SECURETF: A Secure TensorFlow Framework

Do Le Quoc, Franz Gregor, Sergei Arnautov
TU Dresden, Scontain UG

Roland Kunkel
TU Dresden

Pramod Bhatotia
TU Munich

Christof Fetzer
TU Dresden, Scontain UG

Abstract

Data-driven intelligent applications in modern online services have become ubiquitous. These applications are usually hosted in the untrusted cloud computing infrastructure. This poses significant security risks since these applications rely on applying machine learning algorithms on large datasets which may contain private and sensitive information.

To tackle this challenge, we designed SECURETF, a distributed secure machine learning framework based on Tensorflow for the untrusted cloud infrastructure. SECURETF is a generic platform to support unmodified TensorFlow applications, while providing end-to-end security for the input data, ML model, and application code. SECURETF is built from ground-up based on the security properties provided by Trusted Execution Environments (TEEs). However, *it extends the trust of a volatile memory region (or* secure enclave*) provided by the single node TEE to secure a distributed infrastructure required for supporting unmodified stateful machine learning applications running in the cloud*.

The paper reports on our experiences about the system design choices and the system deployment in production use-cases. We conclude with the lessons learned based on the limitations of our commercially available platform, and discuss open research problems for the future work.

Keywords: secure machine learning, confidential computing, intel software guard extensions (Intel SGX), tensorflow

1 Introduction

Machine learning has become an increasingly popular approach for solving various practical problems in data-driven online services [5, 19, 33, 83]. While these learning techniques based on private data *arguably* provide useful online services, they also pose serious security threats for the users. Especially, when these modern online services use the third-party untrusted cloud infrastructure for deploying these computations.

In the untrusted computing infrastructure, an attacker can compromise the confidentiality and integrity of the computation. Therefore, the risk of security violations in untrusted infrastructure has increased significantly in the third-party cloud computing infrastructure [74]. In fact, many studies show that software bugs, configuration errors, and security vulnerabilities pose a serious threat to computations in the cloud systems [16, 41, 75]. Furthermore, since the data is stored outside the control of the data owner, the third-party cloud platform provides an additional attack vector. The clients currently have limited support to verify whether the third-party operator, even with good intentions, can handle the data with the stated security guarantees [52, 91].

To overcome the security risks in the cloud, our work focuses on securing machine learning computations in the untrusted computing infrastructure. In this context, the existing techniques to secure machine learning applications are limiting in performance [38], trade accuracy for security [32] or support only data classification [21]. Therefore, *we want to build a secure machine learning framework that supports existing applications while retaining accuracy, supporting both training and classification, and without compromising the performance. Furthermore, our work strives to provide end-to-end security properties for the input data, ML models, and application code.*

To achieve our design goals, trusted execution environments (TEEs), such as Intel SGX [28] or ARM TrustZone [11], provide an appealing way to build a secure machine learning system. In fact, given the importance of security threats in the cloud, there is a recent surge in leveraging TEEs for shielded execution of applications in the untrusted infrastructure [12, 16, 69, 78, 90]. *Shielded execution* aims to provide strong confidentiality and integrity properties for applications using a hardware-protected secure memory region or *enclave*.

While these shielded execution frameworks provide strong security guarantees against a powerful adversary, the TEEs have been designed to secure single-node in-memory (volatile) computations. Unfortunately, the trust of TEEs does not naturally extend to support distributed stateful applications running in the cloud. To build a secure machine learning framework that supports both training and classification phases, while providing all three important design properties: *transparency*, *accuracy*, and *performance*, we need to address several architectural challenges presented by TEEs, specifically Intel SGX, which acts as the root of trust.

More specifically, in addition to the conventional architectural challenges posed by the SGX architecture in the single node setting, such as limited enclave memory and I/O bottlenecks, we need to address the following three important challenges in the context of distributed cloud computing:

Firstly, we need to extend the trust of SGX to support the distributed TensorFlow framework, where the worker nodes are running in the remote distributed enclaves while ensuring that they execute correct code/computations and

data. However, this is a challenging task since Intel SGX is designed for securing single machine computations.

Secondly, we need to support practical features offered by the virtualized platforms in the public cloud service to enable *elastic and fault-tolerant computing*, i.e., scaling-up/down based on the workloads, and dealing with failures/migrations. To support these important requirements, we need to ensure the new worker node running in a container preserves the integrity, confidentiality of the data, ML models, and application code. However, the traditional remote attestation using the Intel Attestation Service (IAS) [30] is impractical to support the elastic and fault-tolerant computing. Therefore, we need to redesign the mechanism to ensure an elastic trust establishment through a configuration and attestation service.

Lastly, we need to support stateful machine learning applications that rely on reading the input data or write computation results from/to a file system storage as well as to the network. Unfortunately, Intel SGX is designed to protect only the data and computation residing in the volatile enclave memory. It does not provide any security guarantees for stateful machine learning computations across multiple machines.

To overcome these design challenges, we present SECURETF, a secure machine learning framework for the untrusted infrastructure. More specifically, we make the following contributions.

- We have designed and implemented SECURETF as the end-to-end system based on TensorFlow that allows secure execution of the existing unmodified TensorFlow applications without compromising the accuracy.
- We optimized the performance to overcome the architectural limitation of Intel SGX in the context of machine learning workloads for distributed untrusted cloud computing environments.
- We report an extensive evaluation of SECURETF based on microbenchmarks and production use-cases. Our evaluation shows that SECURETF achieves reasonable performance overheads, while providing strong security with low TCB.

SECURETF is a commercially available platform, and it is currently used in production by four major customers. In this paper, we report on our experiences on building SECURETF and deploying it in two production use-cases. We conclude the paper with the lessons learned based on the limitations of our system design, and a discussion on open research problems for the future work.

2 Background and Threat Model

2.1 Machine Learning using TensorFlow

Machine learning aims to automatically extract useful patterns in large-scale data by building probabilistic models [79]. Machine learning approaches are often categorized into supervised, unsupervised and reinforcement learning. All forms have in common that they require datasets, a defined objective, a model and a mechanism to update the model according to new inputs.

To generalize the machine learning approach for masses, Google proposed TensorFlow [8] as a machine learning framework designed for heterogeneous distributed systems. TensorFlow requires the user first to define a directed graph consisting of nodes representing operations on incoming data. Nodes have zero or more inputs and outputs and perform operations on different levels of abstraction such as matrix multiplication, pooling or reading data from disk. Nodes can also have an internal state, depending on their type. Thus the whole graph can be stateful as well.

After defining the graph, the user can perform calculations by starting a session and running the previously defined operations. TensorFlow uses a flow model for calculations.

Through the division of the calculation in the graph into nodes, TensorFlow makes it easy to distribute the execution across different devices. Therefore, TensorFlow can be deployed on mobile devices, single personal computers, as well as computer clusters, by mapping the computation graph on available hardware.

TensorFlow Lite [7] is a feature-reduced version of TensorFlow, designed for mobile and embedded devices. Optimization for mobile devices is achieved by running a mobile-optimized interpreter that keeps the load at a lower level and having the overall binary size smaller when compared to full TensorFlow. The number of available operations for defining a graph is reduced to achieve a smaller memory footprint of the resulting binary. This comes at the cost of trainability of the graph, because TensorFlow Lite can only perform forward passes in graphs. Instead, a model must first be training with the full version of TensorFlow and then exported and converted to a special TensorFlow Lite model format. This format can then be used from the TensorFlow Lite API for inference.

2.2 Intel SGX and Shielded Execution

Intel Software Guard Extension (SGX) is a set of x86 ISA extensions for Trusted Execution Environment (TEE) [30]. SGX provides an abstraction of a secure *enclave*—a hardware-protected memory region for which the CPU guarantees the confidentiality and integrity of the data and code residing in the enclave memory. The enclave memory is located in the Enclave Page Cache (EPC)—a dedicated memory region protected by an on-chip Memory Encryption Engine (MEE). The MEE encrypts and decrypts cache lines that are written and read to EPC, respectively. Intel SGX supports a call-gate mechanism to control entry and exit into the TEE.

Shielded execution based on Intel SGX aims to provide strong confidentiality and integrity guarantees for applications deployed on an untrusted computing infrastructure [12, 16, 69, 78, 90]. Our work builds on the SCONE [12] shielded

execution framework. In the SCONE framework, the applications are linked against a modified standard C library (SCONE libc). In this model, the application's address space is confined to the enclave memory, and interaction with the untrusted memory is performed via the system call interface. In particular, SCONE runtime provides an *asynchronous system call* mechanism [80] in which threads outside the enclave asynchronously execute the system calls. Lastly, SCONE provides an integration to Docker for seamlessly deploying container images.

2.3 Threat Model

We aim to protect against a very powerful adversary even in the presence of complex virtualization stacks in the cloud computing infrastructure [16]. In this setting, the adversary can control the entire system software stack, including the OS or the hypervisor, and is able to launch physical attacks, such as performing memory probes. In addition, we consider an untrusted network in the cloud environment, i.e., the adversary can drop, inject, replay, alter packages, or manipulate the routing of packages. This network model is consistent with the classic Dolev-Yao adversary model [31]. Even under this extreme threat model, our goal is to guarantee the integrity, confidentiality, and freshness of data, code (e.g., Python code), and models of machine learning computation. We also provide bindings with Pesos [52], a secure storage system to protect against rollback attacks [71] on the data stored beyond the secure enclave memory. Our system is adaptable with SGXBounds [56]; therefore, SECURETF is resilient to memory safety vulnerabilities [66].

