X longer than LeNet. We then observe that the larger training time of VGG-7 reduces the percentage of the security overhead compared to that of LeNet in Fig. 4b. The overhead is 0.58% when only one layer is in TrustZone and 2.04% when five layers are in TrustZone. In comparison, the overhead of putting five layers in TrustZone with LeNet is 12.7%.

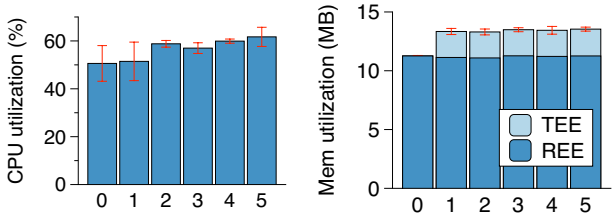
6.3 Runtime breakdown

To further understand the security overhead, we broke down both the LeNet and VGG-7 training time into three major components: aggregation time (*aggr*), edge computing time (*edge*), and communication time (*comm*). This will answer our second question: *What contributes the most to the runtime overhead?* The aggregation time



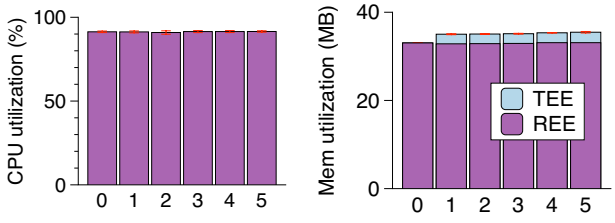
(a) LeNet. (b) VGG-7.

Figure 5: Runtime breakdown. We broke down both the LeNet and VGG-7 training time into three major components: aggregation time (*aggr*), edge computing time (*edge*), and communication time (*comm*). The aggregation time measures the time spent calculating the weight averages from all the edge workers. The edge computing time measures the time overhead of training a model on the edge. The communication time measures the time overhead of exchanging model weights between edge and cloud.



(a) LeNet CPU utilization. (b) LeNet memory utilization.

Figure 6: LeNet training resource utilization. We calculate the CPU utilization for the edge worker by extracting the relevant information from the /proc filesystem. Due to the security isolation between normal world (REE) and secure world (TEE), the utilization of the secure world is calculated with a method proposed by previous work.



(a) VGG-7 CPU utilization. (b) VGG-7 memory utilization.

Figure 7: VGG-7 training resource utilization.

measures the time spent calculating the weight averages from all the edge workers. The edge computing time measures the time overhead of training a model on the edge.

Fig. 5a illustrates the runtime breakdown of unsecured workflow (0 layers in TrustZone) and secured workflow (1 to 5 layers in TrustZone) using LeNet model. The major components of the total runtime overhead are edge computing time and communication time. The edge computing time accounts for, on average, 54% of the total runtime. The communication time is negligible as it is less than 0.4%. The aggregation time is almost invisible in the figure because it accounts for less than 0.8% of the total run time. Although it is insignificant in the total time overhead, the aggregation time of secured workflow is about 50% slower than that of the unsecured workflow. The biggest overhead for the secured aggregator is the data transfer into the EPC using the shared memory.

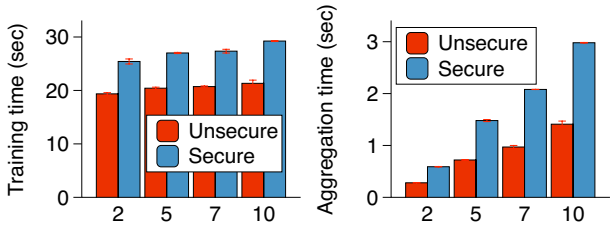
Fig. 5b illustrates the runtime breakdown of unsecured workflow and secured workflow using the VGG-7 model. Similar to the LeNet breakdown, the main components of the total runtime in VGG-7 are the edge computing time and the communication time. The edge computing time accounts for, on average, 96.9% of the total runtime. This percentage is higher than that of LeNet since VGG-7 is more complex than LeNet. The edge computing time also contributes the most to the runtime overhead. The overhead introduced by secured computations is 0.75% for one layer layer placed in TrustZone and up to 14% for 5 layers in TrustZone. The complexity of the VGG-7 model also shows in the number of weights in the model and incurs greater communication time (0.42 seconds) than that of LeNet (0.34 seconds).

