Remote attestation of confidential VMs using ephemeral vTPMs

Vikram Narayanan*
University of Utah
USA
vikram@cs.utah.edu

Claudio Carvalho
IBM Research
USA
cclaudio@ibm.com

Angelo Ruocco
IBM Research
Switzerland
ang@zurich.ibm.com

Gheorghe Almási
IBM Research
USA
gheorghe@us.ibm.com

James Bottomley
IBM Research
USA
jejb@us.ibm.com

Mengmei Ye
IBM Research
USA
mye@ibm.com

Tobin Feldman-Fitzthum
IBM Research
USA
tobin@ibm.com

Daniele Buono
IBM Research
USA
dbuono@us.ibm.com

Hubertus Franke
IBM Research
USA
frankeh@us.ibm.com

Anton Burtsev
University of Utah
USA
anton.burtsev@utah.edu

Abstract

Trying to address the security challenges of a cloud-centric software deployment paradigm, silicon and cloud vendors are introducing *confidential computing* – an umbrella term aimed at providing hardware and software mechanisms for protecting cloud workloads from the cloud provider and its software stack. Today, Intel Software Guard Extensions (SGX), AMD secure encrypted virtualization (SEV), Intel trust domain extensions (TDX), etc., provide a way to shield cloud applications from the cloud provider through encryption of the application's memory below the hardware boundary of the CPU, hence requiring trust only in the CPU vendor. Unfortunately, existing hardware mechanisms do not automatically enable the guarantee that a protected system was not tampered with during configuration and boot time. Such a guarantee relies on a hardware root of trust, i.e., an integrity-protected location that can store measurements in a trustworthy manner, extend them, and authenticate the measurement logs to the user (remote attestation).

In this work, we design and implement a virtual trusted platform module (vTPM) that virtualizes the hardware root of trust without requiring trust in the cloud provider. To ensure the security of a vTPM in a provider-controlled environment, we leverage unique isolation properties of the SEV-SNP hardware that allows us to execute secure services (such as vTPM) as part of the enclave environment protected from the cloud provider. We further develop a novel approach to vTPM state management where the vTPM state is not preserved across reboots. Specifically, we develop a stateless *ephemeral* vTPM that supports remote attestation without any persistent state on the host. This allows us to pair each confidential VM with a private instance of a vTPM completely isolated from the provider-controlled environment and other VMs. We built our prototype entirely on open-source components – Qemu, Linux, and Keylime. Though our work is AMD-specific, a similar approach could be used to build remote attestation protocols on other trusted execution environments (TEE).

ACM Reference Format:
Vikram Narayanan, Claudio Carvalho, Angelo Ruocco, Gheorghe Almási, James Bottomley, Mengmei Ye, Tobin Feldman-Fitzthum, Daniele Buono, Hubertus Franke, and Anton Burtsev. 2023. **Remote attestation of confidential VMs using ephemeral vTPMs**. In *Annual Computer Security Applications Conference (ACSAC '23), December 04–08, 2023, Austin, TX, USA.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3627106.3627112

1 Introduction

Over the last two decades, public clouds have become an inescapable building block of virtually every modern application. The move to the cloud created a unique security challenge. Both application vendors and end-users are required to trust the cloud infrastructure that is often in charge of handling security and privacy-sensitive data. Such trust is fragile as multi-tenant cloud environments are operated by third party providers and include a large and complex virtualization and storage stacks optimized for a wide variety of hardware and software execution scenarios. Unfortunately, vulnerabilities in critical cloud software and infrastructure are unavoidable.

In the last decade, three widely deployed virtual machine monitors (VMMs) – Xen, KVM, and VMware – that provide the foundation of isolation and security in the cloud suffered from 428 [26], 111 [10] and 154 [24] vulnerabilities each. Cloud software stacks like Openstack and Cloudstack suffer from several vulnerabilities, some resulting in total information disclosure and rendering resources unusable [27, 28]. Moreover, physical access to the system opens the door for a range of hardware attacks, e.g., memory extraction such as cold-boot [80], RAMBleed [57, 74], etc.

In an effort to minimize the TCB of cloud applications, hardware vendors and some cloud providers have introduced support for hardware-protected trusted execution environments (TEEs) [3, 8, 35, 46, 54]. TEEs protect data in use from the host software stack including the hypervisor and even the physical attacker. In effect, TEEs remove the cloud provider from the TCB, even though the provider still manages the lifecycle of an application.

Isolation alone, however, is not sufficient to protect a workload or sensitive data. To ensure integrity, modern systems rely on a combination of *measured boot* [65, 81] and *runtime attestation* [42, 53]. A measured boot protocol performs measurement of all binaries involved in the boot of the system to ensure the integrity of all boot-time components, i.e., the platform firmware, bootloader(s),

*Work done while at IBM Research

ACSAC '23, December 04–08, 2023, Austin, TX, USA

ACM ISBN 979-8-4007-0886-2/23/12.
https://doi.org/10.1145/3627106.3627112

and the operating system kernel. Runtime attestation combines measured boot with integrity measurement architecture (IMA) that ensures integrity measurements of all binaries loaded and executed by the system after it booted, i.e., dynamic kernel extensions, system binaries, etc. Attestation works by comparing entries in the measured boot and IMA logs with a pre-defined set of acceptable values (called an *attestation policy*) and exposing any measurements that do not conform to policy expectations.

Support for attestation requires a *root of trust device*, i.e., an integrity-protected location that can store measurements in a trustworthy manner, extend them, and authenticate the measurement logs to the user (remote attestation). On a physical machine, a trusted platform module (TPM) chip can be used as the root of trust. Some cloud providers offer virtual machines with virtual TPMs (vTPMS) attached to them [2, 6, 12, 22]. These vTPMs, however, are emulated by the host virtualization stack. Using this kind of emulated device requires trusting the service provider, which is at odds with confidential computing. In this paper, we show how to implement a confidential vTPM emulated inside a TEE, isolated from both host and guest, linked to the root of trust of the enclave, and providing similar properties to a physical TPM.

In this work, we design and implement a new virtual trusted platform module (vTPM) that virtualizes the hardware root of trust without requiring trust in the cloud provider. To ensure the security of a vTPM in a provider-controlled environment, we leverage unique isolation properties of the SEV-SNP hardware that allows us to execute secure services (such as vTPM) as part of the enclave environment protected from the cloud provider. We further develop a novel approach to the vTPM state management where the vTPM state is not preserved across reboots. Specifically, we develop a stateless *ephemeral* vTPM that supports remote attestation without a persistent state on the host. This allows us to pair each confidential VM with a private instance of a vTPM that is completely isolated from the provider-controlled environment and other VMs.

We design vTPM around the following security requirements:

- **Isolation**: Physical TPMs are isolated at the hardware level. Typical vTPMs emulated on the host are isolated from the guest via virtualization, but exposed to the trusted host. In addition, the vTPM also needs isolation from the guest operating system, since it acts as a root of trust device for attestation. A vTPM should be isolated from both the host and the guest system.
- **Secure communication**: In a physical TPM, communication is isolated at the hardware level, although these assurances can sometimes be subverted [1, 40]. In a typical vTPM, the TPM commands and responses are transmitted through the untrusted hypervisor [33, 38, 55, 61, 70]. An attacker can interpose on the channel and alter the request or response defeating the security guarantees offered by a TPM [40]. Communication with vTPM should be secure.
- **Persistent state**: Physical TPMs have a persistent identity that is set when the device is manufactured. Maintaining persistent state in a virtualized environment usually requires a centralized management system to propagate and store vTPM state. The management system is part of the TCB and is usually managed by the cloud provider. vTPM's state should be managed by the client and protected from the cloud provider.

To implement isolation, we leverage unique properties of the SEV-SNP execution environment. Our confidential vTPM is emulated inside the SEV-SNP enclave (hence it is isolated from the host and the cloud provider). Moreover, we leverage Virtual Machine Privilege Levels (VMPLs) to isolate vTPM from the guest and hence ensure the integrity of remote attestation. Since our confidential vTPM is emulated inside the guest security context, the guest and vTPM can communicate in plaintext without information being exposed to the untrusted host. Moreover, we ensure that neither the guest nor the hypervisor can tamper with the communication. To avoid exposing sensitive vTPM state to a complex management system, we develop a new ephemeral approach to vTPM state management, in which the state of the vTPM never leaves the enclave.

The above security properties allow us to implement a vTPM that is comparable in security and functionality to a physical TPM. Our vTPM does not violate the trust model of confidential computing and extends existing measurement capabilities to support sophisticated attestation flows, enabling the creation of cloud-native workloads with a small TCB that can be rigorously audited.

Our work leverages the unique architectural properties of the AMD SEV-SNP execution environment; however, we will discuss how to generalize this solution at the end of the paper. We will also expand on the properties of the ephemeral vTPM, which does have certain restrictions. The limitations of an ephemeral vTPM do not affect the attestation use cases described here.

Overall, we make the following contributions: We propose using an ephemeral vTPM to remove attacks to the vTPM state. We are the first to leverage the new features of AMD SEV to provide a secure implementation of a vTPM. Finally, we demonstrate a complete remote attestation workflow for our SVSM-vTPM solution, implicitly proving that remote attestation frameworks can provide measured boot and remote attestation with an ephemeral vTPM.

