What Enables Trust in IMA-Based Remote Attestation for CVMs?

Kenichiro Muto and Kuniyasu Suzaki

Institute of Information Security, Yokohama, Japan
dgs247101@iisec.ac.jp, suzaki@iisec.ac.jp

Abstract. Trust in Remote Attestation (RA) based on the Linux Integrity Measurement Architecture (IMA) in Confidential Virtual Machines (CVMs) relies not only on the integrity of measured values, but also on the assumption that the IMA measurement logic remains unmodified. This study empirically evaluates under what conditions this assumption holds in real-world SEV-SNP–based CVM environments. We implemented an evaluation kernel module that dynamically modifies the IMA measurement logic using live patching (kpatch) and kernel probes (kprobe). Using this module, we constructed an evaluation scenario in which measurement values can be altered while maintaining consistency between the IMA log and the vTPM PCR values reported in the Quote, and conducted comparative experiments across Azure, GCP, AWS, and an on-premises environment. Our results show that the feasibility of this scenario strongly depends on the configuration and operational policies of Secure Boot and the Machine Owner Key (MOK). In environments where MOK enrollment is strictly controlled, the kernel modification via the evaluation module is effectively blocked. In contrast, when MOK enrollment is permitted, modification of the measurement logic becomes feasible, and RA based on the modified IMA measurements can still pass verification under a log–PCR consistency model. This study demonstrates that trust in IMA-based RA in CVMs varies depending on operational configuration conditions, and identifies configuration management requirements necessary to preserve trust in IMA-based RA.

Keywords: Remote Attestation · Trusted Platform Module · Integrity Measurement Architecture · Secure Boot · Machine Owner Key · Livepatch · Kernel Probe · Confidential Computing.

1 Introduction

With the growth of cloud services, Confidential Computing has gained increasing attention. Its core technology, the Confidential Virtual Machine (CVM), provides an execution environment isolated from the hypervisor. Since CVMs run on cloud infrastructures managed by third parties, mechanisms to verify their integrity from outside are essential. In CVMs, integrity verification includes Remote Attestation (RA) of the Trusted Execution Environment (TEE) provided by the CPU, and TPM-based RA that verifies the software state inside the TEE.

This study focuses on the latter. Several cloud platforms support virtual TPM (vTPM), enabling TPM-based RA of guest software state [29].

TPM-based RA has traditionally been used to verify boot and runtime file measurements in physical and edge environments [38, 11]. In Linux systems, Secure Boot, TPM, and the Integrity Measurement Architecture (IMA) form a chain of trust from boot to runtime: Secure Boot protects the bootloader and kernel through signature verification, TPM stores measurements in PCRs, and IMA measures file accesses and kernel modules. In CVMs, trust relationships become more complex due to layered virtualization and separation of operational control [18, 36, 13]. It is therefore unclear how these components define trust boundaries in real cloud environments. IMA assumes that its internal measurement logic is not modified [34]. However, the Linux kernel provides mechanisms such as Linux live patching (kpatch) and kernel probes (kprobe), which allow dynamic modification of kernel functions at runtime [31, 25]. If IMA's measurement logic is modified, the IMA log and PCR values may remain consistent while the correctness of the measurements is no longer guaranteed. This means that such violations may go unnoticed in compromised CVMs.

Although this risk has been recognized in prior work [34], the practical trust boundary of IMA-based RA under real-world CVM configurations has not been systematically analyzed. In particular, it remains unclear how Secure Boot and Machine Owner Key (MOK) management, as well as differences across cloud platforms, influence kernel modification feasibility and RA verification outcomes. In this study, we implement a kernel module that modifies the IMA measurement logic using kpatch and kprobe and evaluate its feasibility across SEV-SNP–based CVM environments. We construct evaluation scenarios focusing on (1) the measurement scope defined by IMA policies and (2) modification feasibility under different Secure Boot and MOK configurations. Our results show that under the same IMA policy, the visibility of module loading and the feasibility of modifying the IMA measurement logic differ across CVM environments. When Secure Boot is enabled but MOK enrollment is permitted, modification becomes feasible and RA verification based on modified measurements can still pass when the verifier checks only consistency between the IMA log and the PCR values reported in the Quote. In contrast, when MOK management is strictly controlled, this signed-module-based modification method is effectively blocked. These results indicate that trust in vTPM/IMA-based RA is determined not only by the TPM and IMA mechanisms themselves, but also by configuration and operational policies such as Secure Boot enforcement and MOK management, while the visibility of module loading may additionally depend on IMA policy enforcement.

The contributions of this work are threefold:

- We empirically demonstrate when and how modification of the IMA measurement logic becomes practically feasible in real SEV-SNP–based CVM environments.
- We comparatively evaluate multiple cloud platforms to analyze how Secure Boot configuration and MOK management influence kernel modification feasibility and RA verification outcomes.

– Based on these findings, we identify configuration-dependent trust boundaries of IMA-based RA and derive configuration management requirements necessary to preserve attestation trust in CVMs.

2 Background

2.1 Trusted Platform Module (TPM)

The Trusted Platform Module (TPM) is a security chip that measures system integrity and enables external verification. In Linux, the bootloader, kernel, and selected user-space components are hashed and stored in Platform Configuration Registers (PCRs) inside the TPM. Measurements are recorded through the *Extend* operation, which updates a PCR by hashing the concatenation of its current value and a new measurement [41]:

$$\mathrm{PCR}(n)_{\mathrm{new}} = H(\mathrm{PCR}(n)_{\mathrm{old}} \,\|\, \texttt{measured\_value})$$

Here, n denotes the PCR index, $\|$ indicates concatenation, and `measured_value` is the hash of the newly measured component; H is a TPM-supported hash function (e.g., SHA-1 or SHA-256). TPM 2.0 defines multiple PCRs; in the PC Client specification, `PCR(0)-PCR(7)` are mainly used for boot-time measurements [40]. Boot events are recorded in the TPM event log at `/sys/kernel/security/tpm0/binary_bios_measurements`. A verifier can reconstruct PCR values from this log and compare them with reference values to detect tampering of boot components. TPM can also generate a signed Quote over selected PCR values, enabling cryptographic RA [5]. Thus, TPM provides the basis for externally verifiable boot integrity.

2.2 vTPM on Confidential Computing

Cloud providers such as Azure offer virtual TPMs (vTPMs), allowing guest VMs to access TPM functions through the same interface as hardware TPMs. In Confidential VM (CVM) environments, vTPM enables verifiable recording of boot and OS state measurements [29]. In AMD SEV-SNP environments [2], vTPM may be protected by a Secure VM Service Module (SVSM) [10]. Microsoft's OpenHCL [26] acts as a paravisor supporting multiple Confidential Computing architectures, including AMD SEV-SNP [2] and Intel TDX [16]. Data Center Execution Assurance (DCEA) has also been proposed to cryptographically guarantee that a CVM runs on a trusted host [32]. These efforts extend TPM/vTPM-based trust models into cloud infrastructures.

2.3 Integrity Measurement Architecture (IMA)

Integrity Measurement Architecture (IMA) is a Linux kernel subsystem for runtime integrity measurement. It measures file accesses and kernel module loading, records measurement entries in a log, and extends their hashes into TPM PCRs.