6.4 Resource Utilization

To answer the third question, *What is the workload on the workers?*, we measure the CPU and memory utilization on the workers. We calculate the CPU utilization for the edge worker by extracting the relevant information from the /proc filesystem. Due to the security isolation between normal world (REE) and secure world (TEE), the utilization of the secure world cannot be directly accessed by the /proc filesystem. Therefore, we first determine the total time spent for the training process, and then we determine the time elapsed for the process. The CPU utilization is the percentage of time that is spent on the process of interest. *total_time = utime + stime*, where *utime* is the amount of time that this process has been scheduled in user mode, measured in clock ticks, and *stime* is the amount of time that this process has been scheduled in kernel mode, measured in clock ticks. The time the CPU spends in secure world is part of stime. Next, we get the total elapsed time in seconds since the process started, using the following equation: *seconds = uptime - (starttime / HZ)*. Finally, we calculate the CPU usage in percentage as follows: *cpu_usage = 100 * ((total_time / HZ) / seconds)*

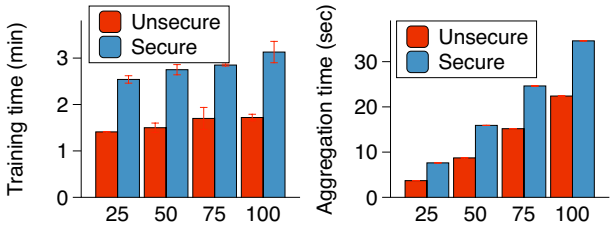
Fig. 6a illustrates the CPU utilization of an edge worker when training the LeNet model. When there is only one layer in TrustZone, the CPU utilization is about 51%. When there are five layers in TrustZone, the CPU utilization is 61%. The increase in the CPU utilization is due to each layer being placed in trusted memory. SecureFL encrypts the layer weights before passing the weights from normal world to secure world. The more weights in the secure world, the more encryption the CPU needs to perform, and hence the higher utilization.

Fig. 6b shows the memory utilization of an edge worker during training. We measured the memory utilization (resident set size (RSS)) of the training process in the normal world by extracting data from the /proc filesystem. The secure world memory utilization



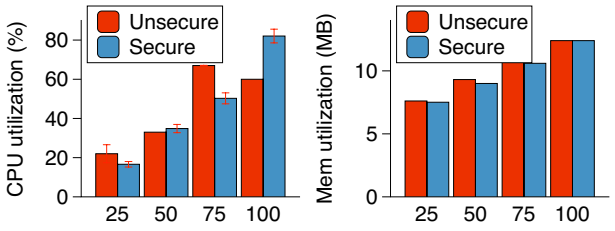
(a) Training time. (b) Aggregation time.

Figure 8: Security overhead with various number of Raspberry Pi workers. Each Raspberry Pi worker keeps only the last layer in the TrustZone. The training time of the secured workflow only incurs marginal overhead with the increase of the participating edge workers.



(a) Training time. (b) Aggregation time.

Figure 9: Security overhead with various number of QEMU workers. With 25 workers in the federated learning system, the training time of the secured workflow is 152 seconds. When there are 50 participating workers, the training time slightly increases to 165 seconds. With a 2X increase in the number of the worker, the increase in the training time is only 8%.



(a) CPU utilization. (b) Memory utilization.