2 Background and related work

2.1 Trusted execution environments

Ubiquitous nature of cloud computing as a de facto large-scale application deployment paradigm resulted in a new security challenge – protecting sensitive user data in a large, complex, and potentially untrusted environment of a cloud provider. To address the growing security concerns, academic [47] and industry [30, 45] projects proposed the idea of trusted execution environments (TEEs) in which the execution of a user program can be shielded from the rest of the software and hardware stack of the cloud provider. TEEs provide isolated environments, or *enclaves*, that ensure confidentiality and integrity of the user workload by relying only on the CPU.

Intel SkyLake architecture introduced software guard extensions (SGX) that implement secure enclave for user-level applications through a combination of novel architectural extensions and CPU microcode. SGX suffered from numerous vulnerabilities [63], ranging from access to the secrets inside the enclave to extracting the quoting enclave's attestation keys that allowed attackers to forge attestation reports [76].

In 2016, AMD introduced secure encrypted virtualization (SEV), where the entire virtual machine — as opposed to just part of an application — was encrypted with an ephemeral key managed by a dedicated co-processor, AMD secure processor (AMD-SP). AMD-SP

takes care of the lifecycle management of the SEV VMs [31] and serves as the integrated root of trust for the AMD processor [59]. By using a unique key per VM, SEV isolates the guest VMs from the rest of the host operating system and from other guests.

Intel trust domain extensions (TDX) introduced their own version of hardware-isolated encrypted virtual machines called trusted domains (TDs). Intel TDX relies on an SGX-based quoting enclave called the TD-quoting enclave to perform remote attestation of trusted domains [8]. Unfortunately, the attestation keys used by the quoting enclave are long-lived, and when leaked, affect millions of devices.

ARM introduced confidential compute architecture (CCA) with their Armv9-A architecture, where the processor provides an isolated hardware execution environment called *Realms*, for hosting entire VMs in a secure space [35]. Similar to other TEEs [3, 8] ARM CCA provides launch measurement for the realms and can do measured boot with their hardware enforced security (HES) module specification [34] which serves as the root of trust [36, 37].

AMD secure encrypted virtualization Since 2016, AMD has incrementally added additional protection features to SEV. SEV-ES (SEV encrypted state) protects the register state in the virtual machine control block (VMCB) with encryption and integrity protection [16]. To communicate and share data with the hypervisor during hypercalls, guest hypervisor communication block (GHCB) was introduced [19] that would remain unencrypted. Finally, with SEV-SNP (secure nested paging), AMD introduced a reverse mapping table (RMP) which performs page validation and keeps track of page ownership to prevent replay attacks [5].

Virtual machine privilege levels To avoid relying on the host infrastructure for running secure services for the confidential VM, AMD also introduced virtual machine privilege levels (VMPLs) in SEV-SNP. Similar to protection rings in x86 architecture, VMPLs allow a guest VM address space to be subdivided into four levels with different privileges (with VMPL0 being the highest privilege level). VMPLs allow design and deployment of secure services that are completely isolated from the untrusted host operating system and the guest VM [5].

To standardize the communication between various services offered by the software running at VMPL0 and the guest operating system AMD introduced a specification called Secure VM service module (SVSM) [17]. The protocol uses registers to pass the arguments and return values. In the absence of SVSM firmware, the entire guest VM can execute under VMPL0 unmodified. However, with SVSM, they run at a lower privilege level, corresponding to a higher VMPL (i.e., 1-3), and require interaction with the SVSM for some privileged operations.

2.2 Integrity

TEEs ensure confidentiality of the workload but do not guarantee integrity. The trusted platform module (TPM) is used along with a TEE to implement a secure root of trust in hardware. A TPM measures and records the cryptographic hash of the software during the boot process and reliably verifies the same at a later point in time. TPM is implemented as a cryptographic co-processor chip that is embedded on the motherboard of a platform. It provides several cryptographic operations (e.g., encryption, signing, hashing) and secure storage for small data such as keys.

Measured boot Measured boot is the process of recording the measurements of all boot components during the system initialization process. Hashes of all components are recorded in a log file that is authenticated using the TPM. This authentication works by extending TPM's Platform Configuration Registers (PCRs) with digests of individual events in the boot log. A TPM-signed *quote* is used to vouch for the accuracy of the log.

Runtime integrity Integrity measurement architecture (IMA) is a Linux subsystem that implements the idea of measured boot after the system is booted, e.g., measures hashes of all kernel extensions before they are executed [69]. Together with measured boot, IMA enables a remote attestation protocol to ensure the runtime integrity of the system. Specifically, it allows an outside observer to ascertain specific properties of a set of devices/machines. As an example, one might be interested to ascertain the booted kernel, on a set of machines in a data center. These properties of interest are cumulatively called an attestation policy. To ensure the integrity of the measurements, IMA relies on the TPM, i.e., extends the measurements into the TPM PCRs, similar to the measured boot log.

Measured boot and remote attestation are designed to stop an attacker who has control over the boot sequence of a system, e.g., an untrusted cloud provider, or an attacker who gains administrative privileges and can load malicious kernel extensions, or downgrade security critical subsystems to exploitable versions. These mechanisms complement a number of security mechanisms aimed to prevent runtime exploitation of the system through a range of low-level vulnerabilities [41], e.g., stack canaries [48], address space randomization (ASLR) [71], data execution prevention (DEP) [77], superuser-mode execution and access prevention [43, 49], and even control-flow [45] and code-pointer integrity [56].

Virtual trusted platform module (vTPM) A vTPM is a pure software implementation of a TPM module as defined by the TPM 2.0 specification [20]. vTPM enables the virtualization of a hardware root of trust across multiple entities, i.e., virtual machines, and is aimed at providing functionality identical to a hardware TPM. Berger et al. [38] proposed the first design for virtualizing a TPM that can be used for providing TPM functionalities to virtual machines. Their design consists of a vTPM manager and a set of vTPM instances, where the vTPM manager executes as part of the VMM and takes care of multiplexing physical hardware across multiple VMs. Berger et al. extend the TPM command specification to include support for creating virtual instances and rely on hardware TPM for establishing trust.

Stumpf et al. [72] proposed a virtual TPM design by applying hardware virtualization techniques from Intel VT-x technology. Their multi-context TPM contains different modes of execution and has a dedicated TPM control structure for every VM, which would be loaded by the VMM before invoking the TPM commands. Several vTPM architectures were proposed over the years: from a generalized vTPM [70] to separating vTPM functionalities across Xen domains with different privileges [33, 55, 61]. Unfortunately, existing designs either place trust on the host environment (VMM, host OS) or rely on the hardware TPM for establishing trust. None of those designs satisfies the security and confidentiality requirements of confidential computing. Recent vTPM designs move their implementation inside a TEE such as Intel SGX [66, 73, 78, 79].

Though this design offers protection from the cloud provider, the state of the TPM must be securely stored and should be protected against rollback attacks. Additionally, to avoid substitution attacks, both the vTPM and the consuming VM must securely identify each other before services can be provided.

Cloud vTPMs Cloud providers that offer confidential VMs typically provide virtual TPM device that can serve as a root of trust and can also be used for remote attestation. Google cloud offers plain SEV confidential VMs and measured boot attestation via a vTPM managed by the hypervisor [23]. Microsoft Azure cloud relies on Azure attestation service for attesting confidential VMs [12] that generates a token to decrypt the vTPM state and the disk. Alibaba cloud offers vTPM support on their elastic compute service VMs [2]. Amazon AWS provides Nitro TPM, a virtual TPM implementation conforming to the TPM 2.0 specification as part of their EC2 offering [6]. Some of these providers use a qemu-backed vTPM that runs on the host, and requires trust in the cloud provider. Additionally, there is very limited public knowledge about the design and implementation of the above cloud vTPMs what limits understanding of their security guarantees. In contrast, our work results in an openly available SVSM-vTPM implementation that is built on top of other standard opensource components (i.e., Qemu, Linux, and Keylime). As our SVSM-vTPM relies only on the hardware-protected isolation environment offered by the AMD-SP hardware, it allows cloud users to leverage our vTPM as SVSM firmware and hence completely eliminate the need for trusting the cloud provider.

TEE-based vTPMs Table 1 presents a summary of differences between our SVSM-vTPM design and other TEE-based vTPMs. CoCoTPM proposes a unified architecture for attestation of confidential VMs where the hypervisor launches a confidential VM that acts as a vTPM manager and handles all the vTPM instances [66]. Several other projects rely on running vTPM under isolation provided by other hardware TEE mechanisms such as Intel SGX [73, 78, 79] and ARM Trustzone [67]. SvTPM aims to protect against NVRAM replacement, and rollback attacks [78] by running the vTPM inside an SGX enclave for KVM-based VMs, whereas eTPM manages several enclave vTPMs in a Xen environment and relies on a physical TPM to provide root of trust [73], similar to Berger et al. [38].

To estasblish root of trust, SvTPM relies on Intel SGX datacenter attestation primitives (DCAP) mechanism whereas CoCoTPM uses a self-signed certificate with which they sign the EK. SVSM-vTPM establishes a chain of trust by generating an SEV-SNP attestation report by passing the $digest(EK_{pub})$ as the user-data along with the attestation request and thus relying only on the AMD hardware.

Both SvTPM and CoCoTPM persists the state of the TPM. SvTPM leverages SGX sealing to tie the persistent state of the TPM to the appropriate VM whereas CoCoTPM stores the state encrypted on the host such that it can only be decrypted by the CoCoTPM. In contrast, by implementing an ephemeral vTPM, we completely eliminate the classes of attacks that come with state protection and endpoint substitution.