However, we do not protect against side-channel attacks based on cache timing and speculative execution [92], and memory access patterns [42, 98]. Mitigating side-channel attacks is an active area of research [67]. We do not consider denial of service attacks since these attacks are trivial for a third-party operator controlling the underlying infrastructure [16], e.g., operating system (OS), and hypervisor. Lastly, we assume that the CPU hardware (including its implementation of SGX) are trusted and the adversary cannot physically open the processor packaging to extract secrets or corrupt the CPU system state.

3 Design

In this section, we present the design of SECURETF.

3.1 System Overview

SECURETF is designed for secure distributed machine learning computations using the hardware-assisted trusted execution environment (TEE) technologies such as Intel SGX. Figure 1 depicts the high-level architecture of SECURETF. Our system ensures not only the confidentiality, integrity and freshness of executions (e.g., training and classifying computations) but also the input data and machine learning models.

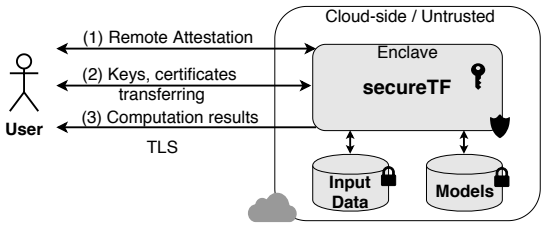

Figure 1. System overview.

At a high-level, the system works as follows: at the first step, when a user deploys a machine learning computation on a remote host (e.g., a public cloud), the user needs to establish trust into the SECURETF instance running in the untrusted environment. To do so, the user performs the remote attestation mechanism provided by the TEE technology to ensure that the computation and the input data deployed in the remote environment are correct and not modified by anyone e.g., an attacker. After trusting the SECURETF running in the remote environment, the user provides secrets including keys for encrypting/decrypting input and output data (e.g., input images and models, certificates for TLS connections), to the machine learning platform. After finishing the computation, SECURETF returns the results back to the user via a TLS connection.

Design goals. Our primary design goal is to achieve strong confidentiality and integrity properties. By confidentiality, we mean that all data including models handled by the machine learning framework and the machine learning framework code itself may not be disclosed to or obtainable by an unauthorized party. By integrity, we mean that modifications of the data handled by SECURETF that were done by an unauthorized party must be detectable and should not compromise the internal state and functioning. In addition, while designing a practical system, we aim to achieve the following goals.

- *Transparency:* The secure framework must offer the same interface as the unprotected framework, and should run unmodified existing applications based on TensorFlow.
- *Performance:* We aim to impose as little overhead as possible when adding security to the machine learning framework.
- *Accuracy:* We do not aim to trade-off accuracy for security. Accuracy will be the same in the native TensorFlow framework as when using no security protection.

3.2 Design Challenges

Building a practical secure distributed machine learning system using TEEs such as Intel SGX is not straightforward, in fact, we need to handle several challenges.

❶ **Code modification.** Intel SGX requires users to heavily modify the source code of their application to run inside

enclaves. Thus, transparently supporting an unmodified machine learning framework to run inside enclaves is not a trivial task.

❷ **Limited EPC size.** Currently, Intel SGX supports only a limited memory space (~ 94MB) for applications running inside enclaves. However, most machine learning computations, especially training, are extremely memory-intensive.

❸ **Establishing the trust in a distributed system.** Trust has to be established in the remote distributed enclaves to ensure that they execute correct code and data. However, this is a challenging task since Intel SGX is originally designed for a single machine.

❹ **Elastic and fault tolerant computing support.** Typically, public cloud services support *elastic computing*, i.e., when the input workload increases, the framework automatically spawns new service containers or instances to handle with the growth of requests. However, whenever spawning a new container, it requires to perform remote attestation to ensure the integrity, confidentiality of the machine learning application in that container before communicating with it. Unfortunately, the traditional attestation mechanism using the Intel Attestation Service (IAS) [30] incurs significant overhead, thus it's impractical in this setting.

❺ **Stateful computing: security of network and file system.** Machine learning applications running inside SGX enclaves need to read input data or write results from/to a file system, storage systems, or network. Unfortunately, Intel SGX is designed to protect only the stateless in-memory data and computation residing inside enclaves. It does not provide any security guarantees for state stateful machine learning computations across multiple machines.

3.3 System Design

In this section, we present the detailed design of distributed SECURETF that handles the aforementioned challenges in §1.

3.3.1 System Components. To overcome the challenge ❶ (see §1), we built SECURETF based on the SCONE shielded execution framework [12]. SCONE enables legacy applications to be executed in Intel SGX enclaves without source code changes. While there are other options available, we choose SCONE, because of the relatively small extra work required to run an application and comparatively small overhead compared to other available options. We leverage SCONE's Docker container support to design secure distributed SECURETF which allows users to perform machine learning computations in a secure manner on an untrusted environment such as a public cloud. Figure 2 shows the distributed architecture of SECURETF. At the high-level, our systems consist of four core components: *Configuration and Remote Attestation Service (CAS), secure machine learning containers* including Tensorflow parameter servers and Tensorflow

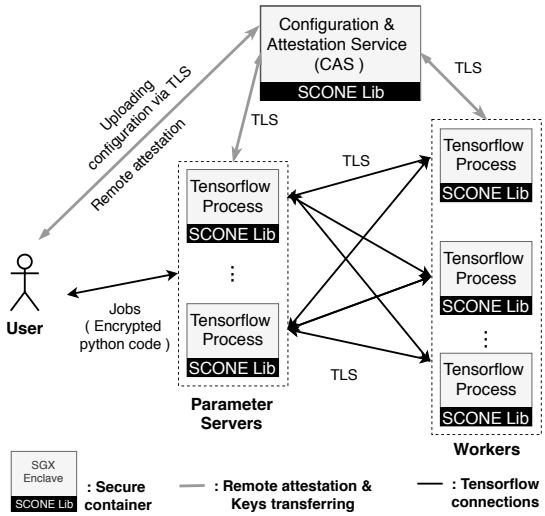

Figure 2. The distributed architecture of SECURETF.

workers, *network shield* and *file system shield*, and *adapted Tensorflow library*.

We design the CAS component to handle the challenges ❸ and ❹. This component takes an important role in the distributed architecture of SECURETF which transparently and automatically performs the remote attestation for secure machine learning containers before transferring secrets and configuration needed to run them. The CAS component is deployed inside an SGX enclave of an untrusted server in the cloud or on a trusted server under the control of the user. When a secure machine learning container is launched, it receives the necessary secrets and configuration from CAS after the remote attestation process, to run machine learning computations using the adapted Tensorflow library running inside an enclave.

We design the network shield and the file system shield components to address the challenge ❺. All communications between secure machine learning containers and the CAS component are protected using the network shield component.

Next, we provide the detailed design of each component.

3.3.2 Configuration and Remote Attestation Service. CAS enhances the Intel attestation service [30] to bootstrap and establish trust across the SECURETF containers and maintain a secure configuration of the distributed SECURETF framework. CAS itself is deployed in an Intel SGX enclave. In the case that CAS is deployed in an enclave on an untrusted server, the user of SECURETF needs to establish trust into the CAS instance, i.e., he/she needs to perform remote attestation of CAS before transferring encryption keys and certificates to process the encrypted input data and machine learning models. By using CAS, we can maintain the original distributed architecture of Tensorflow machine learning framework.

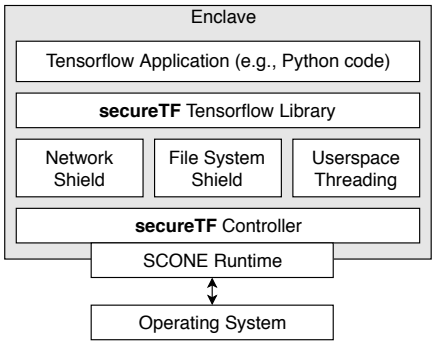

Figure 3. The architecture of a secure machine learning container in SECURETF.

In addition, to guarantee the freshness of data during runtime, we design and implement an auditing service in CAS to keep track the data modification during machine learning computation. This mechanism allows SECURETF to protect against rollback attacks.

3.3.3 Secure Machine Learning Containers. To build secure machine learning containers, we make use of TensorFlow and TensorFlow Lite. TensorFlow Lite has the additional advantage of having a smaller memory footprint which helps us to handle the design challenge ❷. We use SCONE [12] as an additional layer that allows access to SGX features with fewer changes to application code. Figure 3 presents the general architecture of a secure Tensorflow container using SCONE.

Since we build secure machine learning containers based on SCONE in SECURETF, we use Docker [61] to conveniently deploy our system. No changes to the Docker engine is required. The design of a secure machine learning container in SECURETF is composed of two components: (a) the SECURETF controller that provides the necessary runtime environment for securing the TensorFlow library, and (b) SECURETF TensorFlow library that allows deploying unmodified existing TensorFlow applications. We next describe these two components in detail.

SECURETF Controller. The SECURETF controller is based on the SCONE runtime. Inside the SGX enclave, the controller provides a runtime environment for TensorFlow, which includes the *network shield*, the *file system shield*, and the *user-level threading*. Data, that is handled through file descriptors, is transparently encrypted and authenticated through the shields. The shields apply at each location where an application would usually trust the operating system, such as when using sockets or writing files to disk. The shields perform sanity checks on data passed from operating system to enclave to prevent Iago attacks [24]. More specifically, these checks include bound checks and checking for manipulated pointers. This protection is required to fulfill the goal of not requiring the application to deal with untrusted systems (see challenge ❶ in §1).

