Perun: Confidential Multi-stakeholder Machine Learning Framework with Hardware Acceleration Support

Wojciech Ozga, Do Le Quoc, Christof Fetzer

PERUN: Confidential Multi-Stakeholder Machine Learning Framework with Hardware Acceleration Support

Wojciech Ozga[1,2*], Do Le Quoc[3**], and Christof Fetzer[1,4]

[1] TU Dresden, Germany
[2] IBM Research Europe — Zurich
[3] Huawei Munich Research Center, Germany
[4] Scontain UG, Germany

Abstract. Confidential multi-stakeholder machine learning (ML) allows multiple parties to perform collaborative data analytics while not revealing their intellectual property, such as ML source code, model, or datasets. State-of-the-art solutions based on homomorphic encryption incur a large performance overhead. Hardware-based solutions, such as trusted execution environments (TEEs), significantly improve the performance in inference computations but still suffer from low performance in training computations, *e.g.*, deep neural networks model training, because of limited availability of protected memory and lack of GPU support.
To address this problem, we designed and implemented PERUN, a framework for confidential multi-stakeholder machine learning that allows users to make a trade-off between security and performance. PERUN executes ML training on hardware accelerators (*e.g.*, GPU) while providing security guarantees using trusted computing technologies, such as trusted platform module and integrity measurement architecture. Less compute-intensive workloads, such as inference, execute only inside TEE, thus at a lower trusted computing base. The evaluation shows that during the ML training on CIFAR-10 and real-world medical datasets, PERUN achieved a 161× to 1560× speedup compared to a pure TEE-based approach.

Keywords: Multi-Stakeholder Computation · Machine Learning · Confidential Computing · Trusted Computing · Trust Management

1 Introduction

Machine learning (ML) techniques are widely adopted to build functional artificial intelligence (AI) systems. For example, face recognition systems allow paying at supermarkets without typing passwords; natural language processing systems allow translating information boards in foreign countries using smartphones; medical expert systems help to detect diseases at an early stage; image recognition

* Corresponding author: woz@zurich.ibm.com
** Do Le Quoc performed this work at TU Dresden.

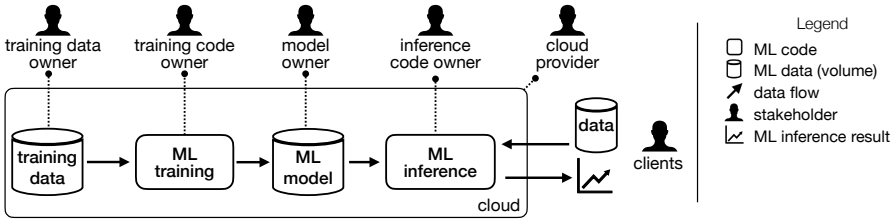

Fig. 1. Stakeholders share source code, data, and computing power to build an ML application. They need a framework to establish mutual trust and share data securely.

systems help autonomous cars to identify road trajectory and traffic hazards. To build such systems, multiple parties or stakeholders with domain knowledge from various science and technology fields must cooperate since machine learning is fundamentally a multi-stakeholder computation, as shown in Figure 1. They would benefit from sharing their intellectual property (IP) – private training data, source code, and models – to jointly perform machine learning computations only if they can ensure their IP remains confidential.

Training data owner. ML systems rely on *training data* to build *inference models*. However, the data is frequently sensitive and cannot be easily shared between disjoint entities, like in the case of the healthcare data used for training diagnostic models contain privacy-sensitive patient information. The strict data regulations, such as general data protection regulation (GDPR) [1], impose an obligation on secure data processing. Specifically, the training data must be under the training data owner's control and must be protected while at rest, during transmission, and training computation.

Training code owner. The training code owner implements a *training algorithm* that trains an *inference model* over the training data. The *training code* (*e.g.*, Python code) typically contains an optimized training model architecture and tuned parameters that build the business value and the inference model quality. Thus, the training code is considered as confidential as the training data. The training requires high computing power, and, as such, it is economically justifiable to delegate its execution to the cloud. However, in the cloud, users with administrative access can easily read the training service source code implemented in popular programming languages, such as Python.

Model owner. The inference model is the heart of any inference service. It is created by training the model with a large amount of training data. This process requires extensive computing power and is time-consuming and expensive. Thus, the model owner, a training code owner, or a third party that buys the model, must protect the model's confidentiality. The trained models may reveal the privacy of the training data [3]. Several works [13,3] demonstrated that extracted images from a face recognition system look suspiciously similar to images from the underlying training data.

Inference code owner. The inference code is an AI service allowing clients to use the inference model on a business basis. The inference code is frequently developed

using Python or JavaScript and hosted in the cloud. Thus, the confidentiality of the code and the integrity of the computation must be protected against an adversary controlling computing systems executing the AI service.

Inference data owner. The inference data owner is a client of an AI service. He wants to protect his input data. Imagine a person sending an X-ray scan of her brain to a diagnostic service to check for a brain tumor. The inference data, *e.g.*, a brain's scan, is privacy-sensitive and must not be accessible by the AI service provider.

To build an AI service, stakeholders must trust that others follow the rules protecting each other's IP. However, it is difficult to establish trust among them. First, some stakeholders might collude to gain advantages over others [41]. Second, even a trustworthy stakeholder might lack expertise in protecting their IP from a skilled attacker gaining access to its computing resources [43,12]. We tackle the following problem: How to allow stakeholders to jointly perform machine learning to unlock all AI benefits without revealing their IP?

Recent works [37,33] demonstrated that cryptographic techniques, such as secure multi-party computation [56] and fully homomorphic encryption [15], incur a large performance overhead, which currently prevents their adoption for computing-intensive ML. Alternative approaches [42,34] adopted trusted execution environments (TEEs) [36] to build ML systems showing that TEEs offer orders of magnitude faster ML computation, at the cost of weaker security guarantees compared to pure cryptographic solutions. Specifically, the pure cryptographic solutions compute on encrypted data, while in TEE-based approaches, a trusted ML software processes the plaintext data in a CPU-established execution environment (called *enclave*), which is isolated from the untrusted operating system and administrator. Although promising for the ML inference, TEEs still incur considerable performance overhead for memory-intensive computations, like deep training, because of the limited memory accessible to the enclave and lack of support for hardware accelerators, like graphical processing units (GPUs). Thus, since TEEs alone are not enough for the ML training processes, we raise the question: What trade-off between security and performance has to be made to allow the ML training to access hardware accelerators?