Both CoCoTPM and SvTPM require modifying parts of the software stack to implement transport layer security (TLS) for securing the communication channel between a VM and its vTPM. We implement an interface where both the command request and response are part of the encrypted VM pages and hence secure by design.

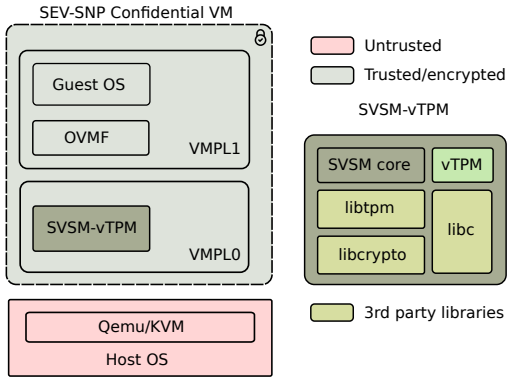

Figure 1: SVSM-vTPM architecture and its components

To manage the state machine of the vTPM instance and to maintain the association betweeen a VM and its vTPM, SvTPM and CoCoTPM take different approaches. SvTPM follows a decentralized model where each vTPM instance is hosted on a separate SGX enclave whereas CoCoTPM employs a central vTPM manager where multiple vTPM instances are hosted on the same CoCoTPM confidential VM. Though the CoCoTPM VM is running inside a confidential VM, a central design suffers from several attacks ranging from denial of service to colluding with other confidential VMs. Though it is possible to launch a dedicated CoCoTPM for every confidential VM, it results in wastage of architectural resources as the number of address space identifiers (ASIDs) are limited. In contrast, our SVSM-vTPM architecture equips each confidential VM with their own private vTPM instance by leveraging the SVSM architecture that implements VM privilege levels. We propose a minimalistic vTPM design that avoids the need to support secure communication and management of persistent state. Also, by having a self-contained design and a simple API interface for performing remote attestation, we avoid the complexities that are associated with orchestrating a remote attestation protocol [44].

3 Threat model

We assume that an attacker has physical access to the machine and unrestricted privileges on the software and firmware executing on the host machine, i.e., firmware, hypervisor and virtualization stack, and the host operating system. However, the memory of the confidential VM is protected by the AMD SEV technology, i.e., encrypted with a key only known to the AMD secure processor (AMD-SP). We trust the AMD hardware and the implementation of SEV-SNP and SVSM. We also assume the attacker can interact with software services hosted on the confidential VM (e.g., a key-value store, e-commerce platform, etc) and can potentially exploit vulnerabilities in the guest kernel (e.g., by crafting a malicious network packet).

Ciphertext side channel attacks [58, 60] on the SEV encrypted VM (by building a dictionary of plaintext-ciphertext pairs) are out of scope. Attacks against the integrity measurement architecture (IMA) such as TOCTOU [39], other measurement gaps such as code injected by extended berkeley packet filter (eBPF) are out of scope. Also, runtime attacks exploiting stack or heap overflows such as return-oriented programming on the guest VM are out of scope as IMA measures only the persistent files.

Table 1: Feature comparison of SVSM-vTPM and other TEE-based vTPMs

	Trust anchor	Persistent State	Secure communication	Rollback protection	TPM management
SvTPM [78]	SGX ->vTPM	Encrypted (SGX-Seal)	SSL	Yes	Self-contained
CocoTPM [66]	Self-signed sub-CA	Encrypted (on disk)	SSL	Yes	Central
SVSM-vTPM	AMD ->vTPM	Ephemeral	Secure by design	N/A	Self-contained

4 TPM virtualization with SVSM

SVSM-vTPM is a secure virtual TPM designed to enable remote attestation and runtime integrity measurement in a provider-controlled confidential computing environment backed by an AMD SEV hardware. Specifically, we do not trust any software on the host machine. To achieve strong isolation from the host, we leverage unique capabilities of AMD SEV environment and execute a virtual TPM instance along with the guest system inside a hardware-protected TEE enclave (Figure 1). The entire SEV-SNP confidential VM memory is encrypted by the AMD-SP. SVSM-vTPM runs inside the VM privilege level 0 (VMPL0), which allows us to both isolate it from the rest of the guest system and provide secure communication between the guest and the TPM. Specifically, we load a minimal bare-metal execution environment in VMPL0 when a new confidential VM is created. Finally, we completely eliminate the burden of TPM state management such as preserving the state, injecting it to the correct confidential VM during boot-up, and also prevent a whole class of attacks based on exfiltration of the TPM state with a novel idea of an ephemeral TPM – our TPM instances have no persistent state to save or guard against.

4.1 Isolation

As vTPM offers a virtual root of trust for the virtual machine, it has to be hosted in an environment that provides strong isolation of its state and is designed to minimize the attack surface for a potential attacker. Arguably, two design flaws undermine the security of existing vTPMs to be used in a confidential computing environment.

First, until recently, the cloud provider was a de facto part of the trust domain. vTPMs were often managed and implemented as a component inside the hypervisor [38] or as a part of the virtualization stack [33, 55, 61]. To reduce the attack surface on the component hosting the vTPM, several alternative vTPM architectures were proposed. Triglav vTPM utilized dynamic root of trust (DRTM) as a mechanism to ensure the integrity of the hypervisor [64]. Another vTPM solution utilized x86 system management mode (SMM) for isolation and protection of the TPM [62]. Though such designs offer some form of protection against a non-malicious cloud environment, they do not satisfy the requirements of confidential computing where the entire host environment is untrusted. Recent TEE-based vTPMs run the vTPM manager and several instances in a hardware isolated TEE such as SGX [73, 78, 79], AMD SEV confidential VM [66] or in ARM Trustzone [67].

Second, historically, virtualization of TPM relied on a centralized architecture. The core part of the vTPM, a vTPM manager, responsible for instantiating a TPM, multiplexing the communication between multiple VMs and vTPMs, and saving the TPM state in a secure location was shared across all vTPM instances [33, 38, 55, 61, 66]. As the manager handles the lifecycle of all vTPMs on a machine and has access to the physical TPM hardware, it naturally becomes a central point for attack. A malicious VM can launch attacks ranging from a simple denial-of-service to sophisticated attacks trying to exfiltrate the secrets by exploiting the vulnerabilities in a centralized vTPM manager. If exploited, the security of all the vTPMs handled by the manager is compromised.

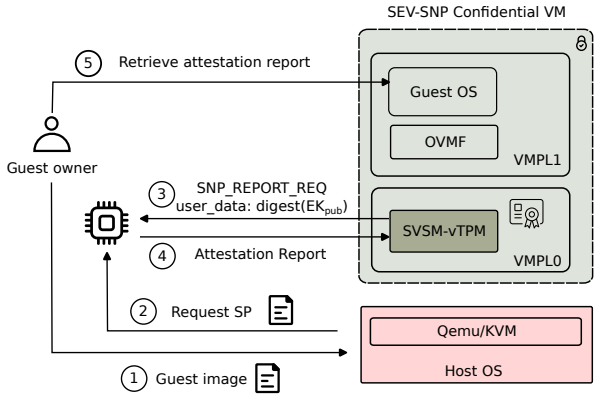

Figure 2: Generating SEV-SNP attestation report inside SVSM-vTPM

Private, isolated TPMs Instead of relying on a central vTPM manager that manages several instances of vTPM in an untrusted environment, we base our design on two insights. First, to provide strong isolation of the vTPM code, we leverage the architectural support offered by AMD SEV. Second, to avoid centralized management, we rely on SVSM specification that offers a way to implement secure services inside the guest VM.

Specifically, to ensure isolation, we leverage the VM privilege levels inside the confidential VM address space provided by the SVSM specification as part of the SEV-SNP architecture. In our architecture, every confidential VM has its own private vTPM that runs at a higher privilege level (i.e., VMPL0) inside each confidential VM and is encrypted by AMD-SP and has the same isolation guarantees of an encrypted VM.

By running our vTPM within an isolated privilege level within the guest address space, we eliminate all the attacks that could be mounted on the component that runs the vTPM. Additionally, operating at the VMPL0 offers additional protection that it cannot be interfered by the guest or the host OS.

We use Qemu/KVM environment for running the confidential VM. Figure 2 shows how a confidential VM is launched. A user provides the boot-time binaries (typically SVSM and OVMF) to be loaded as part of the guest image (①). Qemu communicates with the KVM which communicates with the SEV firmware running inside the AMD-SP through an API interface to create a confidential VM (②). The SVSM firmware is placed in VMPL0 and the OVMF firmware and the rest of the guest environment (i.e., the kernel and initrd in case of direct boot) is placed at VMPL1.

Unlike a regular programming environment that provides operating system abstractions (e.g., syscalls, timers, etc) and feature-rich libraries, SVSM firmware runs on a restrictive bare-metal environment without access to such features. Enclave environments often

come with such restrictions, for instance, one would need a sophisticated library OS [75] to run unmodified applications inside SGX. However, in a bare-metal environment such as SVSM, one does not have operating system abstractions such as timers/clocks, availability of crypto libraries, etc. However, a vTPM needs to have access to timers, random numbers, and cryptographic libraries for realizing a software TPM module. We manually port the necessary libraries to satisfy the dependencies of the TPM module. Due to the encrypted code pages and the lack of interfaces between the debugger and Qemu to install breakpoints inside the encrypted pages, we had to rely on print statements during development for debugging.