For each event, IMA creates a template entry containing the file hash and file path, computes a template hash from these fields, and extends it to a PCR. Many distributions use `PCR(10)` for IMA measurements, although assignments may vary. Measurements are recorded sequentially in the IMA runtime measurement log, which is exposed through the file `/sys/kernel/security/ima/ascii_runtime_measurements` [22]. A verifier can recompute template hashes from the IMA log, replay Extend operations in order, and compare the reconstructed value with the PCR in the Quote; if they match, the log is cryptographically consistent with the TPM state [34]. IMA-based RA therefore relies on consistency between the IMA log and the PCR values reported in the TPM Quote. The measurement scope is policy-controlled; many distributions use `ima_policy=tcb`, which measures major binaries and libraries [21].

2.4 Secure Boot and Machine Owner Key (MOK)

Secure Boot is a UEFI-based signature verification mechanism that ensures bootloaders and kernels are signed by trusted keys, preventing execution of unsigned or tampered code. It establishes a chain of trust using the Platform Key (PK), Key Exchange Keys (KEK), and signature databases (db) from firmware to the OS [44, 30]. While official keys (e.g., Microsoft keys) are typically preinstalled, users can enroll their own public keys through the Machine Owner Key (MOK) mechanism [7, 37]. Once enrolled, modules or kernels signed with the corresponding private key can pass Secure Boot verification. MOK enrollment typically requires issuing a request within the OS (e.g., `mokutil -import`) and confirming it during reboot, serving as an operational control mechanism [14].

2.5 Kernel Modules and Dynamic Modification

The Linux kernel supports runtime extension through kernel modules and provides mechanisms to dynamically alter kernel behavior, including Linux live patching (e.g., kpatch) and kernel probes (kprobe) [31, 25]. Live patching replaces specific kernel functions at runtime; its origin traces back to Ksplice [4], and later systems such as kGraft and kpatch were integrated into the official Livepatch subsystem [20]. kpatch typically uses `ftrace` to redirect execution from old functions to patched versions. In contrast, kprobe inserts hooks into arbitrary kernel functions for analysis or intervention [25]. Although useful for maintenance and debugging, these mechanisms can alter internal kernel behavior and thus affect assumptions underlying integrity measurement.

3 Threat Model and Research Question

3.1 Threat Model

IMA-based RA relies on the assumption that IMA's internal measurement logic is not modified [34]. In addition, the measurement scope of IMA depends on policy

settings. In many distributions using `ima_policy=tcb`, accesses to certain file systems (e.g., `tmpfs`) are excluded from measurement by default [21]. Based on these assumptions and configuration dependencies, we define a threat model in which the conditions required for RA trust are violated.

Figure 1 illustrates the threat model considered in this study. If IMA's measurement logic is modified through mechanisms such as dynamically loaded kernel modules, the resulting measurement values may no longer reflect the actual system state. Moreover, when the modified module is placed in a policy-excluded file system (e.g., `tmpfs`), its introduction may not appear in the IMA log. Even if the altered measurements are recorded in the IMA log and extended into TPM PCRs, a verifier that checks only consistency between the IMA log and the PCR values reported in the Quote has no mechanism to detect modification of the measurement logic itself. As long as log–PCR consistency is preserved, the system may still be considered valid under such a verification model. In CVM environments, if the same assumptions as in traditional systems are adopted, this threat remains theoretically feasible.

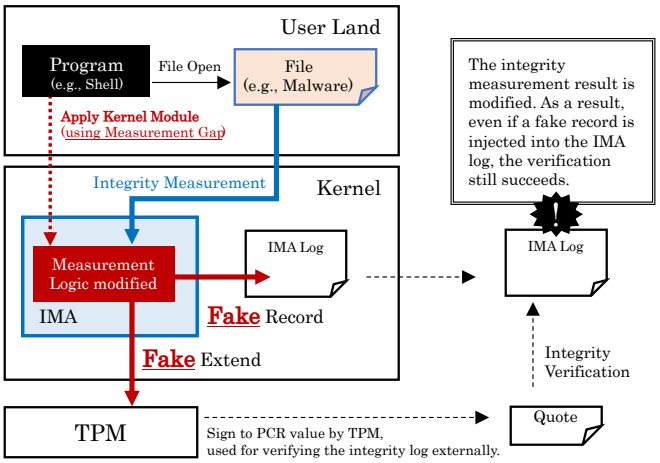

Fig. 1. Threat model considered in this study. The IMA measurement logic is modified while the resulting IMA log entries and TPM PCR values remain mutually consistent. RA verifies consistency between the IMA log and the PCR values reported in the Quote, but does not directly validate the integrity of the measurement logic itself.

3.2 Research Question

CVM environments also support vTPM/IMA-based RA, and the basic mechanisms for integrity measurement and verification are similar to those in traditional systems, except that the TPM is virtual rather than a hardware root

of trust. Therefore, trust in IMA-based RA in CVMs depends on the same underlying assumptions. However, in CVMs, kernel and module loading conditions may differ due to Secure Boot configurations, MOK key management policies, and provider-specific platform controls. It is therefore unclear to what extent modification of IMA's measurement logic at kernel level can be realized under real CVM configurations.

In this study, we reproduce the threat described in Section 3.1 as an evaluation scenario and empirically assess its feasibility under different CVM configurations. We focus on how runtime modification of IMA's measurement logic affects RA verification within the chain of trust formed by Secure Boot, TPM, and IMA, and how its feasibility changes depending on configuration conditions. To achieve this goal, we define the following research questions (RQs):

RQ1. To what extent is the loading of a kernel module that modifies IMA's measurement logic observable under existing IMA policies and measurement mechanisms?

RQ2. How do configuration conditions such as Secure Boot and MOK management affect the feasibility of modification of IMA's measurement logic at kernel level?

RQ3. Based on these findings, what configuration management requirements are necessary to preserve the trust of IMA-based RA in CVM environments?

3.3 Assumption

To reproduce the scenario described in Section 3.1, we make the following assumptions. First, we assume that the attacker has kernel-level privileges (e.g., root access) on the target system. This assumption represents post-compromise scenarios in cloud environments, such as container escape, kernel vulnerabilities, or insider misuse, and does not imply that root compromise is trivial. Instead, it is adopted to examine whether trust in RA can be maintained even when the system is partially compromised. Second, to compare how Secure Boot and MOK management function as trust boundaries, we include configurations in which console or serial access is available and interactions required for MOK enrollment can be performed. This does not assume a specific operational model but allows us to evaluate how configuration differences affect feasibility.

We assume that Linux IMA is enabled with the default policy (`ima_policy=tcb`) and the `ima-ng` template format [21]. Secure Boot status and key management follow each evaluated environment. This study focuses on software-layer behavior and does not consider physical attacks on hardware or firmware. TPM (or vTPM) and Secure Boot are assumed to operate according to specification, and we do not modify internal mechanisms such as Quote generation or Extend operations. The verifier is assumed to follow a standard RA implementation.

4 Experimental Design and Implementation

4.1 Evaluation Scenario

This section describes an evaluation scenario that reproduces the threat model introduced in Section 3.1. We define a *pseudo-attacker* as an experimental actor that re-creates the threat conditions (rather than an actual adversary), and analyze how configuration and operational constraints affect RA outcomes. Figure 2 provides an overview.