We propose PERUN, a framework allowing stakeholders to share their code and data only with certain ML applications running inside an enclave and on a trusted OS. PERUN relies on encryption to protect the IP and on a trusted key management service to generate and distribute the corresponding cryptographic keys. TEE provides confidentiality and integrity guarantees to ML applications and to the key management service. Trusted computing technologies [14] provide integrity guarantees to the OS, allowing ML computations to access hardware accelerators. Our evaluation shows that PERUN achieves 0.96× of native performance execution on the GPU and a speedup of up to 1560× in training a real-world medical dataset compared to a pure TEE-based approach [34].

Altogether, we make the following contributions:

– We designed a secure multi-stakeholder ML framework that: *(i)* allows stakeholders to cooperate while protecting their IP (§3.1, §3.2)), *(ii)* allows stake-

holders to select trade-off between the security and performance, allowing for hardware accelerators usage (§3.3,§3.4).
– We implemented the PERUN prototype (§4) and evaluated it using real-world datasets (§5).

2 Threat Model

Stakeholders are financially motivated businesses that cooperate to perform ML computation. Each stakeholder delivers an input (*e.g.*, input training data, code, and ML models) as its IP for ML computations. The IP must remain confidential during ML computations. The stakeholders have limited trust. They do not share their IP directly, but they encrypt them so that only other stakeholders' applications, which source code they can inspect under a non-disclosure agreement or execute in a sandbox, can access the encryption key to decrypt it.

An adversary wants to steal a stakeholder's IP when it resides on a computer executing ML computation. Such a computer might be provisioned in the cloud or a stakeholder's data center, *e.g.*, a hybrid cloud model. In both cases, an adversary has no physical access to the computer. For this, we rely on state-of-the-art practices controlling and restricting access to the data center to trusted entities.

However, an adversary might exploit an OS misconfiguration or use social engineering to connect to the OS remotely. We assume she can execute privileged software to read an ML process's memory after getting administrative access to the OS executing ML computation. One of the mitigation techniques used in PERUN, integrity measurement architecture (IMA) [45], effectively limits software that can execute on the computer under the assumption that this software, which is considered trusted, behaves legitimately also after it has been loaded to the memory, *i.e.*, an adversary cannot tamper with the process' code after it has been loaded to the memory. This might be achieved using existing techniques, like enforcing control flow integrity [27], fuzzing [57], formally proving the software implementation [58], using memory-safe languages [35], using memory corruption mitigation techniques, like position-independent executables, stack-smashing protection, relocation read-only techniques, or others.

The CPU with its hardware features, hardware accelerators, and secure elements (*e.g.*, TPM) are trusted. We exclude micro-architectural and side-channel attacks, like Foreshadow [51] or Spectre [30]. We rely on the soundness of the cryptographic primitives used within software and hardware components.

3 Design

Our objective is to provide an architecture that: (i) supports multi-stakeholder ML computation, (ii) requires zero code changes to the existing ML code, (iii) allows for a trade-off between security and performance, (iv) uses hardware accelerators for computationally-intensive tasks.

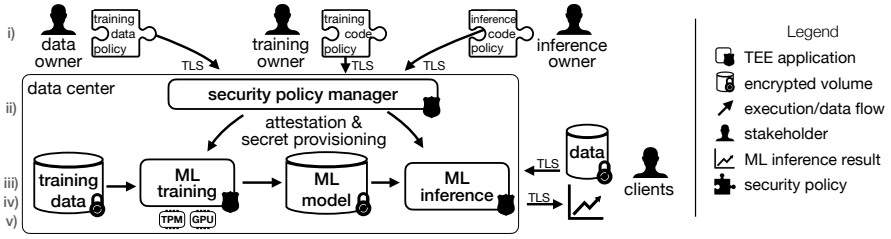

Fig. 2. PERUN framework supports multi-stakeholder ML computation. Stakeholders trust the security policy manager. Inside security policies, they define which stakeholder's application can access a cryptographic key allowing decryption of confidential code or data. TEE protects code, data, and cryptographic keys.

3.1 High-level Overview

Figure 2 shows the PERUN framework architecture that supports multi-stakeholder computation and the use of dedicated hardware accelerators. The framework consists of five components: (i) *stakeholders*, the parties who want to perform ML jointly while keeping their IP protected; (ii) *security policy manager*, a key management and configuration service that allows stakeholders to share IPs for ML computations without revealing them; (iii) *ML computation* including training and inference; (iv) *GPU*, hardware accelerators enabling high-performance ML computation; and (v) *TEE* and *TPM*, *secure elements* enabling confidentiality and integrity of ML computations on untrusted computing resources.

To allow multiple stakeholders to perform ML and keep their IP confidential, we propose that the IP remains under the stakeholder's control. To realize that idea, we design the security policy manager that plays the role of the *root of trust*. Stakeholders establish trust in this component using the *remote attestation* mechanism, like [25], offered by a TEE. The TEE, *e.g.*, Intel software guard extensions (SGX) [11], guarantees the confidentiality and integrity of processed code and data. After stakeholders ensure the security policy manager executes in the TEE, they submit security policies defining access control to their encryption keys. Each stakeholder's IP is encrypted with a different key, and the security policy manager uses security policies to decide who can access which keys. From a technical perspective, the security policy manager generates the keys inside the TEE and sends them only to authenticated ML computations executing inside the TEE. Thus, these keys cannot be seen by any human.

Depending on individual stakeholders' security requirements, PERUN offers different throughput/latency performances for ML computations. For stakeholders willing strong integrity and confidentiality guarantees, PERUN executes ML computations only inside TEEs enclaves, *i.e.*, input and output data, code, and models never leave the enclave. For stakeholders accepting a larger trusted computing base in exchange for better performance, PERUN enables trusted computing technologies [14] to protect ML computations while executing them on hardware accelerators, *e.g.*, GPU. Specifically, it uses IMA, which is an integrity enforcement mechanism that prevents adversaries from running arbitrary software on the OS, *i.e.*, software that allows reading data residing in the main memory

or being transferred to or processed by the GPU. The security policy manager verifies that such a mechanism is enabled by querying a secure element compatible with the trusted platform module (TPM) [19] attached to the remote computer.

3.2 Keys Sharing

Stakeholders use security policies to share encryption keys protecting their IP. For example, the training data owner specifies in his security policy that he allows the ML computation of the training code owner to access his encryption key to decrypt the training data. The security policy manager plays a key role in the key sharing process. It generates an encryption key inside the TEE and securely distributes it to ML computations accordingly to the security policy. The training code owner cannot see the shared secret in the example above because it is transferred only to his application executing inside the TEE.

