Vordr: Verifiable, Scalable and Anonymous Remote Attestation for Confidential Virtual Machines

Nirajan Koirala
University of Notre Dame
nkoirala@nd.edu

Kevin Vuong
University of Notre Dame
kvuong@nd.edu

Micah Brody
University of Notre Dame
mbrody@nd.edu

Jihye Kim
Kookmin University
jihyek@kookmin.ac.kr

Hyunok Oh
Hanyang University
hoh@hanyang.ac.kr

Taeho Jung
University of Notre Dame
tjung@nd.edu

Abstract

Confidential virtual machines (CVMs) provide hardware-rooted attestation and isolation, protecting data in use from untrusted cloud infrastructure. However, current CVM attestation frameworks are limited to a two-party trust model between the cloud provider and the CVM owner, and primarily guarantee only the boot-time state. As CVMs increasingly host public-facing workloads (e.g., LLM inference APIs, web applications) that process end-user data, this creates a three-party trust mismatch among the cloud provider, the CVM owner, and end users. Once a workload is deployed, end-users lack cryptographic assurance of runtime integrity and must implicitly trust the CVM owner for any post-launch modifications. Furthermore, extending continuous runtime attestation to a large number of end users introduces severe scalability bottlenecks and is vulnerable to co-location-based attacks. Existing methods either enforce static post-launch lockdown or rely on CVM owner-maintained software logs that are not hardware-anchored. We present Vordr, a framework that removes the CVM owners from the end-user's trust domain across the full CVM lifecycle while still allowing workload-level updates/installations with auditability. We introduce a novel architecture that establishes an exclusive administrative binding between a process-based TEE (Warden Enclave (WEN)) and the CVM. This binding strictly blocks the CVM owners (or cloud) from directly manipulating the CVM. Vordr continuously tracks runtime integrity via a hardware-rooted Linux IMA event log anchored to PCR 10, serving time-bounded, platform-unlinkable cached or audit-ready quotes for independent end-user auditing. We optimize the costly IMA-log extraction via a novel incremental attestation design leveraging the IMA log's append-only structure and the WEN's sealed state. We implement Vordr, validate it across several workloads, and show that it provides up to 60.8× speedup for runtime monitoring with huge communication reductions in steady-state incremental rounds compared to prior methods. Vordr delivers highly scalable and verifiable runtime attestation, providing substantially stronger guarantees for runtime integrity and platform unlinkability.

CCS Concepts

• **Security and privacy → Domain-specific security and privacy architectures**.

Keywords

Intel TDX, Intel SGX, Remote Attestation, Linux IMA

ACM Reference Format:
Nirajan Koirala, Kevin Vuong, Micah Brody, Jihye Kim, Hyunok Oh, and Taeho Jung. 2026. Vordr: Verifiable, Scalable and Anonymous Remote Attestation for Confidential Virtual Machines. In . ACM, New York, NY, USA, 21 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 Introduction

Public-facing cloud services, such as LLM inference APIs [57, 105], Machine-Learning-as-a-Service (MLaaS) platforms [36], and back-end web services [10] often handle sensitive end users' data; however, the established threat models for protecting this data fundamentally exclude the end-user. To secure data in use, the industry relies on hardware-based Trusted Execution Environments (TEEs) [21] over purely cryptographic methods [11, 29] to achieve practical performance. While early process-based TEEs (e.g., Intel SGX) offered application-level attestations that could, in theory, provide end-user trust, their operational complexity [85] and poor scalability [4, 91] have made them impractical for modern cloud deployments. Consequently, Cloud Service Providers (CSPs) and Application Service Providers (ASPs) have aggressively transitioned to Confidential Virtual Machine (CVM) architectures (e.g., Intel TDX, AMD SEV-SNP, Nvidia Confidential Computing) [39, 69]. While CVMs offer easy deployment and practical performance, their design relies on a two-party trust model strictly focused on protecting the ASP from the CSP. This current paradigm inherently assumes the ASP is benign, leaving consumers vulnerable to the ASPs who, if malicious, might abuse their data. Currently, end users have no mechanism to verify that a public-facing CVM is running the intended software. Consequently, consumers face severe privacy risks, as a malicious/compromised ASP can read, harvest, or monetize user data processed within the CVM. Extending the confidentiality and integrity guarantees to the end-users who actually use these services requires fundamentally rethinking CVM architectures beyond the current trust model.

In this public-facing cloud services setting with the three-party model (CSP, ASP, end user), the primary tool for establishing verifiable trust is remote attestation (RA), but it structurally fails the consumer. RA provides a cryptographic mechanism for a remote verifier to confirm that a specific software stack is being executed on genuine TEE hardware without modifications [44, 74]. However, in typical CVM deployments, this attestation evidence only covers the initial boot image and launch-time security configurations [3, 83].

Conference'17, Washington, DC, USA
2026. ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Consequently, post-boot administration, runtime execution, and the application's runtime state changes remain completely invisible to the end user. Consumers must therefore either blindly trust the ASP or demand a complete CVM lockdown that prevents any updates [26]. Bridging this gap requires transitioning from static launch-time measurements to dynamic runtime monitoring that explicitly includes the end user in the trust boundary.

In this work, we propose Vordr, a framework that extends CVM-based confidentiality and attestation directly to the end users. Vordr introduces a novel hybrid architecture that explicitly exploits the distinct strengths of both modern TEE models. It relies on CVMs to execute large-scale, performance-critical applications, while interposing lightweight process-based TEEs to securely mediate all administrative access to the CVM and monitor the runtime state. This allows Vordr to produce cryptographically verifiable evidence of the CVM's measured launch state, alongside a strictly auditable log of all post-launch administrative and runtime changes on the CVM. By securely linking static boot measurements with dynamic runtime logs, Vordr removes the need for blind end-user trust. Vordr provides end users with continuous and verifiable assurance that the code executing inside the CVM matches the approved post-boot software state. This state is anchored to hardware-rooted measurements, which protects the end user from both untrusted CSPs and potentially malicious ASPs.

At the first glance, it may seem possible to solve this three-party trust problem with a naïve solution. Namely, one may simply embed a trusted agent inside the CVM to serve attestations directly, thereby guaranteeing the authenticity and integrity of the services. However, public-facing applications may need to serve thousands to millions of concurrent end users, and when they are hosted on CVMs for security, there will be scalability issues [23]. We empirically demonstrate that this approach fundamentally fails at cloud scale (§7.4) as hardware-based RA was originally designed for a single prover and verifier [44, 74]. Furthermore, enabling end-user attestation is a double-edged sword because RA may be abused to infer the locations of CVMs (e.g., via co-location attacks [17]). To prevent this, Vordr guarantees not only verifiability and scalability, but also anonymity. By treating unlinkability as a first-class requirement [102], our framework explicitly decouples the security-relevant workload state from the stable platform identity. Vordr utilizes process-based TEE mediators to cache and amortize periodically refreshed, time-bounded attestation evidence and strips all hardware-linked material from the attestation artifact served to end-users. To achieve the above security, privacy, and scalability goals, we establish the following core requirements for Vordr:

(R1) Runtime Executed-Code Integrity: End users must be able to cryptographically verify the CVM's initial launch state alongside all post-boot transitions to its executed-code surface. This covers software installations and updates, library loads, kernel module loads, and any code-execution path triggered by ASP-authorized administrative actions.

(R2) Non-Repudiated and Auditable Changes: All post-launch modifications to a CVM must be recorded in a tamper-evident, append-only log, rooted in hardware, that is auditable upon request. Each transition must be cryptographically bound to its authorizer, ensuring that neither the ASP nor the executing process-based TEE can deny an authorized transition once it is recorded.

(R3) Scalable and Anonymous Attestation: The attestation mechanism should be scalable to serve a large number of concurrent end users. This distributed evidence must provide unlinkability for the platform hosting CVMs.

Existing systems address only fragments of our challenge, failing to simultaneously satisfy the comprehensive requirements outlined above. For instance, Revelio [26] enables end-user attestation for web-facing CVMs by largely eliminating post-boot mutability through measured direct boot, immutable storage, and blocked management channels. While suitable for static appliances, it severely restricts operational flexibility, failing to support long-lived services that require routine post-launch updates. Duet [2] shares architectural similarities with our setting; it places an attested SGX controller in front of a CVM and exposes the controller's logs to users. However, it lacks mechanisms for scalable or anonymous attestation, and its post-boot evidence remains strictly controller-centric rather than providing a hardware-rooted, whole-system runtime measurement. Antonino *et al.* [8] similarly compose SGX with SEV to provide more flexible attestation, but they target legacy pre-SNP SEV architectures and point-in-time provisioning rather than the continuous verification of mutable CVM state. Furthermore, while industry deployments often rely on virtual TPMs (vTPMs) or confidential service meshes like Edgeless Systems' MarbleRun [25] to manage CVMs, these architectures are inherently designed to serve the ASP's internal infrastructure management [97]. They rely on ASP-defined configurations and lack the continuous, anonymized, and tamper-evident runtime verification required to protect massive end-user populations from the ASP itself. Finally, OPERA [17] identifies the privacy and scalability pitfalls of direct attestation, including linkability from platform-identifying collateral and bottlenecks under frequent verification, but it is designed strictly for process-based SGX enclaves rather than scalable, long-lived CVMs.

Our Solution. To address these limitations, we propose Vordr, the first end-to-end heterogeneous TEE architecture to simultaneously provide runtime verifiability, strict auditability, and massive scalability specifically designed for cloud-hosted ASP services that serve a large user base. At its core, Vordr establishes an exclusive, cryptographically enforced administrative binding between a process-based TEE and a VM-based TEE (a mutable CVM). By architecturally inverting control, the process-based TEE acts as the sole trusted mediator for both administrative management and end-user attestation, entirely removing the ASP from the post-launch trust boundary. Rather than relying on static, launch-time evidence, the Vordr continuously monitors the CVM via an optimized incremental attestation mechanism that uses Linux IMA measurements rooted in vTPM PCR 10 with OS-level integrity measurements. It maintains a strictly auditable, non-repudiable log of all ASP-requested state changes. To achieve cloud-scale performance and privacy, Vordr fundamentally decouples the generation of RA evidence from its distribution. The process-based TEE verifies original hardware quotes locally, strips stable platform-identifying collateral to prevent cross-session colocation attacks, and issues anonymized, cached attestation artifacts. Consequently, end users receive continuous, scalable proof of the workload's runtime integrity anchored in hardware-rooted measurements verified by the WEN, with platform identifiers withheld by default to preserve unlinkability. For

users who require independent hardware verification, an access-controlled audit mode exposes the raw DCAP evidence, with the resulting unlinkability degradation confined to only credentialed verifier users. In summary, our contributions are as follows:

- We present Vordr, the first end-to-end heterogeneous TEE architecture designed specifically for cloud-hosted ASP services. By securely binding a process-based enclave (SGX) to a CVM (TDX) and enforcing the enclave as the exclusive administrative mediator, we successfully shift the root of trust for post-boot mutability away from the untrusted ASP.
- We design a runtime monitoring mechanism for mutable CVMs based on an incremental attestation method that optimizes the extraction of IMA logs from quadratic to linear complexity and continuously verifies the executed-code state of a mutable CVM on an append-only, tamper-evident log rooted in hardware with a clean extensibility path.
- We expose a range of end-user attestation modes: delegated cached RA, IMA-audit RA with platform unlinkability, and full-audit RA that lets operators trade independent verifiability against platform unlinkability and scalability without changing the underlying enclave-side verification pipeline.
- We implement a full end-to-end prototype of Vordr using Intel SGX and TDX. We demonstrate high practical throughput across diverse workloads and end-user scalability, achieving up to a 60.8× speedup for incremental attestation while strictly preserving end-user verifiability and privacy. Our (anonymous) artifacts are released for reproducibility via https://anonymous.4open.science/r/vordr-4242.

2 Background

2.1 Intel SGX and TDX

TEE architectures broadly fall into two categories. Process-based TEEs (e.g., Intel SGX) execute a single application within a hardware-isolated enclave secured even against the host OS. Virtual Machine (VM)-based TEEs or CVM (e.g., Intel TDX, AMD SEV) isolate an entire guest OS from the hypervisor. While CVMs offer near-native execution performance [16, 20] and easier legacy integration [2, 99], they do so at the cost of a significantly larger Trusted Computing Base (TCB) and weaker threat model than process-based enclaves [18]. Process-based TEEs, on the other hand, are well-suited for small security-critical components, but they scale poorly for I/O-heavy workloads [4, 91] and also require high operational costs for porting existing applications into SGX enclaves [85]. Intel TDX keeps a CVM in a Trust Domain (TD) and isolates it from the hypervisor via hardware-enforced memory encryption and integrity protection, preventing even a malicious cloud operator from inspecting or tampering with CVM memory [18]. Intel SGX operates at the process level, isolating application enclaves from the OS, hypervisor, and all other processes [21]. Its TCB is limited to the processor hardware, microcode, and enclave code itself, yielding a significantly smaller attack surface than VM-based TEEs. Each enclave is identified by its MrEnclave (a hash of code and initial data) and MRSIGNER (the signing authority) measurements. Both TDX and SGX rely on DCAP-based attestation model, in which a platform-specific PCK certificate anchors trust to Intel's PCS.

2.2 Linux IMA

Linux's Integrity Measurement Architecture (IMA) extends the measured boot chain into the operating system by intercepting the kernel's file-open path and computing cryptographic hashes of files at the point of access, specifically, when files are opened for read or execute [81]. It records a cryptographic measurement for every security-relevant event, including executable launches, shared library loads, kernel module insertions, and file reads by privileged processes [81]. Each measurement produces an entry in the IMA event log containing the PCR index, a SHA-1 template hash, the file content hash, and the file path. The template hash is computed over the TLV-encoded template data [37]: $h_i = \text{SHA-1}(\texttt{len}(d_i)\|d_i\|\texttt{len}(n_i)\|n_i)$, where $d_i = \texttt{algo}\|\texttt{":"}\|\texttt{0x00}\|\text{digest}_i$ is the digest field and $n_i = \texttt{filename}_i\|\texttt{0x00}$ is the name field, both null-terminated. Each template hash is extended into PCR 10 register via the standard TPM extend operation:

$$\text{PCR}_{10}^{(i)} = H\left(\text{PCR}_{10}^{(i-1)}\|h_i'\right), \quad \text{PCR}_{10}^{(0)} = \mathbf{0}$$

where h_i' is the template hash recomputed under the bank's algorithm (H), and $\mathbf{0}$ denotes the initial all-zeros register value. The append-extend pair is atomic with respect to other append-extend pairs, ensuring that the measurement log preserves a total order consistent with the PCR value [30]. Based on the policy (e.g., `tcb` policy), IMA measures various files such as executed binaries, memory-mapped libraries, loaded kernel modules, firmware, or all files read by root [76]. A remote verifier obtains a TPM quote, a signature over the PCR bank values, and replays the IMA event log to recompute the expected PCR 10 value. If the replayed value matches the quoted value, the log's integrity can be trusted for appraisal against a reference manifest of expected measurements [42]. It is important to note that IMA measures files at the point of runtime access, not at the point of installation. A package written to disk but never read or executed will not appear in the measurement log. Conversely, every binary, shared library, or configuration file that is accessed during execution will be recorded. This semantic distinction is significant as IMA provides a cryptographic record of what code actually executed, rather than what software is present on disk. For attestation purposes, this is a stronger property, as it captures the executed-code surface that is actually done at runtime.

2.3 Remote Attestation (RA)

Remote attestation (RA) allows a remote verifier to evaluate evidence about whether an attester is in an expected state [12]. In process-based TEEs, such as Intel SGX, the attested object is a single protected process or enclave where the quote binds enclave identity and measurement information to a hardware-backed platform TCB [44]. In VM-based TEEs, such as Intel TDX, AMD SEV-SNP the attested object is instead an entire CVM [45, 46, 74]. This change in granularity shifts the meaning of attestation where process-based RA identifies a specific enclave binary and its configuration, CVM RA primarily establishes that the guest was launched with an expected initial image, configuration, and platform TCB. Consequently, in typical CVM deployments, a valid attestation report does not by itself establish the ongoing application-level runtime integrity of the guest after launch. To provide any runtime guarantees,

the guest-owner (or ASP) must be trusted to explicitly extend the attestation chain using additional runtime measurement mechanisms such as RTMR registers or vTPM-based methods [5, 45, 74].