4.2 Communication between VM and vTPM

The communication channel between a VM and a corresponding vTPM is a potential target for a range of security attacks, e.g., by altering the TPM command requests and response buffers it is possible to subvert measured boot and runtime attestation protocols [40]. One way to mitigate such attacks is to secure the communication channel by implementing standards such as TPM HMAC and encryption [20] or DMTF secure protocol and data model (SPDM) specification [18]. Though the TPM specification describes encryption and HMAC security layers, there are very few TPM implementations that support them. Developing a complex secure communication protocol such as SPDM requires a large engineering effort. Recent vTPM designs that rely on a hardware-protected TEEs implement secure communication channel using transport layer security (TLS) protocol [15]. Unfortunately, even a standard TLS protocol negatively affects the TCB size of the TPM.

Secure communication Instead of implementing a secure communication protocol, we rely on the mechanism provided by AMD SEV and its ability to pass execution between virtual machine privilege levels. While the transition between VMPL1 and VMPL0 triggers an exit into an untrusted hypervisor controlled by the cloud provider, the internals of the message remain protected inside the hardware-encrypted memory. Moreover, AMD SEV specification ensures that the hypervisor can only resume execution of the VM at a corresponding privilege level, i.e., VMPL0, if the guest system triggers an exit into the hypervisor. Hence, the hypervisor is unable to suppress messages unless the whole VM is halted.

We rely on a generic platform device to interact with the SVSM-vTPM which simply uses a page in memory for communicating the request and response between the confidential VM and the SVSM-vTPM [29]. The guest kernel triggers an exit into the hypervisor, after every write to the TPM command page. Upon re-entry, the hypervisor puts the vCPU in VMPL0 where SVSM-vTPM handler looks for TPM command ready flag and inturn invokes the appropriate TPM command API to formulate the response buffer. Then, the vCPU exits into VMPL1 and continues with the execution of the guest VM. We also make modifications to the TPM driver in OVMF to interact with our SVSM-vTPM.

4.3 vTPM state

A discrete physical TPM stores all the persistent state of the module inside the chip's non-volatile (NV) store which holds the seeds for generation of endorsement key (EK), storage root key (SRK) and also retains other values such as NV Index values, objects made persisted by the TPM user, and state saved when TPM is shutdown. The TCG specification requires a TPM implementation to have some amount of non-volatile storage for the operation of the TPM [20].

As opposed to a physical TPM where the state of the TPM is securely stored inside the TPM hardware chip inside a non-volatile RAM (NVRAM), a vTPM must manage its state in software. Software vTPMs typically implement the NV store in a disk-backed file [33, 38, 55, 61, 66, 67, 78]. Along with the software that implements the vTPM, this NVRAM file is part of the trusted computing base. When a vTPM is first initialized, the state file has to be created on-the-fly or loaded from a file that is pre-created.

However, the state stored in the file needs to be secured against tampering and rollback attacks [50]. This could be achieved by encrypting the NV store file such that it could be decrypted only by the vTPM module. This design calls for securely storing the secret key used to encrypt/decrypt the NV state and inject it as a secret during the boot-up of vTPM module. This brings in several complexities in the context of confidential computing as the secret could only be injected during the launch phase. First the user has to verify the *launch measurement* of the load-time components (i.e., firmware, OVMF, etc.) before delivering the encrypted TPM state along with the key to decrypt the TPM state. The booting of the platform is blocked, waiting for the user to inject the secret. Additional care has to be taken to not give up the state to a confidential VM that is under the control of an attacker.

Ephemeral vTPM Instead, our design choice of using an ephemeral vTPM is much more simplistic and pragmatic. The vTPM goes through the manufacturing process to generate a fresh set of seeds, keys on every boot. We avoid all the problems of handling persistent state, injecting it on every boot, and guarding the encrypted state file by designing an ephemeral vTPM with no state. First, ephemeral vTPM is simple to implement: the NV storage becomes a volatile storage and does not preserve any values across power cycles. Second, it does not require any form of secrets to boot-up the vTPM and the platform. Though there are downsides to this design such as: secrets cannot be preserved across reboots, this offers much more flexibility as there is no secret to guard against the aforementioned attacks. Moreover, the programming environment for SVSM is extremely constrained in terms of capabilities. To save the TPM state on shutdown and to load the state on a reboot, the SVSM should implement additional software to encrypt and decrypt the state file.

4.4 SVSM-vTPM provisioning

After launching the confidential VM, the hypervisor first loads and executes the SVSM binary in VMPL0. Our modified SVSM follows the standard manufacturing process of instantiating a vTPM instance as specified by the TPM2.0 specification [20]. First, we create a new endorsement key (EK) pair $\langle EK_{pub}, EK_{priv} \rangle$ from random seeds. However, we do not create an endorsement key certificate (EK_{cert}) or a platform certificate, as there is no entity to sign these certificates.

A significant, and much under discussed problem in Confidential Computing is seeding the random number generator. A VM when it boots has no natural sources of entropy that are not under the control of the untrusted host. In an ordinary VM, the x86 instructions `RDRAND` and `RDSEED` cause `VMEXIT`s; however, in confidential VMs, these

instructions are guaranteed to provide direct access to the CPU hardware random number in a way that the host cannot influence. We use these instructions as the initial random number entropy source for generating the random seeds.

4.5 Adding vTPM to the trust chain

Since our SVSM-vTPM module is instantiated with random seeds and does not come with a manufacturer's certificate to verify the identity of the TPM, we need to ensure the following security properties:

- **S1** Certify that the SVSM-vTPM is running in a real confidential VM on genuine AMD hardware
- **S2** Certify that the vTPM module is not tampered with.
- **S3** Communicate EK_{pub} in a secure, tamper-proof way.

To ensure these security properties, we rely on the attestation report from the AMD-SP hardware.

SEV-SNP attestation report Software running at any VMPL level can request an attestation report by sending a message to the SEV firmware running inside the AMD-SP. The request structure contains the VMPL level and 512-bits of space for user-provided data which would be included as part of the attestation report signed by the AMD hardware.

Figure 2 shows the steps involved in getting an attestation report. On receiving a request to launch a VM, the platform loads the image and cryptographically measures the contents of the image (①). Once the guest image is launched, the hypervisor puts the vCPU in VMPL0 mode passing control to the SVSM firmware (after ②). The SVSM firmware initializes the guest CPU, memory and sets up a pagetable for execution and finally instantiates a vTPM. The vTPM is provisioned as described in Section 4.4. Then, the vTPM module requests an attestation report by sending a `SNP_REPORT_REQ` message to the AMD-SP hardware (③). We place the digest of the public part of the generated endorsement key (i.e., EK_{pub}) in the user-data field of the request to communicate the identity of the TPM to the guest VM. The request message is encrypted with the appropriate VM platform communication key (VMPCK) for that VMPL level and prepended with a message header which is integrity protected with authenticated encryption (AEAD). The AMD-SP hardware decrypts the message, verifies the integrity and responds with an attestation report(④) that contains the *launch measurements*, VMPL level and the user-data (i.e., $digest\left(EK_{pub}\right)$). We write this report into the NVIndex where the TPM would normally place its EK certificate. We can retrieve the saved attestation report at any point in time (⑤) as long as the guest VM is operational. If needed, the guest VM can also place a report request to the AMD-SP hardware from other VMPL levels to generate a new attestation report.

Ensuring S1 We can easily verify **S1** because the attestation report is generated by the AMD-SP processor and signed using AMD's versioned chip endorsement keys (VCEK) [32]. Verifying that the attestation report is genuine implicitly guarantee that we obtained it from a genuine AMD processor, within a confidential VM.

Ensuring S2 Before launching the confidential VM, the AMD-SP hardware measures all the load-time binaries as part of the *launch measurement*. This includes the SVSM and our SVSM-vTPM code. By verifying these measurements that are included as part of the attestation report, we can ensure that our SVSM-vTPM binary, and anything else running in VMPL0, has not been tampered.

Ensuring S3 By verifying that the report request originated from VMPL0, we can confirm that the report was requested by a legitimate SVSM-vTPM, based on **S2**. By including the $digest(EK_{pub})$ as part of the attestation report (via user-data field), we offer a tamper-proof way to communicate the identify of the TPM (EK_{pub}) to the entities interacting with this specific vTPM. Since EK_{pub} and EK_{priv} are generated from random seeds provided by the hardware (i.e., `RDRAND` and `RDSEED`), as long as the generator is tamper-proof, no entity can recreate EK_{priv} and impersonate this vTPM.

5 Implementation

We base our implementation on the software stack recommended by AMD which is publicly available on github [4]. It consists of Qemu, Open Virtual Machine Firmware (OVMF) and Linux kernel for both the host and the guest, all of which are modified to support the AMD SEV-SNP architecture and will eventually be upstreamed. We make minor modifications to the open-source framework Keylime [9] for performing remote attestation of VMs that has the SVSM-vTPM in the root of trust.

To implement SVSM-vTPM, we extend the open-source SVSM implementation [11] with a minimal C library (a stripped-down version of Musl [13]), WolfSSL library [25] for cryptographic primitives, and Microsoft's TPM that provides a software reference implementation of TCG's TPM 2.0 specification [14].

5.1 Software TCB

We add 1500 lines of code to the existing SVSM implementation in Rust. To implement vTPM, we utilize third party libraries: a minimal C library, WolfSSL crypto library and Microsoft's reference TPM implementation [14]. The software TCB of our implementation is very similar to that of a physical TPM consisting of a processor core (e.g., ARM SecureCore) that can host the software components such as crypto libraries and TPM state machine. Also, the APIs we expose is similar to a hardware TPM implementing a CRB interface.