File system shield. The file system shield protects confidentiality and integrity of data files. Whenever the application would write a file, the shield either encrypts and authenticates, simply authenticates or passes the file as is. The choice depends on user-defined path prefixes, which are part of the configuration of an enclave. The shield splits files into chunks that are then handled separately. Metadata for these chunks is kept inside the enclave, meaning it is protected from manipulation. The secrets used for these operations are different from the secrets used by the SGX implementation. They are instead configuration parameters at the startup time of the enclave.

Network shield. TensorFlow applications do not inherently include end-to-end encryption for network traffic. Users who want to add security must apply other means to secure the traffic, such as a proxy for the Transport Layer Security (TLS) protocol. According to the threat model however, data may not leave the enclave unprotected, because the system software is not trusted. Network communication must therefore always be end-to-end protected. Our network shield wraps sockets, and all data passed to a socket will be processed by the network shield before given to system software. The shield then transparently wraps the communication channel in a TLS connection on behalf of the user application. The keys for TLS are saved in files and protected by the file system shield.

User-level threading. Enclave transitions are costly and should therefore be avoided when possible. Many system calls require a thread to exit userspace and enter kernel space for processing. To avoid thread transitions out of enclaves as much as possible, the controller implements user space threading.

When the OS assigns a thread to an enclave, it first executes an internal scheduler to decide, which application thread to execute. These application threads are then mapped to SGX thread control structures. When an application thread blocks, the controller is run again to assign the OS thread to a different application thread instead of passing control back to the operating system. In this way, the number of costly enclave transitions is reduced. When no application thread is ready for execution, the OS either backs off and waits inside the enclave, or outside, depending on the time required for an enclave transition. A side effect of this user-level threading scheme is that the controller does not require more OS threads than CPUs available to achieve full CPU utilization, which is usually the case for applications running under a conventional OS.

3.3.4 SECURETF TensorFlow Library. Machine learning applications consist of two major steps. In the first step, the model is trained, and thereafter, the model is employed for classification or inference tasks. Next, we explain the detailed design to run both training process and classification process with Intel SGX.

Training process. For the training process, we use the full version of TensorFlow. Training in TensorFlow is usually performed on acceleration hardware such as GPUs and distributed across multiple machines. However, the SECURETF controller requires SGX which is only available for CPUs. We therefore only support training on *CPU*. This limitation reduces the performance of the training process, but it is required to achieve the security goals.

The SECURETF controller allows easy distribution of the application in the form of Docker images. The training instances of SECURETF can be distributed on multiple nodes, each running separate SGX hardware. The network shield applies transparent protection of the communication channel between instances. Scaling on the same instance, that is, on the same CPU is possible, but does decrease relative performance, because the limiting factor in our environment is EPC size, which is fixed for each CPU. Only horizontal scaling with more nodes can substantially increase performance.

Classification process. The main reason for dividing the classification and training process in our design is that we can use different TensorFlow variants for each step. Running with Intel SGX imposes less overhead, if applications have a smaller memory footprint, because the limited *EPC* size is the major bottleneck (see challenge ❷ in §1). TensorFlow Lite has a smaller memory footprint because it targets mobile devices. The drawback is however that it cannot perform training by design. Therefore, we can only use it for classification or inference. When protecting TensorFlow Lite with SCONE, the framework uses the SCONE C library instead of the common system library. The internals of TensorFlow Lite do otherwise not require change, as the interface of the SCONE C library is compatible. The interface for using the classification method of SECURETF is the same as for TensorFlow Lite. Graph definitions created for TensorFlow Lite are compatible.

4 Implementation

We implement SECURETF based on Tensorflow version 1.9.0 and the SCONE framework [12] to run machine learning computations within Intel SGX enclaves. We also consider other TEEs technologies such as ARM TrustZone [11] and AMD's TEE, SME/SEV [3]. However, they have several limitations, e.g., ARM TrustZone supports only a single secure zone, and does not have any remote attestation mechanism, meanwhile, AMD's TEE does not support integrity protection [63].

We rely on SCONE to implement some features such as file system shield and network shield. However, it is not straightforward to use these features out-of-the-box to build SECURETF. For example, SCONE does not support TLS connection via UDP which is required in Tensorflow. SCONE provides only confidentiality and integrity in network/storage shields, whereas, SECURETF ensures also the freshness of data, code and models of machine learning computation. In addition, the memory management and user-level multithreading need to adapt/extend it to fit the custom scheduler and memory management of TensorFlow framework. Thus, we need to develop these missing parts of these features to implement the design of SECURETF.

In this section, we describe several challenges we faced during implementing SECURETF and how we addressed them. We first present how to enable the security features in SECURETF during the training process (§4.1) and classifying process (§4.2). Thereafter, we describe the implementation of the CAS component in §4.3.

4.1 Training Process

The typical user of TensorFlow uses the Python API for defining and training graphs, because it is the richest API. Using Python with SCONE would impose additional complexity because it requires the **d**ynamic **l**ibrary **open** (`dlopen`) system call for imports. As the name implies, `dlopen` dynamically loads libraries during runtime of a program. However, *SGX* does not allow an enclave to be entered by a thread, unless it has been finalized according to the procedures of enclave creation. A library that is dynamically loaded would therefore not be represented in the enclave's attestation hash. Consequently, `dlopen` is disabled by default for SCONE applications. To allow *dlopen*, we need to change the SCONE environment accordingly (i.e., SCONE_ALLOW_DLOPEN=yes). To ensure the security guarantee, we need to authenticate loaded libraries during runtime using the file system shield (see §3.3).

We support not only Python but also C++ API as native Tensorflow framework. In the previous version of SECURETF, we did not support the Python API since, at that time, SCONE did not support *fork* system call which is required in the Python package [55]. The C++ version covers the low-level API of TensorFlow, meaning many convenience features such as estimators or monitored training are not available. However, implementation using C++ API provides much better performance compared to using Python API. There is one approach that let us use the convenience of the Python API for the definition of the computation graph. TensorFlow allows exporting graphs and parameters, such as learned biases that were created in the current session. Graph definitions and checkpoints containing the parameters can later be imported by another program. Importing and exporting are available in both the C++ and the Python API, and they use interchangeable formats. The user can therefore define a graph with the more high level Python API, including data inputs, and later import and run it with C++. If the application does not by default already export its model with a

named interface, changes are required to the original program, so that either the name of operations in the graph can be known, or an interface is defined.

For the training process, we used the full version of TensorFlow, not to be confused with TensorFlow Lite. A graph definition must be provided by the user in form of a graph *frozen* by a script packaged together with TensorFlow, when using either the Python or C++ API. If the user has used the C++ API for the definition, the full source definition of the graph can also be used.

A frozen graph can be created from a graph definition exported from the Python script that defines the graph in the *Protocol Buffers* ([35]) exchange format. A checkpoint file containing all values of a graph that are not part of the graph definition, such as weights, biases and counters can be exported as well.

Alternatively, the graph can also be exported as a blank slate without any initialized internal values. The initialization can then be done inside the SECURETF environment, which is useful if a user wants to train the graph protected by SGX for the entire training process. The initialization operations are required when using the Python API and are therefore usually part of the exported graph.

The user must also provide the inputs for training, such as a set of annotated images. SECURETF protects the input data and code by activating the file system shield (see §3.3).

4.2 Classification /Inference Process

We implemented our design for inference/classifying computations in SECURETF, by integrating the full Tensorflow with SCONE as we developed for the training computations. In addition, we provide a light-weight version for inference by adapting Tensorflow Lite [7] framework to run with SCONE. We first ensured that Tensorflow and TensorFlow Lite compiles with the *musl* C library [2] on Alpine Linux [1], because SCONE enhanced the *musl* library to support legacy application running with Intel SGX. The *musl* libc is designed to be compatible with The GNU C Library *(glibc)* but more secure with a smaller code base. The issue we faced is that Tensorflow currently uses *Identical code folding* (ICF) [84], which is a compiler or linker feature, to eliminate identical function bodies at compile or link time in order to reduce the binary size. However, it is currently supported by *gcc* and the gold linker, but not by the *musl* linker or the compiler wrapper for *musl*. We therefore removed the *ICF* option for the binary targets in the TensorFlow source tree. Compiling the TensorFlow framework with and without ICF provides similar binary sizes. Therefore, the performance cost when deactivating ICF will also be minimal.

The next issue is that TensorFlow also uses *backtrace* by default. This library is specific for*glibc*. We therefore could not use it directly with musl. To solve this issue, we either recompiled dependencies against the musl libc, or disabled *backtrace* in the building configuration of Tensorflow.

After adapting the Tensorflow source code, compiling it with SCONE is quite straightforward by merely setting the environment variables `CC` and `CXX` to the SCONE C and C++ compilers (i.e., **scone-gcc** and **scone-g++**).