Intel TDX distinguishes the launch measurement, captured in MRTD, from runtime measurements, captured in RTMRs. RTMRs are initialized at TD build time and may be extended by guest software using `TDG.MR.RTMR.EXTEND` [45, 46]. Thus, RTMRs provide a mechanism for reporting runtime measurements, but they do not enforce what must be measured, when it must be extended, or whether an application update was omitted from the measurement chain. If the ASP controls the in-guest update path, it can modify the running application without extending the relevant RTMR; the resulting quote remains valid, but the verifier cannot distinguish an unchanged guest from an unmeasured update. Even when the ASP faithfully extends RTMR, after each update, end users must still know the expected post-update measurement chain and repeatedly verify fresh attestation evidence. Concretely, quote generation is invoked via Intel's `tdx_att_get_quote()` interface in `libtdx_attest` library, which obtains a signed TDX quote via the platform's quoting infrastructure. Although quote verification can be performed locally via PCCS, or remotely via ITA at an even higher cost, verification is not the dominant bottleneck. The limiting factor is fresh quote generation itself, where each end-user request forces the platform to enter the quoting path and serialize access to attestation key material and signing operations, so additional concurrent verifiers mainly increase queueing delay rather than throughput. As a result, the CVM itself is not a suitable endpoint for serving attestation to a large verifier population. This creates both a trust issue, because users must trust the ASP to measure every security-relevant change, and a scalability bottleneck, because many users independently perform equivalent verification.

In 2023, Intel deprecated the EPID-based RA model [40] and moved to a DCAP-based RA model to improve quote verification scalability via a local PCCS service [43]. DCAP improved the verifier-side scalability, but it did not eliminate the prover-side quote-generation cost. In the standard DCAP model, each fresh end-user challenge must be bound to a newly generated quote [45]. As demand increases, the CVM becomes a queued service point whose throughput is bounded by hardware-backed quote generation rather than local quote verification. In our measurements, a TDX CVM sustains only about 23.5 fresh attestations per second (§7.4). DCAP also changed the privacy model. Unlike EPID, which provided group-signature-based anonymity [13], DCAP quotes are certified using platform-specific Provisioning Certification Keys (PCKs). A PCK is unique to a processor package or platform instance, and TDX quote certification data identifies the corresponding PCK certificate using platform-derived identifiers [41, 45] Thus, exposing raw DCAP evidence to arbitrary end users can reveal stable platform-correlated artifacts, enabling repeated attestations to be linked to the same underlying host. In public clouds, such linkability can strengthen co-location strategies, a known prerequisite for cross-VM side-channel attacks [79, 98].

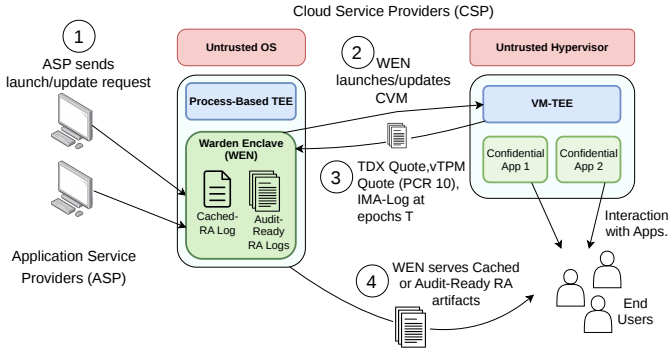


Figure 1: Vordr Architecture

3 Vordr Overview

3.1 System Model

We consider three entities in our framework. First, the CSP owns and operates the physical infrastructure (servers, networks, and the virtualization stack/hypervisor). They could be, for instance, GCP, Azure, Amazon, etc. CSP assists in executing the launch (as enforced by hardware). Second, an ASP requires the TEE resources hosted by CSP to deploy an application that runs on the CVM. They could be early-stage startups [32] or some financial service-providing companies [88] that require TEEs to run privacy-preserving applications serving a large number of end-users. Third, end users interact with the ASP's application or workloads running on the CVM. They require cryptographic evidence that (a) their data is processed by the intended application on the CVM, and (b) the TEE platform and application remain trustworthy not only at launch but throughout CVM runtime (including updates). The process-based TEE used in Vordr for CVM-monitoring is called the warden enclave (WEN).

3.2 Architectural Overview

In Vordr, a WEN launches and monitors the runtime integrity of a CVM hosted inside a VM-based TEE. Runtime monitoring is performed by replaying the guest's IMA event log against an aggregate measurement register. This aggregate can be anchored either directly in TDX RTMRs [104] or in a vTPM PCR such as PCR 10 [24, 47], and the enclave-side replay and verification logic would remain the same in both cases, and only the location and attestation mechanism of the aggregate register differ. To achieve hardware-anchored integrity, the Vordr's architecture requires a cryptographic binding between the runtime measurements and the underlying CVM hardware. We require that the PCR 10 value be authenticated either by extending IMA measurements directly into TDX RTMRs, or by utilizing a vTPM quote over PCR 10 whose attestation key is cryptographically bound to the TDX launch state. We note that PCR 10 with our policy attests executed-code state and does not directly measure data files (e.g., model weights, configuration). Furthermore, our approach is more portable across VM-based TEEs, which do not expose TDX-style runtime measurement registers (e.g., AMD SEV-SNP), and because Intel's RTMR-based IMA approach currently requires kernel changes that are not yet available in upstream Linux [104]. Our approach requires no kernel modifications and runs on production cloud infrastructure, including Google Cloud C3 Confidential VMs. The WEN periodically

attests the CVM at time interval T (epochs) and verifies its runtime state by replaying the IMA event log against the guest's PCR 10 state. Figure 1 outlines our overall architecture.

3.3 Threat Model

We consider the following adversaries during the lifecycle of a CVM and operating WENs. CSPs are considered malicious and may attempt to inspect, modify, or replay protocol messages and may attempt to regain administrative access to the CVM through cloud control-plane mechanisms. We assume that TDX hardware isolation, MRTD-rooted measurement, and the SGX enclave's identity are not compromised, and under this assumption, CSP control-plane manipulations cannot bypass the in-guest authentication path that the WEN configures at commissioning. ASPs are considered malicious as well, and they can request the WEN to launch or make runtime modifications to their CVM alongside remotely attesting it. All CVM modifications requested by the ASP are mediated through the WEN. Each CVM is owned and operated by exactly one ASP, similar to prior works [2, 27]; multi-tenant CVMs with shared administrative authority are outside the scope of this work. Since we operate under a single-WEN model, it can create a single point of failure. We provide details about extending Vordr to a multi-WEN setup in Section F. We consider adversaries that seek to compromise post-launch CVM integrity. ASPs and CSPs are not assumed to remain honest after deployment. ASPs may issue unaccountable updates or repudiate changes they initiated. End users who obtain the RA artifacts may try to co-locate the CVM running the applications they interact with to launch co-location-based attacks.

We assume the WEN bootstrapping a CVM is correctly provisioned and attested. The source code and relevant deployment configuration for applications hosted on CVMs are assumed to be publicly available to enable independent verification of the expected measurements. This transparency assumption is common in attested-service designs [87]. We assume the underlying hardware for all TEEs used in our work correctly implements isolation, memory encryption, and measurement as specified [18, 21]. We do not consider rollback attacks on enclave-persistent state. Such attacks can be mitigated by using an external freshness mechanism, such as a tenant-controlled append-only log [63] or consensus-backed storage that records a monotonic checkpoint version [100].

Platform Privacy and Unlinkability. Adversaries can repeatedly interact with the service as an end user (i.e., verifier) and may launch their own instances and attempt to co-locate. This matches the standard co-location threat model in public clouds, where attacks combine a launch/placement strategy and a co residency detection mechanism [79, 98]. Vordr provides platform unlinkability for external verifiers. The evidence released to end users omits stable hardware-bound platform-identifying attestation artifacts (e.g., PCK certificates, `fmspc`, `qe_id`) of the underlying CVM host platform, so an efficient adversary observing only this evidence cannot link two successful sessions to the same physical host assuming uncompromised TEEs and manufacturer provisioning infrastructure [17]. We note that this property is limited to attestation-layer linkability. We do not prevent adversarial placement or VM-launch strategies for inducing co-residency [79, 98], co-residency detection through performance or side-channel signals [98]. Since WENs are also remotely attested, their raw DCAP evidence can expose similar platform-correlated artifacts. To enable WEN unlinkability, we can achieve it operationally by dedicating the whole platform to WEN host only, preventing adversarial enclave co-location on the same platform. The unlinkability guarantee is held for modes 1 and 2 in our framework (see §5.2.2). Mode 3 achieves full auditability at the cost of unlinkability in Vordr but only to a set of authenticated users.

4 CVM Commissioning via Enclave

The CVM needs to be commissioned via WEN before the runtime-monitoring protocol runs in Vordr. We refer to commissioning as the bounded, WEN-mediated procedure that transitions a CVM from an unattested launch request into an operational state suitable for Vordr's runtime monitoring. We organize the commissioning phase into the following three stages: launch authorization, enclave-only administrative provisioning, and IMA-based launch integrity verification, as discussed in the sections below. We assume the WEN is equipped with a mechanism to obtain credentials for accessing the cloud services to launch CVM without relying on host OS-controlled storage. Concretely, one can achieve this by sealing data to the enclave (e.g., MrEnclave in SGX), where the SGX hardware encrypts a blob such that only the exact enclave binary can decrypt it [21]. Alternatively, WEN can also connect to a separate credential server that verifies the enclave's quote before issuing a token for connection to the cloud [35, 61].

4.1 WEN Init and Launch Authorization

At startup, the WEN loads authorized ASP public keys and generates an enclave-bound key pair (pk, sk), where sk acts as the provisioning SSH key. To anchor its identity, WEN embeds the hash $h = \mathsf{H}(pk)$ into its enclave report data. Before sending a provisioning request, the ASP verifies the WEN's quote Q and confirms the embedded hash matches $\mathsf{H}(pk)$. The ASP then submits a signed CVM configuration request. Upon verifying the ASP's signature, the WEN translates the requested configuration into a platform-specific launch command (e.g., a compatible Intel TDX machine).

4.2 Enclave-Only Administration.

In order to meet the requirements established in §1, after commissioning, the WEN must be the only entity permitted to establish administrative remote access to the CVM in our model. No other party, including the ASP, the CSP, or any CSP IAM principal, should be able to obtain remote shell, root, or management access to the CVM. We note this does not preclude the workload running inside from exposing application-level service endpoints to end users.

We achieve this through a combination of cryptographic and operational controls, and we distinguish these explicitly. First, the remote-access private key sk generated during provisioning is kept exclusively in enclave-protected memory and is never written to disk or transmitted. Second, the WEN installs firewall rules that restrict ingress on management ports (e.g., TCP 22 for SSH) to the WEN's external IP address only, preventing administrative connections from any other source. Third, at CVM creation, the WEN disables cloud-managed remote access paths, including project-wide SSH key injection, IAM-based SSH login, and metadata-based

guest communication channels [33]. We additionally address a post-launch key-injection path not considered by prior work [2]. More broadly, major CSPs expose provider-managed mechanisms that can modify in-guest SSH access after VM launch through metadata services, guest agents, or management extensions [6, 31, 62]. To counter this, the WEN disables provider-managed access mechanisms and blocks guest access to the corresponding metadata or control endpoints. Finally, the WEN marks `authorized_keys` used for SSH immutable using `chattr +i`, which prevents any subsequent modification of the file.

4.3 IMA-based Launch Integrity Verification

In Vordr, we employ IMA-based runtime attestation, hardware-rooted in the PCR-10 register, to be able to achieve the requirements R1 and R2. IMA measurement is commonly enabled by specifying `ima_policy=tcb` as a kernel boot parameter, which the kernel parses during initialization [59, 93]. This is the standard approach documented by Keylime, Red Hat, and openSUSE, all of which require modifying the boot configuration and rebooting the system [52, 72, 77, 78, 82]. SNPGuard [103] avoids rebooting by embedding `ima_policy=tcb` into a custom first-stage image, but this requires a custom image build pipeline. Since GCP TDX CVMs use direct kernel boot (bypassing GRUB), Vordr enables IMA dynamically via `/sys/kernel/security/ima/policy`. Prior to any system configuration, the WEN atomically writes rules for binaries (`BPRM_CHECK`), shared libraries (`MMAP_CHECK`), and kernel modules (`MODULE_CHECK`), mirroring the executable subset of the built-in `tcb` policy [59]. These rules take effect immediately and are permanently locked by the kernel's write-once design [59]. We note that this yields first-launch measurement from the point of policy activation onward, including subsequent package installation and configuration steps, but it does not retroactively cover earlier boot-time accesses. Such accesses can be measured using authenticated evidence of the boot chain covered via MRTD in Intel TDX or measured direct boot [64] in AMD SEV-SNP.

Reference manifest verification. During commissioning, the WEN verifies each IMA entry against a reference manifest $\mathcal{M}_{\text{ref}}$, a precomputed allowlist mapping tuple ($algo$: $hash$) to known-good file paths. Indexing by content hash makes verification order-independent and constant-time per entry, which is necessary because benign executions may access the same files in different orders across runs. We generate $\mathcal{M}_{\text{ref}}$ from a known-good commissioning run on a clean image after hardening and package installation, recording all file-backed IMA measurements except `boot_aggregate`. For instance, for Ubuntu 22.04, the manifest contains 590 distinct hashes covering 595 paths. The manifest is embedded in the WEN enclave and is therefore covered by MrEnclave; any modification is detectable through WEN attestation. During verification, any IMA entry absent from $\mathcal{M}_{\text{ref}}$ indicates unexpected code execution, causing the WEN to abort and tear down the CVM. When the base image or installed package set changes, the manifest must be regenerated from a fresh known-good run. This is an operational requirement shared by prior allowlist-based IMA verification systems such as Keylime. [52, 82].

Security via IMA. Our IMA-based verification detects several attack classes that baseline CVM-based attestations alone cannot detect. The original IMA design [81] was motivated precisely by this gap to enable static boot measurements (analogous to MRTD in TDX) that cannot capture post-boot execution of malicious code. For instance, it can detect trojanized shared libraries (e.g., the XZ Utils supply chain attack [58]) or malicious systemd services or init scripts injected into the VM image that executes at boot [14, 22, 65]. It also protects against unauthorized loadable kernel modules, which are the de facto delivery mechanism for the Linux kernel rootkits [55, 90] and user-space library rootkits that replace or preload shared objects or direct file replacement [54, 96], whose modified hashes would not match the reference.

5 Runtime CVM Monitoring and Attestation

In our model, the ASPs are allowed to update/install various packages and applications into the CVM during its runtime via WEN. To enable end-users to verify such changes, every command requested can be logged with a timestamp, the command, and CVM's output.

Prior work [2] uses enclave-protected logs of SSH commands and their outputs to provide accountability for enclave-mediated administrative actions. However, they are insufficient on their own to establish runtime integrity of the CVM. The reason is threefold. First, a command transcript records the request and the CVM's textual response, but not the exact executable artifacts that were subsequently loaded, their content hashes, or whether the resulting binaries match the intended software execution. Second, output logs returned to the enclave are incomplete by construction as they do not cover execution paths that bypass the enclave, such as code launched by a compromised network-facing process or other in-guest activity. Third, their evidence value depends on userspace output generated by the CVM itself that is not hardware rooted, and thus may be misleading if the guest is compromised. We therefore complement the enclave with IMA-based runtime measurement. IMA measures file contents in the kernel before access according to policy, records the resulting event log, and extends the measurements into PCR 10, allowing validation of post-launch executable content via WEN [52, 59, 81].