The scenario consists of three phases (Phase 1–3) and evaluates how RA behaves when IMA's measurement logic is modified while preserving consistency between the IMA log and TPM PCR values. Phase 2 is divided into Phase 2a (key enrollment) and Phase 2b (module loading) to separate the effects of MOK enrollment from those of module loading. The scenario focuses on: (1) whether kernel module loading—normally restricted by Secure Boot—becomes possible depending on MOK enrollment in Phase 2a, and (2) how IMA policy and measurement scope affect both modification feasibility and the observability of module loading in Phase 2b.

In Phase 2, we examine whether the evaluation module leaves observable traces in the IMA log and whether log–PCR consistency is maintained after modification. In Phase 3, we run a standard RA verifier using the collected Quote and IMA log to evaluate how a verification model based solely on log–PCR consistency treats the modified measurements. This scenario therefore reproduces a violation of IMA's non-modification assumption and empirically evaluates what an external verifier can (and cannot) observe when log–PCR consistency is preserved.

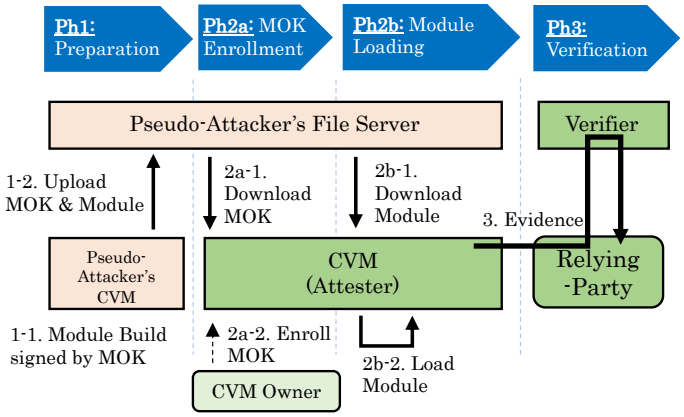

Fig. 2. Evaluation scenario reproducing the threat conditions under different configurations. A signed kernel module modifies IMA's measurement logic while keeping the resulting IMA log and TPM PCR values mutually consistent; we then observe the RA verification outcome.

Phase 1: Preparation (Pseudo-attacker) In Phase 1, the pseudo-attacker prepares a signed evaluation kernel module that modifies IMA's measurement logic, reproducing the condition described in Section 3.1. The module is built on an OS/kernel environment identical or compatible with the target CVM and hooks selected IMA measurement functions using livepatch or kprobe. The module is signed with a self-generated private key, and the corresponding public key (PEM/DER) is prepared for enrollment as a MOK. The module is then transferred to the target environment (e.g., via a file server), establishing the prerequisites for loading a signed module in a Secure Boot–enabled system.

Phase 2a: MOK Enrollment (CVM Owner) Phase 2a reproduces the operational step required to load the evaluation module under Secure Boot. The prepared public key is enrolled as a MOK. MOK enrollment typically involves issuing a request within the guest OS (e.g., `mokutil -import <pubkey>`) and confirming it through an interactive screen during reboot [14, 7]. Therefore, feasibility depends on whether console interaction is available and whether key enrollment is permitted by the target platform configuration. We evaluate both cases—MOK enrollment permitted vs. restricted—to assess how Secure Boot and key-management policies constrain module loading.

Phase 2b: Module Loading (Pseudo-attacker) In Phase 2b, the pseudo-attacker loads the evaluation module and modifies IMA's measurement logic at runtime. After MOK enrollment, the module is loaded on the CVM and hooks are activated via livepatch or kprobe. To evaluate policy-dependent observability, the module is placed on a file system excluded from measurement under the default `ima_policy=tcb` (e.g., `tmpfs`) and loaded from there; we observe whether module-related accesses appear in the IMA log. The module alters measurement inputs (e.g., file hash values and file paths) so that falsified values are processed through the standard IMA template mechanism and extended into TPM PCRs. As a result, the IMA log and PCR values remain consistent, while the measurements no longer correspond to the actual file hashes. We also compare the observability of module loading when the module is placed in measured and non-measured file systems.

Phase 3: Verification (External Verifier) In Phase 3, an external verifier follows a standard RA procedure to collect the IMA log and TPM Quote from the target system. The verifier recomputes template hashes from log entries and checks consistency with the PCR values in the Quote. We observe how RA verification behaves when the modified IMA log and PCR values remain mutually consistent, i.e., how a verifier based solely on log–PCR consistency treats modification of the measurement logic. We also analyze log entries and template fields related to module loading to assess the observability of module loading.

4.2 Implementation

Environment Setup We conducted experiments on four SEV-SNP–based CVM environments: three major public-cloud offerings (Azure, GCP, and AWS) and an on-premises environment based on the open-source COCONUT-SVSM implementation [10]. Figure 3 shows the experimental setup.

Each CVM served both as a target environment and as the build environment for the evaluation module. The module was built on each CVM and then uploaded to a file server. The Relying-Party environment hosted both the Verifier (which validates TPM Quotes and reconstructs IMA logs) and the file server for storing evaluation modules, on the same VM. This single Relying-Party environment was shared across all CVM experiments.

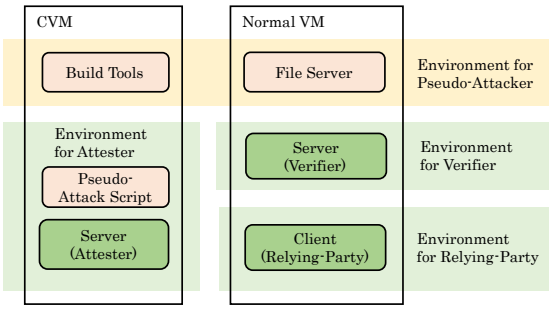

Fig. 3. Experimental setup: Each CVM builds and runs the evaluation module and the Attester component, while a separate normal VM hosts the file server, Verifier, and the Relying-Party component for the end-to-end RA workflow.

Table 1 summarizes the experimental environments. All CVMs were based on Ubuntu 24.04 LTS and enabled Linux IMA with the default configuration (`ima_policy=tcb` and template `ima-ng`). To build the evaluation module, we used standard build tools (`build-essential`, `flex`, `bison`, `fakeroot`) and additional libraries for handling ELF/DWARF information (`dwarves`, `libelf-dev`, `libdw-dev`), which are required by `kpatch-build` to analyze kernel internals and generate livepatch modules. To load the module under Secure Boot, we generated a signing key using `openssl` and signed the module, then enrolled the corresponding public key as a MOK to realize a configuration where signed third-party modules can be loaded even with Secure Boot enabled. For PCR reads and Quote collection, we used `tpm2-tools` on all VMs, enabling a consistent method for collecting IMA measurement logs and TPM Quotes across platforms. A Python agent on the Attester side collected the evidence and sent it to the Relying-Party, where the Verifier performed validation to realize an end-to-end RA workflow consisting of measurement, attestation, and verification in CVMs. Although kernel versions differ across environments, our evaluation indicates that the feasibility of the scenario was primarily determined by Secure Boot configuration and MOK enrollment policy rather than kernel version differences.