To provision ML computations with encryption keys, the security policy manager authenticates them using a remote attestation protocol offered by a TEE engine, *e.g.*, the SGX remote attestation protocol [25]. During the remote attestation, the TEE engine provides the security policy manager with a cryptographic measurement of the code executing on the remote platform. The cryptographic measurement – the output of the cryptographic hash function over the code loaded by the TEE engine to the memory – uniquely identifies the ML computation, allowing the security policy manager to authorize access to the encryption key based on the ML computation identity and stakeholder's security policies.

3.3 Security Policy and Trade-offs

Perun relies on security policies as a means to define dependencies among stakeholders' computation and shared data.

Listing 1.1 shows an example of a policy. The policy has a unique name (line 1), typically combining a stakeholder's name and its IP name. The name is used among stakeholders to reference *volumes* containing code, input, or output data. A volume is a collection of files encrypted with an encryption key managed by the security policy manager. Only the authorized ML computations have access to the key required to decrypt the volume and access the IP. To prevent an adversary from changing the policy, the stakeholder embeds his public key inside the policy (line 21). The security policy manager accepts only policies containing a valid signature issued with a corresponding stakeholder's private key.

The ML computation definition consists of a command required to execute the computation inside a container (line 2) and a cryptographic hash over the source code content implementing the ML computation (line 8). The security policy manager uses the hash to authenticate the ML computation before providing it with the encryption key.

The policy allows selecting trade-offs between security and performance. For example, a training code owner who wants to use the GPU to speed up the ML training computation might define conditions under which he trusts the OS. In such a case, a stakeholder defines a certificate chain permitting to verify the

Listing 1.1. Security policy example

```
1  name: training_owner/training_code
2  command: python /app/training.py
3  volumes:
4    - path: /training_data
5      import: training_data_owner/training_data_service
6    - path: /inference_model
7      export: inference_owner/inference_service
8  integrity_hash: {"0a11...bb3f"}
9  operating_system:
10   certificate_chain: |-
11     -----BEGIN CERTIFICATE-----
12     # certificate chain allowing
13     # verification of the secure
14     # element manufacturer
15     -----END CERTIFICATE-----
16   integrity:
17     measure0: e0f1...4be6
18     measure1: ae44...3a6e
19     measure2: 3d45...796d
20 stakeholder: |-
21   -----BEGIN PUBLIC KEY-----
22   # the policy owner's public key
23   -----END PUBLIC KEY-----
```

authenticity of a secure element attached to the computer (line 10) and expected integrity measurements of the OS (lines 16-19). The security policy manager only provisions the ML computations with the encryption key if the OS integrity (kernel sources and configuration) are trusted by the stakeholder. Specifically, the OS integrity measurements reflect what kernel code is running and whether it has enabled the required security mechanisms. Only then, the ML computations can access the confidential data and send it to the outside of the TEE, *e.g.*, GPU.

We discuss now and evaluate later (§5.2) two security levels that are particularly important for the ML computation. The first one, the *high-assurance security level*, fits well the inference because it offers strong security guarantees provided by the TEE, allowing the inference model to execute in an untrusted data center controlled by an untrusted operator. It comes at performance limitation, which is acceptable for inference because, typically, inference operates on much smaller data than ML training and does not need access to hardware accelerators. The high-assurance security level offers confidentiality and integrity of code and data at rest and in runtime. The trusted computing base (TCB) is low; It includes only the inference model executing inside the TEE, the hardware providing the TEE functionality, and the key distribution process. The second security level, the *high integrity level*, fits well the ML training because it enables access to hardware accelerators required for intensive computation. It comes at the cost of a larger TCB compared to the high-assurance security level because the code providing access to the hardware accelerators, *i.e.*, an operating system, must be trusted. PERUN relies on the TPM to establish trust with load-time kernel integrity and on IMA to extend this trust to the OS runtime integrity.

3.4 Hardware ML Accelerators Support

Typically, ML computations (*e.g.*, deep neural networks training) are extremely intensive because they must process a large amount of input data. To decrease the computation time, popular ML frameworks, such as TensorFlow [2], support

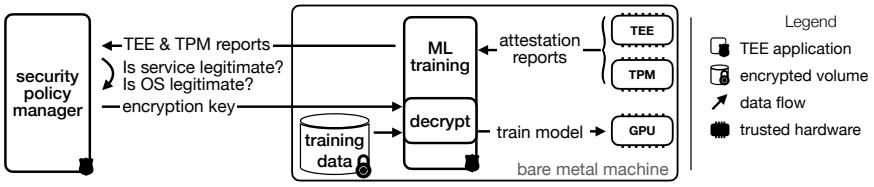

Fig. 3. The high-level overview of Perun supporting secure computation using hardware accelerator, *e.g.*, the GPU. Perun performs both the SGX and TPM attestation before provisioning the ML code with cryptographic keys. The successful TPM attestation informs that the legitimate OS with enabled integrity-enforcement mechanisms controls access to the GPU.

hardware accelerators, such as GPUs or Google tensor processing units (TPUs). Unfortunately, existing hardware accelerators do not support confidential computing, thus not offering enough security guarantees for the multi-stakeholder ML computation. For example, an adversary who exploits an OS misconfiguration [55] can launch arbitrary software to read data transferred to the GPU from any process executing in the OS. Even if ML computations execute inside the TEE enclaves, an adversary controlling the OS can read the data when it leaves the TEE, *i.e.*, it is transferred to the GPU or is processed by the GPU. Because of this, we design Perun to support additional security mechanisms protecting access to the data (also code and ML models) while being processed out of the TEE. This also allows stakeholders to trade-off between security level and performance they want to achieve when performing ML computations.

Figure 3 shows how Perun enables hardware accelerator support. ML computations transfer to the security policy manager a report describing the OS's integrity state. The report is generated and cryptographically signed by a secure element, *e.g.*, a TPM chip, physically attached to the computer. The security policy manager authorizes the ML computation to use the encryption key only if the report states that the OS is configured with the required security mechanisms. Precisely, the integrity enforcement mechanism, such as integrity measurement architecture (IMA) [45], controls that the OS executes only software digitally signed by a stakeholder. Even if an adversary gains root access to the system, she cannot launch arbitrary software that allows her to sniff on the communication between the ML computation and the GPU, read the data from the main memory, or reconfigure the system to disable security mechanisms or load a malicious driver. This also allows Perun to mitigate software-based micro-architectural and side-channel attacks [51,9,54,16], which are vulnerabilities of TEEs.