5.1 Logs and Audit Model via IMA

IMA measures files when they are accessed under the active policy. Our policy is scoped to three hook types that include `BPRM_CHECK` (executables being launched), `MMAP_CHECK` (shared libraries being mapped into memory), and `MODULE_CHECK` (kernel modules being loaded). This scope reflects our threat model. For runtime integrity, we verify that the code currently executing is the code that was approved/logged through the WEN-mediated administrative path, but we do not attempt to measure every data file the workload may read. We therefore omit `FILE_CHECK uid=0`, which is included in the standard `ima_policy = tcb` preset and measures every file opened by root. This rule is the primary driver of the large measurement counts and boot slowdowns reported in prior work [89], and the bulk of those entries are non-executable data files (logs, runtime state, transient I/O) that fall outside our threat model for executed-code integrity. We make this scoping decision explicit as our policy is designed to attest the executed-code surface of the CVM, not the contents of every data file the workload may consume. Configuration files, model weights, tokenizers, and other workload-specific

data files that are accessed via `read()` or non-executable `mmap()` are not measured under this policy. We discuss how operators can extend coverage for application-level data integrity in §6. We note that IMA does not provide a complete account of every filesystem change. To make this evidence independently auditable, the WEN can further export the IMA measurement list together with the vTPM quote over PCR 10 and sign the combined artifact inside the enclave before releasing it to end users. An end user can then replay the log against the quoted PCR 10 value and verify the measured hashes against an expected policy, such as a reference manifest for the approved base image and package set [52, 53, 82].

IMA Measurement to Enforceable Audit. Our design transforms IMA from a passive monitor into an enforceable audit trail. While conventional IMA detects unauthorized execution, it cannot attribute the event to a specific administrative action. Because only the WEN exclusively mediates all CVM administration, every file-level IMA event must correlate with an authenticated ASP request. When the WEN executes a requested update, it verifies that the resulting IMA measurements match the operation's expected executed-code footprint, and any deviation immediately flags an unapproved execution path. By cryptographically binding authorized administrative intent to actual runtime effects, this gives IMA a stronger role than passive measurement alone, as WEN can relate authorized operations to their measured file-access effects and expose deviations for audit.

5.2 Incremental Attestation

After provisioning, the WEN periodically attests the CVM at certain epochs T and serves the latest attestation quote to end users (R1), stripping platform-identifying information to preserve quote anonymity (R3). WEN deploys an in-guest agent inside the CVM during launch that periodically returns three artifacts: (i) the TDX hardware quote binding the platform state, (ii) the IMA, the newly appended IMA event log entries, and (iii) a vTPM quote over the current PCR 10 value, signed by a vTPM key that is cryptographically anchored to the TDX platform. Inside the enclave, the WEN verifies the quote's signature and nonce to validate the CVM's quote and anchor the vTPM's identity. It then verifies the signature on the vTPM quote to guarantee the authenticity of the reported PCR 10 state. Finally, it replays the IMA log against this cryptographically verified PCR 10 value, confirming each entry's template hash. If any entry fails this consistency check, or if PCR replay does not match the quoted PCR 10 state, the WEN returns a runtime violation flag to the end-users. The per-round check is syntactic as it establishes that the log is well-formed, append-only, and consistent with the quoted PCR 10 state, detecting truncation, reordering, and fabrication. Semantic approval, i.e., whether the measured binaries are approved software, is workload-specific and exposed to end users via the IMA-audit and full-audit modes (§5.2.2).

A naïve implementation of the above IMA-based runtime attestation re-reads and re-verifies the *entire* IMA measurement log at every attestation round. We found that, on Linux, the bottleneck of this design is not cryptographic replay, but kernel-side regeneration of the IMA log through the `seq_file` interface [60, 94]. To mitigate such a bottleneck, we design an optimized incremental attestation protocol that exploits two key properties of our setting: (i) the IMA log is append-only within a boot epoch, and (ii) Vordr's WEN maintains enclave-protected state across attestation rounds. This reduces the per-epoch attestation cost from full-log replay to $O(\Delta n)$, rendering the common case ($\Delta n = 0$) constant-time.

5.2.1 Bottleneck in Full-Replay Attestation. The Linux kernel exposes the IMA log through the pseudo-file under `/sys/kernel/security/ima/ascii_runtime_measurements` [60]. This interface is implemented using `seq_file`, which is designed for virtual files whose contents are generated on demand rather than stored contiguously on disk [94]. Unlike a regular file, this pseudo-file generates its content on demand by iterating over a doubly-linked list of IMA's queue entry structures, where the read path is lock-free, using RCU (read copy update) lock, so concurrent IMA measurements are not blocked during reads [95]. A `seq_file` reader advances through the linked list via an iterator, and the next `read()` session resumes from the most recent logical position. However, for iterator implementations backed by linked structures, reaching that logical position requires walking the structure from its head on each session boundary [94]. Concretely, each `read()` system call fills a kernel buffer (typically about 4 KB, holding 15–20 IMA entries) and returns to userspace. On the *next* `read()` call, `seq_file` invokes `ima_measurements_start()`, which re-traverses the linked list from the head to the current logical position before resuming iteration [94]. This behavior is problematic for the IMA log because a large measurement list is consumed through many successive `read()` system calls. Suppose N denotes the total number of log entries, and let B denotes the number of entries returned per buffer fill (typically $B = 20$). A full replay requires about N/B read calls. If each successive call reconstructs the iterator state by traversing to the current logical position, then the cumulative traversal work is,

$$T_{\text{traversal}} = \sum_{k=1}^{N/B} k \cdot B \approx \frac{N^2}{2B}. \tag{1}$$

Thus, the kernel-side cost of only generating the measurement list can dominate the total attestation latency, even before the verifier (WEN) replays hashes against PCR 10. For large values of N, this yields huge pointer dereferences (about 490 million for 140K entries) far exceeding the single-pass N traversals one might expect.

Prior systems [80] have also recognized that IMA attestation should be incremental, and the IMA documentation explicitly recommends designing incremental attestations [60]. Currently, Keylime follows this approach at the verifier protocol layer by maintaining per-agent incremental attestation state, including the next IMA entry to request, remembered PCR state, and boot-time metadata, and its APIs request only the suffix of the measurement list beginning at a stored offset [49–51]. However, prior designs optimize only the verifier-side replay cost from $O(N)$ to $O(\Delta n)$. They do not address the attester-side extraction bottleneck. In Keylime, the agent reopens the IMA pseudo-file on each request and seeks to the stored offset, which still incurs $O(N^2/B)$ kernel-side traversal cost as the seek-to-offset phase alone requires N/B `read()` calls, each re-walking the linked list from its head (Equation 1).

Our design differs from existing IMA-based attestation systems in three key respects. First, it optimizes the attester-side extraction path by keeping the IMA pseudo-file descriptor positioned across rounds and reading only newly appended entries, reducing

Table 1: Per-round asymptotic cost comparison. N: total IMA entries, Δn: new entries, B: entries per `seq_file` buffer fill.

Component	Naïve	Incr. prior	Ours
Log extraction ($\Delta n = 0$)	$O(N^2/B)$	$O(N^2/B)$	$O(1)$
Log extraction ($\Delta n > 0$)	$O(N^2/B)$	$O(N^2/B)$	$O(N \cdot \Delta n/B)$
Replay/verification	$O(N)$	$O(\Delta n)$	$O(\Delta n)$

extraction cost to $O(\Delta n)$ and to $O(1)$ when $\Delta n = 0$. Second, the rolling attestation checkpoint in Vordr is maintained inside the WEN's SGX enclave and sealed across attestation rounds, rather than being entrusted to a conventional verifier host and its unprotected software state. Third, because the checkpoint is sealed inside the enclave rather than held only in verifier process memory, it survives WEN restart or replacement without forcing a full replay from entry 0. [19]. Because the WEN is a persistent, stateful entity that repeatedly attests the same CVM across rounds, it can coordinate the in-guest agent's file descriptor lifecycle through control commands (e.g., reset and Δn generation). A stateless or one-shot verifier would require the agent to reopen the IMA pseudo-file on every request, falling back to the $O(N^2/B)$ extraction cost. We note that SGX sealing protects checkpoint confidentiality and integrity, but not freshness, since a host can replay an older, valid sealed checkpoint unless the enclave is coupled to an external rollback-resistant state service. Table 1 compares the asymptotic costs of three IMA-based runtime attestation approaches.

5.2.2 Vordr's Incremental Attestation. Vordr's incremental attestation protocol exploits the append-only nature of IMA logs within a boot epoch, along with WEN's sealed enclave state across attestation rounds. Instead of re-reading and re-verifying the full IMA log, Vordr verifies only the newly appended suffix and updates a sealed checkpoint. Our protocol exposes three end-user attestation modes that trade independent verifiability against platform unlinkability and serving throughput. All three modes share the same incremental enclave-side verification pipeline, and they only differ in how much evidence the WEN forwards.

• Mode 1 — Cached RA (delegated verification). WEN exports a compact, enclave-authenticated summary of the latest verified snapshot, including freshness, verdicts, TD identity, and hashes of the DCAP quote, vTPM quote, IMA state, and authorized command history. End user verifies only the WEN's enclave signature and trusts the measured WEN to have correctly executed the full chain verification (DCAP report data, vTPM quote over PCR10, IMA replay). It is intended for typical end users (e.g., users of an LLM inference API) who interact with the service and have no operational means of running DCAP verification themselves. This mode delivers full unlinkability and the throughput to serve large concurrent users.

• Mode 2 — IMA-audit RA (workload-level independent verification). WEN releases (i) the IMA log or its verified delta, (ii) the vTPM quote over PCR 10, and (iii) the SGX-authenticated ASP command log. No PCK collateral, `fmspc`, or `qe_id` of the underlying TDX platform is released. The user can independently verify the vTPM signature, replay the IMA log against the quoted PCR 10 value, and audit each IMA entry against their own reference manifest if available. We deliberately push semantic approval to the end user rather than enforcing a complete WEN-resident allowlist as the CVM is a live system in which ASPs routinely install/update software. It lets each user apply their own risk criteria (e.g., reject a CVM that installed `curl` or `netcat`), analogous to supply-chain transparency logs. This mode preserves platform unlinkability while giving the user direct visibility into the workload's executed-code state. It is intended for users who require workload transparency (e.g., to audit ASP-installed packages against their own risk model) but accept the WEN as the trust anchor for hardware identity.

• Mode 3 — Full-audit RA (end-to-end independent verification). WEN additionally releases the raw DCAP quote together with its PCK certificate chain, enabling the end-user to verify the entire chain . This mode trades platform unlinkability for end-to-end independent verifiability. To prevent the degradation of the platform's anonymity, mode 3 can be gated by an operator-managed authenticated-verifier list maintained inside the WEN's sealed state. End users not on the list are served only by modes 1 or 2, and mode 3 can be served only to an authenticated subset of users (regulators, internal auditors, contracted security reviewers) for whom co-location risk is already mitigated through external means.

Our protocol assumes a persistent attestation agent inside the CVM, and it is considered a part of the TCB for availability. We note that a compromised agent can refuse the service, but it cannot violate integrity in our model since it cannot forge a valid incremental measurement suffix without causing replay to diverge from the quoted PCR state. Figure 2 summarizes our incremental attestation protocol.

.

6 Security Analysis and Discussions

At boot, the WEN remotely attests the CVM via the TDX quote, which is signed by the platform's Quoting Enclave under the PCK and covers MRTD over firmware (TDVF), initial memory, VM configuration, kernel, and initrd [18]. The WEN verifies MRTD against a whitelist of known-good measurements and any tampering with the kernel, initrd, or firmware by a malicious hypervisor produces a divergent MRTD and is rejected. This also covers rollback attacks on the CVM image itself, since an outdated image yields an outdated MRTD. The TDX quote does not cover the root filesystem by default. Works such as Revelio [26] use `dm-verity` for hash-tree-based rootfs verification. Similarly, we can compose our framework with `dm-verity` where the read-time slowdown is acceptable.

Sealed checkpoint integrity. The security of the incremental protocol depends critically on the integrity of the sealed checkpoint (k, p, b) (Figure 2). An adversary who can tamper with this state can mount two attacks. First, by advancing k past entries that contain violations, the adversary causes the WEN to skip those entries entirely; subsequent rounds verify only the suffix beyond k and never revisit the suppressed violations. Second, by forging the stored PCR value p to one that already accounts for the malicious entries, the adversary ensures that incremental PCR replay starting from p matches the quoted PCR 10 state, masking the inconsistency. Both attacks allow a compromised CVM to pass attestation despite having executed unauthorized binaries. By sealing the checkpoint inside the WEN's SGX enclave, Vordr ensures that (k, p, b) is bound to the enclave's identity (MrEnclave) and inaccessible to the host OS, hypervisor, or cloud provider.

State at WEN: Sealed checkpoint (k, p, b), where k is the number of verified IMA entries, p is the replayed PCR 10 value, and b is boot-time metadata.
State at CVM Agent: Open pseudo-file descriptor `fd` for the IMA measurement log.

(1) **Initialization.** On the first attestation round of a boot epoch, the agent reads the full IMA measurement list from `fd`, obtains a TPM quote over PCR 10, and sends both to the WEN.
(2) **Full verification.** WEN verifies the log against policy, replays it until the replayed digest matches the quoted PCR 10 value, and seals (k, p, b).
(3) **Count check.** Before each later round, the agent reads `runtime_measurements_count`. If the count equals k, it returns only freshness evidence (e.g., a new quote), & round terminates.
(4) **Delta extraction.** If the count is $k + \Delta n$, the agent reads only the newly appended suffix from the still-open `fd`, obtains a fresh quote, and sends the Δn new entries to WEN.
(5) **Delta verification.** WEN checks boot-time consistency with b, replays only the Δn new entries starting from checkpoint p, and accepts iff replay reaches the quoted PCR 10 value.
(6) **State update.** On success, WEN seals the new checkpoint $(k + \Delta n, p', b)$. On failure, it marks the CVM untrusted and may fall back to full replay.
(7) **Disclosure.** WEN releases evidence to end user based on their requested mode and verifier-list status: mode 1 (default), mode 2 additionally returns vTPM over PCR 10, the IMA log/delta, and the command log, and mode 3 additionally returns raw-DCAP quote with PCK cert. (permitted only to verifiers on authenticated-verifier list).

Figure 2: Vordr's incremental attestation protocol. WEN keeps a sealed rolling checkpoint, while the CVM agent preserves an open handle to the IMA pseudo-file and sends only newly appended entries after the initial round.

Network isolation. The WEN is the sole authorized administrative endpoint for the CVM, and inbound access is restricted at the cloud firewall to the WEN's authenticated channel, eliminating remote login and exposed-service exploitation as code-injection vectors. For outbound traffic required by workloads, the WEN configures egress allowlists. Any externally fetched binary that is subsequently executed is captured by IMA and reflected in PCR 10, making such activity detectable at the next attestation epoch.

Verification vs. Unlinkability. As noted by OPERA [17], releasing PCK certificates and other platform-identifying material in TDX quotes enables cross-session host re-identification, a known prerequisite for cross-VM co-location attacks [79]. Vordr's three attestation modes (§5.2.2) enable distinct verification-anonymity trade-offs. Cached RA delegates verification to the WEN in exchange for full unlinkability and maximum throughput. IMA-audit RA gives the user independent verification of workload-level evidence (vTPM quote, IMA replay) while still preserving platform unlinkability. Full-audit RA returns the raw DCAP quote and PCK chain, which restores end-to-end independent verifiability at the price of platform identifiers.

Coverage and Extensibility. The combination of (a) SGX controlled network isolation, (b) IMA measurement providing a TPM-anchored, append-only record of executable launches, library mappings, and kernel module loads from policy activation onward, and (c) MRTD-rooted attestation of the launch image enables a verifier to establish what the CVM was at launch and what code has executed since policy activation. Vordr's incremental attestation protocol (§5.2) periodically verifies new IMA entries, further bounding the window of undetected unauthorized execution. We note that our IMA policy attests the executed-code surface (binaries, shared libraries, kernel modules) and detects classical post-boot integrity attacks such as binary replacement, library trojanization, kernel rootkits, and unauthorized package installation. Workloads requiring integrity of data files (e.g., LLM model weights, tokenizers, prompt templates) are not covered by this policy, but Vordr can extend to such cases via workload-specific IMA hooks, IMA-Appraisal for signed artifacts, or file system integrity via `dm-verity`.