Figure 10: Resource utilization of the aggregator server when handling different numbers of workers. We observed that both the CPU and the memory are not fully utilized. When there are 100 workers, the CPU utilization is around 70% and the memory utilization is about 12 MB.

cannot be directly measured in the /proc filesystem. The normal world OS is not allowed to access this information of the secure world to prevent side-channel attacks [44]. To measure the memory utilization of the secure world, we make use of the abort dump [34] and reference the call stack, similar to the method proposed by previous work [30]. In the call stack dump, we calculate the sizes of the different memory regions. This estimates the amount of memory utilized when training differing model sizes.

Fig. 7 shows the CPU and memory utilization of an edge worker when training a VGG-7 model. The total memory utilization consists of the memory used in REE and the memory used in TEE. When all of the layers are calculated in the normal world, there is no memory used for TEE. When a few layers are moved into TrustZone, the TEE memory is around 2 MB. The TEE memory remains almost the same with different numbers of layers in TrustZone. The TEE runtime allocates a fixed mapping size for secure memory [30]. Placing more layers in TrustZone has no effect on the secure memory size. The CPU utilization slightly increases with each additional layer placed into TrustZone. This is due to the data transfer between worlds and the encryption/decryption in the secure world. We conclude that the workload on the edge is consistent in memory usage and increases in CPU utilization with additional secured layers.

6.5 Scalability

In this section, we answer the question *How scalable is our approach?*. We evaluated SecureFL using both real devices and scaled QEMU workers. Scaling the number of workers also allows us to gain insights into the question: *What is the workload on the aggregation server?*

6.5.1 Raspberry Pi Cluster. We investigate the runtime overhead when scaling the number of Raspberry Pi workers. This gives a good understanding of the performance overhead and scalability using real worker devices.

Fig. 8a shows the total training time of all workers when different numbers of edge workers participate in federated learning. Each Raspberry Pi worker keeps only the last layer in the TrustZone. The training time of the secured workflow only incurs marginal overhead with the increase of the participating edge workers. When there are two edge workers participating in federated learning, the training time is 37.82 seconds. The training time only increases by 6% and 14% when there are five edge workers and ten edge workers, respectively. Comparing the secure and unsecured workflow for ten edge workers, the time overhead introduced by the security mechanism is 5.02 seconds, which is about 13% of the unsecured training time. The overhead comes from data transfer between the non-EPC and EPC memory regions, and the encryption. Note that the results in Fig. 3b involve only one worker; the security overhead above consists of multiple workers.

The aggregator is centralized in our system. We investigated the aggregation runtime and the potential bottleneck. Fig. 8b illustrates the aggregation time with a varying number of edge workers. Since all of the workers share a single SGX-enabled aggregator server, a larger number of edge workers can introduce additional overhead to the server compared to only running with one worker. The aggregation time measures how long it takes to perform averaging among all of the model weights received from all of the workers. Since each worker sends all the model weights to the aggregation server, the size of the weights is proportional to the number of participating workers. When there are two participating edge workers, the secured aggregation time is 0.65 seconds, on average. This is

around 0.35 seconds slower than the unsecured workflow. If we increase the number of workers to 5, the secured aggregation time increases from 0.65 seconds to 1.58 seconds and the unsecured aggregation time is 0.71 seconds. If we further increase the number of workers to 10, the secured aggregation time increases to 3.16 seconds and the unsecured aggregation time is 1.425 seconds. Increasing from two workers to ten workers, the increase in the secured aggregation time is only 2.52 seconds. With a total training time of 43.35 seconds using 10 workers, the overhead from using the security mechanisms accounts for only 5.8%. When there are more workers, SecureFL needs to copy more data into the enclave and perform more decryption and encryption inside the enclave.

6.5.2 QEMU Workers. In order to better evaluate the scalability of our secure federated learning system, we need a test environment with more than ten workers. Using QEMU, we can increase the number of participating workers in the following experiment to a total of 100. Fig. 9 shows the training time with different numbers of QEMU workers. With 25 workers in the federated learning system, the training time of the secured workflow is 152 seconds. When there are 50 participating workers, the training time slightly increases to 165 seconds. With a 2X increase in the number of the worker, the increase in the training time is only 8%. We further increase the number of workers to 75 and 100. Training time of these two cases increases by 12% and 23.6%, respectively, compared to the result of 25 workers. With 4X more workers in the training, the training time overhead increases by 23.6%. This result further illustrates that our federated learning system has reasonable scalability overhead given the security benefits.