To enable hardware accelerator support, a stakeholder specifies expected OS integrity measurements inside the security policy (Listing 1.1, lines 9-19) and certificates allowing verification of the secure element identity. The OS integrity measurements are cryptographic hashes over the OS's kernel loaded to the memory during the boot process. A secure element collects such measurements during the boot process and certifies them using a private key linked to a certificate issued by its manufacturer. The certificate and integrity measurements are enough for the security policy manager to verify that the IMA enforces the OS integrity.

Although the hardware accelerator support comes at the cost of weaker security guarantees (additional hardware and software must be trusted when compared to a pure TEE-based approach), it greatly improves the ML training computation's performance (see §5.2).

3.5 Zero Code Changes

PERUN framework requires zero code changes to run existing ML computations, thus providing a practical solution for legacy ML systems. To achieve it, PERUN adapts platforms supporting running legacy applications inside the TEE, such as SCONE [5] or GrapheneSGX [49]. These platforms allow executing unmodified code inside the TEE by recompiling the code using dedicated cross-compilers or running them with a modified interpreter executing in the TEE.

3.6 Policy Deployment and Updates

A stakeholder establishes a transport layer security (TLS) connection to the security policy manager to deploy a policy. During the TLS handshake, the stakeholder verifies the identity of the security policy manager. The security policy manager owns a private key and corresponding certificate signed by an entity trusted by a stakeholder. For example, such a certificate can be issued by a TEE provider who certifies that given software running inside a TEE and identified by a cryptographic hash is the security policy manager. Some TEE engines, such as SGX, offer such functionality preventing even a service administrator from seeing the private key [29]. For other TEEs, a certificate might be issued by a cloud provider operating the security policy manager as part of cloud offerings.

PERUN requires that the security policy manager authorizes changes to the deployed policy. Otherwise, an adversary might modify the stakeholder's policy allowing malicious code to access the encryption key. In the PERUN design, the stakeholder includes his public key inside the digitally signed security policy. Since then, the security policy manager accepts changes to the policy only if a new policy has a signature issued with the stakeholders' private key corresponding to the public key present in the existing policy. By having a public key embedded in the security policy, other stakeholders can verify that the policy is owned by the stakeholder they cooperate with. The details of the policy security manager regarding key management, high availability, tolerance, and protection against rollback attacks are provided in [18].

4 Implementation

We implemented the PERUN prototype based on TensorFlow version 2.2.0 and the SCONE platform [5] because SCONE provides an ecosystem to run unmodified applications inside a TEE. We also rely on the existing key management system provided by the SCONE [50] and its predecessor [18] to distribute the

configuration to applications. We rely on Intel SGX [11] as a TEE engine because it is widely used in practice.

Our prototype uses a TPM chip [19] to collect and report integrity measurements of the Linux kernel loaded to the memory during a trusted boot [46] provided by tboot [10] with Intel trusted execution technology (TXT) [17]. The Linux kernel is configured to enforce the integrity of software, dynamic libraries, and configuration files using Linux IMA [45], a Linux kernel's security subsystem. Using the TPM chip, verifies that the kernel is correctly configured and interrupts its execution when requirements are not met.

We use an nvidia GPU as an accelerator for ML computation. The ML services are implemented in Python using TensorFlow framework, which supports delegating ML computation to the GPU.

4.1 Running ML Computations Inside Intel SGX

To run unmodified ML computations inside the SGX enclaves, we use the SCONE cross-compiler and SCONE-enabled Python interpreters provided by SCONE as Docker images. They allow us to build binaries that execute inside the SGX enclave or run Python code inside SGX without any source code changes.

The SCONE wraps an application in a dynamically linked loader program (*SCONE loader*) and links it with a modified C-library (*SCONE runtime*) based on the musl libc [39]. On the ML computation startup, the SCONE loader requests SGX to create an isolated execution environment (enclave), moves the ML computation code inside the enclave, and starts. The SCONE runtime, which executes inside the enclave along with the ML computations, provides a sanitized interface to the OS for transparent encryption and decryption of data entering and leaving the enclave. Also, the SCONE runtime provides the ML computations with its configuration using configuration and attestation service (CAS) [50].

4.2 Sharing the Encryption Key

We implement the security policy manager in the Perun architecture using the CAS, to generate, distribute, and share encryption keys between security policies. We decided to use the CAS because it integrates well with SCONE-enabled applications and implements the SGX attestation protocol [25]. Other key management systems supporting the SGX attestation protocol might be used [8,32] but require additional work to integrate them into SCONE.

We create a separate CAS policy for each stakeholder. The policy contains an identity of the stakeholder's IP (data, code, and models) and its access control and configuration. It is uploaded to CAS via mutual TLS authentication using a stakeholder-specific private key corresponding to the public key defined inside the policy. This fulfills the Perun requirement of protecting unauthorized stakeholders from modifying policies. The IP identity is defined using a unique per application cryptographic hash calculated by the SGX engine over the application's pages and their access rights. The SCONE provides this value during the application build process. The CAS allows for the specification of

the encryption key as a program argument, environmental variable, or indirectly as a key related to an encrypted volume. Importantly, the CAS allows defining which policies have access to the key. Thus, with the proper policy configuration, stakeholders share keys among enclaves as required in the PERUN architecture.

Our prototype uses the CAS encrypted volume functionality, for which the SCONE runtime fetches from the CAS the ML computation configuration containing the encryption key. Specifically, following the SGX attestation protocol, the SCONE runtime sends to CAS the SGX attestation report in which the SGX hardware certifies the ML computation identity. The CAS then verifies that the report was issued by genuine SGX hardware and the ML computation is legitimate. Only afterward, it sends to the SCONE runtime the encryption key. The SCONE runtime transparently encrypts and decrypts data written and read by the ML computation from and to the volume. The ML computations, *i.e.*, training and inference authorized by stakeholders via policies, can access the same encryption key, thus gaining access to a shared volume.

4.3 Enabling GPU Support with Integrity Enforcement

Our prototype implementation supports delegating ML computations to the GPU under the condition that the integrity-enforced OS handles the communication between the enclave and the GPU. The integrity enforcement mechanism prevents intercepting confidential data that leaves the enclave because it limits the OS functionality to a subset of programs essential to load the ML computation and the GPU driver. Thus, a malicious program cannot run alongside the ML computations on the same computing resources. We use trusted boot and TPM to verify it, *i.e.*, that the remote computer runs a legitimate Linux kernel with enabled integrity enforcement that limits software running on the computer to the required OS services, the GPU driver, and ML computations.

Trusted boot. Trusted computing technologies define a set of technologies that measure, report, and enforce kernel integrity. Specifically, during the computer boot, we rely on a trusted bootloader [10], which uses a hardware CPU extension [17] to measure and securely load the Linux kernel to an isolated execution environment [46]. The trusted bootloader measures the kernel integrity (a cryptographic hash over the kernel sources) and sends the TPM chip measurements.