We measure the SVSM code and other third-party crates that are part of the dependency chain as everything is open source. We also assume that WolfSSL and Microsoft's TPM implementations are bug-free. It should be noted that the Microsoft reference implementation TPM is also the code which is running in firmware inside a hardware TPM. For this reason, we expect our vTPM to have the same security characteristics as a hardware TPM: state exfiltration is prevented by the VPML0 SNP security so the only attack vector is via the TPM command interface.

5.2 Remote attestation with Keylime

We use Keylime for remote attestation. Keylime is designed to perform both boot-time and runtime attestation on a fleet of systems, using the attested nodes' TPM devices as the root of trust [9]. Keylime is comprised of three major components: A Keylime *agent* is installed on every attested node. The agent announces itself with a Keylime *registrar* when it starts up. The Keylime *verifier* is in charge of performing attestations on every node.

Registration protocol The purpose of Keylime registration is to record the availability of the registering agent for attestation and to establish mutual trust between the agent and the registrar. To this

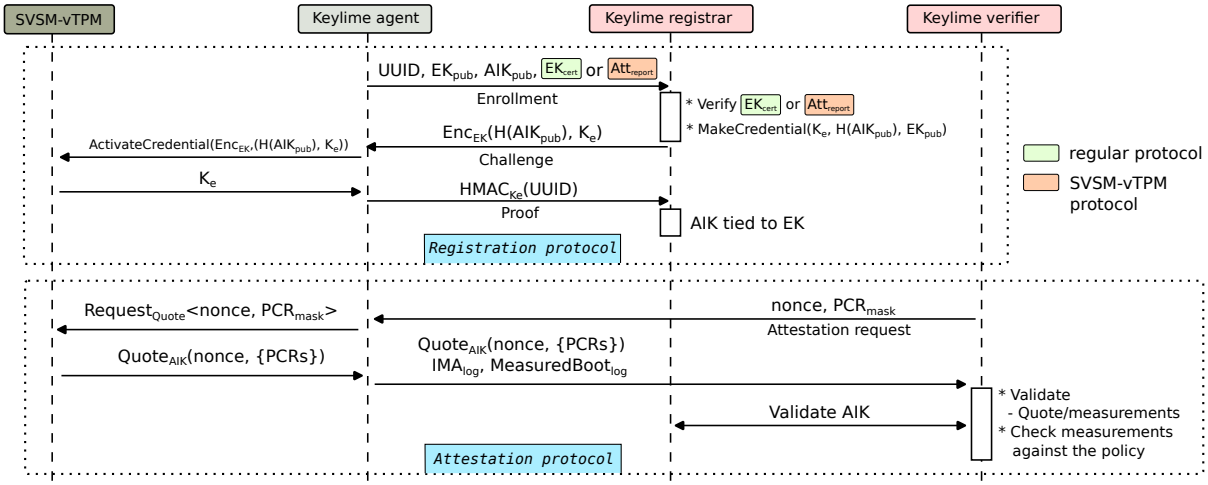

Figure 3: Remote attestation of a confidential VM using keylime and SVSM-vTPM

end the agent's credentials are checked and an attestation key is negotiated between the agent and the registrar for use for subsequent attestation challenges. As shown in Figure 3, the agent initiates the enrollment process by sending its TPM credentials - i.e., the public part of its endorsement key (EK) and attestation identity key (AIK), as well as the EK certificate, and the node's UUID to the registrar. The registrar verifies that the TPM's identity and authenticity using the public EK and the EK certificate. Next, the validity of the AIK is established through the *MakeCredential/ActivateCredential* function pair by using a carefully constructed secret that can only survive the registrar to agent roundtrip when both the TPM, AIK and UUID are authentic. Identity verification of a normal TPM device involves checking that the EK certificate correctly signs the public EK, and furthermore that the EK certificate (EK_{cert}) is signed by a trusted root (such as a manufacturer key or an intermediary key).

Attestation protocol Having successfully registered with the *registrar*, the *agent* is now ready to service attestation challenges. The Keylime *verifier* initiates the attestation protocol by sending a TPM quote request to the *agent*, containing a nonce (to guard against replay attacks) and a PCR mask (list of PCRs). The *agent* sends back the requested quote signed by the TPM, using the AIK associated during the registration phase. In addition, a number of logs (e.g. measured boot log, IMA log) are sent back with the quote. The *verifier* validates the TPM quote by decrypting it with the registered AIK; validates the logs by testing them against the PCRs contained in the quote; and finally checks the contents of the logs against the attestation policy to render a trustworthy/untrustworthy verdict.

Protocol changes to handle SVSM-vTPMs Since Keylime is built around interaction with TPM devices, we needed to make only minor modifications in the code to handle SVSM-vTPMs. Basically, we only had to modify how the Keylime verifier checks the authenticity of a TPM device (function `check_ek`). As mentioned above, a "normal" TPM device is authenticated through its EK certificate, which signs the public EK and in turn is verified by a manufacturer certificate. Keylime carries a list of acceptable manufacturer certificates, and any TPM in use by Keylime has to be signed by one of these. Our ephemeral SVSM-vTPM, by its very nature, is not provisioned with an EK certificate. However, the (ephemeral) public EK is signed by the SEV attestation report, which we validate by checking it against the platform manufacturer's signature (i.e., AMD). In order to minimize the required changes in Keylime, we decided to simply replace the EK certificate with an SEV attestation report (Att_{report}) in our SVSM-vTPM (that is, we reuse the NVIndex in the TPM where the EK certificate normally resides). The *agent* reads and submits the attestation report instead of the EK certificate during registration. The modified *registrar* validates the attestation report (ensuring that it is signed by an authentic AMD platform) instead of the validating the EK certificate (marked by a different color in Figure 3). No other parts of the registration/attestation protocols require changes for correct Keylime function.

6 Evaluation

We ran all our experiments on publicly-available Cloudlab infrastructure [68]. We utilize a Dell Poweredge R6525 server equipped with two AMD EPYC 7543 32-core processor and 256 GiB RAM. The host machine runs a 64-bit Ubuntu 20.04 Linux with v5.19 kernel and Qemu v6.1.50, whereas the confidential guest VM runs a 64-bit Ubuntu 22.04 Linux with a v5.17 kernel with Open Virtual Machine Firmware (OVMF) version `edk2-stable202208`, all of which are modified to enable SEV-SNP [4]. We have also evaluated our software stack on a Lenovo ThinkServer equipped with an AMD EPYC 7763 64-core processor and 128 GiB RAM.

6.1 Performance analysis

To understand the overheads of commonly used TPM functionalities, we study the performance of several TPM commands on SVSM-vTPM and compare that with a vanilla virtual machine that utilizes a vTPM hosted by Qemu. We rely on Qemu/KVM to launch both the regular and confidential VM. Qemu-vTPM setup uses the native TPM CRB interface as its frontend with an `swtpm` backend where the backed communicates with the vTPM running on the host userspace via a UNIX socket interface. The SVSM-vTPM setup uses a generic platform driver [29] to communicate with the vTPM inside the SVSM (as discussed in Section 4.2) running under VMPL0.

We compare the performance of four different TPM commands which are essential for remote attestation, i.e., `PCRREAD`, `PCREXTEND`, `TPM2_QUOTE`, `CREATEPRIMARY`:

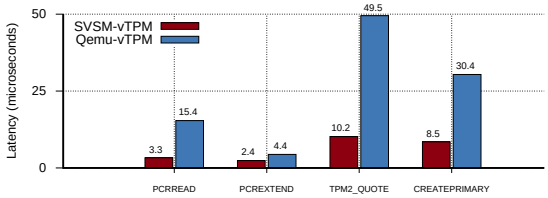

Figure 4: Performance overhead of SVSM-vTPM vs Qemu-vTPM

- **PCR read** This command reads the platform configuration registers of the TPM. A TPM may maintain multiple banks of PCR, where each bank is a collection of PCRs extended with a specific hashing algorithm (e.g., sha1, sha256). In our benchmark, we read all the PCR values from all the banks (i.e., sha1, sha256, sha384).
- **PCR extend** performs an extend operation on a specific PCR from a bank, i.e., it computes the hash of the old PCR value concatenated with the input data, i.e., $PCR_{new} = hash(PCR_{old}||input\_data)$. We extend a single PCR register from a sha256 bank.
- **Quote** A TPM quote contains a subset of PCRs from a bank and a nonce (to prevent replay attacks) signed by the attestation key (AIK) of the TPM. We request a quote of three PCRs (16-18) from two different banks (sha1 and sha256).
- **Create primary** The TPM command creates a primary object under the chosen hierarchy (Endorsement, Platform, Owner or NULL) and loads it into the TPM. The TPM returns only a context with which one can interact with this object and the public and private portions of the key are not returned. We create an ECC keypair with the default curve (ecc256).

We perform all the experiments by booting the confidential VM with the corresponding setup (Qemu or SVSM), and invoke the TPM commands from the guest user space using the `tpm2-tools` package [21]. For each TPM command, we ran the benchmark for 3000 iterations. We ran these experiments three times to measure the average latency (Figure 4). SVSM-vTPM incurs 5x lower latency than Qemu-vTPM on `PCRREAD`s and to get a `TPM2_QUOTE`. We incur 1.8x and 3.5x lower latency on `PCREXTEND` and `CREATEPRIMARY` TPM operations respectively. Both qemu-hosted vTPM and SVSM-vTPM incur an exit into the hypervisor to communicate with the TPM. Our SVSM-vTPM suffers from much less overhead compared to the qemu-hosted vTPM as the latter involves communicating with the TPM emulator backend (i.e., `swtpm`) through the socket interface.