We then measure the aggregation time to understand the aggregation overhead when scaling. The secured aggregation time of 25 workers is 8.15 seconds, and that of 100 workers is 36.97 seconds. The aggregation time increases by 4.5X from 25 workers to 100 workers. The aggregation time measures the execution of the entire SGX function. This includes non-SGX to SGX CPU mode switches, copying all data to and from EPC, and performing computations in SGX. The amount of work is linearly proportional to the number of workers participating in the training.

The aggregation time of the unsecured workflow is lower than that of the secured workflow, but the scaling overhead is similar. The aggregation time of 25 workers is 3.74 seconds, which is only around 46% of the secured aggregation time with the same number of workers. When there are 100 workers, the unsecured aggregation time increases by 5.5X. Both the unsecured and the secured aggregation time increase linearly with the increase in the number of workers.

Since the aggregation server can become a single point bottleneck of our system, we investigate the resource utilization to determine if the aggregation server is overwhelmed by the number of workers. We focus on CPU and memory utilization.

Fig. 10a shows the CPU utilization of the aggregator when handling different numbers of workers. When there are 25 workers, the CPU utilization for both unsecured and secured workflow is relatively low, only around 20%. When the number of workers increases to 50, the CPU utilization increases to around 34%. We observe a similar pattern with even more workers. The aggregator CPU is much more engaging when handling the aggregation workload of 100 workers. The utilization increased to around 70%. In particular, the utilization of unsecured workflow is 60% and the secured workflow is 82%. However, the CPU is still not fully utilized even handling such a large amount of workers. For a quad-core CPU, the utilization number is 400% when fully utilized. The CPU is only 82% utilized because the calculation of weight averages is quite efficient on modern CPUs.

Fig. 10b shows the memory utilization of the aggregator when handling different numbers of workers. We obtain the RSS for the process to monitor the memory utilization of the aggregator while performing the tests. The memory utilization of the aggregator is quite low, even when there are 100 workers. The total of all models only reaches approximately 12 MB, showing that the memory of the aggregation server is far from overloaded. We conclude that the workload on the aggregator server is low, even in the case of 100 workers.

7 CONCLUSIONS

This paper presents SecureFL, a secure federated learning framework that uses both SGX and TrustZone to defend against information disclosure in the cloud and membership inference attacks on the cloud and edge workers. It is a hardware secured end-to-end federated learning framework that protects both the aggregator and edge workers in practical settings. We show that our implementation is feasible using widely available hardware to provide privacy to the federated learning framework. Our approach is orthogonal to other federated learning security techniques, allowing additional security techniques to be implemented on top of SecureFL.

We present a rigorous security analysis that proves SecureFL can effectively protect user privacy against HBC adversaries on cloud and edge resources. We provide an evaluation of our framework using real edge worker devices and show a quantitative analysis of the impact of enabling security features on the performance of federated learning. SecureFL demonstrates reasonable overhead given the provided security benefits. The overhead is 14% for a system with ten Raspberry Pi based workers, and 23.6% for a system with 100 QEMU-based workers. We believe that SecureFL can better protect user privacy and allow users to adopt more intelligent applications powered by federated learning.

8 ACKNOWLEDGEMENT

This research is sponsored by U.S. National Science Foundation awards CNS-1629888, IIS-1633381, and CNS-1955593. We thank the anonymous reviewers for their helpful suggestions.

REFERENCES

[1] Naman Agarwal, Ananda Theertha Suresh, Felix Xinnan X Yu, Sanjiv Kumar, and Brendan McMahan. 2018. cpSGD: Communication-efficient and differentially-private distributed SGD. In *Advances in Neural Information Processing Systems*. 7564–7575.