Table 1. Experimental environments

CVM*	**Machine Spec with OS and Tools**
Azure	Hyper-V UEFI Release v4.1 Processor: 2 vCPU (AuthenticAMD, 2.4 GHz), 15.4 GB RAM Ubuntu 24.04.3 LTS (kernel: 6.11.0-1018-azure)
GCP	Google Compute Engine Processor: 2 vCPU (AMD EPYC 7B13, 2.4 GHz), 7.7 GB RAM Ubuntu 24.04.3 LTS (kernel: 6.14.0-1014-gcp)
AWS	Amazon EC2 Processor: 8 vCPU (AMD EPYC 7R13, 2.7 GHz***), 30.9 GB RAM Ubuntu 24.04.3 LTS (kernel: 6.14.0-1011-aws)
COCONUT-SVSM	QEMU (v8.2.0-81-g260df2b36b-dirty) Processor: 4 vCPU (AMD EPYC-v4, 3.0 GHz), 7.7 GB RAM (Host Processor: AMD EPYC 7313P) Ubuntu 24.04 LTS (kernel: 6.11.0-coconut-svsm-sevsnp+)
Common Software (installed in CVM)	python3 (3.12.3-0ubuntu2.1), openssl (3.0.13-0ubuntu3.6) git (1:2.43.0-1ubuntu7.3), kpatch-build (0.9.11) build-essential (12.10ubuntu1), bc (1.07.1-3ubuntu4) flex (2.6.4-8.2build1), bison (2:3.8.2+dfsg-1build2) fakeroot (1.33-1), dwarves (1.25-0ubuntu3) libssl-dev (3.0.13-0ubuntu3.6), libelf-dev (0.190-1.1ubuntu0.1) libdw-dev (0.190-1.1ubuntu0.1), tpm2-tools (5.6-1build4)
Normal VM**	**Machine Spec with OS and Tools**
VirtualBox VM	Oracle VirtualBox (7.1.4 r165100) Processor: AMD Ryzen 3 7330U 2 vCPU (2.3 GHz), 2.0 GB RAM Ubuntu 24.04.2 LTS (kernel: 6.8.0-84-generic) python3 (3.12.3-0ubuntu2), tpm2-tools (5.6-1build4)

* The Attester and the evaluation-module build environment reside on the same CVM.
** The Relying-Party, Verifier, and file server reside on the same VM.
*** Average value across vCPU cores.

Evaluation Module Implementation We implemented an evaluation module as a signed kernel module to reproduce the threat conditions in a controlled manner. The module was signed with a self-generated key and made loadable under Secure Boot via the MOK enrolled in Phase 2a. We implemented two variants: a kpatch-based livepatch module and a kprobe-based module using function probes. Both variants hook selected functions in the IMA measurement flow and modify measurement inputs (e.g., hash values and file paths) for specific file accesses so that the resulting IMA records differ from those derived from the original file contents.

As shown in Figure 4, the module modifies template data between `process_measurement` and the routines that append IMA log entries and extend TPM PCRs (`ima_add_digest_entry` and `ima_pcr_extend`). By changing the data passed into the Extend operation, we can alter IMA measurements while preserving the correspondence between the IMA log and PCR values, without modifying the TPM protocol, TPM driver, or TPM (vTPM) implementation. Consequently, the IMA log and PCR values remain mutually consistent even though the measurements no longer reflect the original file contents. For the kprobe variant, we inserted probes on `ima_store_template` to modify data immediately before template generation. For the kpatch variant, directly replacing `static` functions can cause `kpatch-build` to fail; to avoid this limitation, we applied patches around `ima_alloc_init_template`, which is not declared `static`.

Building a kpatch module also requires symbol information that exactly matches the running kernel.

We installed the debug symbol packages in advance, replaced `Module.symvers` generated in the source tree with `/lib/modules/$(uname -r)/build/Module.symvers` for the running kernel, and executed `kpatch-build` with the corresponding `vmlinux`. These steps ensured that the generated module could be loaded into the running kernel.

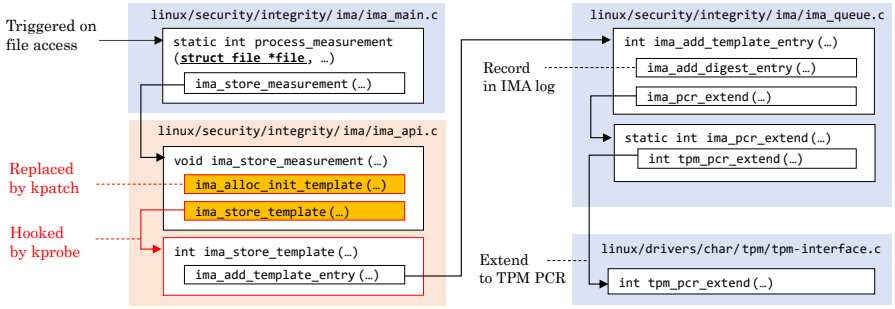

Fig. 4. Hook points of the evaluation module in the IMA measurement flow. The module modifies template inputs between IMA measurement processing and log/PCR updates, yielding modified yet internally consistent IMA log entries and PCR values.

Enabling IMA in UKI-based Environments To enable IMA, we first added `ima=on ima_policy=tcb` to the GRUB kernel command line. This works in conventional GRUB-based systems. However, on Azure CVM, IMA was not enabled with the same configuration. We found that Azure CVMs use a Unified Kernel Image (UKI) boot flow.

In a UKI-based environment, the kernel image, initrd, and command-line parameters are bundled into a single signed image that is loaded by `shim` after signature verification, bypassing GRUB at boot time [43]. Because the effective kernel command line is embedded in the UKI, modifying GRUB configuration files does not affect boot parameters. Therefore, UKI-based systems require a different mechanism to enable IMA. Since Azure CVMs do not allow passing IMA parameters via GRUB, we placed a policy file at `/etc/ima/ima-policy` and activated the policy after boot. Using this method, we confirmed that IMA measurement can be enabled on UKI-based Azure CVMs with behavior equivalent to `ima_policy=tcb`. The policy file used in this study corresponds to the file-form description of `ima_policy=tcb` in the IMA documentation [21] (see Appendix A).

Attestation and Verification Component To verify correspondence between IMA measurements and TPM PCR values, we implemented experimental

components following a standard IMA-based RA procedure. We replay TPM Extend operations over the IMA log to reconstruct PCR values and compare them with the PCR values reported in the Quote, thereby validating measurement-log and PCR consistency. We followed the Background-Check Model in RFC 9334 (RATS) [5] and implemented an Attester, Relying-Party, and Verifier.

The Attester collects the IMA log and TPM event logs (`ascii_runtime_measurements` and `binary_bios_measurements`) inside the CVM and uses `tpm2-tools` [39] to obtain `PCR(0...10)` and a signed Quote. The Verifier provides a nonce, which the Attester includes as the `-qualification` input when generating the Quote to ensure freshness. The Attester sends the IMA log, TPM event logs, and Quote to the Relying-Party, where the Verifier independently validates them. Figure 5 shows the verification flow.