For completeness, we also ran the same experiments on a machine that has an on-board physical TPM 2.0 device (Nuvoton NPCT75x). On an average, `TPM2_QUOTE` and `CREATEPRIMARY` is 25,000 times slower compared to our emulated SVSM-vTPM at 262,143 μs and 192,918 μs respectively whereas `PCRREAD` is 9,000 times slower (30,026 μs), and `PCREXTEND` is 3,900 times slower (9,359 μs). In general, physical TPMs are an order of magnitude slower than emulated ones because they are often connected to the mainboard via a low-bandwidth bus such as serial peripheral interface (SPI).

6.2 Security Analysis

A regular physical TPM is fully-isolated from the CPU and has its own crypto engine, TPM state machine and a secure RNG inside the chip. Moreover they do not store any of the TPM secrets on the DRAM and are not vulnerable to memory attacks. The gist of our security argument is that we are tying an ephemeral vTPM to the AMD-SP hardware's root of trust to perform runtime attestation. In this section we examine a number of potential security attacks that are impossible to perform with a physical TPM and explain how our vTPM design prevents them. Our hypothetical attacker's goal would be to infiltrate and alter a guest confidential VM without being detected by the remote attestation system (Keylime).

Fake vTPM The guest confidential VM boots with the SVSM firmware containing our SVSM-vTPM as part of the VM launch process. The essence of this attack is that after the system is booted and the keylime *agent* is registered, an attacker could spawn a new software vTPM in the guest userspace to hijack all the vTPM commands and redirect to the newly spawned vTPM. The new fake software vTPM is no longer running at a higher privilege level and can be controlled by the attacker to forge TPM quotes in an attempt to authenticate fake boot and IMA logs, and therefore hide unauthorized software alterations from keylime.

Once the registration protocol is complete, the keylime registrar has associated the *EK* of our ephemeral vTPM with the *AIK* that would be used for signing the TPM quote. With the above redirection of TPM commands to a fake vTPM an attacker would not be able to forge the TPM quote, as the fake vTPM has no access to the private *AIK* of the original vTPM, safely hidden by VMPL0.

The attacker could possibly force the registration protocol to restart where an attacker could feed the TPM credentials from the newly created vTPM. Again, keylime would detect this because of the mismatch of the fake TPM's EK_{pub} with its digest in the attestation report. A fake attestation report cannot be generated because the report contains the VMPL of the entity that requested it, and the guest is not running at VMPL0.

Fake SEV-SNP attestation report We save the attestation report requested by SVSM-vTPM at the same NVIndex as the EK_{cert} to make it available to the keylime agent. The essence of this attack is that the attacker could overwrite this NVIndex with either garbage data or another attestation report after compromising the guest. Garbage data would be detected by the keylime *registrar*, resulting in attestation failure. When overwritten with a genuine attestation report, an attacker can potentially change the identity of the vTPM, i.e., create another vTPM (similar to Fake vTPM attack) with a new set of keys and record the new EK_{pub} as part of the user-data field of the attestation report. If successful, they can perform all the attacks mentioned under the "Fake vTPM" attack (i.e., spoof PCRs, forge quotes, etc).

Even though one could retrieve an attestation report from a different VM privilege level, the platform guarantees that no one could spoof the VMPL level in the attestation report as it could be generated only by the software running inside VMPL0 (i.e., the keys for encrypting the request message is available only at the corresponding level). Thus, the replaced attestation report, if valid, would contain a VMPL level greater than 0. To prevent this attack, we check the VMPL level while validating the attestation report to ensure the requester VMPL level is set to zero.

An attacker can overwrite the attestation report NVIndex with a genuine attestation report off another confidential VM or from a previous boot of this confidential VM. Though the attestation report is signed by the AMD hardware, the user-data will not match

with the digest of EK_{pub} we have inside the SVSM-vTPM, making the attack detectable.

Confidential VMs with no SVSM Though VMPL levels are supported in the SEV-SNP specification, it cannot be enforced by the end-user on a provider-controlled environment. A malicious cloud provider could host a regular SEV VM and pretend that it is running with an SEV-SNP firmware. In this scenario, the confidential VM would run without the SVSM firmware, where the entire guest operating system will run under VMPL0. This makes it possible for a guest VM to generate its own attestation report where the requester VMPL level is set to 0. To prevent this attack, the user could verify that the confidential VM is booted with the SVSM firmware running at VMPL0 by measuring the boot-time binaries that includes the SVSM firmware running at VMPL0 and validate it against the measurements reported in the attestation report provided by the cloud provider. If the measurements do not match, the confidential VM is likely booted without the SVSM firmware.

Weaknesses in random number generator (RNG) AMD hardware has suffered from a buggy HWRNG in the past where RDRAND instruction gave out a constant value instead of a random number [7]. An attacker could exploit a weak or buggy HWRNG implementation to guess the initial seeds of the vTPM and create the same secret keys as the vTPM. For example, by guessing the attestation key, one could forge TPM quotes and break the guarantees of remote attestation. To be resilient to such hardware bugs, we can seed the RNG with additional sources of entropy such as the hash of a key derived by the AMD-SP upon user's request along with the RDSEED instruction.

6.3 Case Studies

Full disk encryption Full disk encryption (FDE) protects the confidentiality and integrity of data at-rest. To prevent accidental disclosure of the secret key (e.g., disk encryption key), it is a standard practice to encrypt the secret key (*wrap* operation) such that it can be decrypted only by the TPM (*unwrapping*). The wrapping key (i.e., the key which wraps the secret) is often the storage root key (SRK) present in the TPM.

However, in our ephemeral vTPM, there are no persistent storage keys in the TPM to support unwrapping of keys. Figure 5 shows the steps involved in supporting FDE on an ephemeral vTPM. To support FDE, we create an intermediary storage key K_{iSK} (a restricted decryption key with sensitiveDataOrigin [51]). Now, we perform a TPM *seal* operation on the disk encryption key by parenting it to the storage key (K_{iSK}) we just created, outputting a sealed blob which can be unsealed only by a TPM with the same key. On platform boot, the vTPM would generate an ephemeral endorsement key (e_{Ek}) and an ephemeral storage root key ($eSRK$). By retrieving the public part of the eSRK (① in Figure 5), we can wrap the intermediary key K_{iSK} with $eSRK_{pub}$ to create a wrapped key that can be decrypted only by our vTPM (② in Figure 5). It has to be noted that all the above operations can be performed on any TPM, i.e., the user need not necessarily perform these on the vTPM of the confidential VM. Now, the disk encryption key is wrapped to the parent key and the parent is in turn wrapped to the eSRK, forming a hierarchy under the ephemeral storage root key (③ in Figure 5). It is also possible to wrap the parent key with EK_{pub} instead to create a hierarchy under the ephemeral endorsement key.

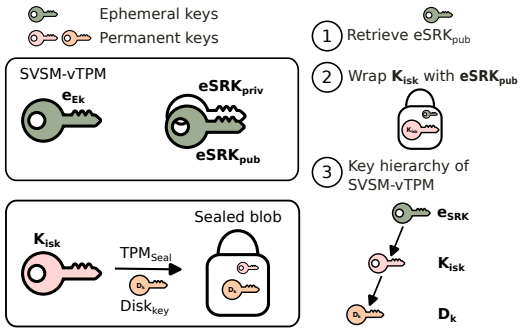

Figure 5: Full disk encryption in an ephemeral vTPM

As both the disk encryption key and its parent key are wrapped for our specific vTPM, they are no longer a secret and can be delivered to the confidential VM in the clear. Since the sealed disk encryption key is invariant, we can embed this into the initrd. Finally, we can deliver the wrapped parent key (② in Figure 5) to the confidential VM once we have performed the initial attestation of the platform to ensure its trustworthiness.

Storing secrets We cannot store secrets directly by wrapping the keys on our ephemeral SVSM-vTPM as the EK and SRK would be newly generated on every boot. One could use a similar technique we used for FDE to form a hierarchy of keys under an intermediary storage key. Once the system is booted, we can parent the intermediary key to the ephemeral SRK or EK forming a hierarchy under the chosen key. Using this technique, one could store a hierarchy of keys, as we do with a regular persistent TPM.

7 Discussion

Applicability to other TEEs With SVSM-vTPM, we rely on a VM privilege level provided by the SVSM specification that offers a secure environment where we implement our vTPM. If there is a hardware feature that guarantees this level of isolation on other TEEs such as Intel TDX, SGX, or ARM CCA, then we can follow similar design choices to build an ephemeral vTPM on those platforms. However, if such mechanisms do not allow a protected space within the TEE, the vTPM could be run in a sibling TEE that can host the vTPM and communicate with the enclave/confidential VM, similar to CoCoTPM.

Supported functionalities We do not store the persistent state with our ephemeral vTPM. Nonetheless, all other TPM functionalities are supported. Similar to a regular TPM one can store persistent keys, but they go through an extra step (similar to the reparenting mechanism with the disk encryption keys). In terms of the TCB, we no longer need the persistent state support which includes storing, encryption/decryption, and verifying the state etc.