[2] Julien Amacher and Valerio Schiavoni. 2019. On the performance of ARM TrustZone. In *IFIP International Conference on Distributed Applications and Interoperable Systems*. Springer, 133–151.

[3] Ittai Anati, Shay Gueron, Simon Johnson, and Vincent Scarlata. 2013. Innovative technology for CPU based attestation and sealing. In *Proceedings of the 2nd international workshop on hardware and architectural support for security and privacy*, Vol. 13. 7.

[4] 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).

[5] 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 (WOOT 17)*.

[6] Guoxing Chen, Sanchuan Chen, Yuan Xiao, Yinqian Zhang, Zhiqiang Lin, and Ten H Lai. 2018. SGXpectre attacks: Leaking enclave secrets via speculative execution. *arXiv preprint arXiv:1802.09085* (2018).

[7] Hong-You Chen and Wei-Lun Chao. 2020. FedBE: Making bayesian model ensemble applicable to federated learning. (2020).

[8] Mingqing Chen, Rajiv Mathews, Tom Ouyang, and Françoise Beaufays. 2019. Federated learning of out-of-vocabulary words. *arXiv preprint arXiv:1903.10635* (2019).

[9] Yitao Chen, Saman Biookaghazadeh, and Ming Zhao. 2019. Exploring the capabilities of mobile devices in supporting deep learning. In *Proceedings of the 4th ACM/IEEE Symposium on Edge Computing*. 127–138.

[10] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* 2016, 86 (2016), 1–118.

[11] John R Douceur. 2002. The sybil attack. In *International workshop on peer-to-peer systems*. Springer, 251–260.

[12] Matt Fredrikson, Somesh Jha, and Thomas Ristenpart. 2015. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. 1322–1333.

[13] google. 2019. Your chats stay private while Messages improves suggestions. https://support.google.com/messages/answer/9327902l.

[14] Zhongshu Gu, Heqing Huang, Jialong Zhang, Dong Su, Hani Jamjoom, Ankita Lamba, Dimitrios Pendarakis, and Ian Molloy. 2018. YerbaBuena: Securing Deep Learning Inference Data via Enclave-based Ternary Model Partitioning. *arXiv preprint arXiv:1807.00969* (2018).

[15] 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).

[16] ARM Holding. 2009. ARM Security Technology, Building a Secure System using TrustZone Technology.

[17] Peter Kairouz, H Brendan McMahan, Brendan Avent, Aurélien Bellet, Mehdi Bennis, Arjun Nitin Bhagoji, Keith Bonawitz, Zachary Charles, Graham Cormode, Rachel Cummings, et al. 2019. Advances and open problems in federated learning. *arXiv preprint arXiv:1912.04977* (2019).

[18] Paul Kocher, Jann Horn, Anders Fogh, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, et al. 2019. Spectre attacks: Exploiting speculative execution. In *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE, 1–19.

[19] Alex Krizhevsky and Geoffrey Hinton. 2009. Learning multiple layers of features from tiny images. (2009).

[20] Ya Le and Xuan Yang. 2015. Tiny ImageNet visual recognition challenge. *CS 231N* 7 (2015), 7.

[21] Yann LeCun et al. 2015. LeNet-5, convolutional neural networks. *URL: http://yann. lecun. com/exdb/lenet* 20, 5 (2015), 14.

[22] Yann LeCun, Corinna Cortes, and CJ Burges. 2010. MNIST handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist* 2 (2010).

[23] Tao Lin, Lingjing Kong, Sebastian U Stich, and Martin Jaggi. 2020. Ensemble Distillation for Robust Model Fusion in Federated Learning. In *Advances in Neural Information Processing Systems*.

[24] Yehida Lindell. 2005. Secure multiparty computation for privacy preserving data mining. In *Encyclopedia of Data Warehousing and Mining*. IGI Global, 1005–1009.