Note that there is no standalone version of TensorFlow Lite available, meaning a user of TensorFlow Lite needs to build their application inside the TensorFlow source folder, with dependency targets set to TensorFlow Lite. Tensorflow uses *Bazel* as a build tool [17], however, Bazel also does not link library targets unless a binary target is created, which means TensorFlow Lite cannot be easily released from the source tree by compiling all libraries, and move them to the system's include directories. Thus, we added compile targets that force linking as a workaround. The libraries could then be moved to other projects along with the header files, and used as third party dependencies. With this, we developed a classifier service from scratch. The service takes classification requests via network, and uses TensorFlow Lite for inference/classifying. For evaluation, we used an example available in the TensorFlow Lite source, which takes its inputs from the hard drive and prints the classification results to console.

4.3 Configuration and Remote Attestation Service

For large-scale deployment SECURETF, we design the Configuration and Remote Attestation Service component (CAS) to transparently perform the remote attestation and transfer keys to distributed SECURETF containers (see §3.3). We implement the CAS component using Rust [59] programming language since it provides strong type safety. To run CAS with Intel SGX, we utilize the SCONE cross compiler to compile our implementation of CAS. We make use of an encrypted embedded SQLite [10] to store encryption keys, certificates, and other secrets for Tensorflow computations (see §3.3). This database itself also runs inside an enclave with the help of the SCONE framework.

To allow a user of CAS can ensure that the code of CAS was not modified and indeed runs inside a valid SGX enclave, besides running CAS with SCONE, we implement CAS in the way that it has zero configuration parameters that can control its behavior. Thus, an attacker with root/privileged accesses cannot break the trust given by the user in CAS. A detail description of CAS regarding protection against rollback attacks and key management is provided in [39].

5 Evaluation

In this section, we present the evaluation results of SECURETF based on both microbenchmarks and macrobenchmarks with real world deployment.

5.1 Experimental Setup

Cluster setup. We used three servers with em SGXv1 support running Ubuntu Linux with a 4.4.0 Linux kernel, equipped

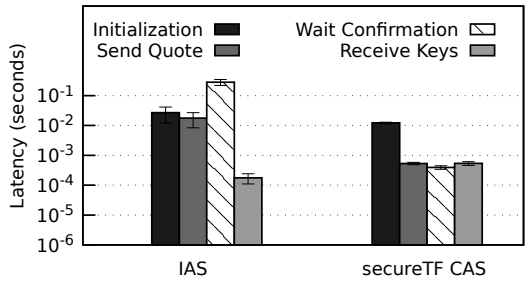

Figure 4. The attestation and keys transferring latency comparison between SECURETF with the traditional way using IAS.

with an Intel© Xeon© *CPU* E3-1280 v6 at 3.90*GHz* with 32 KB L1, 256 KB L2, and 8 MB L3 caches, and 64 *GB* main memory. These machines are connected with each other using a 1 Gb/s switched network. The CPUs update the latest microcode patch level.

In addition, we used a Fujitsu ESPRIMO P957/E90+ desktop machine with an Intel© core i7-6700 CPU with 4 cores at 3.40GHz and 8 hyper-threads (2 per core). Each core has a private 32KB L1 cache and a 256KB L2 cache while all cores share an 8MB L3 cache.

Datasets. We used two real world datasets: *(i)* Cifar-10 image dataset [53] and *(ii)* MNIST handwritten digit dataset [58].

#1: Cifar-10. This dataset contains a labeled subset of a much larger set of small pictures of size 32x32 pixels collected from the Internet. It contains a total of 60,000 pictures. Each picture belongs to one of ten classes, which are evenly distributed, making a total of 6,000 images per class. All labels were manually set by human labelers. Cifar-10 has the distinct advantage that a reasonable good model can be trained in a relatively short time. The set is freely available for research purposes and has been extensively used for benchmarking machine learning techniques [43, 97].

#2: MNIST. The MNIST handwritten digit dataset[58] consists of 60000 28 pixel images for training, and 10000 examples for testing.

Methodology. Before the actual measurements, we warmed up the machine by running at full load with *IO* heavy operations that require swapping of *EPC* pages. We performed measurements for classification and training both with and without the file system shield. For full end-to-end protection, the file system shield was required. We evaluate SECURETF with the two modes: *(i)* hardware mode (HW) which runs with activated TEE hardware and *(ii)* simulation mode (SIM) which runs with simulation without Intel SGX hardware activated. We make use of this SIM mode during the evaluation to evaluate the performance overhead of the Intel SGX and to evaluate SECURETF when the EPC size is getting large enough in the future CPU hardware devices.

5.2 Micro-benchmark: Remote Attestation and Keys Management

In SECURETF, we need to securely transfer certificates and keys to encrypt/decrypt the input data, models and the communication between worker nodes (in distributed training process). To achieve the security goal, we make use of the CAS component (see §3.3) which attests Tensorflow processes running inside enclaves, before transparently provides the keys and certificates to encrypt/decrypt input data, models, and TLS communications. Note that the advantage of using CAS over the traditional way using IAS to perform attestation is that the CAS component is deployed on the local cluster where we deploy SECURETF.

Figure 4 shows the break-down latency in attestation and keys transferring of our component CAS and the method using IAS. The quote verification process in our CAS takes less than 1ms, whereas in the IAS based method is ~ 280ms. In total, our attestation using CAS (~ 17ms) is roughly 19× faster than the traditional attestation using IAS (~ 325ms). This is because the attestation using IAS requires providing and verifying the measured information contained in the quotes [30] which needs several WAN communications to the IAS service.

5.3 Macrobenchmark: Classifying Process

We evaluate the performance of SECURETF in real-world deployments. First, we present the evaluation results of SECURETF in detecting objects in images and classifying images using pre-trained deep learning models. Thereafter, in the next section, we report the performance results of SECURETF in training deep learning models (see §5.4).

In the first experiment, we analyze the latency of SECURETF in Sim mode and HW mode, and make a comparison with native versions using glibc and musl libc (i.e., running Tensorflow Lite with Ubuntu and Alpine linux) and a system [6] provided by Intel using Graphene [90]. Graphene is an open-source SGX implementation of the original Graphene library OS. It follows a similar principle to Haven [16], by running a complete library OS inside of SGX enclaves. Similar to SCONE [12], Graphene offers developers the option to run their applications with Intel SGX without requiring code modifications. All evaluated systems except the Graphene-based system run inside a Docker container.

To conduct this experiment, we use the Desktop machine (see § 5.1) to install Ubuntu 16.04 since the Graphene based system does not work with Ubuntu 18.04. The evaluated systems run with single thread because of the current version of the Graphene-based system does not support multiple threads, i.e., to run the classification process, we use the same input arguments for the classification command line: $\$ label\_image - m\ model.tflite - i\ input.bmp - t\ 1$. For the latency measurement, we calculate the average over 1, 000

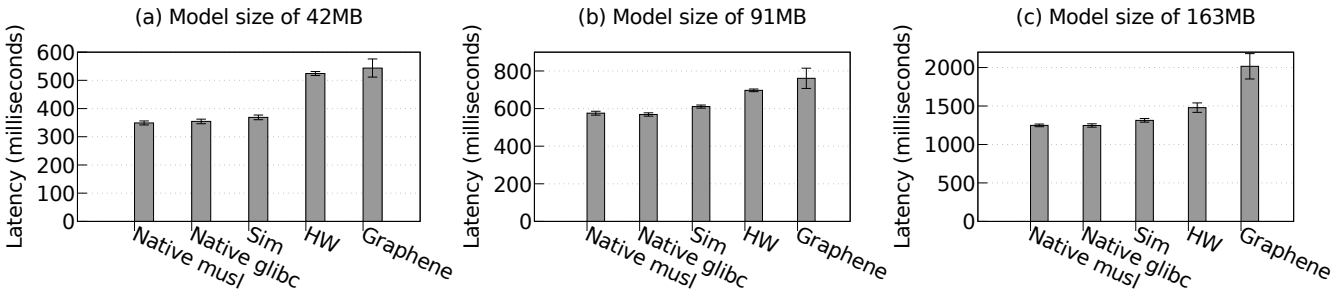

Figure 5. Comparison between SECURETF, native versions and the state-of-the-art Graphene system in terms of latency with different model sizes, **(a)** Densenet (42MB), **(b)** Inception_v3 (91MB), and **(c)** Inception_v4 (163MB).

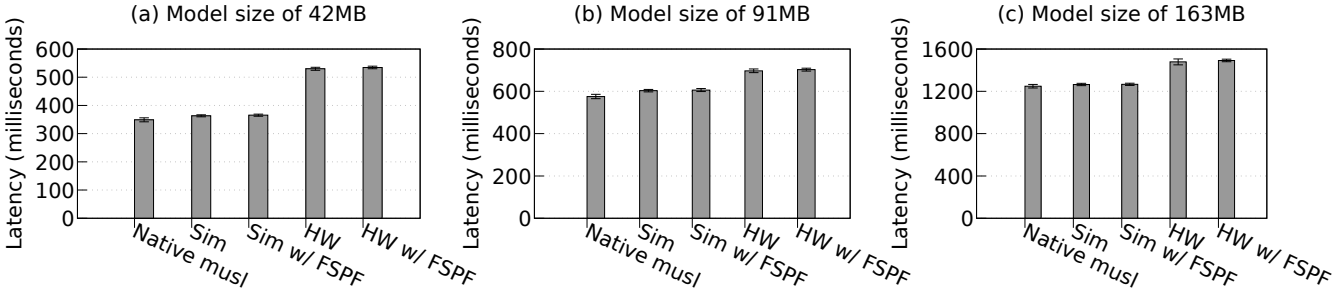

Figure 6. The effect of file system shield on the classification latency with different model sizes, **(a)** Densenet (42MB), **(b)** Inception_v3 (91MB), and **(c)** Inception_v4 (163MB).

runs. We use a single bitmap image from the Cifar-10 dataset as an input of evaluated systems.