7 Experimental Evaluation

We implement our Vordr using Intel SGX as the WEN and launched the CVM as Intel TDX via the WEN on Google Cloud. For the WEN, we ran it on a machine with Intel Xeon Gold 5412U equipped with 405 GB RAM and SGXv2 on Linux 6.8.0-63. All CVM experiments were conducted using a Google Cloud Intel TDX c3-standard machines with 4 CPUs, 14 GB RAM, 50 GB NVMe boot disk, and 8 CPUs, 32 GB RAM, 200 GB NVMe boot disk, respectively, for incremental attestation test and workload benchmarking. All VMs were using a Ubuntu 22.04.5 LTS (Linux kernel 6.8.0-1053-gcp). To isolate the performance of our $O(\Delta n)$ extraction and cached RA scaling, our prototype retrieves the IMA log and PCR 10 directly via the in-guest agent. We defer the full vTPM cryptographic binding due to current GCP infrastructure limits, as the default `6.8.0-gcp` kernel lacks upstream support to route IMA directly into TDX RTMRs, and securely binding GCP's software vTPM to the hardware quote requires a custom TDX-enlightened vTPM architecture. Because vTPM quote generation is a constant-time operation with respect to the IMA log length [68], its absence does not significantly alter our asymptotic scaling results.

7.1 Commissioning CVM with Vordr

First, we report the time to commission (launch) a ready-to-use CVM under Vordr compared to Native and TDX-only execution in Figure 3. We require a one-time commissioning cost when an ASP requests launch of a new CVM through Vordr. The Native and TDX bars measure only infrastructure bring-up to first administrative access. By contrast, the TDX-Vordr bar measures the full ASP-triggered commissioning workflow needed to make the CVM remotely attestable. Its GCE create phase includes WEN-side orchestration around CVM launch, including ephemeral SSH-key generation, firewall-rule setup, and VM creation. The SSH-ready phase largely consists of WEN's fixed 60s, post-boot stabilization wait rather than a pure first-SSH latency. Security setup covers the guest-side preparation step including loading the IMA policy, hardening SSH, installing baseline packages, and preparing the CVM for integrity checking. Here, SSH hardening refers to tightening the CVM's default SSH configuration, e.g., disabling password-based access and restricting login behavior, before the instance is exposed for normal operation. During IMA verification, WEN validates the guest's IMA log and PCR-10 against the expected launch state and DCAP-libs installation stage installs the Intel TDX/DCAP

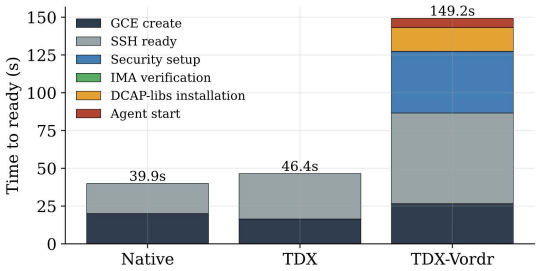


Figure 3: Total time to launch a VM via GCP

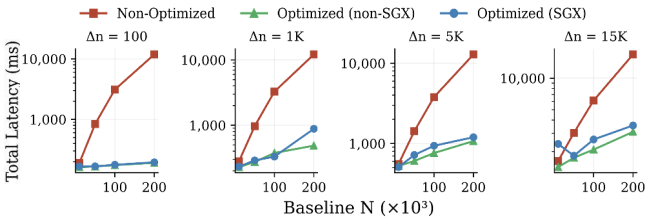


Figure 4: Total runtime attestation across baseline IMA logs

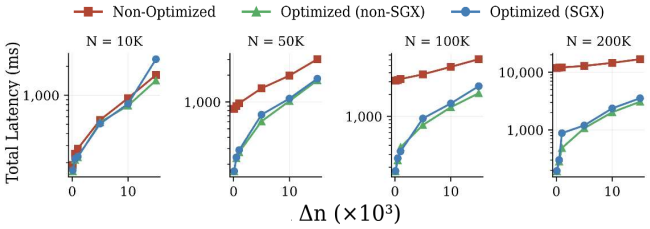


Figure 5: Total Attestation latency across update size (Δn)

user-space libraries and supporting services needed for local quote generation. Agent start is the time to launch the attestation agent and wait until it is reachable via TCP.

7.2 Incremental Attestation Benchmarks

We microbenchmark our optimized incremental attestation method by running an agent server on CVM that returns the runtime attestation quote that contains the IMA log, TDX quote, and PCR 10 value to the WEN. Inside the SGX enclave, each attestation round performs the TLS client exchange with the CVM, simple DCAP quote verification (ECDSA-P256 signature over the TD Report plus nonce binding), and per-entry SHA-1 template-hash self-consistency check over the received IMA log delta plus rolling hash on those values to check if it matches the received PCR 10 hash.

To quantify the performance advantage of our optimized incremental attestation protocol, we conducted a controlled microbenchmark comparing three attestation strategies across varying IMA log sizes and update sizes ($\Delta \cdot n$). We vary two parameters across the experiments: the baseline IMA log size $N \in \{10K,\ 50K,\ 100K,\ 200K\}$ entries, representing the accumulated measurement history of the CVM and the per-epoch update size $\Delta n \in \{100,\ 500,\ 1K,\ 5K,\ 10K,\ 15K\}$ new entries, representing the runtime activity since the last attestation. IMA entries are generated by creating and executing shell scripts on the CVM, each of which the kernel measures and

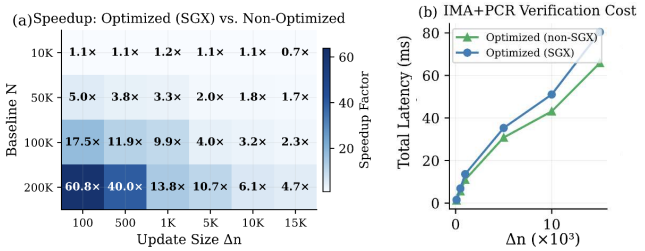


Figure 6: Speed-up with security guarantees of SGX (left). Overhead of SGX enclaves for IMA + PCR verification (right).

appends to the IMA log before execution. We provide more details behind this experimental setup in Appendix A.

Figure 4 shows total attestation latency for four update sizes Δn. The non-optimized baseline scales linearly as it re-reads the entire IMA log on every round, whereas both optimized variants remain nearly flat in N and track Δn instead . The optimized (SGX) curve lies within 10–20% of optimized (non-SGX). It shows that non-optimized mode is essentially insensitive to Δn. Both optimized variants grow approximately linearly in Δn. Once we establish the baseline, the runtime cost in our incremental design is governed by the update size only, rather than the accumulated log length. Figure 6 (a) plots the speedup of optimized (SGX) over the non-Optimized baseline across the $(N, \Delta n)$ grid. Our optimization is most valuable precisely where production CVMs operate, i.e., large, long-lived logs with a small number of new measurements per round. Figure 6(b) isolates the cost of the main component that actually runs inside the SGX enclave, joint IMA integrity checking, and PCR-10 extend-chain replay, as a function of the update size Δn (averaged across N). We found the average SGX overhead to be around +21%. The protocol transfers the same payload in all three methods, a request message that carries the current IMA byte offset (50 B) and a reponse carrying (i) a DCAP quote (4 KB), (ii) the boot-time attestation fields, (iii) the CVM's latest PCR-10 value (a hex-encoded 20-byte SHA-1 digest, 40 B) together with its running IMA entry count, and (iv) the Δn new IMA entries. The IMA payload dominates, growing linearly in Δn at roughly 0.21 KB per entry, 21 KB for Δn=100, 208 KB for Δn=1K, and 3.15 MB for Δn=15K, irrespective of the baseline N. All modes (non-optimized, optimized (non-SGX), and optimized (SGX)) send exactly these Δn entries. Hence, the communication cost is identical across methods and is fully bounded by Δn.

7.3 Evaluating Vordr on Different Workloads

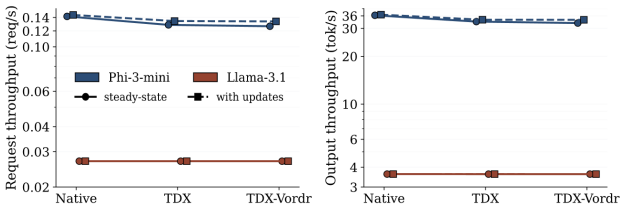
We evaluate Vordr on two end-to-end application workloads to measure the operating cost of a CVM with runtime monitoring via our method. The first is an LLM-serving deployment running `vLLM` [56] (compute-bound, multi-second per-request latency) and the second is a tuned NGINX static-file server (syscall-bound, sub-millisecond per-request). For each workload, we compare three target-VM configurations: Native (no TDX), TDX (TDX on, no Vordr), and TDX-Vordr (full Vordr stack) on the same `c3-standard-8` hardware on Google Cloud. We define the attestation epoch T as the time period in which WEN performs the runtime attestation for CVM once. For each workload, we vary the attestation epoch

$T \in \{15, 30, 60, 300\}$ s, an IMA log-size axis (cold-5K vs. warm-100K entries), and an IMA workload-pattern axis (steady-state vs. with-updates: an `apt install` at t+120 s and a `pip install` at t+300 s). We provide more details about our methodology, including calibration and validation details in Appendix C. We note that TDX-Vordr is the only setup that protects consumer data against ASPs, and we show that the extra overhead introduced by this additional layer of protection is minimal.

7.3.1 LLM Serving Real Inference Service. We deploy `vLLM` [56] on CPU, serving Phi-3-mini [1] and Llama-3.1-8B (AWQ-INT4) [34] at calibrated arrival rates of 0.15 and 0.03 req/s respectively. The per-request latency in these LLM workloads is dominated by multi-second matrix-multiplication time, hence we want to measure how our confidential setup affects the overall throughput under Vordr's IMA policy, which guarantees that the unmodified `vLLM` binary and its dependencies execute securely.

Figure 7 shows request and output-token throughput across the three conditions (T=30 s, warm IMA baseline, steady-state, and with-updates). On Phi-3-mini, moving from native to TDX costs 9.27% on request and 8.74% on output-token throughput, respectively. Comparing TDX with TDX-Vordr, we see that Vordr adds at most 2.12% overhead in steady-state and 0.42% overhead in with-updates. Llama-3.1 shows essentially no measurable throughput loss across all runs. Llama-3.1 was run at only 0.03 RPS (Phi-3-mini was run at 0.15 RPS) since we found that the model can only process at 0.027 req/s on a c3-standard VM. Hence, the Llama-3.1 runs are much farther from saturation, and the completed throughput remains pinned by the offered load. For compute-bound LLM workloads where per-request inference time dominates over attestation work, Vordr's overhead falls within the variability of TDX itself. We found this conclusion in line with prior works [67] that host LLMs on TEEs. We further describe the details behind our findings in Section B. Next, we measure Vordr's CPU demand per unit of wall-clock time to evaluate how much compute capacity it competes for. For each TDX–Vordr run, we compute the fraction of wall-clock time spent in attestation as $A_r = 100 \times \frac{\sum_i t^{(r)}_{\text{total},i}}{\text{dur}^{(r)} \times 1000}$, where $t^{(r)}_{\text{total},i}$ is the total latency (ms) of an attestation round i in run r, and $\text{dur}^{(r)}$ is the total run duration in seconds. Figure 8 reports the corresponding wall-clock attestation time fraction, the fraction of measurement-window time the verifier spends inside Vordr rounds. Across both models and all four IMA-workloads patterns at each epoch, attestation time fraction stays in the low single-digit range and declines monotonically as T grows. Even at the most aggressive T=15 s, the CVM spends only 2% of time in attestation.

7.3.2 NGINX Static-File Serving. We deploy NGINX 1.27 [70] on the target VMs and serve two static payloads: a `1kb` response chosen to measure the syscall-bound setting where `accept/epoll/sendfile` dominate request service, and a `100kb` response chosen to measure the bandwidth-bound setting that stresses the TCP transmit path and the cross-boundary memory transfers that TDX intercepts. The load generator is `wrk2` [92], run at 80,000 req/s and 20,000 req/s on `1kb` and `100kb` respectively to achieve 80% of native saturation in both cases. We defer the other application-specific details and tail latency figures to Section D. In Figure 9, we can see that throughput



*Throughputs of Native/TDX are the theoretical upper bounds for TDX-Vordr.

Figure 7: Vordr's throughputs for Phi-3-mini and Llama-3.1. WEN performs runtime-attestation every T=30s (epoch).

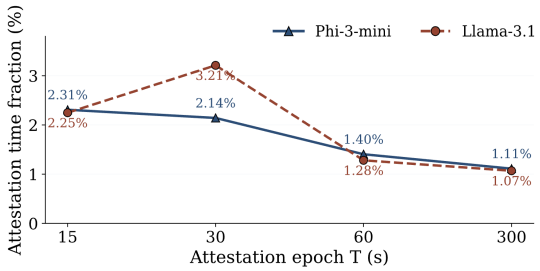
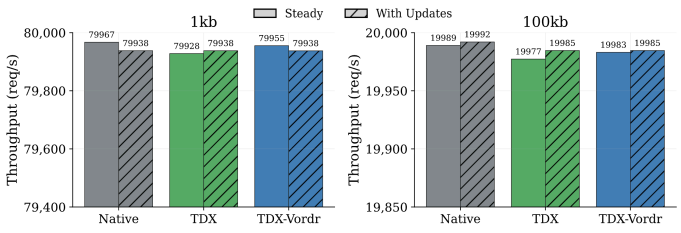


Figure 8: Attestation time fraction under varying attestation epochs T. Each point reports the mean percentage of runtime spent in Vordr attestation across cold/warm IMA states and steady-state/with-update IMA-workloads.



*Throughputs of Native/TDX are the theoretical upper bounds for TDX-Vordr.

Figure 9: Throughput of NGINX workload. WEN performs runtime-attestation every T=30s (epoch).

for TDX and TDX-Vordr remains extremely close to native. TDX alone adds a very small overhead and adding Vordr on top of TDX adds almost no additional throughput cost. Across the bars, retained throughput is roughly 99.94% to 99.99% of native. Hence, we can observe that NGINX keeps almost all of its throughput even with TDX and Vordr enabled. We note that the throughput is set by the load generator, not by the server in these experiments. These findings show that Vordr imposes negligible overhead on latency-sensitive services like NGINX, and employing hardware-rooted verifiability does not compromise practical cloud performance.

7.4 Evaluating the Scalability

To evaluate the scalability of the runtime RA for Vordr, we evaluate three different RA models: (i) direct DCAP, in which every end-user request triggers a fresh TDX quote, (ii) cached RA (mode 1 in Vordr), and (iii) audit-ready RA (mode 3 or full-audit in Vordr). Mode 2 (IMA-audit) throughput is within a small constant factor of

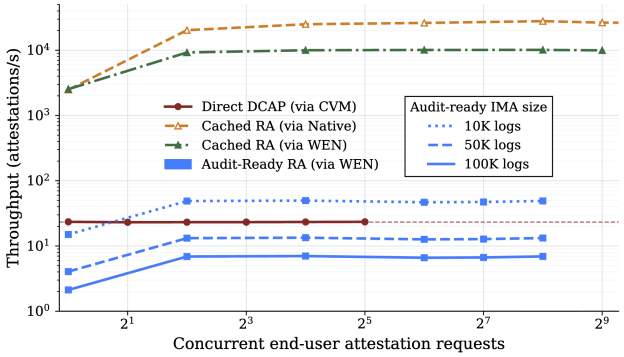


Figure 10: Throughput for direct DCAP and our model. WEN performs runtime-attestation every T=15s.

mode 3 and shares the same network-bounded scaling profile, hence we omit it in these experiments (see Section E for more details). The direct DCAP baseline was measured on a TDX CVM using `libtdx_attest` library, and the Vordr experiments were measured on a WEN (SGX enclave) with RA epochs running against a remote TDX CVM. We note that direct DCAP here carries a stronger trust assumption about runtime state than Vordr. A DCAP quote authenticates the TD measurements and RTMR values presented by the guest, but the end user must trust the CVM owner to extend the RTMRs correctly after installing or updating the intended applications, and to avoid making any malicious changes later that might not be reflected in RTMRs.