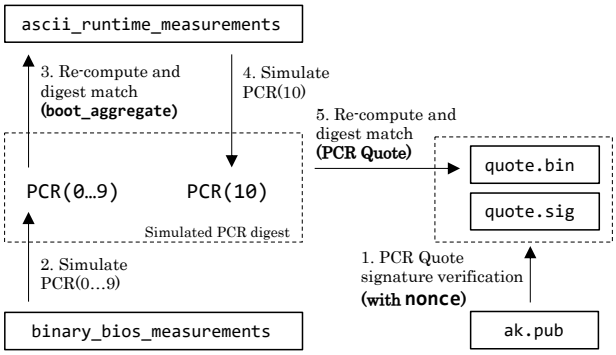

Fig. 5. Verification flow: The Verifier checks the Quote signature and nonce, reconstructs PCR values from event logs, replays IMA entries to derive `PCR(10)`, and accepts attestation only if the reconstructed PCR set matches the values reported in the Quote.

First, the Verifier checks the Quote signature and confirms nonce freshness (Step 1). It then reconstructs `PCR(0...9)` from the TPM event log (Step 2) and compares them with `boot_aggregate` in the IMA log (Step 3). Next, it replays Extend operations over the IMA log to reconstruct `PCR(10)` (Step 4). Finally, it recomputes `pcrDigest` from the reconstructed PCR set and verifies that it matches the `pcrDigest` contained in the Quote (Step 5). If all checks succeed, the Verifier concludes that the IMA log is cryptographically consistent with the PCR values contained in the TPM-signed Quote.

5 Results and Findings

5.1 Experimental Results

Table 2 summarizes the results of the evaluation scenario (Phases 1–3) across the CVM environments. Each phase was repeated several times with consistent

outcomes across runs. The table also shows the Secure Boot (SB) and vTPM configuration of each environment.

Table 2. Feasibility of the evaluation scenario across CVM environments

Target CVM	SB	vTPM	P1	P2a	P2b	P3	Feasibility
Azure	Enabled	Enabled	✓	✓	✓	✓*	Feasible*
GCP	Enabled	Enabled	✓	✗	✗	✗	Not feasible
AWS	Disabled	Disabled	✓	✗	✓	N/A	N/A
COCONUT-SVSM	Not supported	Enabled	✓	✗	✓	✓	Feasible

* RA verification succeeded, but module loading was observable in the IMA log.

Legend: ✓ completed; ✗ not completed; N/A not applicable.
Phases: P1 Preparation; P2a MOK enrollment; P2b Module loading; P3 Verification.

As shown in Table 2, the primary difference across environments appeared in Phase 2a (MOK enrollment). On Azure, MOK enrollment was completed via the serial console, whereas on GCP the enrollment interface was unavailable. COCONUT-SVSM [10] does not support Secure Boot and therefore has no MOK mechanism, making Phase 2a not applicable. AWS did not provide either Secure Boot or vTPM in our configuration; therefore Phase 3 (vTPM/IMA-based RA verification) was not evaluated and AWS results are discussed only up to Phase 2b. This environment was included to contrast cases where Secure Boot and vTPM-based attestation are unavailable, highlighting how configuration determines the practical attestation trust boundary.

In SB–enabled environments (Azure and GCP), the feasibility of MOK enrollment determined whether the evaluation module could be loaded (Phase 2b) and whether modification of the IMA measurement logic could be realized, allowing RA verification in Phase 3 to succeed. In environments where Secure Boot was disabled or unsupported (AWS and COCONUT-SVSM), module loading was not constrained by signature verification and MOK was not a determining factor. We also observed differences in the visibility of module loading. On Azure, the Phase 3 consistency check (agreement between the Quote and reconstructed PCR values) succeeded, but module-loading events appeared in the IMA log. On COCONUT-SVSM, the consistency check also succeeded while module loading left no observable traces in the IMA log.

These results indicate that feasibility depends on (i) kernel-level control (root privileges), (ii) Secure Boot configuration, and (iii) the operational policy governing MOK enrollment. They further show that the observability of module loading depends on how the IMA policy is enforced in the CVM environment and on module placement (i.e., whether a non-measured filesystem region exists).

5.2 Practical Feasibility of Building Evaluation Modules

To assess the practical feasibility of building evaluation modules that modify the IMA measurement logic, we compared two approaches—kpatch and kprobe—in terms of build time, required artifacts, and platform dependency (Table 3).

Generating a livepatch module with kpatch requires `vmlinux`, `Module.symvers`, and a kernel source tree that exactly matches the running kernel. While these artifacts are available when the kernel is built locally, their availability varies across cloud-provider images, which can constrain building the module directly on a CVM. In our experiments, patch generation and symbol analysis increased build time to tens of minutes, making this approach highly environment-dependent. In contrast, the kprobe approach requires only standard build tools and kernel headers, typically available in cloud environments. Modules can therefore be built entirely within the CVM, with build times on the order of seconds across all tested platforms.

Overall, kprobe was less platform-dependent than kpatch and required significantly less preparation and build time. This indicates that the feasibility of dynamically modifying IMA depends not only on theoretical possibility but also on the availability of kernel artifacts and provider-specific packaging decisions.

Table 3. Comparison of feasibility for building evaluation modules

Category	**kpatch**	**kprobe**
Build time (Average across 5 builds)		
- Azure	36 m 55.06 s	1.54 s
- GCP	54 m 11.04 s	4.72 s
- AWS	49 m 57.39 s	6.43 s
- COCONUT-SVSM	17 m 5.19 s	1.88 s
Required files	Kernel source + vmlinux + Module.symvers	Kernel headers only
Platform dependency	High	Low

5.3 Findings and Countermeasures to the Threat Model

Based on the experimental results, we summarize the configuration conditions under which dynamic modification of the IMA measurement logic becomes feasible and the factors that prevent it, corresponding to RQ1–RQ3.

For RQ1 (observability), visibility depended on IMA policy enforcement and module placement. When a module was loaded from a filesystem excluded under `ima_policy=tcb` (e.g., `tmpfs`), traces of module loading were not recorded in the IMA log in the evaluated on-premise CVM environments, whereas on Azure CVM module loading remained observable even from `tmpfs`. Thus, observability depends on policy enforcement conditions, and module loading can be partially hidden under certain configurations.

For RQ2 (impact of configuration conditions), the decisive factor was not Secure Boot enablement itself, but the operational policy governing MOK enrollment. Even with Secure Boot enabled, if reboot-time enrollment is permitted, a signing key can be enrolled and a signed module loaded. This modification

method was feasible on Azure but not on GCP, where enrollment was effectively restricted. In environments where Secure Boot was disabled or unsupported, modules could be loaded without signature verification. Although AWS lacked vTPM support and Phase 3 was not evaluated, Phase 2b confirmed that module loading was feasible when Secure Boot was disabled. These results show that feasibility depends not only on Secure Boot state but also on configuration conditions such as MOK management strictness.

For RQ3 (configuration management requirements), three factors jointly determine feasibility: (1) root privilege acquisition, (2) Secure Boot configuration, and (3) the MOK enrollment policy. Controlling any of these factors can prevent modification of the IMA measurement logic. Preserving IMA-based RA trust in CVMs therefore requires Secure Boot enforcement and strict control or auditing of MOK enrollment. If Secure Boot cannot be used, root privileges and module-loading mechanisms must be strictly controlled. Figure 6 summarizes these relationships.