Models. For classifying images, we use several pre-trained deep learning models with different sizes including *Inception-v3* [82] with the size of 91MB, *Inception-v4* [81] with the size of 163MB and Densenet [45] with the size of 42MB. We manually checked the correctness of a single classification by classifying the image with the TensorFlow *label_image* application involving no self-written code and running directly on the host without containerization. We later compared the results to the ones provided by SECURETF and other evaluated systems, we could confirm that indeed the same classifying result was produced by the evaluated systems.

#1: Effect of input model sizes. Figure 5 shows the latency comparison between SECURETF with Sim and HW mode, native Tensorflow Lite with glibc, native Tensorflow Lite with musl libc, and Graphene-based system. SECURETF with Sim mode incurs only ~ 5% overhead compared to the native versions with different model sizes. In addition, SECURETF with Sim mode achieves a latency 1.39×, 1.14×, and 1.12× lower than SECURETF with HW mode with the model size of 42MB, 91MB, and 162MB, respectively. This means that operations in the libc of SECURETF introduce a lightweight overhead. This is because SECURETF handles certain system calls inside the enclave and does not need to exit to the kernel. In the Sim mode, the execution is not performed inside hardware SGX enclaves, but SECURETF still handles some system calls in userspace, which can positively affect performance. We perform an analysis using *strace* tool to confirm that some of the most costly system calls of SECURETF are indeed system calls that are handled internally by the SCONE runtime.

Interestingly, the native Tensorflow Lite running with glibc is the same or slightly faster compared to the version with musl libc. The reason for this is that both C libraries excel in different areas, but glibc has the edge over musl in most areas, according to microbenchmarks [4], because glibc is tailored for performance, whereas musl is geared towards small size. Because of this difference in goals, an application may be faster with musl or glibc, depending on the performance bottlenecks that limit the application. Differences in performance of both C libraries must therefore be expected.

In comparison to Graphene-based system, SECURETF with HW mode is faster and faster than Graphene-based system when we increase the size of input models, specially when it exceeds the limit of the Intel SGX EPC size (~ 94MB). In particular, with the model size of 42MB, SECURETF with HW mode is only 1.03× faster compared to Graphene-based system, however, with the model size of 163MB, SECURETF with HW mode is ~ 1.4× compared to Graphene-based system. The reason for this is that when the application allocates memory size larger than the EPC size limit, the performance of reads and writes severely degraded because it performs encrypting data and paging operations which are very costly. To reduce this overhead, we reduce the size of our libraries loaded into SGX enclaves. Instead of adding the whole OS libc into SGX enclaves as Graphene did, we make use of SCONE libc [12] which is a modification of musl libc having

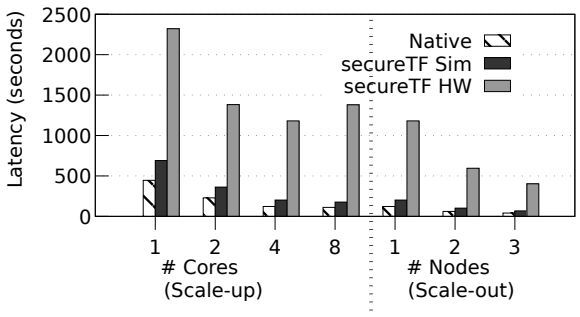

Figure 7. The latency comparison in classifying cifar-10 images with different numbers of CPU cores and nodes.

Figure 8. The training latency comparison between SECURETF with different modes and native Tensorflow.

much smaller size. In this library, system calls are not executed directly but instead are forwarded to the outside of an enclave via the asynchronous system call interface (see §3.3). This interface together with the user level scheduling allows SECURETF to mask system call latency by switching to other application threads. Thus, we expect this speedup factor of SECURETF compared to Graphene-based system will increase more when the size of the input model size is increased and when the application runs with multiple threads.

#2: Effect of file system shield. One of real world usecases of SECURETF is that a user not only wants to acquire classifying results but also wants to ensure the confidentiality of the input images since they may contain sensitive information, e.g., handwritten document images. At the same time, the user wants to protect her/his machine learning models since he/she had to spend a lot of time and cost to train the models. To achieve this level of security, the user activates the file system shield of SECURETF which allows he/she to encrypt the input including images and models and decrypt and process them within an SGX enclave (see §3.3).

In this experiment, we evaluate the effect of this file system shield on the overall performance of SECURETF. As previous experiments, we use the same input Cifar-10 images. Figure 6 shows the latency of SECURETF when running with/without activating the file system shield with different models. The file system shield incurs significantly small overhead on the performance of the classification process. SECURETF with Sim mode running with the file system shield is 0.12% slower than SECURETF with Sim mode running without the file system shield. Whereas in the SECURETF with HW mode, the overhead is 0.9%. The lightweight overhead comes from the fact that our file system shield uses Intel-CPU-specific hardware instructions to perform cryptographic operations and these instructions can reach a throughput of up to 4 GB/s, while the model is about 163 MB in size. This leads to a negligible overhead on the startup of the application only.

#3: Scalability. To evaluate the scalability of SECURETF, we measure the latency of SECURETF in classifying 800 cifar-10 images, with different number of CPU cores (scale-up), and different number of physical nodes (scale-out). Figure 7 shows that SECURETF both in Sim and HW mode scale quite well from 1 CPU core to 4 CPU cores. However, SECURETF in HW mode does not scale from 4 CPU cores to 8 CPU cores. The reason for this is that the EPC size is limited to ~94*MB*. When SECURETF runs with 8 cores it requires more than the capacity of the current version of Intel SGX. Thus, it requires to perform the paging mechanism which is very expensive. For scale-out evaluating, we keep each node to run with 4 CPU cores. As we expected, SECURETF in both Sim and HW mode scale well with different numbers of physical nodes. The latency of SECURETF in HW mode with 1 node is 1180s whereas with 3 nodes the latency is 403s.

#4: TensorFlow and TensorFlow Lite comparison. To show the advantage of using TensorFlow Lite in SECURETF instead of TensorFlow for inference or classification, we make a comparison between them. In this experiment, we use the same input model (i.e., Inception_v3 model) and input image to evaluate the performance of SECURETF using TensorFlow and TensorFlow Lite in HW mode. SECURETF with TensorFlow Lite achieves a ~ 71× lower latency (0.697s) compared to SECURETF with TensorFlow (49.782s). The reason for this is that, the binary size of SECURETF with TensorFlow Lite is only 1.9MB, meanwhile the binary size of SECURETF with TensorFlow is 87.4MB; and note that the Intel SGX enclave EPC size is limited to ~ 94MB.

5.4 Macrobenchmark: Distributed Training

In this section, we evaluate the performance of SECURETF in training distributed deep learning models at scale. In these experiments, we use MNIST handwritten digit dataset (see §5.1) and three physical servers having the same configuration described in §5.1. We keep the same batch size of 100 and learning rate as 0.0005, then measure the end-to-end latency of SECURETF with different modes including HW mode, Sim mode, with and without activating the network shield, and a native version of Tensorflow.

Figure 8 shows that SECURETF with different modes scales almost linearly with the number of workers. SECURETF, with full features running in HW mode, achieves a speedup of 1.96× and 2.57× when it runs with 2 and 3 workers, respectively. Unsurprisingly, this mode of SECURETF is roughly 14× slower compared to the native version due to the fact

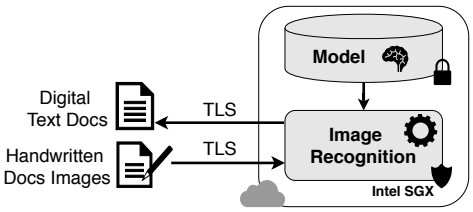

Figure 9. Deployment #1: secure document digitization.

that the training process requires memory-intensive computations and the enclave EPC size is limited to ~ 94MB. However, we believe that Intel will release new generation of its hardware which supports much large EPC sizes, thus we performed the experiments to evaluate SECURETF in the SIM mode, to see the overhead of SECURETF in the case the EPC size is enough for the training process. The slowdown factor in comparison to the native version, is reduced to 6× and 2.3× with SECURETF in the SIM mode with and without activating the network shield, respectively. This indicates that the main overhead of the current implementation is the network shield. In addition, note that the slowdown in SIM mode is because of a scheduling issue in SCONE. We have reported this issue, it's now fixed in the current version of SCONE.

From the results of experiments, we can learn that with the current Intel SGX hardware capacity, performing securely inference/classification inside Intel SGX is practical, but it is not feasible for securely training deep learning (see §7.1).

6 Real-World Deployments

SECURETF is a commercial platform, and it is actively used by four customers (names omitted) in production. We next present the SECURETF deployment for two use cases.

6.1 Secure Handwritten Documents Analysis

The first use case of SECURETF is to perform secure handwritten documents analytics (see Figure 9). A company (name omitted) is using a public cloud to automatically translate handwritten documents into digital format using machine learning. Customers of this company not only want to acquire the inference results, but also want to ensure the confidentiality of the input since the handwritten document images contain sensitive information. At the same time, the company wants to protect its Python code for the inference engine as well as its machine learning models. To achieve this level of security, the company has deployed our framework—SECURETF. The company uses the file system shield to encrypt Python code and models used for the inference. Meanwhile, the customers make use of the attestation mechanism of SECURETF to attest the enclave running the service, and then send their handwritten document images via the TLS connections to this service to convert them into digital text documents.