The TPM stores integrity measurements in a dedicated tamper-resistant memory called platform configuration registers (PCRs). A PCR value cannot be set to an arbitrary value. It can only be extended with a new value using a cryptographic hash function: $PCR\_extend = hash(PCR\_old\_value \,||\, data\_to\_extend)$. This prevents tampering with the measurements after they are extended to a PCR. The TPM implements a TPM attestation protocol [20] in which it uses a private key known only to the TPM to sign a report containing PCRs. Our prototype uses the TPM attestation protocol to read the TPM report certifying that an ML computation executes on an integrity-enforced OS, *i.e.*, a Linux kernel with enabled IMA.

Integrity enforcement. IMA is a kernel mechanism that authenticates files

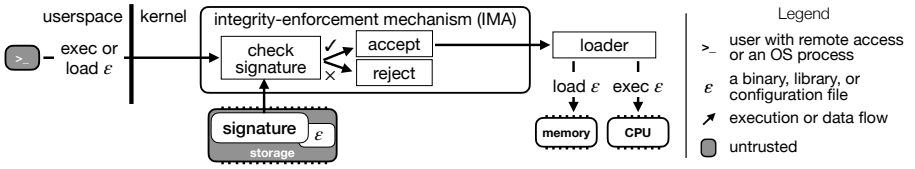

Fig. 4. The kernel integrity-enforcement system authenticates a file by checking its digital signature before loading it to the memory.

before allowing them to be loaded to the memory. Figure 4 shows how the IMA works. A process executing in userspace requests the kernel to execute a new application, load a dynamic library, or read a configuration file. IMA calculates the cryptographic hash over the file's content, reads the file's signature from the file's extended attribute, and verifies the signature using a public key stored in the kernel's *ima keyring*. If the signature is correct, IMA extends the hash to a dedicated PCR and allows the kernel to continue loading the file.

Trusted boot service. Because SCONE is proprietary software, we could not modify the SCONE runtime to provide the CAS with the TPM report. Instead, we implemented this functionality in a *trusted boot service* that uses the TPM to verify that the ML computations execute in the integrity-enforced OS.

The CAS performs the SGX attestation of the trusted boot service and provisions it with the TPM certificate as well as a list of the kernel integrity measurements. The trusted boot service reads the integrity measurements stored in PCRs using the TPM attestation protocol. The TPM genuineness is ensured by verifying the TPM certificate using a certificate chain provided by the CAS. The Linux kernel integrity is verified by comparing the integrity measurements certified by the TPM with the measurements read from the CAS.

We implemented the trusted boot service as an additional stage in the ML data processing. It enables other ML computations to access the confidential data only if the OS state conforms to the stakeholder's security policy. It copies the confidential data from an encrypted volume of one ML computation to a volume accessible to another ML computation after verifying the kernel integrity using the TPM. Our implementation is complementary with Linux unified key setup (LUKS) [7]. LUKS allows the kernel to decrypt the file system only if the kernel integrity has not changed. This prevents accessing the trusted boot service's volume after modifying the kernel configuration, *i.e.*, disabling the integrity-enforcement mechanism.

5 Evaluation

Testbed. Experiments were executed on a ASUS Z170-A mainboard equipped with an Intel Core i7-6700K CPU supporting SGXv1, Nvidia GeForce RTX 2080 Super, 64 GiB of RAM, Samsung SSD 860 EVO 2 TB hard drive, Infineon SLB 9665 TPM 2.0, a 10 Gb Ethernet network interface card connected to a 20 Gb/s switched network. Hyper-threading is enabled. The enclave page cache (EPC) is

configured to reserve 128 MB of RAM. CPUs are on the microcode patch level 0xe2. We run Ubuntu 20.04 with Linux kernel 5.4.0-65-generic. Linux IMA is enabled. The hashes of all OS files are digitally signed using a 1024-bit RSA asymmetric key. The signatures are stored inside files' extended attributes, and the certificate signed by the kernel's build signing key is loaded to the kernel's keyring during initrd execution.

Datasets. We use two datasets: (i) the classical CIFAR-10 image dataset [31], and (ii) the real-world medical dataset [47].

5.1 Attestation Latency

We run an experiment to measure the overhead of verifying the OS integrity using the TPM. Precisely, we measure how much time it takes an application implementing the trusted boot service to receive configuration from the security policy manager, read the TPM, and verify the OS integrity measurements.

The security policy manager executes on a different machine located in the same data center. It performs the SGX attestation before delivering a configuration containing two encryption keys – a typical setup for ML computations – and measurements required to verify the OS integrity. The security policy manager and the trusted boot service execute inside SCONE-protected Docker containers.

Table 1 shows that launching the application inside a SCONE-protected container takes 1573 ms. Running the same application that additionally receives the configuration from the security policy manager incurs 118 ms overhead. Additional 719 ms are required to read the TPM quote, verify the TPM integrity and authenticity, and compare the read integrity measurements with expected values provided by the security policy manager. As we show next, 2.5 sec overhead required to perform SGX and TPM attestation is negligible considering the ML training execution time.

5.2 Security and Performance Trade-off

To demonstrate the advantage of PERUN in allowing users to select the trade-off between security and performance, we compare the performance of different security levels provided by PERUN and the pure SGX based system called SecureTF [34]. We run the model training using the following setups: (i) only CPU (*Native*); (ii) GPU (*Native GPU*); (iii) PERUN, IMA enabled (PERUN*+IMA*); (iv) PERUN, IMA and SGX enabled (PERUN*+IMA+SGX*); (v) PERUN with GPU, IMA enabled (PERUN*+IMA+GPU*).

Table 1. End-to-end latency of verifying software authenticity and integrity using SGX and TPM attestation. Mean latencies are calculated as 10% trimmed mean from ten independent runs. *sd* stands for standard deviation.

	Execution time
Application in a container	1573 ms (sd=16 ms)
+ SGX attestation	1691 ms (sd=37 ms)
+ SGX and TPM attestation	2410 ms (sd=33 ms)

The *Native* and *Native GPU* levels represent scenarios where no security guarantees are provided. PERUN+*IMA* and PERUN+*IMA*+*GPU* represent the high integrity level (§3.3) in which ML training can execute directly on the CPU or GPU (high performance) while require to extend trust to the operating system (large TCB). Finally, PERUN+*IMA*+*SGX* represents the high-assurance security level where all computations are performed inside the TEE (limited performance) but requires a minimal amount of trust in the remote execution environment (low TCB). In all setups, the trusted boot service executes inside the enclave.