[25] 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. *Hasp@ isca* 10, 1 (2013).

[26] 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.

[27] H Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, et al. 2016. Communication-efficient learning of deep networks from decentralized data. *arXiv preprint arXiv:1602.05629* (2016).

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

[29] 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 (SP)*. IEEE, 691–706.

[30] Fan Mo, Ali Shahin Shamsabadi, Kleomenis Katevas, Soteris Demetriou, Ilias Leontiadis, Andrea Cavallaro, and Hamed Haddadi. 2020. DarkneTZ: towards model privacy at the edge using trusted execution environments. In *Proceedings of the 18th International Conference on Mobile Systems, Applications, and Services*. 161–174.

[31] Milad Nasr, Reza Shokri, and Amir Houmansadr. 2018. Comprehensive privacy analysis of deep learning: Stand-alone and federated learning under passive and active white-box inference attacks. *arXiv preprint arXiv:1812.00910* (2018).

[32] Oleksii Oleksenko, Bohdan Trach, Robert Krahn, Mark Silberstein, and Christof Fetzer. 2018. Varys: Protecting SGX enclaves from practical side-channel attacks. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*. 227–240.

[33] OP-TEE. 2017. OP-TEE Trusted OS Documentation. https://www.op-tee.org/.

[34] OP-TEE. 2019. OP-TEE Abort dumps. https://optee.readthedocs.io/en/latest/debug/abort_dumps.html.

[35] OP-TEE. 2020. OP-TEE Documentation. https://https://readthedocs.org/projects/optee/downloads/pdf/latest/.

[36] AJ Paverd, Andrew Martin, and Ian Brown. 2014. Modelling and automatically analysing privacy properties for honest-but-curious adversaries. *Tech. Rep.* (2014).

[37] 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).

[38] Joseph Redmon. 2013–2016. Darknet: Open Source Neural Networks in C. http://pjreddie.com/darknet/.

[39] Samsung. 2021. Knox White Paper. https://docs.samsungknox.com/admin/whitepaper/kpe/samsung-knox.htm.

[40] 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.

[41] Ravid Shwartz-Ziv and Naftali Tishby. 2017. Opening the black box of deep neural networks via information. *arXiv preprint arXiv:1703.00810* (2017).

[42] Amol Vasudeva and Manu Sood. 2018. Survey on sybil attack defense mechanisms in wireless ad hoc networks. *Journal of Network and Computer Applications* 120 (2018), 78–118.

[43] Hongyi Wang, Mikhail Yurochkin, Yuekai Sun, Dimitris Papailiopoulos, and Yasaman Khazaeni. 2020. Federated Learning with Matched Averaging. In *International Conference on Learning Representations*. https://openreview.net/forum?id=BkluqlSFDS

[44] Wenhao Wang, Guoxing Chen, Xiaorui Pan, Yinqian Zhang, XiaoFeng Wang, Vincent Bindschaedler, Haixu Tang, and Carl A Gunter. 2017. Leaky cauldron on the dark land: Understanding memory side-channel hazards in SGX. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. 2421–2434.

[45] Timothy Yang, Galen Andrew, Hubert Eichner, Haicheng Sun, Wei Li, Nicholas Kong, Daniel Ramage, and Françoise Beaufays. 2018. Applied federated learning: Improving google keyboard query suggestions. *arXiv preprint arXiv:1812.02903* (2018).

[46] Seehwan Yoo, Hyunik Kim, and Joongheon Kim. 2018. Secure compute-vm: Secure big data processing with SGX and compute accelerators. In *Proceedings of the 3rd Workshop on System Software for Trusted Execution*. 34–36.

[47] Jason Yosinski, Jeff Clune, Anh Nguyen, Thomas Fuchs, and Hod Lipson. 2015. Understanding neural networks through deep visualization. *arXiv preprint arXiv:1506.06579* (2015).

[48] Matthew D Zeiler and Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *European conference on computer vision*. Springer, 818–833.

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