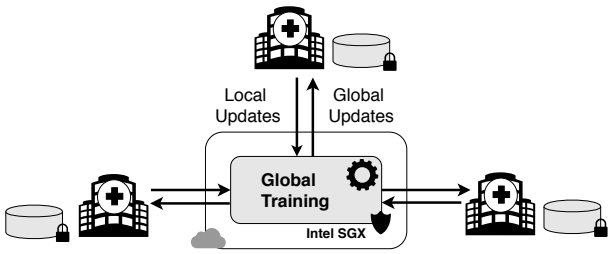

Figure 10. Deployment #2: secure federated learning.

6.2 Secure Federated Learning: Medical Use-case

The second use case of SECURETF is secure federated learning (FL) [60] (see Figure 10). FL is proposed to allow multiple parties to jointly train a model that takes benefits from diverse datasets from the parties. In our second use-case, several hospitals are actively collaborating to train a model for diagnosing brain tumors. However, at the same time, they want to protect patients' data regarding their privacy. Thus, each hospital performs the training locally using its local data and thereafter shares the model parameters with the global training computation without sharing its actual data. Unfortunately, the local model may reveal private information [9]. These local models have been demonstrated to be vulnerable to several privacy attacks [44]. In addition, there is empirical evidence to the risks presented by machine learning models, e.g., the work by Matt *et al* [34] demonstrates that extracted images from a face recognition system look similar to images from the underlying training dataset. To handle this issue, these hospitals make use of SECURETF to run the global training inside Intel SGX enclaves. They only share their local model after performing the attestation over the enclaves. The communication with the global training enclaves are performed via TLS connections.

7 Discussion and Lessons Learned

In this section, we discuss the lessons learned based on the limitations of our commercially available platform, and also, present open research problems for the future work.

7.1 Training Vs Classification

The limited EPC size has different implications for training vs classifications. As shown in §5, training deep learning with the larger datasets inside the enclave is performance-wise limiting due to EPC paging. However, the EPC size is quite practical for classifying/inference processes since the size of the deployed ML model is usually much smaller than the original training data. As discussed in §6, we are effectively using SECURETF for image classification (§6.1), and federated machine learning use case (see §6.2).

To improve the performance of the training phase in the limited enclave memory regions, we are exploring two avenues: (1) *data normalization:* we can further improve the training performance, by normalizing input data, e.g., in image recognition services, all input images can be normalized

to the size of 32×32; and (2) *Ice lake CPUs* Intel announced the next generation processors called *ice lake* which supports larger EPC size [27].

7.2 ML Model Optimizations

To further improve the performance, we are exploring perform optimizations for the ML models leveraging pruning and quantization tools, such as Intel OpenVINO Toolkit [99]. Since TensorFlow models are typically abstracted as directed computation graphs (see §2), where nodes are operations and edges present the communication between operations. By performing optimization on the model graphs such as pruning unnecessary edges and nodes, we can significantly improve the performance of classification/inference computations. The optimization also provides an opportunity to deploy ML inference service at edge devices supporting SGX [29] in edge computing. In fact, we have been working with an IoT-based company to use SECURETF for securely deploying the latest trained models at the edge, while achieving high-performance.

7.3 Security Analysis and Properties

SECURETF protects machine learning computations against attackers with privileged access by executing securely these computations inside Intel SGX enclaves. All data (input training/inference data, model, and Python code) and communications outside enclaves are always encrypted. The encrypted data is only decrypted inside enclaves. The keys or secrets to decrypt the data are protected inside the CAS component which is also running inside an enclave. The CAS component only provides these secrets via TLS connections to the machine learning enclaves after attesting these enclaves. A detailed security analysis of CAS is provided in [39].

Intel SGX is typically vulnerable to side-channel attacks [22, 26, 37, 50, 93, 95, 96]. Although this type of attacks are out-of-scope of our work, it is worth to mention that the version of SCONE, which was integrated in SECURETF, can not only protect against L1-based side channels attacks [67] but also Iago attacks [25]. We can also make use of LLVM-extensions, e.g., speculative load hardening [23] to prevent exploitable speculation which helps us to present the variants of Spectre attacks [26, 50]. In addition, the next generation of Intel CPUs [27] seems to provide hardware-based solutions to handle several types of side-channel attacks.

SECURETF supports only TLS-based communications to protect against eavesdropping on any communication between the CAS and computation nodes in a distributed setting. In SECURETF, the TLS certificates are generated inside the SGX enclave running CAS, and thus they cannot be seen by any human. This mechanism allows SECURETF to handle man-in-the-middle attacks. However, TLS and its predecessor are also vulnerable to side-channel attacks, e.g., attacks on RSA [13, 20]. Thus, in SECURETF, we recommend to completely disable RSA encryption and replace it by forward-secret key exchanges e.g., Elliptic-curve Diffie–Hellman (ECDHE) encryption [68].

7.4 GPUs Support

Graphics Processing Units (GPUs) have become popular and essential accelerators for machine learning [18]. Unfortunately, trusted computing in GPUs is not commercially available, except research prototypes, such as Graviton [94]. Therefore, SECURETF provides security properties by relying on Intel SGX which is supported only for CPUs.

Technically, SECURETF can also offer the GPU support, however, it requires weakening the threat model, i.e., we need to assume that the GPU computations and the communication between GPU and CPU are secure. The relaxation of the threat model may be acceptable in practice for several use cases, e.g., when users just want to protect their Python code and models for machine learning computations. SECURETF can ensure the code and models are encrypted. However, this extension may not practical for many other use cases [94]. Therefore, we are currently investigating GPU enclave research proposals, e.g., Graviton [94] and HIX [47] which proposed hardware extensions to provide a secure environment on GPUs.

8 Related Work

In this section, we summarize the related work about secure machine learning and shielded execution using Intel SGX.

Early works on preserving-privacy data mining techniques have relied on randomizing user data [21, 32, 73]. These approaches trade accuracy for privacy. They include a parameter that allows making a trade-off between privacy and accuracy. The proposed algorithms aim to provide privacy of computation, but they do not protect the results themselves in the cloud, nor do they secure the classification phase. While this can protect the users privacy, it does not cover training as in SECURETF. Further, we target to provide the same accuracy level as the native execution.

Graepel et al. [38] developed machine learning algorithms to perform both training and classification on encrypted data. The solution is based on the properties of homomorphic encryption. However, homomorphic encryption schemes provide restrictive computing operations, and incur high performance overheads. There have been a series of recent works [48, 54, 62, 64] aimed to provide secure machine learning platforms with Secure multiparty computation (MPC). Especially, Delphi [62] and CrypTFlow [54] demonstrated that they outperform previous works. However, these systems also were designed only for securing inferences. SECURETF is instead based on a hardware-based encryption approach (i.e., Intel SGX) and it supports both training and inference computations.

Shielded execution provides strong security guarantees for legacy applications running on untrusted platforms [16]. Prominent examples include Haven [16], SCONE [12], and Graphene-SGX [90]. Our work builds on the SCONE framework. Intel SGX has become available in clouds [36, 49], unleashing a plethora of services to be ported, including Web search [72], actor framework [76], storage [15, 52], leases [85], monitoring and profilers [14, 51], software update [70], FaaS [88], networking [86, 87], and data analytics systems [57, 77, 100].

Recently, several secure machine learning systems have been proposed, which rely on Intel SGX to support secure machine learning [40, 46, 65, 89]. Privado [40] proposes a mechanism to obtain oblivious neural networks. Then, it executes the oblivious neural network inside SGX enclaves for secure inferencing. Slalom [89] makes use of a combination of Intel SGX and untrusted GPUs to secure Deep Neural Networks (DNNs) computations. The idea of Slalom is that it splits the DNN computations into linear operations (e.g., matrix multiplications) on GPUs, whereas performing the non-linear operations (eg. ReLUs operations) inside Intel SGX enclaves. This approach allows achieving much better performance since the intensive computation is performed with GPUs. Unfortunately, Slalom still has several limitations. First, as Privado, it focuses only on secure inferences. It refers to secure training computations as a research challenge. Second, it requires Tensorflow users to heavily modify or redevelop their existing code. Third, it does not support distributed settings, i.e., it does not support secure connections between SGX enclaves. Finally, Slalom is not production ready, in fact, it indicates that it can be used only for testing. Chiron [46] is the most relevant for SECURETF, where they leveraged Intel SGX for privacy-preserving machine learning services. Unfortunately, Chiron is a single-threaded system within an enclave. In addition, Chiron requires adding an interpreter and model compiler into enclaves which introduce significant runtime overhead since the limited EPC size. The work from Ohrimenko et al. [65] also used Intel SGX to secure machine learning computations, however, it supports only a limited number of operators. In contrast, we propose SECURETF — a practical distributed machine learning framework for securing both training and inference computations.

9 Conclusion

In this paper, we report on our experience with building and deploying SECURETF, a secure TensorFlow-based machine learning framework leveraging the hardware-assisted TEEs, specifically Intel SGX. SECURETF extends the security properties of a secure stateless enclave in a single node to secure unmodified distributed stateful machine learning applications. Thereby, it provides a generic platform for end-to-end security for the input data, ML model, and application code. Moreover, it supports both training and classification phases while providing all three important design properties for the secure machine learning workflow: *transparency*, *accuracy*, and *performance*. SECURETF is a commercially available platform, and is currently being used in production by four major customers. While there are several open challenges and limitations of our system, our experience shows that SECURETF strives for a promising approach: it incurs reasonable performance overheads, especially in the classification/inference process, while providing strong security properties against a powerful adversary. Lastly, we also discussed several open challenges and on-going extensions to the system.