Figure 10 plots aggregate throughput as a function of the number of concurrent end-user attestation request streams. The red line shows direct per-request DCAP attestation from the CVM, the orange line shows cached RA served by the native machine (no SGX), and the green line shows cached RA served by the WEN (inside SGX). The blue family corresponds to audit-ready RA served by the WEN, where the measured audit bundle is projected to larger raw IMA logs of 10K, 50K, and 100K entries while holding the quote and command-log components fixed. The initial throughput rise for WEN-served requests reflects improved pipeline utilization as higher concurrency allows the closed-loop workload to fully saturate previously idle server resources. DCAP throughput remains flat because each DCAP request forces fresh hardware quote generation through a single-threaded enclave. Hence, concurrent requests mainly add queueing delay rather than additional throughput.

WEN-served cached RA achieves massive scalability (about 10.1K attestations/s) compared to the hardware-bottlenecked direct DCAP (23.5 req/s) as we amortize the RA. Since end users trust WEN's internal verification, cached RA acts as a high-throughput serving mode, returning a constant-size (1.4KB) authenticated metadata object. Audit-Ready RA is designed for deferred, deep verification and is strictly bottlenecked by network transport as it must transmit the raw quote (8KB), full IMA log, PCR 10 value, and WEN command log, and its payload size and transmission cost grow linearly with accumulated runtime evidence, causing a corresponding drop in throughput. Standard Intel DCAP verification fundamentally requires the PCK certificate chain to root the hardware quote's signature back to the Intel CA. In Vordr, cached RA mitigates platform linkability [17] because the WEN performs this DCAP verification internally and drops the PCK collateral, serving only a synthetic, anonymized artifact signed by the enclave. However, this unlinkability is inherently difficult to preserve in audit-Ready RA because the primary goal of full audit mode is to allow end users to independently verify the underlying hardware evidence (without blindly trusting the WEN's synthetic claims). The WEN must then export the raw DCAP quote and its associated PCK certificate chain. Consequently, audit-ready RA fundamentally trades platform anonymity for independent, deferred verifiability.

8 Related Works

There are many works in the literature that aim to provide runtime integrity of the application/workloads in TEEs. Duet [2] combines a process-based TEE controller (SGX) with a CVM, where the controller holds exclusive SSH access and simply logs all administrative commands in plain. However, Duet's runtime integrity evidence consists solely of command strings and their text output recorded by the controller, a software-only log with no cryptographic binding to the CVM's actual system state or hardware. Revelio [26] provides runtime integrity by enforcing immutability such that the VM image is reproducibly built, the root filesystem is mounted read-only with dm-verity integrity protection, and all inbound connections are blocked (lockdown). However, this blocks runtime updates to the CVM and can only be used for static applications and workloads. Other works [48] have used Linux IMA to provide runtime integrity guarantees. Upadhyay et al. [97] employ custom IMA templates for per-pod isolation in Kubernetes, enabling pod-level continuous integrity verification for 5G virtual network functions. Triglav [73] uses SGX enclaves to host virtual TPMs (vTPMs), IMA inside VMs for runtime measurement, and Dynamic Root of Trust for Measurement (DRTM) for hypervisor integrity. The protocol performs implicit attestation during secure login, binding VM integrity state (IMA measurements + vTPM quote) to the session. However, it targets pre-TDX standard VMs using SGX and DRTM as the trust anchor. Similarly, its implicit attestation during login limits verification to login events only, without supporting any continuous runtime monitoring. Galanou et al. [28] extended Triglav by introducing eBPF and Linux Security Module hooks for dynamic policy-driven attestation of post-boot runtime state, enabling verifier-specific evidence generation across different CVM lifecycle stages without requiring guest OS changes. However, their work focuses on multi-tenant CVMs, which is a problem setup different than ours. More importantly, none of these works provides unlinkability, scalability for RA, or end-user auditability. Traditional RA typically focuses on a single prover-verifier interaction [101]. Due to the increasing interconnectedness of modern systems, mainly in IoT and cloud environments, many works have focused on attestation for entire distributed applications or large groups of devices. SEDA [9] uses hop-by-hop attestation with symmetric keys, aggregating responses along a spanning tree to report the number of devices that passed attestation. SANA [7] improves upon SEDA by making it more scalable and publicly verifiable. Both of these works are collective attestation protocols optimized for IoT environments, which do not yield much benefit to the VM platforms that are much more powerful. Works such as [8] operate in a similar model as ours, using SGX enclave as a trusted guest

owner for SEV VMs. In practice, their model can be used to scale up RA to a large number of end-users; however, they do not provide any runtime integrity (boot-time only), and target pre-SNP SEV, which has been outdated. NestedSGX [101] creates SGX-like enclaves inside an SEV-SNP CVM by leveraging VMPL features, with a security monitor emulating SGX leaf functions. This complements our work, as NestedSGX provides enclave-in-CVM isolation, while Vordr provides external, trustworthy control over the CVM.

9 Conclusion

In this work, we present Vordr, a framework for enabling runtime attestation of workloads that run on CVMs. Vordr combines the strengths of process-based TEEs with VM-based TEEs and enables highly scalable, unlinkable RA via an optimized incremental attestation method that is significantly faster than existing methods. We implement our framework, show its practical performance and scalability, and achieve up to 60.8× speed for the runtime incremental attestation via our optimized method.

Acknowledgment

The authors used ChatGPT and Gemini to polish the language, and Claude to help create skeleton scripts for experiments. All texts and codes were manually reviewed and entered by the authors.

References

[1] Marah Abdin, Jyoti Aneja, Harkirat Behl, Sébastien Bubeck, Ronen Eldan, Suriya Gunasekar, Michael Harrison, Rylan Hewett, Mojan Javaheripi, Piero Kauffmann, Yin Tat Lee, Mario Lucic, Adam Oesterling, Aleksandar Petrov, Michael Picheny, Guilherme Perin Ribeiro, Ram Rungta, Ezioma Seung, Yi Wang, et al. 2024. Phi-3 Technical Report: A Highly Capable Language Model Locally on Your Phone. arXiv preprint arXiv:2404.14219. https://arxiv.org/abs/2404.14219

[2] Istemi Ekin Akkus and Ivica Rimac. 2024. Duet: Combining a trustworthy controller with a confidential computing environment. In *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. IEEE, 436–442.

[3] Erdem Aktas, Cfir Cohen, Josh Eads, James Forshaw, and Felix Wilhelm. 2023. Intel trust domain extensions (TDX) security review. *Google security review* (2023).

[4] Mansour Alharthi, Fan Sang, Dmitrii Kuvaiskii, Mona Vij, and Taesoo Kim. 2025. Rakis: Secure fast i/o primitives across trust boundaries on intel sgx. In *Proceedings of the Twentieth European Conference on Computer Systems*. 1177–1193.

[5] Amazon Web Services. 2026. *Attest an Amazon EC2 Instance with AMD SEV-SNP*. Amazon Web Services. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/snp-attestation.html Accessed: 2026-04-27.

[6] Amazon Web Services (EC2). [n. d.]. Tutorial: Complete the Configuration Required to Connect to Your Instance Using EC2 Instance Connect. https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ec2-instance-connect-tutorial.html. Accessed: 2026-04-14.

[7] Moreno Ambrosin, Mauro Conti, Ahmad Ibrahim, Gregory Neven, Ahmad-Reza Sadeghi, and Matthias Schunter. 2016. SANA: secure and scalable aggregate network attestation. In *Proceedings of the 2016 ACM SIGSAC conference on computer and communications security*. 731–742.

[8] Pedro Antonino, Ante Derek, and Wojciech Aleksander Wołoszyn. 2023. Flexible remote attestation of pre-SNP SEV VMs using SGX enclaves. *IEEE access* 11 (2023), 90839–90856.

[9] Nadarajah Asokan, Ferdinand Brasser, Ahmad Ibrahim, Ahmad-Reza Sadeghi, Matthias Schunter, Gene Tsudik, and Christian Wachsmann. 2015. Seda: Scalable embedded device attestation. In *Proceedings of the 22nd ACM SIGSAC conference on computer and communications security*. 964–975.

[10] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2014. Shielding Applications from an Untrusted Cloud with Haven. In *11th USENIX Symposium on Operating Systems Design and Implementation (OSDI 14)*. USENIX Association, 267–283.

[11] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. 2012. Foundations of garbled circuits. In *Proceedings of the 2012 ACM conference on Computer and communications security*. 784–796.

[12] Henk Birkholz, David Thaler, Michael Richardson, Ned Smith, and Wei Pan. 2023. *Remote ATtestation procedureS (RATS) Architecture*. RFC 9334. Internet Engineering Task Force. https://www.rfc-editor.org/rfc/rfc9334

[13] Ernie Brickell and Jiangtao Li. 2009. Enhanced Privacy ID from Bilinear Pairing. Cryptology ePrint Archive, Paper 2009/095. https://eprint.iacr.org/2009/095 Accessed: 2026-04-25.

[14] Calvin Brierley, Jamie Pont, Budi Arief, David J Barnes, and Julio Hernandez-Castro. 2020. Persistence in linux-based iot malware. In *Nordic Conference on Secure IT Systems*. Springer, 3–19.

[15] Mike Burrows. 2006. The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th symposium on Operating systems design and implementation*. 335–350.

[16] Bruno Casella. 2025. A performance analysis of vm-based trusted execution environments for confidential federated learning. In *2025 33rd Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*. IEEE, 204–208.

[17] Guoxing Chen, Yinqian Zhang, and Ten-Hwang Lai. 2019. OPERA: Open Remote Attestation for Intel's Secure Enclaves. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security* (London, United Kingdom) *(CCS '19)*. Association for Computing Machinery, New York, NY, USA, 2317–2331. doi:10.1145/3319535.3354220

[18] Pau-Chen Cheng, Wojciech Ozga, Enriquillo Valdez, Salman Ahmed, Zhongshu Gu, Hani Jamjoom, Hubertus Franke, and James Bottomley. 2024. Intel TDX Demystified: A Top-Down Approach. *Comput. Surveys* 56, 9 (2024), 1–40. doi:10.1145/3652597

[19] Keylime Contributors. 2023. IMA and Keylime in the presence of package updates (Issue #1515). https://github.com/keylime/keylime/issues/1515.

[20] Luigi Coppolino, Salvatore D'Antonio, Giovanni Mazzeo, and Luigi Romano. 2025. An experimental evaluation of TEE technology: Benchmarking transparent approaches based on SGX, SEV, and TDX. *Computers & Security* 154 (2025), 104457.

[21] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. Cryptology ePrint Archive, Paper 2016/086. https://eprint.iacr.org/2016/086

[22] Giulio De Pasquale, Ilya Grishchenko, Riccardo Iesari, Gabriel Pizarro, Lorenzo Cavallaro, Christopher Kruegel, and Giovanni Vigna. 2024. {ChainReactor}: Automated Privilege Escalation Chain Discovery via {AI} Planning. In *33rd USENIX Security Symposium (USENIX Security 24)*. 5913–5929.

[23] Giuseppe DeCandia, Deniz Hastorun, Madan Jampani, Gunavardhan Kakulapati, Avinash Lakshman, Alex Pilchin, Swaminathan Sivasubramanian, Peter Vosshall, and Werner Vogels. 2007. Dynamo: Amazon's highly available key-value store. *ACM SIGOPS operating systems review* 41, 6 (2007), 205–220.

[24] Keylime Developers. [n. d.]. keylime/keylime. https://github.com/keylime/keylime. GitHub repository: A CNCF Project to Bootstrap & Maintain Trust on the Edge / Cloud and IoT. Accessed: 2026-03-20.

[25] Edgeless Systems. 2026. MarbleRun. https://github.com/edgelesssys/marblerun. Accessed: 2026-04-29.

[26] Anna Galanou, Khushboo Bindlish, Luca Preibsch, Yvonne-Anne Pignolet, Christof Fetzer, and Rüdiger Kapitza. 2023. Trustworthy confidential virtual machines for the masses. In *Proceedings of the 24th International Middleware Conference*. 316–328.

[27] Anna Galanou, Khushboo Bindlish, Luca Preibsch, Yvonne-Anne Pignolet, Christof Fetzer, and Rüdiger Kapitza. 2024. Trustworthy confidential virtual machines for the masses. arXiv:2402.15277 [cs.CR] https://arxiv.org/abs/2402.15277

[28] Anna Galanou, Florian Lubitz, Hajeong Jeon, Christof Fetzer, and Rüdiger Kapitza. 2025. Full Trust Alchemist: Reforging Attestation for Cloud-based Confidential Workloads. In *Proceedings of the 26th International Middleware Conference*. 369–382.

[29] Craig Gentry. 2009. Fully homomorphic encryption using ideal lattices. In *Proceedings of the forty-first annual ACM symposium on Theory of computing*. 169–178.

[30] Ken Goldman. 2024. IMA Event Log — IMA 1.0 Documentation. https://ima-doc.readthedocs.io/en/latest/event-log-format.html. Accessed March 2026.

[31] Google Cloud. [n. d.]. Guest Agent Functionality. https://docs.cloud.google.com/compute/docs/images/guest-agent-functions. Accessed: 2026-04-14.

[32] Google Cloud. 2024. Emerging startups working with Google Cloud. https://cloud.google.com/blog/topics/startups/emerging-startups-working-with-google-cloud. Accessed: 2026-04-14.

[33] Google Cloud. 2026. Restrict SSH keys from VMs. https://docs.cloud.google.com/compute/docs/connect/restrict-ssh-keys. Google Cloud Documentation. Last updated 2026-04-08 UTC. Accessed 2026-04-09.

[34] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Anjali Kadian, Amal Al-Dahle, Aiesha Letman, Anish Mathur, Alan Schelten, Amy Vaughan, et al. 2024. The Llama 3 Herd of Models. *arXiv preprint arXiv:2407.21783* (2024). https://arxiv.org/abs/2407.21783

[35] Franz Gregor, Wojciech Ozga, Sébastien Vaucher, Rafael Pires, Do Le Quoc, Sergei Arnautov, André Martin, Valerio Schiavoni, Pascal Felber, and Christof Fetzer. 2020. Trust Management as a Service: Enabling Trusted Execution in the Face of Byzantine Stakeholders. In *2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. 502–514. doi:10.1109/DSN48063.2020.00063

[36] Ehsan Hesamifard, Hassan Takabi, Mehdi Ghasemi, and Rebecca N. Wright. 2018. Privacy-preserving Machine Learning as a Service. *Proceedings on Privacy Enhancing Technologies* 2018, 3 (2018), 123–142. doi:10.1515/popets-2018-0024

[37] IMA. [n. d.]. Policy Syntax. *IMA 1.0 documentation*. https://ima-doc.readthedocs.io/en/latest/policy-syntax.html Accessed: 2026-03-20.

[38] IMA Project. [n. d.]. IMA Event Log. https://ima-doc.readthedocs.io/en/latest/event-log-format.html. IMA 1.0 documentation. Accessed: 2026-04-25.

[39] Intel Corporation. 2022. Intel Trust Domain Extensions (Intel TDX) White Paper. https://cdrdv2-public.intel.com/690419/TDX-Whitepaper-February2022.pdf. Revision February 2022.

[40] Intel Corporation. 2023. Intel SGX Attestation Service Utilizing Intel EPID End-of-Life Announcement. https://community.intel.com/t5/Intel-Software-Guard-Extensions/IAS-End-of-Life-Announcement/td-p/1545831. Accessed: 2026-04-25.