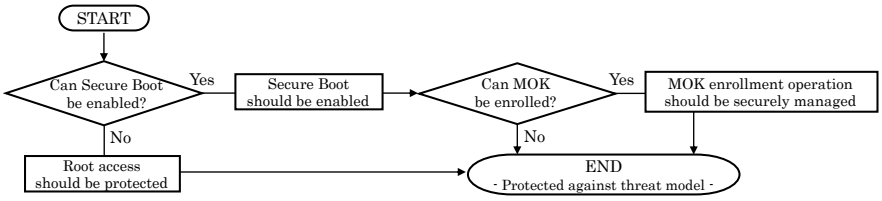

Fig. 6. Countermeasure flow showing how Secure Boot configuration and MOK enrollment control define the trust boundary of IMA-based RA.

6 Discussion

This section discusses the scope limitations of this study and their implications for future work regarding how the trust boundary of IMA-based RA is shaped by configuration conditions, as indicated by our results. We first clarify the assumptions and scope limitations of this study, and then discuss the determinants of RA trust from the perspectives of measurement policy, boot configuration, and kernel modification surfaces.

6.1 Limitations

First, regarding assumptions and evaluation scope, this study focuses on software-layer configuration and operational conditions affecting IMA-based attestation in CVM environments. We assume that an adversary can perform kernel-level operations (e.g., has root privileges) and do not consider physical attacks, firmware modification, or compromise of the hypervisor or TEE implementation. We further assume that Secure Boot and MOK function according

to specification and that the TPM (including vTPM) correctly performs Extend and Quote operations. Accordingly, our conclusions assume that the attestation base (TPM and Secure Boot) operates as specified. Under this assumption, we characterize the configuration-dependent feasibility of modifying the IMA measurement logic. Lower-layer trust anchors, such as the TPM key hierarchy–particularly the binding between the Attestation Key (AK) and Endorsement Key (EK)–have been analyzed in prior work [8]. However, these aspects are outside the scope of our implementation-level evaluation, and a fully integrated assessment of them remains future work.

Second, regarding environment and implementation dependence, our evaluation targets SEV-SNP–based CVMs. Other architectures, such as Intel TDX and Arm CCA, differ in vTPM realization and Secure Boot integration [16, 9, 3]. For example, Intel TDX introduces MRTD and RTMR, which maintain measurements independently of TPM PCRs [17]. Integrating IMA-based PCR extension with such architecture-specific measurement registers into a unified verification model is non-trivial. While evaluation on other CVM architectures is beyond the scope of this study, there is a report suggesting that RTMR may not always be used in an Azure TDX environment [19]. In such vTPM/IMA-based attestation models, similar configuration-dependent behaviors may arise. Moreover, our Verifier checks consistency between the IMA log and the TPM Quote; existing frameworks such as Keylime [35] and Azure Attestation [27] may apply different PCR selections and policy evaluation logic. Therefore, platform architecture and RA implementation choices may influence how the trust boundary is defined and evaluated.

6.2 Future Work

First, regarding policy integration and measurement design, differences in IMA template formats (e.g., `ima-ng`, `ima-sig`), policy settings (e.g., alternatives to `ima_policy=tcb`), and their enforcement can influence both template hash composition and the visibility of module loading events in the IMA log. For example, under `ima_policy=tcb`, module loading from `tmpfs` is generally expected to fall outside the measured scope, creating observability gaps. However, our experiments showed that while such module loading was not recorded in the on-premise environment, it remained observable on Azure CVM under the same policy. Expanding the measurement scope increases visibility but also runtime overhead. Recent work explores fine-grained policy mechanisms for IMA measurement filtering [24]. A systematic analysis of the interaction between platform configuration and measurement policy, and the resulting trade-offs under different configuration conditions, remains future work.

Second, our findings indicate that configuration-aware attestation is a promising direction. As shown in this study, measurement policy alone does not determine whether trust is preserved; Secure Boot and MOK configuration states are practical determinants of feasibility. TPM PCRs reflect boot-time configuration in stages, with `PCR(7)` indicating Secure Boot state and `PCR(14)` reflecting MOK enrollment [42]. Including these PCR values in attestation evidence would enable

verification models that explicitly incorporate configuration state. In practice, verifiers should evaluate configuration-related PCRs (e.g., `PCR(7)` for Secure Boot and `PCR(14)` for MOK state) in addition to log–PCR consistency. Such designs must balance transparency and confidentiality of configuration information. Prior work has also considered confidentiality-aware RA, for example in the work of Eckel et al. [12].

Third, further study is needed on kernel modification and measurement granularity. Even when Secure Boot signature verification is enforced, mechanisms such as eBPF allow dynamic kernel modification independently of module-signature controls [23, 15]. A systematic classification of remaining kernel modification surfaces and corresponding defense layers is necessary. In addition, IMA operates primarily at file granularity and cannot distinguish internal control-flow paths within multi-call binaries such as BusyBox, nor reliably capture in-memory execution that bypasses `exec`. Approaches such as C-FLAT [1] may complement IMA and strengthen integrity verification in CVM environments.

7 Related Work

Research on TPM/IMA-based RA spans multiple layers of the trust chain. To clarify how our study relates to prior work, we organize the literature according to which layer of the RA trust chain is directly addressed. Table 4 positions representative prior work according to whether it directly or indirectly targets the TPM layer or the IMA measurement layer within the RA trust chain.

Table 4. Positioning of Related Work across the RA Trust Chain

Related Work	**TPM Layer**	**IMA Layer**
TPM-FAIL [28]	✓	–
Subvert IMA [6]	–	✓
Ruffin et al. [33]	–	△
This study	△	✓

✓: Directly targets this layer △: Indirectly affects this layer –: Not targeted

Existing studies can be broadly grouped into (i) work targeting hardware-rooted trust in the TPM, including cryptographic processing and key management, and (ii) work targeting the integrity of the IMA measurement layer. The former evaluates the security of signature generation and key handling in attestation, whereas the latter examines how measurement values are generated, recorded, and incorporated into the PCR extension process. This study belongs to the second category. Unlike many prior studies that focus on measurement consistency or its operational use, we experimentally demonstrate that the IMA measurement logic can be modified under real configuration conditions. Our objective is not to introduce a new attack vector, but to clarify when a known modification capability becomes practically realizable in deployed environments.

7.1 Attacks on TPM Hardware Trust

Studies on TPM security primarily analyze the security of hardware and cryptographic implementations. TPM-FAIL [28] analyzes timing side channels in ECDSA signing in TPM 2.0 and demonstrates that secret key material can be inferred externally. With recovered key material, an adversary can generate valid signatures over arbitrary PCR values and Quotes, directly violating the hardware-rooted trust assumption underlying TPM-based attestation. In contrast, our study does not target TPM internals, cryptographic operations, or key material. Instead, we evaluate how the inputs to the TPM Extend operation—namely, IMA-generated measurement values—can be altered through modification of the measurement logic. We therefore do not compromise the attestation signature layer; rather, we analyze how the integrity of the IMA measurement logic affects the overall RA trust chain. This distinction places our focus at a different layer from TPM-FAIL.