Acknowledgements. We thank our shepherd Professor Sara Bouchenak and the anonymous reviewers for their insightful comments and suggestions. This work has received funding from the Cloud-KRITIS Project and the European Union's Horizon 2020 research and innovation programme under the LEGaTO Project (legato-project.eu), grant agreement No 780681.

References

[1] Alpine Linux. https://alpinelinux.org/. Accessed: May, 2020.
[2] Alpine Linux FAQ. https://wiki.musl-libc.org/faq.html. Accessed: May, 2020.
[3] AMD Secure Technology. https://www.amd.com/en/technologies/security. Accessed: May, 2020.
[4] Comparison of C/POSIX standard library implementations for Linux. http://www.etalabs.net/compare_libcs.html. Accessed: May, 2020.
[5] Deepmind health and research collaborations. https://deepmind.com/applied/deepmind-health/working-partners/health-research-tomorrow/. Accessed: May, 2020.
[6] Graphene Tensorflow Lite benchmark. https://github.com/oscarlab/graphene-tests/tree/master/tensorflow/. Accessed: May, 2020.
[7] Tensorflow lite. https://www.tensorflow.org/lite. Accessed: Jan, 2020.
[8] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, et al. TensorFlow: A System for Large-Scale Machine Learning. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.
[9] M. Abadi, A. Chu, I. Goodfellow, H. B. McMahan, I. Mironov, K. Talwar, and L. Zhang. Deep learning with differential privacy. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2016.
[10] G. Allen and M. Owens. *The Definitive Guide to SQLite*. Apress, 2010.
[11] ARM. Building a secure system using TrustZone technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, 2009. Accessed: May, 2020.
[12] S. Arnautov, B. Trach, F. Gregor, T. Knauth, A. Martin, C. Priebe, J. Lind, D. Muthukumaran, et al. SCONE: Secure Linux Containers with Intel SGX. In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2016.
[13] N. Aviram, S. Schinzel, J. Somorovsky, N. Heninger, M. Dankel, J. Steube, L. Valenta, D. Adrian, J. A. Halderman, V. Dukhovni, E. Käsper, S. Cohney, S. Engels, C. Paar, and Y. Shavitt. DROWN: Breaking TLS using sslv2. In *25th USENIX Security Symposium (USENIX Security)*, 2016.
[14] M. Bailleu, D. Dragoti, P. Bhatotia, and C. Fetzer. Tee-perf: A profiler for trusted execution environments. In *2019 49th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2019.

[15] M. Bailleu, J. Thalheim, P. Bhatotia, C. Fetzer, M. Honda, and K. Vaswani. SPEICHER: Securing lsm-based key-value stores using shielded execution. In *17th USENIX Conference on File and Storage Technologies (FAST)*, 2019.

[16] A. Baumann, M. Peinado, and G. Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2014.

[17] Bazel. The Bazel project. https://bazel.build/. Accessed: May, 2020.

[18] R. Bekkerman, M. Bilenko, and J. Langford. *Scaling up machine learning: Parallel and distributed approaches*. Cambridge University Press, 2011.

[19] J. Bennett, S. Lanning, et al. The netflix prize. In *Proceedings of KDD cup and workshop*, 2007.

[20] H. Böck, J. Somorovsky, and C. Young. Return of bleichenbacher's oracle threat (ROBOT). In *27th USENIX Security Symposium (USENIX Security)*, 2018.

[21] R. Bost, R. A. Popa, S. Tu, and S. Goldwasser. Machine learning classification over encrypted data. In *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*, 2015.

[22] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi. Software grand exposure:{SGX} cache attacks are practical. In *11th {USENIX} Workshop on Offensive Technologies (WOOT)*, 2017.

[23] C. Carruth. Speculative load hardening. https://llvm.org/docs/SpeculativeLoadHardening.html, 2019.

[24] S. Checkoway and H. Shacham. Iago Attacks: Why the System Call API is a Bad Untrusted RPC Interface. In *Proceedings of the 18th ACM International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[25] S. Checkoway and H. Shacham. Iago attacks: Why the system call api is a bad untrusted rpc interface. In *Proceedings of the Eighteenth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2013.

[26] G. Chen, S. Chen, Y. Xiao, Y. Zhang, Z. Lin, and T. H. Lai. Sgxpectre attacks: Stealing intel secrets from sgx enclaves via speculative execution. *arXiv e-prints*, 2018.

[27] I. Corp. 10th Generation Intel Processors Core Families. https://www.intel.com/content/dam/www/public/us/en/documents/datasheets/10th-gen-core-families-datasheet-vol-1-datasheet.pdf. Accessed: May, 2020.

[28] I. Corp. Intel Software Guard Extensions (Intel SGX). https://software.intel.com/en-us/sgx. Accessed: May, 2020.

[29] I. Corporation. Intel nuc kits. Accessed: 28 May 2020.

[30] V. Costan and S. Devadas. Intel SGX Explained. *IACR Cryptology ePrint Archive*, 2016.

[31] D. Dolev and A. C. Yao. On the security of public key protocols. In *Proceedings of the 22nd Annual Symposium on Foundations of Computer Science (SFCS)*, pages 350–357, 1981.

[32] W. Du and Z. Zhan. Using randomized response techniques for privacy-preserving data mining. In *Proceedings of the ninth international conference on Knowledge discovery and data mining (SIGKDD)*, 2003.

[33] K. R. Foster, R. Koprowski, and J. D. Skufca. Machine learning, medical diagnosis, and biomedical engineering research-commentary. *Biomedical engineering online*, 2014.

[34] M. Fredrikson, S. Jha, and T. Ristenpart. Model inversion attacks that exploit confidence information and basic countermeasures. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2015.

[35] Google. Google protocol buffers. https://developers.google.com/protocol-buffers/. Accessed: May, 2020.

[36] J. C. Gordon. Microsoft azure confidential computing with intel sgx. Accessed: 28 May 2020.

[37] J. Götzfried, M. Eckert, S. Schinzel, and T. Müller. Cache attacks on intel sgx. In *Proceedings of the 10th European Workshop on Systems Security*, 2017.

[38] T. Graepel, K. Lauter, and M. Naehrig. Ml confidential: Machine learning on encrypted data. In *Proceedings of the International Conference on Information Security and Cryptology*, 2012.

[39] F. Gregor, W. Ozga, S. Vaucher, R. Pires, D. L. Quoc, S. Arnautov, A. Martin, V. Schiavoni, P. Felber, and C. Fetzer. Trust Management as a Service: Enabling Trusted Execution in the Face of Byzantine Stakeholders. In *Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, 2020.

[40] K. Grover, S. Tople, S. Shinde, R. Bhagwan, and R. Ramjee. Privado: Practical and secure dnn inference with enclaves. 2018.

[41] H. S. Gunawi, M. Hao, T. Leesatapornwongsa, T. Patana-anake, T. Do, J. Adityatama, K. J. Eliazar, A. Laksono, J. F. Lukman, V. Martin, and A. D. Satria. What Bugs Live in the Cloud? A Study of 3000+ Issues in Cloud Systems. In *Proceedings of the ACM Symposium on Cloud Computing (SoCC)*, 2014.

[42] M. Hähnel, W. Cui, and M. Peinado. High-resolution side channels for untrusted operating systems. In *Proceedings of the USENIX Annual Technical Conference (ATC)*, 2017.

[43] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[44] B. Hitaj, G. Ateniese, and F. Perez-Cruz. Deep models under the gan: Information leakage from collaborative deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2017.

[45] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017.

[46] T. Hunt, C. Song, R. Shokri, V. Shmatikov, and E. Witchel. Chiron: Privacy-preserving machine learning as a service. *CoRR*, 2018.

[47] I. Jang, A. Tang, T. Kim, S. Sethumadhavan, and J. Huh. Heterogeneous isolated execution for commodity gpus. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*, 2019.

[48] C. Juvekar, V. Vaikuntanathan, and A. Chandrakasan. Gazelle: A low latency framework for secure neural network inference. In *Proceedings of the 27th USENIX Conference on Security Symposium (USENIX Security)*, 2018.

[49] P. Karnati. Data-in-use protection on ibm cloud using intel sgx. Accessed: 28 May 2020.

[50] P. Kocher, J. Horn, A. Fogh, , D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom. Spectre Attacks: Exploiting Speculative Execution. In *40th IEEE Symposium on Security and Privacy (S&P)*, 2019.

[51] R. Krahn, D. Dragoti, F. Gregor, D. Le Quoc, V. Schiavoni, P. Felber, C. Souza, A. Brito, and C. Fetzer. TEEMon: A continuous performance monitoring framework for TEEs. In *Proceedings of the 21th International Middleware Conference (Middleware)*, 2020.

[52] R. Krahn, B. Trach, A. Vahldiek-Oberwagner, T. Knauth, P. Bhatotia, and C. Fetzer. Pesos: Policy enhanced secure object store. In *Proceedings of the Thirteenth EuroSys Conference (EuroSys)*, 2018.

[53] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. Technical report, Citeseer, 2009.