[41] Intel Corporation. 2024. *Design Guide for Intel SGX and Intel TDX Provisioning Certificate Caching Service*. Accessed: 2026-04-25.

[42] Intel Corporation. 2024. Integrity Measurement Architecture (IMA) Logs — Intel Trust Authority Documentation. https://docs.trustauthority.intel.com/main/articles/articles/ita/tpm-ima-logs.html. Accessed March 2026.

[43] Intel Corporation. 2024. *Intel SGX Data Center Attestation Primitives: ECDSA Orientation*. Accessed: 2026-04-25.

[44] Intel Corporation. 2025. *Intel Software Guard Extensions (Intel SGX) Data Center Attestation Primitives: ECDSA Quote Library API*. Intel Corporation. https://download.01.org/intel-sgx/latest/dcap-latest/linux/docs/Intel_SGX_ECDSA_QuoteLibReference_DCAP_API.pdf Rev Production.

[45] Intel Corporation. 2025. *Intel Trust Domain Extensions Data Center Attestation Primitives (Intel TDX DCAP): Quote Generation Library and Quote Verification Library*. Intel Corporation. https://download.01.org/intel-sgx/latest/dcap-latest/linux/docs/Intel_TDX_DCAP_Quoting_Library_API.pdf Rev. 0.9.

[46] Intel Corporation. 2025. *Intel Trust Domain Extensions (Intel TDX) Module: Base Architecture Specification*. Intel Corporation. https://cdrdv2-public.intel.com/867568/intel-tdx-module-base-spec-348549007.pdf Document number 348549-007US.

[47] Intel Trust Authority. [n. d.]. Integrity Measurement Architecture (IMA) logs. https://docs.trustauthority.intel.com/main/articles/articles/ita/tpm-ima-logs.html. Accessed: 2026-03-20.

[48] Trent Jaeger, Reiner Sailer, and Umesh Shankar. 2006. PRIMA: policy-reduced integrity measurement architecture. In *Proceedings of the eleventh ACM symposium on Access control models and technologies*. 19–28.

[49] Keylime Contributors. [n. d.]. agentstates.py at master — keylime/keylime. https://github.com/keylime/keylime/blob/master/keylime/agentstates.py. Accessed: 2026-04-15.

[50] Keylime Maintainers. [n. d.]. cloud_verifier_common.py at master — keylime/keylime. https://github.com/keylime/keylime/blob/master/keylime/cloud_verifier_common.py. Accessed: 2026-04-15.

[51] Keylime Project. [n. d.]. Agent — Keylime Documentation 7.13.0. https://keylime.readthedocs.io/en/v7.13.0/rest_apis/2_1/agent.html. Accessed: 2026-04-15.

[52] Keylime Project. 2024. Runtime Integrity Monitoring — Keylime Documentation. https://github.com/keylime/keylime/blob/master/docs/user_guide/runtime_ima.rst. Accessed: 2026-04-10. States:.

[53] Keylime Project. 2024. Use Measured Boot — Keylime Documentation. https://keylime.readthedocs.io/en/latest/user_guide/use_measured_boot.html. Accessed: 2026-04-10..

[54] Samuel T King and Peter M Chen. 2006. SubVirt: Implementing malware with virtual machines. In *2006 IEEE Symposium on Security and Privacy (S&P'06)*. IEEE, 14–pp.

[55] Christopher Kruegel, William Robertson, and Giovanni Vigna. 2004. Detecting kernel-level rootkits through binary analysis. In *20th Annual Computer Security Applications Conference*. IEEE, 91–100.

[56] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient Memory Management for Large Language Model Serving with PagedAttention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*.

[57] Caihua Li, In Gim, and Lin Zhong. 2025. Confidential Prompting: Privacy-preserving LLM Inference on Cloud. arXiv:2409.19134 [cs.CR] doi:10.48550/arXiv.2409.19134

[58] Mario Lins, René Mayrhofer, Michael Roland, Daniel Hofer, and Martin Schwaighofer. 2024. On the critical path to implant backdoors and the effectiveness of potential mitigation techniques: Early learnings from XZ. *arXiv preprint arXiv:2404.08987* (2024).

[59] Linux IMA Project. 2024. IMA Configuration — IMA Documentation. https://ima-doc.readthedocs.io/en/latest/ima-configuration.html. Accessed: 2026-04-10..

[60] Linux IMA Project. 2024. IMA Event Log Format. https://ima-doc.readthedocs.io/en/latest/event-log-format.html Accessed: 2026-03-21.

[61] Microsoft. 2023. Azure Attestation client library for Python. https://learn.microsoft.com/en-us/python/api/overview/azure/security-attestation-readme. Accessed: 2026-03-02.

[62] Microsoft Azure. [n. d.]. VMAccess Extension for Linux - Virtual Machines. https://learn.microsoft.com/en-us/azure/virtual-machines/extensions/vmaccess-linux. Accessed: 2026-04-14.

[63] Shubham Mishra, João Gonçalves, Chawinphat Tankuranand, Neil Giridharan, Natacha Crooks, Heidi Howard, and Chris Jensen. 2026. Proteus: Append-Only Ledgers for (Mostly) Trusted Execution Environments. *arXiv preprint arXiv:2602.05346* (2026).

[64] Masanori Misono, Dimitrios Stavrakakis, Nuno Santos, and Pramod Bhatotia. 2024. Confidential vms explained: An empirical analysis of amd sev-snp and intel tdx. *Proceedings of the ACM on Measurement and Analysis of Computing Systems* 8, 3 (2024), 1–42.

[65] MITRE ATT&CK. 2025. Boot or Logon Initialization Scripts: RC Scripts, Sub-technique T1037.004. https://attack.mitre.org/techniques/T1037/004/. MITRE ATT&CK Enterprise. Last modified October 24, 2025. Accessed April 13, 2026.

[66] Michael Mitzenmacher. 2001. The Power of Two Choices in Randomized Load Balancing. *IEEE Transactions on Parallel and Distributed Systems* 12, 10 (2001), 1094–1104.

[67] Avanika Narayan, Dan Biderman, and Christopher Re. 2025. Proof-of-Concept for Private Local-to-Cloud LLM Chat via Trusted Execution Environments. In *ES-FoMo III: 3rd Workshop on Efficient Systems for Foundation Models*.

[68] Vikram Narayanan, Claudio Carvalho, Angelo Ruocco, Gheorghe Almasi, James Bottomley, Mengmei Ye, Tobin Feldman-Fitzthum, Daniele Buono, Hubertus Franke, and Anton Burtsev. 2023. Remote attestation of confidential VMs using ephemeral vTPMs. In *Proceedings of the 39th Annual Computer Security Applications Conference*. 732–743.

[69] Rob Nertney. 2026. *Unified Deployment Guide for Confidential Computing*. NVIDIA Corporation. https://docs.nvidia.com/cc-deployment-guide-tdx.pdf Accessed: April 28, 2026.

[70] NGINX, Inc. 2024. nginx: Changes. https://nginx.org/en/CHANGES. Official nginx change log, including the nginx 1.27.x release series. Accessed: 2026-04-27.

[71] Diego Ongaro and John Ousterhout. 2014. In Search of an Understandable Consensus Algorithm. In *Proceedings of the 2014 USENIX Annual Technical Conference (ATC)*. 305–319.

[72] openSUSE Wiki. 2024. SDB:Ima_evm — IMA and EVM on openSUSE. https://en.opensuse.org/SDB:Ima_evm. Accessed: 2026-04-10..

[73] Wojciech Ozga, Christof Fetzer, et al. 2021. Triglav: Remote attestation of the virtual machine's runtime integrity in public clouds. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*. IEEE, 1–12.

[74] Jeremy Powell. 2022. AMD SEV-SNP Attestation: Establishing Trust in Guests. Linux Security Summit Europe. https://www.amd.com/content/dam/amd/en/documents/developer/lss-snp-attestation.pdf

[75] Michael Preisach. 2022. *System Integrity and Attestation for Biometric Sensors*. Ph. D. Dissertation. thesis. linz,(January 2022).

[76] Red Hat. 2025. How to Use the Linux Kernel's Integrity Measurement Architecture. https://www.redhat.com/en/blog/how-use-linux-kernels-integrity-measurement-architecture. Accessed March 2026.

[77] Red Hat, Inc. 2024. Enhancing Security with the Kernel Integrity Subsystem. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/7/html/kernel_administration_guide/enhancing_security_with_the_kernel_integrity_subsystem. RHEL 8 Kernel Administration Guide..

[78] Red Hat, Inc. 2024. Enhancing Security with the Kernel Integrity Subsystem. https://docs.redhat.com/en/documentation/red_hat_enterprise_linux/9/html/managing_monitoring_and_updating_the_kernel/enhancing-security-with-the-kernel-integrity-subsystem_managing-monitoring-and-updating-the-kernel. RHEL 9 Kernel Administration Guide..

[79] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In *Proceedings of the 16th ACM conference on Computer and communications security*. 199–212.

[80] Margie Ruffin, Chenkai Wang, Gheorghe Almasi, Abdulhamid Adebayo, Hubertus Franke, and Gang Wang. 2025. Towards Continuous Integrity Attestation and Its Challenges in Practice: A Case Study of Keylime. In *Proceedings of the 55th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN '25)*. IEEE, 256–265. doi:10.1109/DSN61648.2025.11068805

[81] 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)*. USENIX Association, San Diego, CA. https://www.usenix.org/conference/13th-usenix-security-symposium/design-and-implementation-tcg-based-integrity-measurement

[82] Nabil Schear, Patrick T. Cable II, Frederic P. Gurry, Andrew F. Westerinen, Matthew J. Moynihan, Michael P. Wenstrom, and David C. Kravitz. 2016. Bootstrapping and Maintaining Trust in the Cloud. In *Proceedings of the 2016 USENIX Annual Technical Conference (USENIX ATC '16)*. USENIX Association, Denver, CO, USA, 65–77.

[83] AMD Sev-Snp. 2020. Strengthening VM isolation with integrity protection and more. *White Paper, January* 53, 2020 (2020), 1450–1465.

[84] Alex Shamis, Peter Pietzuch, Burcu Canakci, Miguel Castro, Cédric Fournet, et al. 2023. Confidential Consortium Framework: Secure Multiparty Applications with Confidentiality, Integrity, and High Availability. *Proceedings of the VLDB Endowment* 17, 2 (2023), 225–240.

[85] Kripa Shanker, Arun Joseph, and Vinod Ganapathy. 2020. An evaluation of methods to port legacy code to sgx enclaves. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1077–1088.

[86] Max Sherborne, Heidi Howard, et al. 2024. LSKV: A Confidential Distributed Datastore to Protect Critical Data in the Cloud. In *arXiv preprint arXiv:2407.12623*.

[87] Signal. 2017. Technology preview: Private contact discovery for Signal. https://signal.org/blog/private-contact-discovery/. Accessed: 2026-04-15.

[88] Rohit Sinha, Sivanarayana Gaddam, and Ranjit Kumaresan. 2019. Luciditee: A tee-blockchain system for policy-compliant multiparty computation with fairness. *Cryptology ePrint Archive* (2019).

[89] Juhyung Son, Sungmin Koo, Jongmoo Choi, Seong-je Choi, Seungjae Baek, Gwangil Jeon, Jun-Hyeok Park, and Hyoungchun Kim. 2017. Quantitative analysis of measurement overhead for integrity verification. In *Proceedings of the Symposium on Applied Computing*. 1528–1533.

[90] Jakob Stühn, Jan-Niclas Hilgert, and Martin Lambertz. 2024. The hidden threat: Analysis of linux rootkit techniques and limitations of current detection tools. *Digital Threats: Research and Practice* 5, 3 (2024), 1–24.

[91] Jakob Svenningsson, Nicolae Paladi, and Arash Vahidi. 2021. Faster enclave transitions for IO-intensive network applications. In *Proceedings of the ACM SIGCOMM 2021 Workshop on Secure Programmable Network INfrastructure*. 1–8.

[92] Gil Tene. 2015. wrk2: A constant throughput, correct latency recording variant of wrk. https://github.com/giltene/wrk2. Accessed: 2026-02-21.

[93] The Linux Kernel Documentation. 2024. The Kernel's Command-Line Parameters. https://www.kernel.org/doc/html/v4.14/admin-guide/kernel-parameters.html. Documents `ima_policy=`.

[94] The Linux Kernel Documentation. 2024. The seq_file Interface. https://docs.kernel.org/filesystems/seq_file.html Accessed: 2026-03-24.

[95] The Linux Kernel Project. 2024. Integrity Measurement Architecture (IMA) Source Code. https://git.kernel.org/pub/scm/linux/kernel/git/torvalds/linux.git/tree/security/integrity/ima Kernel v6.17, Accessed: 2026-03-24.

[96] Donghai Tian, Rui Ma, Xiaoqi Jia, and Changzhen Hu. 2019. A kernel rootkit detection approach based on virtualization and machine learning. *IEEE Access* 7 (2019), 91657–91666.

[97] Rajendra Upadhyay, Rajendra Paudyal, Lisa Donnan, Duminda Wijesekera, et al. 2025. Tpm-based continuous remote attestation and integrity verification for 5g vnfs on kubernetes. In *2025 IEEE 7th International Conference on Trust, Privacy and Security in Intelligent Systems, and Applications (TPS-ISA)*. IEEE, 435–444.

[98] Venkatanathan Varadarajan, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. 2015. A placement vulnerability study in {Multi-Tenant} public clouds. In *24th USENIX Security Symposium (USENIX Security 15)*. 913–928.

[99] Jianqiang Wang, Pouya Mahmoody, Ferdinand Brasser, Patrick Jauernig, Ahmad-Reza Sadeghi, Donghui Yu, Dahan Pan, and Yuanyuan Zhang. 2022. VirTEE: A full backward-compatible TEE with native live migration and secure I/O. In *Proceedings of the 59th ACM/IEEE Design Automation Conference*. 241–246.

[100] Weili Wang, Sen Deng, Jianyu Niu, Michael K Reiter, and Yinqian Zhang. 2022. Engraft: Enclave-guarded raft on byzantine faulty nodes. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*. 2841–2855.

[101] Wenhao Wang, Linke Song, Benshan Mei, Shuang Liu, Shijun Zhao, Shoumeng Yan, XiaoFeng Wang, Dan Meng, and Rui Hou. 2024. The road to trust: Building enclaves within confidential VMs. *arXiv preprint arXiv:2402.11438* (2024).

[102] Tara Whalen, Thibault Meunier, Mrudula Kodali, Alex Davidson, Marwan Fayed, Armando Faz-Hernández, Watson Ladd, Deepak Maram, Nick Sullivan, Benedikt Christoph Wolters, et al. 2022. Let the right one in: attestation as a usable {CAPTCHA} alternative. In *Eighteenth Symposium on Usable Privacy and Security (SOUPS 2022)*. 599–612.

[103] Luca Wilke, Jan Wichelmann, Florian Sieck, and Thomas Eisenbarth. 2024. SNPGuard: Remote Attestation of SEV-SNP VMs Using Open Source Tools. In *Proceedings of the 7th Workshop on System Software for Trusted Execution (SysTEX)*. Co-located with EuroS&P 2024.

[104] Haidong Xia, Ken Lu, Ruoyu Ying, Xiaocheng Dong, and Yanhui Zhao. [n. d.]. Runtime Integrity Measurement and Attestation in a Trust Domain. https://www.intel.com/content/www/us/en/developer/articles/community/runtime-integrity-measure-and-attest-trust-domain.html. Accessed: 2026-03-20.