8 Conclusions

The landscape of cloud security is changing with the growing need to remove the cloud provider from the trust domain. Hardware vendors lay the foundation for implementing this vision through a collection of mechanisms that ensure confidentiality of a cloud execution, i.e., encryption of application memory, but, unfortunately, lack support for ensuring runtime integrity. Our work develops a novel approach for virtualizing the hardware root of trust through a combination of hardware mechanisms and a new ephemeral

approach to managing the TPM state. We demonstrate how an ephemeral vTPM can be used for providing remote attestation of a confidential VM. In the spirit of transparency, our implementation is open source and can be audited, verified, and extended. As more and more cloud providers are gearing up to offer confidential VMs, we believe our SVSM-vTPM architecture would provide a reference point for implementing a vTPM on SEV-SNP infrastructure.

Acknowledgments

We would like to thank ACSAC 2023 reviewers for numerous suggestions helping us to improve this work. This research is supported in part by the National Science Foundation under OAC-2341138. Vikram Narayanan is partly supported by an IBM PhD fellowship.

References

[1] 2022. Add integrity and security to TPM2 transactions. https://www.spinics.net/lists/linux-integrity/msg24093.html. Online; accessed Dec 17, 2022.
[2] 2022. Alibaba Cloud Security White Paper. Online; accessed Dec 17, 2022.
[3] 2022. AMD Memory encryption. https://amd.wpenginepowered.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v9-Public.pdf. Online; accessed Dec 19, 2022.
[4] 2022. AMD Secure Encrypted Virtualization. https://github.com/AMDESE. Online; accessed Nov 11, 2022.
[5] 2022. AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf. Online; accessed Dec 17, 2022.
[6] 2022. AWS Nitro System. https://aws.amazon.com/ec2/nitro/. Online; accessed Dec 17, 2022.
[7] 2022. How a months-old AMD microcode bug destroyed my weekend. https://arstechnica.com/gadgets/2019/10/how-a-months-old-amd-microcode-bug-destroyed-my-weekend/. Online; accessed Dec 7, 2022.
[8] 2022. Intel Trust Domain CPU Architectural Extensions. https://www.intel.com/content/dam/develop/external/us/en/documents/intel-tdx-cpu-architectural-specification.pdf. Online; accessed Dec 17, 2022.
[9] 2022. Keylime. https://github.com/keylime/keylime. Online; accessed Nov 11, 2022.
[10] 2022. KVM CVEs. https://nvd.nist.gov/vuln/search. Online; accessed Dec 17, 2022.
[11] 2022. Linux SVSM for secure x86 virtualization in Rust. https://github.com/AMDESE/linux-svsm. Online; accessed Nov 11, 2022.
[12] 2022. Microsoft Azure Attestation. https://learn.microsoft.com/en-us/azure/attestation/overview. Online; accessed Dec 6, 2022.
[13] 2022. Musl libc. https://musl.libc.org/. Online; accessed Dec 17, 2022.
[14] 2022. Official TPM 2.0 Reference Implementation (by Microsoft). https://github.com/microsoft/ms-tpm-20-ref. Online; accessed Dec 17, 2022.
[15] 2022. OpenSSL. https://github.com/openssl/openssl. Online; accessed Nov 11, 2022.
[16] 2022. Protecting VM register state with SEV-ES. https://www.amd.com/system/files/TechDocs/Protecting%20VM%20Register%20State%20with%20SEV-ES.pdf. Online; accessed Dec 17, 2022.
[17] 2022. Secure VM Service Module for SEV-SNP Guests. https://developer.amd.com/wp-content/resources/58019.pdf. Online; accessed Dec 17, 2022.
[18] 2022. Security Protocol and Data Model (SPDM) Specification. https://www.dmtf.org/sites/default/files/standards/documents/DSP0274_1.1.0.pdf. Online; accessed Dec 17, 2022.
[19] 2022. SEV-ES Guest-Hypervisor Communication Block Standardization. https://developer.amd.com/wp-content/resources/56421.pdf. Online; accessed Dec 17, 2022.
[20] 2022. TPM 2.0 library. https://trustedcomputinggroup.org/resource/tpm-library-specification/. Online; accessed Dec 17, 2022.
[21] 2022. Trusted Platform Module (TPM2.0) tools. https://github.com/tpm2-software/tpm2-tools. Online; accessed Dec 17, 2022.
[22] 2022. Validating instances using Cloud Monitoring. https://cloud.google.com/compute/confidential-vm/docs/monitoring. Online; accessed Dec 6, 2022.
[23] 2022. Virtual Trusted Platform Module for Shielded VMs: security in plaintext. https://cloud.google.com/blog/products/identity-security/virtual-trusted-platform-module-for-shielded-vms-security-in-plaintext. Online; accessed Dec 6, 2022.
[24] 2022. VMWare CVEs. https://www.vmware.com/security/advisories.html. Online; accessed Dec 17, 2022.
[25] 2022. WolfSSL Embedded SSL/TLS Library. https://github.com/wolfSSL/wolfssl/. Online; accessed Dec 17, 2022.
[26] 2022. Xen vulnerability statistics. https://www.cvedetails.com/product/23463/XEN-XEN.html?vendor_id=6276. Online; accessed Dec 17, 2022.
[27] 2023. Apache Cloudstack CVEs. https://www.cvedetails.com/vulnerability-list/vendor_id-45/product_id-23458/Apache-Cloudstack.html. Online; accessed May 20, 2023.
[28] 2023. Openstack CVEs. https://www.cvedetails.com/vulnerability-list/vendor_id-11727/Openstack.html. Online; accessed May 20, 2023.
[29] 2023. [RFC 0/3] Enlightened vTPM support for SVSM on SEV-SNP. https://lore.kernel.org/all/acb06bc7f329dfee21afa1b2ff080fe29b799021.camel@linux.ibm.com/. Online; accessed May 20, 2023.
[30] Tiago Alves. 2004. Trustzone: Integrated hardware and software security. *Information Quarterly* 3 (2004), 18–24.
[31] AMD. 2022. Secure Encrypted Virtualization API Version 0.24. https://www.amd.com/system/files/TechDocs/55766_SEV-KM_API_Specification.pdf. Online; accessed Dec 17, 2022.
[32] AMD. 2023. Versioned Chip Endorsement Key (VCEK) Certificate and KDS Interface Specification. https://www.amd.com/system/files/TechDocs/57230.pdf. Online; accessed May 17, 2023.
[33] Melvin J Anderson, Micha Moffie, Chris I Dalton, et al. 2007. Towards Trustworthy Virtualization Environments: Xen Library OS Security Service Infrastructure. *Trusted Systems Laboratory, HP Laboratories Bristol* (2007), 88–111.
[34] ARM Inc. 2022. ARM CCA Security Model 1.0. https://developer.arm.com/documentation/DEN0096/latest. Online; accessed Dec 17, 2022.
[35] ARM Inc. 2022. Introducing Arm Confidential Compute Architecture. https://developer.arm.com/documentation/den0125/0200. Online; accessed Dec 19, 2022.
[36] ARM Inc. 2022. Runtime Security Subsystem (RSS). https://tf-m-user-guide.trustedfirmware.org/platform/arm/rss/readme.html. Online; accessed Dec 17, 2022.
[37] Tamas Ban. 2022. Attestation and Measured Boot. https://www.trustedfirmware.org/docs/Attestation_and_Measured_Boot.pdf. Online; accessed Dec 17, 2022.
[38] Stefan Berger, Ramón Cáceres, Kenneth A. Goldman, Ronald Perez, Reiner Sailer, and Leendert van Doorn. 2006. vTPM: Virtualizing the Trusted Platform Module. In *Proceedings of the 15th Conference on USENIX Security Symposium (USENIX Security 06)*. Article 21, 305–320 pages.
[39] Felix Bohling, Tobias Mueller, Michael Eckel, and Jens Lindemann. 2020. Subverting Linux' Integrity Measurement Architecture. In *Proceedings of the 15th International Conference on Availability, Reliability and Security (ARES 20)*. Article 27, 10 pages. https://doi.org/10.1145/3407023.3407058
[40] Jeremy Boone. 2018. *TPM Genie: Interposer Attacks Against the Trusted Platform Module Serial Bus*. White paper. NCC Group. https://www.nccgroup.com/globalassets/about-us/us/documents/tpm-genie.pdf.
[41] Haogang Chen, Yandong Mao, Xi Wang, Dong Zhou, Nickolai Zeldovich, and M. Frans Kaashoek. 2011. Linux Kernel Vulnerabilities: State-of-the-Art Defenses and Open Problems. In *Proceedings of the Second Asia-Pacific Workshop on Systems (APSys '11)*. 1–5. https://doi.org/10.1145/2103799.2103805
[42] George Coker, Joshua Guttman, Peter Loscocco, Amy Herzog, Jonathan Millen, Brian O'Hanlon, John Ramsdell, Ariel Segall, Justin Sheehy, and Brian Sniffen. 2011. Principles of remote attestation. *International Journal of Information Security* 10, 2 (2011), 63–81.
[43] Jonathan Corbet. 2012. Supervisor mode access prevention. https://lwn.net/Articles/517475/. Online; accessed Dec 17, 2022.
[44] Intel Corporation. 2019. *Intel® SGX Data Center Attestation Primitives (Intel® SGX DCAP)*. https://download.01.org/intel-sgx/sgx-dcap/1.7/linux/docs/Intel_SGX_DCAP_ECDSA_Orientation.pdf.
[45] Intel Corporation. 2022. *Intel® 64 and IA-32 Architectures Software Developer's Manual*. https://software.intel.com/content/www/us/en/develop/download/intel-64-and-ia-32-architectures-sdm-combined-volumes-1-2a-2b-2c-2d-3a-3b-3c-3d-and-4.html.
[46] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Paper 2016/086. https://eprint.iacr.org/2016/086 https://eprint.iacr.org/2016/086.
[47] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *25th USENIX Security Symposium (USENIX Security 16)*. 857–874. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan
[48] Crispin Cowan, Calton Pu, Dave Maier, Heather Hinton, and Jonathan Walpole. 1998. StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks. In *Proceedings of the 7th USENIX Security Symposium*. 63–78.
[49] Stephen Fischer. 2011. Supervisor Mode Execution Protection. NSA Trusted Computing Conference.
[50] Trusted Computing Group. 2019. Trusted Platform Module Library Part 1: Architecture. See [52], Chapter 37.7.2 External NV, 232–233. https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf
[51] Trusted Computing Group. 2019. Trusted Platform Module Library Part 1: Architecture. See [52], Chapter 25.2.3 "sensitiveDataOrigin",