CIFAR-10 dataset. We perform training using the CIFAR-10 dataset, a convolutional neural network containing four *conv* layers followed by two fully connected layers. We use *BatchNorm* after each conv layer. We apply the *ADAM* optimization algorithm [28] with the learning rate set to 0.001.

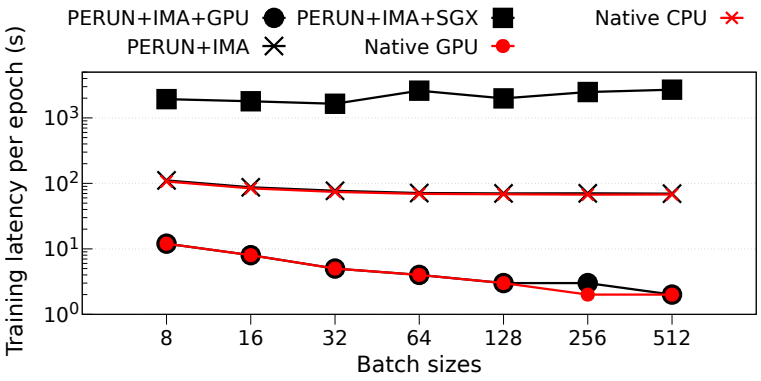

Fig. 5. The CIFAR-10 training latency comparison among different security levels offered by PERUN. Mean latencies are calculated from five independent runs.

Figure 5 shows the training latency, and Figure 6 shows the PERUN speedup depending on setups and batch sizes. At the high-assurance security level (PERUN+*IMA*+*SGX*), PERUN achieves almost the same performance as the pure SGX-based system, secureTF. This is because the training data is processed only inside the enclave, and SGX performs compute-intensive paging caused by the limited EPC size (128 MB) that cannot accommodate the training computation data (8 GB). PERUN+*IMA*+*GPU* and PERUN+*IMA* achieve 1321× and 40× speedup when relying just on the high integrity level compared to secureTF (batch size of 512). With these setups, the PERUN performance is similar to native systems (~ 0.96× of native latency) because the integrity protection mechanism performs integrity checks only when it loads files to the memory for the first time, leading to almost native execution afterward.

Real-world medical dataset. Next, we evaluate PERUN using a large-scale real-world medical dataset [47]. The dataset contains a wide range of medical images, including images of cancer and tumor treatment regimens for various

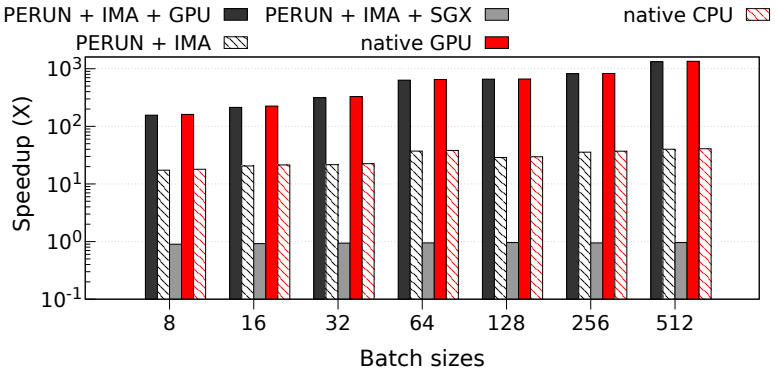

Fig. 6. The CIFAR-10 training speedup of evaluated systems in comparison to PERUN with the highest security level (PERUN + IMA + SGX).

parts of the human body, *e.g.*, brain, colon, prostate, liver, and lung. It was created via CT or MRI scans by universities and research centers from all around the world. We perform training over the brain tumor images dataset (6.1 GB) using the 2-D U-Net [44] TensorFlow architecture from Intel AI [4]. It makes use of the *ADAM* optimizer that includes 7 760 385 parameters with 32 feature maps. We set the learning rate to 0.001 and the batch size to 32.

Table 2 shows that at the high-assurance security level (the data is processed entirely inside the enclave), PERUN*+IMA+SGX* achieves the same performance as the referenced SGX-based system. However, when relying just on the high integrity level to protect the data, PERUN*+IMA+GPU* and PERUN*+IMA* achieve a speedup of 1559× and 47× compared to secureTF, respectively. We maintain the accuracy of 0.9875 in all experiments (dice coef: 0.5503, soft dice coef: 0.5503).

6 Related Work

Secure multi-party computation. Although cryptographic schemes, such as secure multi-party computation (MPC) and fully homomorphic encryption, are promising to secure multi-stakeholder ML computation, they have limited application in practice [52,42]. They introduce high-performance overhead [42,37,33,38,26], which is a limiting factor for computing-intensive ML,

Table 2. The training latency comparison among different security levels of PERUN, secureTF, and native. The results were obtained from a single run.

System	Latency per epoch	Speedup
Native CPU	5 h 26 min 14 sec	47×
Native GPU	9 min 54 sec	1561×
PERUN+IMA	5 h 26 min 17 sec	47×
PERUN+IMA+SGX	257 h 27 min 49 sec	~ 1×
PERUN+IMA+GPU	9 min 55 sec	1560×
secureTF	257 h 43 min 53 sec	(baseline)

and require to heavily modify existing ML code. Furthermore, they do not support all ML algorithms, such as, deep neural networks. Some of them also require additional assumptions, like MPC protocol requiring a subset of honest stakeholders. Unlike Perun, most of them lack support for training computation.

Secure ML using TEEs. Many works leverage TEE to support secure ML [21,23,42]. Chiron [23] uses SGX for privacy-preserving ML services, but it is only a single-threaded system. Also, it needs to add an interpreter and model compiler into the enclave. This incurs high runtime overhead due to the limited EPC size. The work from Ohrimenko et al. [42] also relies on SGX for secure ML computations. However, it does not allow using hardware accelerators and supports only a limited number of operators — not enough for complex ML computations. In contrast to these systems, Perun supports legacy ML applications without changing their source code. SecureTF [34] is the most relevant work for Perun because it also uses SCONE. It supports inference and training computation, as well as distributed settings. However, it is not clear how secureTF can be extended to support secure multi-stakeholders ML computation. Also, secureTF does not support hardware accelerators, making it less practical for training computation. Other works [48,40,6] use SGX and untrusted GPUs for secure ML computations. They split ML computations into trusted parts running in the enclave and untrusted parts running in the GPU. However, they require changing the existing code and do not support multi-stakeholder settings.