[105] Ziqian Zeng, Jianwei Wang, Junyao Yang, Zhengdong Lu, Haoran Li, Huiping Zhuang, and Cen Chen. 2025. PrivacyRestore: Privacy-Preserving Inference in Large Language Models via Privacy Removal and Restoration. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 10821–10855. doi:10.18653/v1/2025.acl-long.532

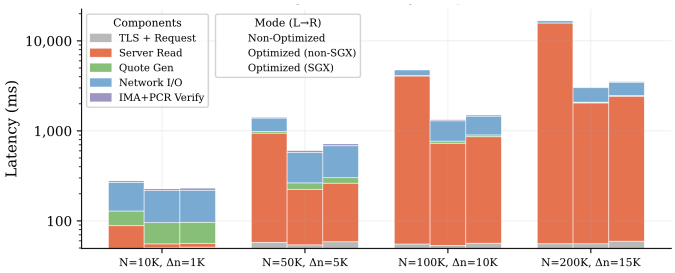


Figure 11: Overall latency break down by component for incremental attestation methods

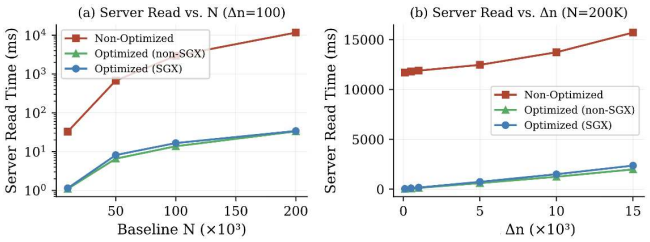


Figure 12: Server Read time across IMA log size (N) and change in IMA log (Δn)

A Full Latency Breakdown of Incremental Attestation

In the non-optimized mode, the CVM agent reopens the IMA pseudo-file descriptor on every attestation epoch. Seeking to offset N requires N/B successive `read()` calls, each re-walking an increasing prefix of the list, yielding $O(N^2/B)$ cumulative extraction cost (see Equation 1 in Section 5.2). Next, for the optimized method, the CVM agent keeps the IMA file descriptor persistently open across attestation epochs, maintaining the kernel's internal `seq_file` iterator position. On each subsequent epoch, the agent reads only the Δn new entries appended since the last read, eliminating the $O(N^2/B)$ seek-to-offset phase entirely. For the optimized method, we ran the WEN code both with and without SGX to isolate the overhead introduced by SGX.

For each attestation round, we record the timing breakdown for TLS handshake, CVM-side IMA log read time, TDX DCAP quote generation, network response transfer, WEN-side DCAP quote verification, WEN-side IMA + PCR verification, and total end-to-end latency. Full latency breakdown and its details are provided in Figure 11. In Figure 12, we isolate the CVM-side IMA read time, which is the term targeted by the optimization. Part (a) fixes Δn=100 and scales N: Non-Optimized read time grows linearly from 32.7 ms to 11.7 s as N grows from 10K to 200K, while Optimized (SGX) read time stays between 1.1 ms and 34.1 ms, a 29× to 343× improvement. Part (b) fixes N=200K and varies Δn: Non-Optimized is flat (12–17 s, dominated by the full traversal), whereas Optimized (SGX) grows linearly with Δn.

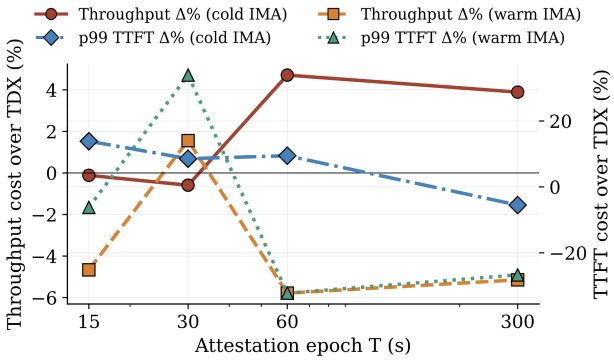


Figure 13: Overhead of TDX-Vordr compared to TDX-only as the attestation epoch T varies from 15 s to 300 s for Phi-3-mini. We report throughput overhead (%) and $p99$ TTFT overhead (%) for both cold and warm IMA baselines under steady-state load.

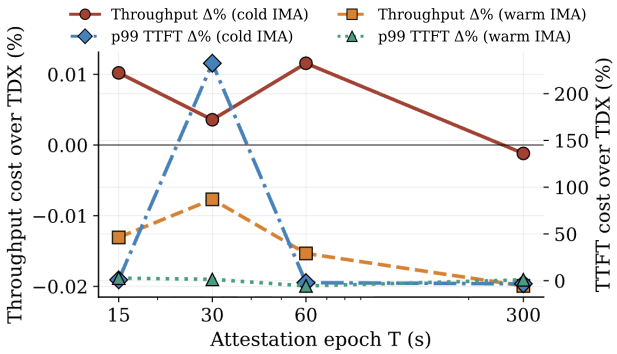


Figure 14: Overhead of TDX-Vordr compared to TDX-only as the attestation epoch T varies from 15 s to 300 s for LLama-3.1. We report throughput overhead (%) and $p99$ TTFT overhead (%) for both cold and warm IMA baselines under steady-state load.

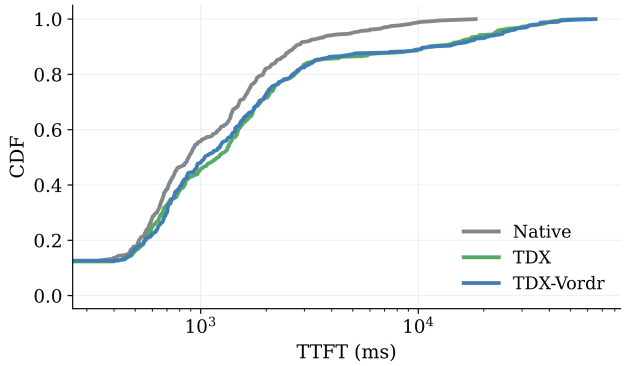


Figure 15: Empirical cumulative distribution function (CDF) of TTFT for Phi-3-mini. TDX-Vordr is pinned at attestation epoch T=30 s.

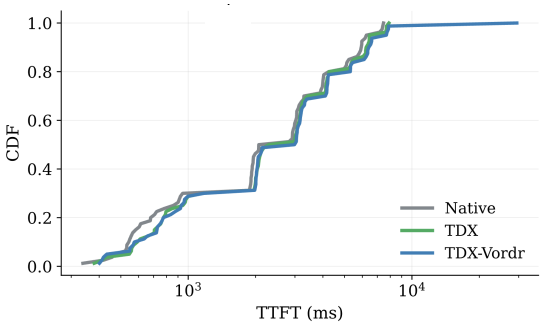


Figure 16: Empirical cumulative distribution function (CDF) of TTFT for Llama-3.1. TDX-Vordr is pinned at attestation epoch T=30 s.

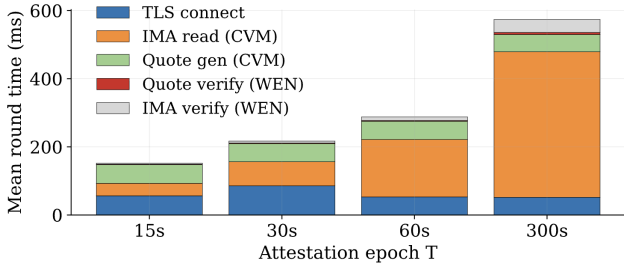


Figure 17: Mean attestation round time across different remote attestation epoch values T for Phi-3-mini.

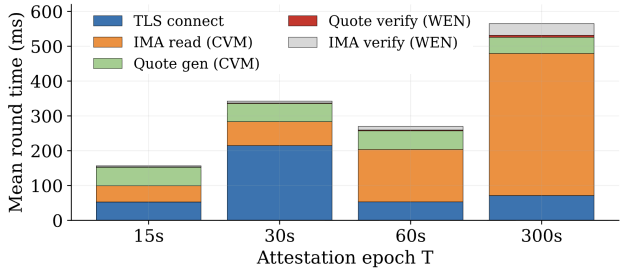


Figure 18: Mean attestation round time across different remote attestation epoch values T for Llama-3.1.

B LLM Workload Full-Evaluation

B.1 Environmetal Setup

The workload uses ShareGPT V3 prompts with open-loop Poisson arrivals at 0.15 req/s, corresponding to roughly 80% of measured saturation, after a 60 s warmup and over a 600 s measurement window. For Vordr, we sweep attestation epoch T ∈ {15,30,60,300} s. For the IMA baseline size, term them based on size:{cold, warm}, and workload pattern: {steady, with-updates}. In the cold setting, the CVM starts with a minimal IMA measurement log, representing a freshly launched instance with little prior execution history. In the warm setting, the IMA log is pre-populated through a burn-in phase (e.g., package installation and prior execution), resulting in a large baseline log. Warm-IMA runs are explicitly pre-burned to about 100,000 entries before the benchmark starts, but cold runs

do not do that pre-burn. During the steady mode, the Δn remains constant with no new IMA log entries. In the with-updates mode, Δn is increased via various commands that get captured by the IMA log. In particular, two real update actions: apt-get install -y sl and pip3 install –no-deps –target /tmp/pipcache wheel are run. These commands cause additional measured file accesses and executable loads on the CVM. This models a long-running production VM, where attestation operates over incremental log growth (Δn) rather than an initially empty log.

The load generator was hosted on a workstation in our local network rather than co-located with the target VM in the cloud, contributing approximately 21 ms of fixed WAN RTT latency to every measured request. Because vLLM CPU inference produces multi-second time-to-first-token (mean p50 = 2.5 s, p99 = 7.1 s), this network floor accounts for less than 1% of measured latency in all conditions and does not affect cross-condition comparisons.

B.2 Evaluation Details

We evaluate Vordr under long-running LLM inference workloads using `microsoft/Phi-3-mini-4k-instruct` and `hugging-quants/Meta-Llama-3.1-8B-Instruct-AWQ-INT4` served by vLLM on CPU. All runs were generated from a single prepared boot-disk snapshot and executed on fresh `c3-standard-8` instances with 32 GiB RAM in `us-central1-a`. We compare three conditions: native, TDX-only, and TDX-Vordr.

For the Phi-3-mini model, we observe that TDX introduces the dominant performance overhead (about 6.2%) relative to native execution, while TDX+Vordr closely tracks TDX. Moving from native to TDX reflects the baseline cost of confidential execution, whereas adding Vordr on top of TDX changes throughput by only +0.2%. Thus, we observe no systematic application-visible throughput overhead beyond TDX itself. A similar trend is observed for latency. The TTFT (time to first token; a critical LLM performance metric measuring the latency between submitting a prompt and receiving the very first character of the response) $p50$ increases modestly from native to TDX due to the underlying confidential execution overhead, but TDX+Vordr remains closely aligned with TDX. For higher quantiles ($p95$, $p99$), the latency values get larger and noisier, reflecting queueing effects under near-saturation CPU load. However, the TDX-Vordr tail quantiles overlap with those of TDX, indicating that Vordr does not introduce any additional tail-latency amplification beyond TDX. Overall, both throughput and latency results show that Vordr adds negligible overhead on top of TDX.

We next vary the attestation epoch over a 20× range to test whether tighter monitoring intervals perturb the workload. Figures 13 and 14 show the relative overhead of TDX-Vordr compared to TDX-only as the attestation epoch varies. Across a 20× range of T, throughput overhead remains within a small band around zero, demonstrating that increasing attestation frequency does not introduce additional application-visible cost for Phi-3-mini. While $p99$ TTFT is noisier due to workload-induced queueing effects, no consistent upward trend is observed. For Llama-3.1, the throughput loss is negligible similarly. In both models, the high TTFT cost is seen at 30s. We found it to be mainly a tail-effect artifact and driven by isolated p99 tail outliers in the matched runs, as the surrounding epochs do not show the same monotonic pattern. This shows that Vordr remains off the critical path even under aggressive polling intervals, indicating that attestation frequency does not impact steady-state performance.

Figures 15 and 16 show the full TTFT distribution across all requests for Phi-3-mini and Llama-3.1 respectively. We can see the CDF of per-request TTFT across the three conditions. For Phi-3-mini, native is left-shifted, indicating lower latency overall, while TDX shifts the distribution to higher latencies compared to native. For Llama-3.1, we do not observe any significant shift. The TDX-Vordr curve remains closely aligned with TDX across nearly the entire range for both models. This indicates that Vordr does not introduce any additional distributional shift or tail amplification beyond the baseline cost of TDX.

Next, we measure the cost of an attestation round (Figures 17 and 18) to identify the dominant factors of runtime-monitoring overhead. We observe that round time is dominated by fixed-cost components, particularly TLS connection establishment and TDX quote generation, while incremental IMA processing remains lightweight in steady state. This shows that Vordr's overhead does not scale with workload activity or log growth, explaining why frequent attestation does not translate into application-visible performance degradation. Even in the worst case (large IMA log at $T = 300$ s), the increased cost is amortized over longer intervals and does not affect steady-state throughput. Hence, the round cost is dominated by fixed-cost cryptographic work, and the monitoring thread is active for only a small fraction of wall-clock time on the CVM. Empirically, we found that Vordr attestation agent occupies 2.31% of the run at T=15 s, 2.14% at T=30 s, 1.40% at T=60 s, and 1.11% at T=300 s. In absolute terms, that is only about 13.9 s, 12.8 s, 8.4 s, and 6.7 s, respectively, over a 600 s run. In sum, these results show that Vordr can provide continuous runtime-integrity monitoring for a realistic LLM service with little observable overhead beyond the baseline cost of TDX.

The default Vordr policy attests vLLM's executed-code surface (the Python interpreter, vLLM's compiled extensions, CUDA/CPU runtime libraries, and the kernel module set) but does not measure the model weight files (`*.safetensors`, AWQ checkpoint files), tokenizer files, or chat templates, since these are accessed via non-executable `read/mmap` paths. For deployments where end users must verify the served model identity, operators extend the policy with workload-specific `FILE_CHECK` rules pinned to the model artifact paths, optionally combined with IMA-appraisal to enforce digital signatures at access time. Alternatively, model artifacts can be placed on a `dm-verity` protected read-only partition whose root hash is bound to the CVM's launch measurement chain, giving end users hardware-rooted assurance that the served model matches a published reference. Both extensions are compatible with Vordr's incremental attestation protocol, and the additional measurements are appended to the same PCR 10 log and verified by the WEN through the same Δn-replay path.

C Detailed Experimental Methodology

All target VMs are `c3-standard-8` instances on Google Cloud (8 vCPU, 32 GB RAM, Ubuntu 22.04). *TDX* cells use Intel TDX

Table 2: Cross-model throughput values and slowdowns for the representative slice (warm IMA baseline; T=30 s for TDX-Vordr). Slowdowns are computed relative to the Native configuration within the same model and workload.

Throughput Numbers for models on different run environments

Model	Workload	Condition	Request (req/s)	Output (tok/s)
Phi-3-mini	steady-state	Native	0.141770	36.313255
Phi-3-mini	steady-state	TDX	0.128633	33.137813
Phi-3-mini	steady-state	TDX-Vordr	0.126631	32.436624
Phi-3-mini	with updates	Native	0.144162	36.631731
Phi-3-mini	with updates	TDX	0.134424	34.034946
Phi-3-mini	with updates	TDX-Vordr	0.133856	34.006100
Llama-3.1	steady-state	Native	0.026918	3.620524
Llama-3.1	steady-state	TDX	0.026904	3.618600
Llama-3.1	steady-state	TDX-Vordr	0.026906	3.618879
Llama-3.1	with updates	Native	0.026915	3.620111
Llama-3.1	with updates	TDX	0.026912	3.619632
Llama-3.1	with updates	TDX-Vordr	0.026907	3.619048

Throughput Slowdown relative to Native

Model	Workload	TDX req	TDX tok	TDX-Vordr req	TDX-Vordr tok
Phi-3-mini	steady-state	9.27%	8.74%	10.68%	10.68%
Phi-3-mini	with updates	6.76%	7.09%	7.15%	7.17%
Llama-3.1	steady-state	0.053%	0.053%	0.045%	0.045%
Llama-3.1	with updates	0.013%	0.013%	0.029%	0.029%

on the same machine type with Confidential Computing mode enabled. Each cell provisions a *fresh* VM from a common GCE image to ensure that no state leaks between cells. Both workloads use a load-generator VM co-located in the same cloud zone as the target so that internal-IP RTT is sub-millisecond, and wide-area latency does not contaminate the sub-millisecond effects we measure. For each workload, the matrix is the cross-product of three axes plus repetitions:

- *Attestation epoch* $T \in \{15, 30, 60, 300\}$ s (TDX-Vordr only).
- *IMA baseline log size*: *cold* (5K entries, just-booted) or *warm* (100K entries pre-burned).
- *Workload pattern*: *steady-state* or *with-updates*; the latter triggers an `apt install` at t+120 s and a `pip install` at t+300 s during the measurement window, injecting realistic IMA log change.