7.2 Threats to IMA Measurement Trust

Studies on IMA primarily addressed trust in the measurement and log consistency. Subvert IMA [6] exploits the time gap between measurement and use (TOCTOU: time-of-check to time-of-use) and demonstrates that the content executed at runtime can differ from the content originally measured. In this case, the measurement logic itself remains unchanged; inconsistency arises at the execution stage after measurement. In contrast, our study modifies the IMA measurement functions that generate template data and supply inputs to the TPM Extend operation. We therefore evaluate the IMA measurement layer as part of the trust boundary, rather than the post-measurement execution step. The modified measurement values are extended into PCRs through standard TPM operations, affecting the resulting Quote while preserving log–PCR consistency. Our contribution is to demonstrate that, without compromising TPM internals, configuration-dependent conditions in real environments can influence the RA trust chain through the IMA measurement layer.

Ruffin et al. [33] analyze Keylime deployments and report that consistency may be lost across collection, transmission, and verification stages. They identify false negatives and false positives caused by caching and asynchronous processing, thereby evaluating reliability at the RA framework layer. Although this work does not alter measurement logic, it shows that weaknesses in post-measurement handling can partially undermine the guarantees expected from IMA-based attestation. Thus, while our study focuses on the integrity of measurement logic, Ruffin et al. address reliability in the evidence handling and verification stages, targeting a different layer of the RA trust chain.

8 Conclusion

This study demonstrates that trust in vTPM/IMA-based Remote Attestation (RA) depends not only on consistency between the IMA log and TPM PCR

values, but also on the assumption that the IMA measurement logic remains unmodified. We evaluated under which configuration conditions this assumption holds in SEV-SNP–based CVM environments. By implementing a signed evaluation kernel module that modifies template-generation data in the IMA measurement flow using kpatch and kprobe, we reproduced a scenario in which altered IMA logs and PCR values remain mutually consistent and pass RA verification based solely on log–PCR consistency, without modifying the TPM (vTPM) or the Quote generation mechanism.

Our experiments show that the feasibility of modifying the IMA measurement logic is determined not by Secure Boot enablement alone, but by the operational key-management policy, particularly the MOK enrollment policy. Even when Secure Boot is enabled, if MOK enrollment is permitted, a signed module can alter the IMA measurement logic while preserving log–PCR consistency. When MOK enrollment is effectively restricted, this modification method is blocked. We also show that the observability of module loading is configuration-dependent: depending on IMA policy enforcement and module placement, module-loading events may remain visible in the IMA log or leave no observable traces under certain configurations.

Overall, our results show that the trust boundary of IMA-based RA in CVM environments is determined not only by TPM and IMA mechanisms, but by the combination of root privilege acquisition, Secure Boot configuration, and MOK enrollment policies, while the visibility of module loading may additionally depend on IMA policy enforcement. We characterize this boundary as a configuration-management problem involving Secure Boot enforcement and MOK control and derive corresponding operational and verification implications. We also present a countermeasure workflow linking Secure Boot configuration, MOK enrollment control, and the resulting trust boundary.

Responsible Disclosure

In accordance with responsible disclosure principles, we reported a subset of the behaviors identified in this study that could potentially be misused under certain configuration conditions to the Microsoft Security Response Center (MSRC) and the Information-technology Promotion Agency, Japan (IPA). Specifically, we reported that, when kernel-level privileges are available, it is possible to conceal traces of module loading by leveraging characteristics of IMA's measurement scope and to modify the IMA measurement logic using dynamic patching mechanisms such as kpatch and kprobe, thereby injecting altered file paths and hash values into the IMA event log and TPM PCRs. We clarified that this behavior reflects a configuration-dependent risk rather than a flaw in TPM hardware. The necessary technical information was provided, and the disclosure process was concluded following review by the respective organizations.

Acknowledgments. We used ChatGPT for implementation assistance and editorial support during the preparation of this manuscript. The authors take responsibility for the content. We acknowledge OpenAI for providing ChatGPT.

References

1. Abera, T., Asokan, N., Davi, L., Ekberg, J.E., Nyman, T., Paverd, A., Sadeghi, A.R., Tsudik, G.: C-FLAT: control-flow attestation for embedded systems software. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security. pp. 743–754 (2016)
2. AMD: AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. White Paper (January 2020)
3. Arm Limited: Learn the architecture - Introducing Arm Confidential Compute Architecture. Tech. rep. (2025), https://documentation-service.arm.com/static/6812468479c4c17ad8037525, version 4.0, Document ID den0125_400_en
4. Arnold, Jeff and Kaashoek, M Frans: Ksplice: Automatic rebootless kernel updates. In: Proceedings of the 4th ACM European conference on Computer systems. pp. 187–198 (2009)
5. Birkholz, H., Thaler, D., Richardson, M., Smith, N., Pan, W.: RFC 9334: Remote ATtestation procedureS (RATS) Architecture (2023)
6. Bohling, F., Mueller, T., Eckel, M., Lindemann, J.: Subverting linux'integrity measurement architecture. In: Proceedings of the 15th International Conference on Availability, Reliability and Security. pp. 1–10 (2020)
7. Canonical Ltd.: Secure Boot and Machine Owner Keys in Ubuntu. https://documentation.ubuntu.com/security/security-features/platform-protections/secure-boot/ (2025), (Accessed: 2026-03-09)
8. Chen, L., El Kassem, N., Newton, C.J.: How to bind a TPM's attestation keys with its endorsement key. The Computer Journal **67**(3), 988–1004 (2024)
9. Cheng, P.C., Ozga, W., Valdez, E., Ahmed, S., Gu, Z., Jamjoom, H., Franke, H., Bottomley, J.: Intel tdx demystified: A top-down approach. ACM Computing Surveys **56**(9), 1–33 (2024)
10. COCONUT-SVSM Project: COCONUT Secure VM Service Module (SVSM). https://github.com/coconut-svsm (2023), (Accessed: 2025-11-16)
11. Dave, A., Wiseman, M., Safford, D.: SEDAT: Security enhanced device attestation with TPM 2.0. arXiv preprint arXiv:2101.06362 (2021)
12. Eckel, M., George, D.R., Grohmann, B., Krauß, C.: Remote attestation with constrained disclosure. In: Proceedings of the 39th Annual Computer Security Applications Conference. pp. 718–731 (2023)
13. Eisoldt, J., Galanou, A., Ruzhanskiy, A., Küchenmeister, N., Baburkin, Y., Dai, T., Gudymenko, I., Köpsell, S., Kapitza, R.: SoK: A cloudy view on trust relationships of CVMs–How Confidential Virtual Machines are falling short in Public Cloud. arXiv preprint arXiv:2503.08256 (2025)
14. Fedora Project: Machine Owner Key Enrollment. https://docs.fedoraproject.org/en-US/quick-docs/mok-enrollment/ (2024), (Accessed: 2026-03-09)
15. Gbadamosi, B., Leonardi, L., Pulls, T., Høiland-Jørgensen, T., Ferlin-Reiter, S., Sorce, S., Brunström, A.: The eBPF runtime in the linux kernel. arXiv preprint arXiv:2410.00026 (2024)
16. Intel Corporation: Intel Trust Domain Extensions (Intel TDX) Whitepaper. Tech. rep. (2022), https://cdrdv2-public.intel.com/690419/TDX-Whitepaper-February2022.pdf, document Number 690419, February 2022
17. Intel Corporation: Intel Trust Domain Extensions (Intel TDX) Module Base Architecture Specification. Tech. rep. (2025), https://cdrdv2-public.intel.com/867568/intel-tdx-module-base-spec-348549007.pdf, document Number 348549-007US