Trusted GPUs. Although trusted computation on GPUs is not commercially available, there is ongoing research. HIX [24] enables memory-mapped I/O access from applications running in SGX by extending an SGX-like design with duplicate versions of the enclave memory protection hardware. Graviton [53] proposes hardware extensions to provide TEE inside the GPU directly. Graviton requires modifying the GPU hardware to disable direct access to the critical GPU interfaces, e.g., page table and communication channels from the GPU driver. Telekine [22] restricts access to GPU page tables without trusting the kernel driver, and it secures communication with the GPU using cryptographic schemes. The main limitation of these solutions is that they require hardware modification of the GPU design, so they cannot protect existing ML computations, and they also do not support multi-stakeholder ML computations.

7 Conclusion

Perun allows multiple stakeholders to perform ML without revealing their intellectual property. It provides strong confidentiality and integrity guarantees at the performance of existing TEE-based systems. With the help of trusted computing, Perun permits utilizing hardware accelerators, reaching native hardware-accelerated systems' performance at the cost of a larger trusted computing base. When training an ML model using real-world datasets, Perun achieves 0.96× of native performance execution on the GPU and a speedup of up to 1560× compared to the state-of-the-art SGX-based system.

Acknowledgment

We thank the anonymous reviewers for their insightful comments and suggestions as well as Maksym Planeta and Hieu Le for their feedback and help. This work has received funding from the European Union's Horizon 2020 research and innovation program under the AI-Sprint project (ai-sprint-project.eu), grant agreement No 101016577.

References

1. Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (2016)
2. Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X.: TensorFlow: A system for large-scale machine learning. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16). pp. 265–283 (2016)
3. Abadi, M., Chu, A., Goodfellow, I., McMahan, H.B., Mironov, I., Talwar, K., Zhang, L.: Deep Learning with Differential Privacy. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS '16). p. 308–318 (2016)
4. AI, I.: Deep Learning Medical Decathlon Demos for Python. https://github.com/IntelAI/unet/ (accessed on 02/2021)
5. Arnautov, S., Trach, B., Gregor, F., Knauth, T., Martin, A., Priebe, C., Lind, J., Muthukumaran, D., O'Keeffe, D., Stillwell, M., Goltzsche, D., Eyers, D., Kapitza, R., Pietzuch, P., Fetzer, C.: SCONE: Secure linux containers with Intel SGX. In: 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16). pp. 689–703 (2016)
6. Asvadishirehjini, A., Kantarcioglu, M., Malin, B.: GOAT: GPU Outsourcing of Deep Learning Training With Asynchronous Probabilistic Integrity Verification Inside Trusted Execution Environment. arXiv preprint arXiv:2010.08855 (2020)
7. Broz, M.: LUKS2 On-Disk Format Specification, Version 1.0.0. In: LUKS documentation (2018)
8. Chakrabarti, S., Baker, B., Vij, M.: Intel SGX Enabled Key Manager Service with OpenStack Barbican. arXiv preprint:1712.07694 (2017)
9. Chen, G., Chen, S., Xiao, Y., Zhang, Y., Lin, Z., Lai, T.H.: SgxPectre: Stealing Intel Secrets from SGX Enclaves Via Speculative Execution. In: 2019 IEEE European Symposium on Security and Privacy (EuroS&P). pp. 142–157 (2019)
10. Corporation, I.: Trusted Boot (tboot). https://sourceforge.net/projects/tboot/ (accessed on May, 2021)
11. Costan, V., Devadas, S.: Intel SGX Explained. IACR Cryptol. ePrint Arch. **2016**(86), 1–118 (2016)
12. Emont, J., Stevens, L., McMillan, R.: Amazon Investigates Employees Leaking Data for Bribes. https://www.wsj.com/articles/amazon-investigates-employees-leaking-data-for-bribes-1537106401 (accessed on 02/2021)

13. Fredrikson, M., Jha, S., Ristenpart, T.: Model Inversion Attacks That Exploit Confidence Information and Basic Countermeasures. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security (CCS '15). p. 1322–1333 (2015)
14. Gallery, E., Mitchell, C.J.: Trusted Computing: Security and Applications. Cryptologia (2009)
15. Gentry, C.: Fully homomorphic encryption using ideal lattices. In: Proceedings of the 41st annual ACM symposium on Theory of computing. pp. 169–178 (2009)
16. Goetzfried, J., Eckert, M., Schinzel, S., Mueller, T.: Cache Attacks on Intel SGX. In: Proceedings of the 10th European Workshop on Systems Security (EuroSec '17) (2017)
17. Greene, J.: Intel trusted execution technology: Hardware-based technology for enhancing server platform security. Intel Corporation (2010)
18. Gregor, F., Ozga, W., Vaucher, S., Pires, R., Le Quoc, D., Arnautov, S., Martin, A., Schiavoni, V., Felber, P., Fetzer, C.: Trust management as a service: Enabling trusted execution in the face of byzantine stakeholders. In: 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '20). pp. 502–514 (2020)
19. Group, T.C.: TPM Library Specification, family "2.0", level 00, revision 01.38. In: TCG Resources, TPM 2.0 Library (2016)
20. Group, T.C.: TCG Trusted Attestation Protocol (TAP) Information Model for TPM Families 1.2 and 2.0 and DICE Family 1.0. Version 1.0, Revision 0.36 (2019)
21. Grover, K., Tople, S., Shinde, S., Bhagwan, R., Ramjee, R.: Privado: Practical and Secure DNN Inference with Enclaves. arXiv preprint arXiv:1810.00602 (2018)
22. Hunt, T., Jia, Z., Miller, V., Szekely, A., Hu, Y., Rossbach, C.J., Witchel, E.: Telekine: Secure Computing with Cloud GPUs. In: 17th USENIX Symposium on Networked Systems Design and Implementation (NSDI '20) (2020)
23. Hunt, T., Song, C., Shokri, R., Shmatikov, V., Witchel, E.: Chiron: Privacy-preserving Machine Learning as a Service. arXiv preprint arXiv:1803.05961 (2018)
24. Jang, I., Tang, A., Kim, T., Sethumadhavan, S., Huh, J.: Heterogeneous Isolated Execution for Commodity GPUs. In: Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS '19). p. 455–468 (2019)
25. Johnson, S., Scarlata, V., Rozas, C., Brickell, E., Mckeen, F.: Intel software guard extensions: EPID provisioning and attestation services. White Paper **1**(1-10), 119 (2016)
26. Juvekar, C., Vaikuntanathan, V., Chandrakasan, A.: GAZELLE: A Low Latency Framework for Secure Neural Network Inference. In: Proceedings of the 27th USENIX Conference on Security Symposium (USENIX Security). p. 1651–1668 (2018)
27. Khandaker, M.R., Liu, W., Naser, A., Wang, Z., Yang, J.: Origin-sensitive control flow integrity. In: 28th USENIX Security Symposium (USENIX Security '19). pp. 195–211 (2019)
28. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
29. Knauth, T., Steiner, M., Chakrabarti, S., Lei, L., Xing, C., Vij, M.: Integrating remote attestation with transport layer security. arXiv preprint arXiv:1801.05863 (2018)

30. Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., Schwarz, M., Yarom, Y.: Spectre attacks: Exploiting speculative execution. In: 2019 IEEE Symposium on Security and Privacy (SP). pp. 1–19. IEEE (2019)
31. Krizhevsky, A., Hinton, G.: Learning multiple layers of features from tiny images. Tech. rep., Citeseer (2009)
32. Kumar, A., Kashyap, A., Phegade, V., Schrater, J.: Self-Defending Key Management Service (SDKMS) with Intel Software Guard Extensions (SGX). White Paper (2018)
33. Kumar, N., Rathee, M., Chandran, N., Gupta, D., Rastogi, A., Sharma, R.: CrypTFlow: Secure TensorFlow Inference. In: IEEE Symposium on Security and Privacy (S&P '20). pp. 336–353 (2020)
34. Le Quoc, D., Gregor, F., Arnautov, S., Kunkeland, R., Bhatotia, P., Fetzer, C.: secureTF: A Secure TensorFlow Framework. In: Proceedings of the 21th International Middleware Conference (Middleware). p. 44–59 (2020)
35. Matsakis, N.D., Klock, F.S.: The rust language. ACM SIGAda Ada Letters pp. 103–104 (2014)
36. McKeen, F., Alexandrovich, I., Berenzon, A., Rozas, C.V., Shafi, H., Shanbhogue, V., Savagaonkar, U.R.: Innovative instructions and software model for isolated execution. In: Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy (HASP 13) (2013)
37. Mishra, P., Lehmkuhl, R., Srinivasan, A., Zheng, W., Ada Popa, R.: Delphi: A Cryptographic Inference Service for Neural Networks. In: 29th USENIX Security Symposium (USENIX Security '20). pp. 2505–2522 (2020)
38. Mohassel, P., Zhang, Y.: Secureml: A system for scalable privacy-preserving machine learning. In: 2017 IEEE Symposium on Security and Privacy (S&P '17). pp. 19–38 (2017)
39. muslc: musl libc. https://musl.libc.org (accessed on 02/2021)
40. Ng, L.K., Chow, S.S., Woo, A.P., Wong, D.P., Zhao, Y.: Goten: GPU-Outsourcing Trusted Execution of Neural Network Training and Prediction. 35th AAAI Conference on Artificial Intelligence (2019)
41. Noor, T.H., Sheng, Q.Z., Zeadally, S., Yu, J.: Trust Management of Services in Cloud Environments: Obstacles and Solutions. ACM Comput. Surv. (2013)
42. Ohrimenko, O., Schuster, F., Fournet, C., Mehta, A., Nowozin, S., Vaswani, K., Costa, M.: Oblivious Multi-Party Machine Learning on Trusted Processors. In: Proceedings of the 25th USENIX Conference on Security Symposium. p. 619–636 (2016)
43. Reuters: Ex-Microsoft employee charged with leaking trade secrets to blogger. https://www.reuters.com/article/us-microsoft-tradesecret-idUSBREA2J07K20140320 (accessed on 02/2021)
44. Ronneberger, O., Fischer, P., Brox, T.: U-net: Convolutional networks for biomedical image segmentation. In: International Conference on Medical image computing and computer-assisted intervention. pp. 234–241. Springer (2015)
45. Sailer, R., Zhang, X., Jaeger, T., Van Doorn, L.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: USENIX Security symposium. pp. 223–238 (2004)
46. Shin, J., Jacobs, B., Scott-Nash, M., Hammersley, J., Wiseman, M., Spiger, R., Wilkins, D., Findeisen, R., Challener, D., Desselle, D., Goodman, S., Simpson, G., Brannock, K., Nelson, A., Piwonka, M., Dailey, C., Springfield, R.: TCG D-RTM Architecture, Document Version 1.0.0. Trusted Computing Group (2013)

47. Simpson, A.L., Antonelli, M., Bakas, S., Bilello, M., Farahani, K., Van Ginneken, B., Kopp-Schneider, A., Landman, B.A., Litjens, G., Menze, B., et al.: A large annotated medical image dataset for the development and evaluation of segmentation algorithms. arXiv preprint arXiv:1902.09063 (2019)
48. Tramèr, F., Boneh, D.: Slalom: Fast, Verifiable and Private Execution of Neural Networks in Trusted Hardware. 7th International Conference on Learning Representations (ICLR) (2019)
49. Tsai, C.C., Porter, D.E., Vij, M.: Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In: Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference (USENIX ATC '17). p. 645–658 (2017)
50. UG, S.: SCONE Configuration and Attestation Service (CAS). https://sconedocs.github.io/CASOverview/ (accessed on 02/2021)
51. Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the keys to the intel SGX kingdom with transient out-of-order execution. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 991–1008 (2018)
52. Volgushev, N., Schwarzkopf, M., Getchell, B., Varia, M., Lapets, A., Bestavros, A.: Conclave: Secure Multi-Party Computation on Big Data. In: Proceedings of the 14th EuroSys Conference (EuroSys '19) (2019)
53. Volos, S., Vaswani, K., Bruno, R.: Graviton: Trusted Execution Environments on GPUs. In: Proceedings of the 13th USENIX Conference on Operating Systems Design and Implementation (OSDI '18). p. 681–696 (2018)
54. Weisse, O., Van Bulck, J., Minkin, M., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Strackx, R., Wenisch, T.F., Yarom, Y.: Foreshadow-NG: Breaking the Virtual Memory Abstraction with Transient Out-of-Order Execution. Technical report (2018)
55. Xu, T., Zhang, J., Huang, P., Zheng, J., Sheng, T., Yuan, D., Zhou, Y., Pasupathy, S.: Do Not Blame Users for Misconfigurations. In: Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles (SOSP '13) (2013)
56. Yao, A.C.: Protocols for secure computations. In: 23rd IEEE Annual Symposium on Foundations of Computer Science (SFCS 1982). pp. 160–164 (1982)
57. Zeller, A., Gopinath, R., Böhme, M., Fraser, G., Holler, C.: The fuzzing book (2019)
58. Zinzindohoué, J.K., Bhargavan, K., Protzenko, J., Beurdouche, B.: HACL∗: A verified modern cryptographic library. In: Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security. pp. 1789–1806 (2017)