[54] N. Kumar, M. Rathee, N. Chandran, D. Gupta, A. Rastogi, and R. Sharma. CrypTFlow: Secure TensorFlow Inference. In *IEEE Symposium on Security and Privacy (S&P)*, 2020.

[55] R. Kunkel, D. L. Quoc, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer. TensorSCONE: A Secure TensorFlow Framework using Intel SGX. *arXiv preprint arXiv:1902.04413*, 2019.

[56] D. Kuvaiskii, O. Oleksenko, S. Arnautov, B. Trach, P. Bhatotia, P. Felber, and C. Fetzer. SGXBOUNDS: Memory Safety for Shielded Execution. In *Proceedings of the 12th ACM European Conference on Computer Systems (EuroSys)*, 2017.

[57] D. Le Quoc, F. Gregor, J. Singh, and C. Fetzer. Sgx-pyspark: Secure distributed data analytics. In *Proceedings of the World Wide Web Conference (WWW)*, 2019.

[58] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[59] N. D. Matsakis and F. S. Klock, II. The rust language. In *Proceedings of the 2014 ACM SIGAda Annual Conference on High Integrity Language Technology*, HILT '14, 2014.

[60] B. McMahan, E. Moore, D. Ramage, S. Hampson, and B. A. y Arcas. Communication-Efficient Learning of Deep Networks from Decentralized Data. In *Proceedings of the 20th International Conference on Artificial Intelligence and Statistics*, 2017.

[61] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux Journal*, 2014.

[62] P. Mishra, R. Lehmkuhl, A. Srinivasan, W. Zheng, and R. A. Popa. Delphi: A cryptographic inference service for neural networks. In *29th USENIX Security Symposium (USENIXSecurity)*, 2020.

[63] S. Mofrad, F. Zhang, S. Lu, and W. Shi. A comparison study of Intel SGX and AMD memory encryption technology. In *Proceedings of the 7th International Workshop on Hardware and Architectural Support for Security and Privacy*, 2018.

[64] P. Mohassel and Y. Zhang. Secureml: A system for scalable privacy-preserving machine learning. In *2017 IEEE Symposium on Security and Privacy (S&P)*, 2017.

[65] O. Ohrimenko, F. Schuster, C. Fournet, A. Mehta, S. Nowozin, K. Vaswani, and M. Costa. Oblivious multi-party machine learning on trusted processors. In *Proceedings of the 25th USENIX Security Symposium (USENIX Security)*, 2016.

[66] O. Oleksenko, D. Kuvaiskii, P. Bhatotia, P. Felber, and C. Fetzer. Intel MPX Explained: A Cross-layer Analysis of the Intel MPX System Stack. *Proceedings of the ACM on Measurement and Analysis of Computing Systems*, 2018.

[67] O. Oleksenko, B. Trach, R. Krahn, M. Silberstein, and C. Fetzer. Varys: Protecting SGX enclaves from practical side-channel attacks. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, 2018.

[68] R. Oppliger. *SSL and TLS: Theory and Practice*. Artech House, 2016.

[69] M. Orenbach, M. Minkin, P. Lifshits, and M. Silberstein. Eleos: Exit-Less OS services for SGX enclaves. In *Proceedings of the 12th ACM European ACM Conference in Computer Systems (EuroSys)*, 2017.

[70] W. Ozga, D. Le Quoc, and C. Fetzer. A practical approach for updating an integrity-enforced operating system. In *Proceedings of the 21th International Middleware Conference (Middleware)*, 2020.

[71] B. Parno, J. R. Lorch, J. R. Douceur, J. Mickens, and J. M. McCune. Memoir: Practical state continuity for protected modules. In *Proceedings of the 32nd IEEE Symposium on Security and Privacy (S&P)*, 2011.

[72] R. Pires, D. Goltzsche, S. B. Mokhtar, S. Bouchenak, A. Boutet, P. Felber, R. Kapitza, M. Pasin, and V. Schiavoni. CYCLOSA: decentralizing private web search through sgx-based browser extensions. In *38th IEEE International Conference on Distributed Computing Systems(ICDCS)*, 2018.

[73] D. L. Quoc, M. Beck, P. Bhatotia, R. Chen, C. Fetzer, and T. Strufe. PrivApprox: Privacy-Preserving Stream Analytics. In *Proceedings of the 2017 USENIX Annual Technical Conference (USENIX ATC)*, 2017.

[74] N. Santos, K. P. Gummadi, and R. Rodrigues. Towards Trusted Cloud Computing. In *Proceedings of the 1st USENIX Workshop on Hot Topics in Cloud Computing (HotCloud)*, 2009.

[75] N. Santos, R. Rodrigues, K. P. Gummadi, and S. Saroiu. Policy-sealed data: A new abstraction for building trusted cloud services. In *Proceedings of the 21st USENIX Security Symposium*, 2012.

[76] V. A. Sartakov, S. Brenner, S. Ben Mokhtar, S. Bouchenak, G. Thomas, and R. Kapitza. Eactors: Fast and flexible trusted computing using sgx. In *Proceedings of the 19th International Middleware Conference (Middleware)*, 2018.

[77] F. Schuster, M. Costa, C. Gkantsidis, M. Peinado, G. Mainar-ruiz, and M. Russinovich. VC3 : Trustworthy Data Analytics in the Cloud using SGX. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P)*, 2015.

[78] S. Shinde, D. Tien, S. Tople, and P. Saxena. Panoply: Low-tcb linux applications with sgx enclaves. In *Proceedings of the Annual Network and Distributed System Security Symposium (NDSS)*, page 12, 2017.

[79] O. Simeone. A brief introduction to machine learning for engineers. *arXiv preprint arXiv:1709.02840*, 2017.

[80] L. Soares and M. Stumm. FlexSC: Flexible System Call Scheduling with Exception-less System Calls. In *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2010.

[81] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *Proceedings of the 31th AAAI Conference on Artificial Intelligence (AAAI)*, 2017.

[82] C. Szegedy, V. Vanhoucke, S. Ioffe, J. Shlens, and Z. Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016.

[83] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014.

[84] S. Tallam, C. Coutant, I. L. Taylor, X. D. Li, and C. Demetriou. Safe icf: Pointer safe and unwinding aware identical code folding in gold. In *GCC Developers Summit*, 2010.

[85] B. Trach, R. Faqeh, O. Oleksenko, W. Ozga, P. Bhatotia, and C. Fetzer. T-lease: A trusted lease primitive for distributed systems. In *ACM Symposium on Cloud Computing 2020 (SoCC)*, 2020.

[86] B. Trach, A. Krohmer, S. Arnautov, F. Gregor, P. Bhatotia, and C. Fetzer. Slick: Secure Middleboxes using Shielded Execution. 2017.

[87] B. Trach, A. Krohmer, F. Gregor, S. Arnautov, P. Bhatotia, and C. Fetzer. ShieldBox: Secure Middleboxes using Shielded Execution. In *Proceedings of the ACM SIGCOMM Symposium on SDN Research (SOSR)*, 2018.

[88] B. Trach, O. Oleksenko, F. Gregor, P. Bhatotia, and C. Fetzer. Clemmys: Towards secure remote execution in faas. In *12th ACM International Conference on Systems and Storage (SYSTOR)*, 2019.

[89] F. Tramèr and D. Boneh. Slalom: Fast, verifiable and private execution of neural networks in trusted hardware. In *7th International Conference on Learning Representations (ICLR)*, 2019.

[90] C.-C. Tsai, D. E. Porter, and M. Vij. Graphene-SGX: A practical library OS for unmodified applications on SGX. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*, 2017.

[91] A. Vahldiek-Oberwagner, E. Elnikety, A. Mehta, D. Garg, P. Druschel, R. Rodrigues, J. Gehrke, and A. Post. Guardat: Enforcing data policies at the storage layer. In *Proceedings of the 10th ACM European Conference on Computer Systems (EuroSys)*, 2015.

[92] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, 2018.

[93] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. F. Wenisch, Y. Yarom, and R. Strackx. Foreshadow: Extracting the keys to the intel sgx kingdom with transient out-of-order execution. In *Proceedings of the 27th USENIX Security Symposium (USENIX Security)*, 2018.

[94] S. Volos, K. Vaswani, and R. Bruno. Graviton: Trusted execution environments on gpus. In *Proceedings of the 13th USENIX Symposium*

on Operating Systems Design and Implementation (OSDI), 2018.

[95] W. Wang, G. Chen, X. Pan, Y. Zhang, X. Wang, V. Bindschaedler, H. Tang, and C. A. Gunter. 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 (CCS)*, 2017.

[96] O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. F. Wenisch, and Y. Yarom. Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution. Technical report, 2018. See also USENIX Security paper Foreshadow [93].

[97] B. Xu, N. Wang, T. Chen, and M. Li. Empirical evaluation of rectified activations in convolutional network. *arXiv preprint arXiv:1505.00853*, 2015.

[98] Y. Xu, W. Cui, and M. Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *Proceedings of the 36th IEEE Symposium on Security and Privacy (S&P)*, 2015.

[99] A. Zaytsev and A. Zaytsev. Openvino toolkit. =https://software.intel.com/content/www/us/en/develop/articles/openvino-relnotes.html. Accessed: 28 May 2020.

[100] W. Zheng, A. Dave, J. G. Beekman, R. A. Popa, J. E. Gonzalez, and I. Stoica. Opaque: An Oblivious and Encrypted Distributed Analytics Platform. In *Proceedings of the 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2017.