198. https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf
[52] Trusted Computing Group. 2019. *Trusted Platform Module Library Part 1: Architecture* (level 00 revision 01.59 ed.). https://trustedcomputinggroup.org/wp-content/uploads/TCG_TPM2_r1p59_Part1_Architecture_pub.pdf
[53] Vivek Haldar, Deepak Chandra, and Michael Franz. 2004. Semantic Remote Attestation: A Virtual Machine Directed Approach to Trusted Computing. In *Proceedings of the 3rd Conference on Virtual Machine Research And Technology Symposium - Volume 3 (VM 04)*. 29–41.
[54] Guerney D. H. Hunt, Ramachandra Pai, Michael V. Le, Hani Jamjoom, Sukadev Bhattiprolu, Rick Boivie, Laurent Dufour, Brad Frey, Mohit Kapur, Kenneth A. Goldman, Ryan Grimm, Janani Janakirman, John M. Ludden, Paul Mackerras, Cathy May, Elaine R. Palmer, Bharata Bhasker Rao, Lawrence Roy, William A. Starke, Jeff Stuecheli, Enriquillo Valdez, and Wendel Voigt. 2021. Confidential Computing for OpenPOWER. In *Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys 21)*. 294–310. https://doi.org/10.1145/3447786.3456243
[55] Xin Jin, Li-na Wang, Rong-wei Yu, Peng Kou, and Cheng-lin Shen. 2010. Administrative Domain: Security Enhancement for Virtual TPM. In *2010 International Conference on Multimedia Information Networking and Security*. 767–771. https://doi.org/10.1109/MINES.2010.162
[56] Volodymyr Kuznetsov, László Szekeres, Mathias Payer, George Candea, R. Sekar, and Dawn Song. 2014. Code-Pointer Integrity. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation (OSDI 14)*. 147–163.
[57] Andrew Kwong, Daniel Genkin, Daniel Gruss, and Yuval Yarom. 2020. RAMBleed: Reading Bits in Memory Without Accessing Them. In *2020 IEEE Symposium on Security and Privacy (SP)*. 695–711. https://doi.org/10.1109/SP40000.2020.00020
[58] Mengyuan Li, Luca Wilke, Jan Wichelmann, Thomas Eisenbarth, Radu Teodorescu, and Yinqian Zhang. 2022. A Systematic Look at Ciphertext Side Channels on AMD SEV-SNP. In *2022 IEEE Symposium on Security and Privacy (SP)*. 337–351. https://doi.org/10.1109/SP46214.2022.9833768
[59] Akash Malhotra. 2022. AMD RYZEN pro 5000 series mobile processors. https://www.amd.com/system/files/documents/amd-security-white-paper.pdf. Online; accessed Dec 17, 2022.
[60] Mengyuan Li and Yinqian Zhang and Huibo Wang and Kang Li and Yueqiang Cheng. 2021. CIPHERLEAKS: Breaking Constant-time Cryptography on AMD SEV via the Ciphertext Side Channel. In *30th USENIX Security Symposium (USENIX Security 21)*. 717–732. https://www.usenix.org/conference/usenixsecurity21/presentation/li-mengyuan
[61] Derek Gordon Murray, Grzegorz Milos, and Steven Hand. 2008. Improving Xen Security through Disaggregation. In *Proceedings of the Fourth ACM SIGPLAN/SIGOPS International Conference on Virtual Execution Environments (VEE 08)*. 151–160. https://doi.org/10.1145/1346256.1346278
[62] Eugene D Myers. 2018. Using the Intel STM for Protected Execution. https://www.platformsecuritysummit.com/2018/speaker/myers/STMPE2Intelv84a.pdf. Online; accessed Dec 17, 2022.
[63] Alexander Nilsson, Pegah Nikbakht Bideh, and Joakim Brorsson. 2020. A survey of published attacks on Intel SGX. *arXiv preprint arXiv:2006.13598* (2020).
[64] Wojciech Ozga, Do Le Quoc, and Christof Fetzer. 2021. TRIGLAV: Remote Attestation of the Virtual Machine's Runtime Integrity in Public Clouds. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. 1–12. https://doi.org/10.1109/CLOUD53861.2021.00013
[65] Bryan Parno, Jonathan M. McCune, and Adrian Perrig. 2010. Bootstrapping Trust in Commodity Computers. In *2010 IEEE Symposium on Security and Privacy (SP)*. 414–429. https://doi.org/10.1109/SP.2010.32
[66] Joana Pecholt and Sascha Wessel. 2022. CoCoTPM: Trusted Platform Modules for Virtual Machines in Confidential Computing Environments. In *Proceedings of the 38th Annual Computer Security Applications Conference (ACSAC 22)*. 989–998. https://doi.org/10.1145/3564625.3564648
[67] Himanshu Raj, Stefan Saroiu, Alec Wolman, Ronald Aigner, Jeremiah Cox, Paul England, Chris Fenner, Kinshuman Kinshumann, Jork Loeser, Dennis Mattoon, Magnus Nystrom, David Robinson, Rob Spiger, Stefan Thom, and David Wooten. 2016. fTPM: A Software-Only Implementation of a TPM Chip. In *25th USENIX Security Symposium (USENIX Security 16)*. 841–856. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/raj
[68] Robert Ricci, Eric Eide, and CloudLab Team. 2014. Introducing CloudLab: Scientific Infrastructure for Advancing Cloud Architectures and Applications. *; login:: the magazine of USENIX & SAGE* 39, 6 (2014), 36–38.
[69] Reiner Sailer, Xiaolan Zhang, Trent Jaeger, and Leendert Van Doorn. 2004. Design and implementation of a TCG-based integrity measurement architecture.. In *13th USENIX Security Symposium (USENIX Security 04)*. 223–238.
[70] Vincent Scarlata, Carlos Rozas, Monty Wiseman, David Grawrock, and Claire Vishik. 2008. *TPM Virtualization: Building a General Framework*. Vieweg+Teubner, 43–56. https://doi.org/10.1007/978-3-8348-9452-6_4
[71] Hovav Shacham, Matthew Page, Ben Pfaff, Eu-Jin Goh, Nagendra Modadugu, and Dan Boneh. 2004. On the effectiveness of address-space randomization. In *Proceedings of the 11th ACM conference on Computer and Communications Security (CCS 04)*. 298–307. https://doi.org/10.1145/1030083.1030124
[72] Frederic Stumpf and Claudia Eckert. 2008. Enhancing Trusted Platform Modules with Hardware-Based Virtualization Techniques. In *2008 Second International Conference on Emerging Security Information, Systems and Technologies*. 1–9. https://doi.org/10.1109/SECURWARE.2008.23
[73] Haonan Sun, Rongyu He, Yong Zhang, Ruiyun Wang, Wai Hung Ip, and Kai Leung Yung. 2018. eTPM: A Trusted Cloud Platform Enclave TPM Scheme Based on Intel SGX Technology. *Sensors* 18, 11 (2018). https://doi.org/10.3390/s18113807
[74] Chihiro Tomita, Makoto Takita, Kazuhide Fukushima, Yuto Nakano, Yoshiaki Shiraishi, and Masakatu Morii. 2022. Extracting the Secrets of OpenSSL with RAMBleed. *Sensors* 22, 9 (2022). https://doi.org/10.3390/s22093586
[75] Chia-Che Tsai, Donald E. Porter, and Mona Vij. 2017. 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)*. 645–658.
[76] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. 991–1008.
[77] Arjan van de Ven. [n. d.]. New Security Enhancements in Red Hat Enterprise Linux v.3, update 3. https://static.redhat.com/legacy/f/pdf/rhel/WHP0006US_Execshield.pdf.
[78] Juan Wang, Jie Wang, Chengyang Fan, Fei Yan, Yueqiang Cheng, Yinqian Zhang, Wenhui Zhang, Mengda Yang, and Hongxin Hu. 2023. SvTPM: SGX-Based Virtual Trusted Platform Modules for Cloud Computing. *IEEE Transactions on Cloud Computing* 11, 3 (2023), 2936–2953. https://doi.org/10.1109/TCC.2023.3243891
[79] Juan Wang, Feng Xiao, Jianwei Huang, Daochen Zha, Chengyang Fan, Wei Hu, and Huanguo Zhang. 2018. A Security-Enhanced vTPM 2.0 for Cloud Computing. In *Information and Communications Security (ICICS 2017)*. 557–569.
[80] Stephen Weis. 2014. Protecting data in-use from firmware and physical attacks. *Black Hat* (2014).
[81] Richard Wilkins and Brian Richardson. 2013. UEFI secure boot in modern computer security solutions. In *UEFI forum*. 1–10.