18. Kocaoğullar, C., Marjanov, T., Petrov, I., Laurie, B., Cutter, A., Kern, C., Hutchings, A., Beresford, A.R.: A Confidential Computing Transparency Framework for a Comprehensive Trust Chain. arXiv preprint arXiv:2409.03720 (2024)
19. Kuniyasu Suzaki: Lesson from Cloud Confidential Computing Remote Attestation Sample. FOSDEM 2026 Confidential Computing Devroom (2026), https://fosdem.org/2026/schedule/event/RVSEMG-cloud-ra-sample/, (Accessed: 2026-03-09)
20. Lawrence, J., de Souza, M.P.: Kernel Livepatching: An Introduction. https://www.linuxfoundation.org/webinars/kernel-livepatching-an-introduction (2024), linux Foundation Webinar, Recorded May 22, 2024
21. Linux Kernel Community: IMA Policy. https://ima-doc.readthedocs.io/en/latest/ima-policy.html (2023), (Accessed: 2026-03-09)
22. Linux Kernel Community: Integrity Measurement Architecture (IMA) - Event Log Format. https://ima-doc.readthedocs.io/en/latest/event-log-format.html (2023), (Accessed: 2026-03-09)
23. Linux Kernel Documentation: Extended Berkeley Packet Filter (eBPF). https://docs.kernel.org/bpf/ (2023), (Accessed: 2026-03-09)
24. Litchfield, A., Du, W.: XFilter: An Extension of the Integrity Measurement Architecture Based on Fine-Grained Policies. Applied Sciences **13**(10), 6046 (2023)
25. Mavinakayanahalli, A., Panchamukhi, P., Keniston, J., Keshavamurthy, A., Hiramatsu, M.: Probing the guts of kprobes. In: Linux Symposium. vol. 6, p. 5 (2006)
26. Microsoft: OpenVMM and OpenHCL. https://github.com/microsoft/OpenVMM (2024), (Accessed: 2026-03-09)
27. Microsoft Learn: Azure Attestation documentation. https://learn.microsoft.com/en-us/azure/attestation/ (2022), (Accessed: 2026-03-09)
28. Moghimi, D., Sunar, B., Eisenbarth, T., Heninger, N.: TPM-FAIL: TPM meets Timing and Lattice Attacks. In: 29th USENIX Security Symposium (USENIX Security 20). pp. 2057–2073. USENIX Association (Aug 2020), https://www.usenix.org/conference/usenixsecurity20/presentation/moghimi-tpm
29. Narayanan, V., Carvalho, C., Ruocco, A., Almasi, G., Bottomley, J., Ye, M., Feldman-Fitzthum, T., Buono, D., Franke, H., Burtsev, A.: Remote attestation of confidential VMs using ephemeral vTPMs. In: Annual Computer Security Applications Conference (ACSAC 2023). pp. 732–743 (2023)
30. Oracle Corporation: Oracle Linux 10 Working With UEFI Secure Boot. Tech. rep. (2025), https://docs.oracle.com/en/operating-systems/oracle-linux/10/secure-boot/OL10-SBOOT.pdf
31. Red Hat kpatch project: kpatch: live kernel patching for Linux. https://github.com/dynup/kpatch (2014), (Accessed: 2025-11-16)
32. Rezabek, F., Mahhouk, M., Miller, A., Genchev, S., Kilbourn, Q., Carle, G., Passerat-Palmbach, J.: Proof of Cloud: Data Center Execution Assurance for Confidential VMs. arXiv preprint arXiv:2510.12469 (2025)
33. Ruffin, M., Wang, C., Almasi, G., Adebayo, A., Franke, H., Wang, G.: Towards Continuous Integrity Attestation and Its Challenges in Practice: A Case Study of Keylime. In: 2025 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 256–265. IEEE (2025)
34. Sailer, R., Zhang, X., Jaeger, T., Van Doorn, L.: Design and implementation of a TCG-based integrity measurement architecture. In: USENIX Security symposium. vol. 13, pp. 223–238 (2004)
35. Schear, N., Cable, P.T., Moyer, T.M., Richard, B., Rudd, R.: Bootstrapping and maintaining trust in the cloud. In: Proceedings of the 32nd Annual Conference on Computer Security Applications. pp. 65–77 (2016)

36. Scopelliti, G., Baumann, C., Mühlberg, J.T.: Understanding Trust Relationships in Cloud-Based Confidential Computing. In: IEEE European Symposium on Security and Privacy Workshops (EuroS&PW 2024). pp. 169–176. IEEE (2024)
37. Smith, R.: Managing EFI Boot Loaders for Linux: Dealing with Secure Boot. https://www.rodsbooks.com/efi-bootloaders/secureboot.html (2024), originally written: 11/4/2012; last update: 3/24/2024
38. Steffen, A.: The linux integrity measurement architecture and tpm-based network endpoint assessment. Linux Security Summit (2012)
39. tpm2-software community: tpm2-tools: Trusted Platform Module 2.0 user-space tools. https://github.com/tpm2-software/tpm2-tools (2018), (Accessed: 2025-11-16)
40. Trusted Computing Group: TCG PC Client Platform Firmware Profile Specification. Specification (2023), https://trustedcomputinggroup.org/wp-content/uploads/TCG-PC-Client-Platform-Firmware-Profile-Version-1.06-Revision-52_pub-3.pdf, version 1.06, Revision 52
41. Trusted Computing Group: Trusted Platform Module 2.0 Library Part 1: Architecture. Specification (2025), https://trustedcomputinggroup.org/wp-content/uploads/Trusted-Platform-Module-2.0-Library-Part-1-Version-184_pub.pdf, version 184
42. UAPI Group: Linux TPM PCR Registry. https://uapi-group.org/specifications/specs/linux_tpm_pcr_registry/ (v10 Initial Release), (Accessed: 2026-03-09)
43. UAPI Group: Unified kernel image. https://uapi-group.org/specifications/specs/unified_kernel_image/ (v10 Initial Release), (Accessed: 2026-03-09)
44. UEFI Forum: Unified Extensible Firmware Interface (UEFI) Specification, Version 2.10. Tech. rep. (2022), https://uefi.org/specs/UEFI/2.10/, section 32: Secure Boot and Driver Signing

Appendix A IMA Policy Used in Our Experiments

An excerpt of the IMA policy used in our experiments is shown below. These rules measure events such as loading executables and shared libraries, as well as loading kernel modules. At the same time, certain file systems—including `tmpfs` (fsmagic `0x1021994`)—are explicitly excluded from measurement.

```
dont_measure fsmagic=0x9fa0
dont_measure fsmagic=0x62656572
dont_measure fsmagic=0x64626720
dont_measure fsmagic=0x1021994
...
measure func=MMAP_CHECK mask=MAY_EXEC
measure func=BPRM_CHECK mask=MAY_EXEC
measure func=FILE_CHECK mask=^MAY_READ euid=0
measure func=FILE_CHECK mask=^MAY_READ uid=0
measure func=MODULE_CHECK
measure func=FIRMWARE_CHECK
measure func=POLICY_CHECK
```