For each workload, we calibrate the per-cell offered load to 80% of native saturation. Saturation is identified as the highest target rate for which (i) achieved throughput stays within 1% of target, (ii) there are zero socket errors, and (iii) $p99 < 2 \times p50$. For LLM, we used vLLM's `benchmark_serving.py` on a fresh native VM, sweeping arrival rates to identify per-model saturation; the resulting matrix RPS is 0.15 req/s for Phi-3-mini and 0.03 req/s for Llama-3.1. For NGINX//1kb, we swept `wrk2` target rates from 1,000 to 200,000 req/s on a fresh native VM and fixed the matrix RPS at 80,000 req/s (80% of the 100,000 req/s flat-region top, with the latency knee appearing at 150,000 req/s). For NGINX//100kb, the same procedure identifies a clean ceiling at 25,000 req/s (21 Gbps of HTTP body throughput), giving a matrix RPS of 20,000 req/s. We separately verify that the load generator retains ≥ 50% CPU headroom at the matrix RPS, so the load generator itself is not a bottleneck. Each

Table 3: Cross-model throughput values and slowdowns at the representative slice (warm IMA, T=30 s for TDX-Vordr). Slowdowns are computed relative to Native within the same model and workload.

Model	Workload	Request thr. (req/s)		Output thr. (tok/s)	
		TDX	+Vordr	TDX	+Vordr
Phi-3-mini	steady	−9.27%	−10.68%	−8.74%	−10.68%
Phi-3-mini	w/updates	−6.76%	−7.15%	−7.09%	−7.17%
Llama-3.1	steady	−0.053%	−0.045%	−0.053%	−0.045%
Llama-3.1	w/updates	−0.013%	−0.029%	−0.013%	−0.029%

cell runs a discarded warm-up phase followed by a recorded measurement window, during which we collect workload-specific service metrics, per-attestation-round timings (Δn, end-to-end round time, `pcr_match` verdict) from the verifier, and a 5-second-interval system sampler trace on the target (CPU, memory, IMA log size). Across the entire campaign, every TDX+Vordr round where new IMA entries arrived ($\Delta n > 0$) verified cleanly (`pcr_match`=*True*, `runtime_verdict=CLEAN`), confirming end-to-end correctness of the protocol on real workloads.

D NGINX Workload Details

NGINX 1.27 (alpine) is deployed inside the target VM via Docker with host networking. The configuration enables `epoll`, `reuseport`, `sendfile`, and `tcp_nopush`, sets `worker_connections=16384` and a 65,535 backlog, and disables access logging to remove a non-attestation-related I/O contribution from the measurement. Sysctl tuning on the target raises `somaxconn`, the ephemeral-port range, and `tcp_max_syn_backlog` to values consistent with the matrix

Table 4: NGINX `/1kb` matched delta of TDX-Vordr vs. TDX, by (log-size, workload-pattern). Throughput in req/s; latency in ms.

Log-size	Workload	Δ throughput	Δ p50	Δ p99
cold	steady	+0.05%	+3.59%	+0.84%
cold	with-updates	−0.07%	−2.86%	−1.50%
warm	steady	+0.01%	+0.81%	+0.33%
warm	with-updates	+0.07%	+2.97%	+0.99%

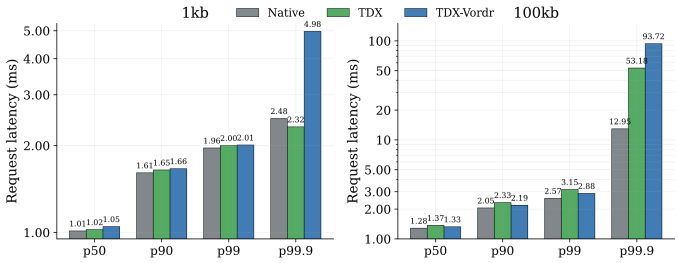


Figure 19: Latency tail by condition (steady, attestation epoch, T=30s)

RPS. Two static routes are exposed: `/1kb` (1024 bytes, syscall-bound regime) and `/100kb` (102,400 bytes, bandwidth-bound regime).

The load generator is `wrk2` [92], the coordinated-omission-corrected variant of `wrk`, configured with 4 threads and 200 persistent connections per cell. The two-phase wrapper runs a 30-second discarded warm-up phase followed by a 300-second recorded phase with `-latency` for HdrHistogram percentile output; warm-up data is excluded from all reported percentiles. We verified that `wrk2` on the load generator retains ≥ 50% CPU headroom at the matrix RPS so the load generator does not become the bottleneck. Table 4 reports the matched-delta $(TDX\text{-}Vordr - TDX)/TDX$ per cell. Two of four cells show *negative* deltas on p99, and one on throughput, indicating that Vordr's effect is at our measurement noise floor.

The 102,400-byte payload exercises the TCP transmit path rather than the syscall-and-keepalive path. At the matrix RPS of 20,000 req/s (≈16 Gbps of body throughput), all three conditions sustain the offered load with the same ≤ 0.1% deviation as on `/1kb`, and the TDX-Vordr vs TDX matched delta is again within run-to-run noise. The mean per-round wall-clock time for TDX-Vordr on `/1kb` is 77 ms in cold and 4,538 ms in warm. The warm number is dominated by round 0, the one-time replay of the pre-burned 100K-entry IMA baseline and subsequent rounds run in ∼50–100 ms regardless of Δn bursts produced by the with-updates pattern. This validates the design assumption motivating the incremental protocol; the cost of bootstrapping the verifier's view of the IMA log is paid once per session and amortized across all subsequent rolling rounds.

Figure 19 shows that Vordr imposes negligible overhead on the vast majority of web traffic, with `p50` and `p99` latencies remaining nearly identical to the Native and TDX baseline across two payload sizes. However, predictable latency spikes emerge at the extreme tail (`p99.9`), which are driven by the periodic background attestation checks and boundary transitions required for continuous runtime verification.

E Scalability of Remote Attestation in Vordr

E.0.1 Audit-Ready RA vs. Cached-RA. Vordr supports two major (three total) end-user serving modes that share the same WEN-side verification path but expose different amounts of evidence to the end user. At each remote-attestation epoch, the measured WEN obtains the TDX evidence from the CVM, verifies the DCAP quote and replays PCR 10 against the received IMA log or delta-log material inside the enclave. The WEN also keeps track of the authorized update/installation history requested by the ASP in a command log. In both modes, WEN verifies the evidence, but they differ in what the end users receive after that WEN verification has been completed. Cached-RA is the compact response format exported by the WEN after it has verified the latest CVM evidence snapshot. A cached-RA artifact bundle contains the following WEN-authenticated elements:

(1) **Nonce binding**: binds the response to the requesting end user and prevents replay of an old cached response to a fresh challenge.
(2) **Issue time**: records when the WEN enclave emitted this RA object.
(3) **Refresh epoch identifier**: identifies which periodic WEN re-attestation epoch produced the cached state.
(4) **Staleness**: tells the end user how old the verified snapshot is relative to the latest WEN refresh.
(5) **CVM update/install-in-progress flag**: indicates whether the WEN is currently executing an ASP-authorized software installation, package update, or other command-driven state transition on the CVM. This lets the end user reject or defer acceptance while the guest is in a transient state whose final measurements may not yet be stable.
(6) **TDX attestation verdict**: states whether the enclave-side DCAP quote verification succeeded for the latest epoch.
(7) **Runtime integrity verdict**: states whether the enclave-side PCR 10 replay and IMA integrity checks succeeded for the latest epoch.
(8) **Measured TD identity**: lets the end user apply policy over the expected boot-time TD image.
(9) **TCB and debug status**: exposes security-relevant platform state needed for acceptance policy.
(10) **Quote digest**: names the exact DCAP quote that the enclave verified, without retransmitting it.
(11) **IMA digest** and entry count $|L_{\text{IMA}}|$: name the exact IMA evidence snapshot whose replay was checked and indicate its scale.
(12) **PCR 10 digest** : binds the response to the specific replay target accepted by the enclave.
(13) **Command-log digest**, size, and entry count: name the exact SGX-authenticated command history used to interpret authorized state changes and to relate the current runtime state to the WEN's approved update actions.
(14) **Enclave authenticator**: authenticates all preceding fields as one WEN-issued RA object.

The raw quote, raw IMA log, raw PCR 10 value, and raw command log are not transmitted in cached-RA. The end user trusts the measured enclave on which WEN runs to have already verified the DCAP quote and replayed PCR 10 over the IMA evidence correctly during the most recent refresh epoch.

Audit-Ready RA contains the entire cached-RA artifact above and additionally includes the raw DCAP quote, the raw IMA log, the raw PCR 10 value, and the SGX-authenticated command log. The end user can therefore recompute the returned hashes immediately and may defer a full independent audit until needed, including re-running DCAP quote verification and PCR 10 replay over the IMA log. This mode preserves the protocol's audit path. However, this mode shifts the bottleneck from attestation to evidence serialization and transport.

Table 5 compares the response sizes for cached RA and audit-ready RA modes. The cached-RA response size is approximately 1.4 KB based on the actual compact JSON response schema returned by the server, because the benchmark logs payload bytes only for the full-bundle mode. The reported audit-ready RA throughput should be interpreted as the cost of serving and transporting a complete evidence bundle after the WEN has already performed quote verification and PCR 10 replay during the background refresh epoch. It is not the cost of re-running the expensive checks at every end user on every request. Cached-RA throughput is effectively constant because the response is compact and size-stable. Audit-Ready RA throughput is conditioned on the evidence snapshot size because, as the IMA and command logs grow, the audit bundle grows, and serving throughput declines approximately in inverse proportion to the serialized RA artifact bundle size once transport becomes the bottleneck.

IMA measurement logs are workload- and policy-dependent rather than time-bounded. The official IMA documentation reports that a reasonable policy produces roughly 5K records at boot and can grow to around 100K records over time, while broader or poorly scoped policies that measure rapidly changing files can produce hundreds of thousands of records [38]. Empirical measurements show comparable orders of magnitude: an IMA-enabled system was reported to reach 214K log lines after about 15 days of uptime, corresponding to roughly 40 MB, with system updates, initramfs generation, and source compilation contributing several tens of thousands of entries [75]. A recent DSN study of Keylime also found an Ubuntu executable-policy baseline of 323K entries, or 46 MB, and reported that daily system updates added 1,271 executable-file entries and 0.16 MB on average [80]. Thus, long-running systems using executable-content IMA policies should expect logs in the thousands after boot and potentially tens of thousands to around 100K entries over time, especially when updates, service restarts, module loads, or build/install activity introduce new executable objects.

Mode	Size
Direct DCAP (via CVM)	8 KB
Cached RA (via Native)	1.4 KB
Cached RA (via WEN)	1.4 KB
Audit-Ready RA (1K IMA entries)	1.52 MB
Audit-Ready RA (10K IMA entries)	3.95 MB
Audit-Ready RA (50K IMA entries)	14.61 MB
Audit-Ready RA (100K IMA entries)	27.95 MB

Table 5: Approximate serialized response size by attestation mode. Audit-Ready RA projections keep the measured 8KB quote and 950KB command log fixed while scaling only the raw IMA log size.

F Employing Multiple Enclaves

Our design so far uses a single Warden Enclave (WEN). While this simplifies the framework, it creates a single point of failure: if that WEN crashes, becomes unavailable, or is saturated by concurrent requests, the system cannot process either remote-attestation (RA) or runtime-modification (RM) requests. In addition, as the number of CVMs, ASPs, and end users grows, a single WEN becomes a throughput bottleneck. To address these limitations, we extend Vordr to a multi-WEN architecture in which a pool of WENs manages a shared set of CVMs. This helps improve read-side availability and throughput while maintaining a consistent view of each CVM's attested state.

For each CVM v, the WEN pool maintains replicated WEN-side state, including the latest committed attestation checkpoint, boot-epoch metadata, and audit-log head. We distinguish between *reads*, which serve attestation evidence to end users, and *writes*, which execute runtime modifications on the CVM. Reads may be distributed across multiple WENs, but writes must remain serialized per CVM to preserve consistency.

F.1 Consistency of Attestation and Runtime Updates

Introducing multiple WENs raises a consistency problem. RA requests are logically read operations over a CVM's state, whereas RM requests change the CVM state and therefore act as writes. If multiple WENs were allowed to issue concurrent modifications, they could diverge in their view of the CVM's IMA log, attestation checkpoint, or audit record. We therefore assign each CVM v a unique write leader $\mathcal{L}(v)$ and require that only $\mathcal{L}(v)$ may issue runtime modifications to v. Leader maintains a per-CVM *write-in-progress* flag that is visible to all WENs. When the leader issues any modification command to CVM, other WENs are informed so they can continue serving RA requests (with a write-in-progress flag). After the modification completes, the leader re-attests the CVM, advances the WEN-side checkpoint, appends the corresponding audit entry, and broadcasts the new committed state to others. A leader for each CVM can be selected and maintained using a lease-based coordination or consensus protocols such as Raft [71] alongside the enclaves, as explored in prior works [84, 86, 100]. Such leader selection and coordination in distributed systems is an orthogonal problem, and for simplicity, we assume that every CVM has a known current leader and that leader changes occur only through an external reconfiguration or recovery procedure [15].

F.2 Load Scheduling of Attestation Requests

We distribute RA requests across the non-leader WENs using a simple round-robin algorithm. For each CVM v, let $\mathcal{F}(v)$ denote the set of follower WENs that are currently eligible to serve attestation for v. A follower belongs to $\mathcal{F}(v)$ only if it is alive, admitted to the WEN pool, and synchronized to the latest committed state for v. Round robin is then applied over $\mathcal{F}(v)$.

This approach removes the single-WEN bottleneck on the read path, requires no centralized scheduler, and scales naturally as new eligible follower WENs are added. We note that it does not eliminate the write-side single point of failure, because RM requests remain serialized through $\mathcal{L}(v)$. If no eligible follower is available for a given CVM, the request may be served by the leader or deferred. If the deployments exhibit substantial heterogeneity across WENs or highly skewed request costs, the round-robin policy can be replaced by a load-aware scheduler (e.g., power-of-two choices [66]) without changing the underlying consistency model. We leave such an extension to future work.

F.3 WEN Admission and Revocation

To join the pool, a candidate WEN C_{new} contacts any existing WEN C_{spon}, which performs WEN-to-WEN local/remote attestation [21] to verify that C_{new} runs the expected (public) WEN code (i.e., has the correct enclave measurement). Upon successful attestation, C_{new} is admitted to the WEN pool for RR scheduling, establishes mutually authenticated channels with the existing WENs, and is subscribed to the in-guest attestation agent for the CVMs it will serve. It receives the current attestation checkpoint, boot-epoch metadata, audit-log head, and the current write-status flag for each assigned CVM.

A WEN may be removed from the pool for routine decommissioning, persistent unavailability, failure to attest, or suspected compromise. Revocation has two stages. First, the WEN is immediately removed from all scheduling sets so that it no longer receives RA traffic. Second, its authenticated peer channels and any role-specific credentials are revoked. If the removed WEN held write authority for any CVM, that authority must be reassigned before further RM requests are accepted for those CVMs. If the removed WEN possessed per-CVM administrative credentials, those credentials should be rotated before another WEN assumes leadership. A revoked WEN that later rejoins must repeat the full admission procedure, including attestation and state catch-up.