TÉCNICO
UNIVERSIDADE
DE LISBOA

A Unified Framework for Attested Confidential VM Workloads in Public Clouds

João Henriques Sereno

Thesis to obtain the Master of Science Degree in

Computer Science and Engineering

Supervisors: Prof. Nuno Miguel Carvalho dos Santos
Prof. Luís Eduardo Teixeira Rodrigues

Examination Committee

Chairperson: Prof. Name of the Chairperson
Supervisor: Prof. Nuno Miguel Carvalho dos Santos
Member of the Committee: Prof. Bernardo Luís da Silva Ferreira

May 2026

Declaration
I declare that this document is an original work of my own authorship and that it fulfills all the requirements of the Code of Conduct and Good Practices of the Universidade de Lisboa.

Acknowledgments

My supervisors, Prof. Nuno Santos and Prof. Luís Rodrigues, brought both depth and clarity to this work. Their advice kept me focused, and their judgment helped me give the best I had. I am grateful for their insightful advice, the freedom they gave me, and for the trust that came with it.

Daniel Castro was a constant presence throughout this work. His intuition helped solve the hardest technical problems we faced. His guidance and encouragement helped overcome many of the barriers encountered while working on this. I learned a lot from him.

To my mother, Telma, and my father, Abílio, thank you for your unconditional support. Every time, you believed in my choices and capabilities, even when those were hard to comprehend from the outside. You made this possible.

I would like to thank my friends and family for their companionship, support, and for making these years worth remembering.

Lastly, I want to thank my girlfriend, Beatriz, for her immeasurable assistance in every part of my life. You lifted me up in the worst moments and made the good ones better. I will be forever grateful for your friendship and kindness.

To you all, my sincerest thanks.

This work was developed within the scope of: proj. no.62–“Responsible AI”, financed by European Funds, namely “Recovery and Resilience Plan”–Component 5: “Agendas Mobilizadoras para a Inovação Empresarial”, included in the NextGenerationEU funding program; the EU’s Horizon Europe research and innovation programme under Grant Agreement No 101189689; FCT under grants UID/50021/2025 and UID/PRR/50021/2025; and, IAPMEI under grant C6632206063-00466847 (PT Smart Retail).

Abstract

Confidential Virtual Machines (VMs) backed by AMD Secure Encrypted Virtualization–Secure Nested Paging (SEV-SNP) allow workloads to execute on public cloud infrastructure while excluding the infrastructure operator from accessing the VM memory or execution state. Remote attestation makes this protection verifiable, but in practice the verification is rarely performed independently. The tools and provider-specific interfaces needed to derive expected measurements exist in fragments, and no coherent pipeline assembles them into a workflow that a verifying party can use to predict and verify the measurements of an arbitrary workload across providers.

This thesis analyzes the attestation properties of AMD SEV-SNP confidential VM offerings from Amazon Web Services, Microsoft Azure, and Google Cloud Platform, and classifies the software verification depth assurance each provider makes achievable according to a proposed taxonomy. It then introduces *Evident*, a lifecycle management framework that integrates and extends tooling into a single pipeline spanning image construction, measurement derivation, deployment, and cross-provider remote attestation. The verifying party derives expected measurements directly from the VM image artifacts, removing the dependency on externally supplied reference values. The framework requires no modifications to the deployed workload. Evaluation through a confidential inference use case confirmed the predicted measurements match, with remote attestation completing under two or eight seconds, depending on the cloud provider, and the accompanying server component posing little to no interference with the workload execution.

Keywords

Confidential Computing; AMD SEV-SNP; Remote Attestation; Trusted Execution Environments; Virtual TPM; Reproducible Measurements

Resumo

As máquinas virtuais confidenciais protegidas pela tecnologia AMD Secure Encrypted Virtualization–Secure Nested Paging (SEV-SNP) permitem que cargas de trabalho sejam executadas em infraestruturas de nuvem públicas, impedindo que o operador da infraestrutura aceda à memória ou ao estado de execução da máquina virtual. A atestação remota torna esta proteção verificável, mas, na prática, a verificação raramente é realizada de forma independente. As ferramentas e interfaces específicas de cada operador, necessárias para obter as medições esperadas, estão fragmentadas e não há um processo coerente que permite a uma entidade verificadora prever e verificar as medições de uma carga de trabalho arbitrária entre operadores.

Esta tese analisa as propriedades de atestação das ofertas de máquinas virtuais confidenciais AMD SEV-SNP da Amazon Web Services, Microsoft Azure e Google Cloud Platform, e classifica a profundidade de verificação de software alcançável em cada operador segundo uma taxonomia proposta. Apresenta então o *Evident*, uma estrutura de gestão do ciclo de vida que integra e amplia as ferramentas num processo único que inclui a construção de imagens, a obtenção de medições e a atestação remota entre operadores. A entidade verificadora obtém as medições esperadas diretamente a partir dos artefactos da imagem, eliminando a dependência de valores de referência externos. Validou-se a solução através de um caso de uso de inferência confidencial, que confirmou a correta previsão de medições. A atestação remota demora menos de dois ou oito segundos (dependendo do operador) e o servidor acompanhante não interfere com a carga de trabalho.

Palavras Chave

Computação Confidencial; AMD SEV-SNP; Atestação Remota; Ambientes de Execução Confiáveis; TPM Virtualizada; Medições Reproduzíveis

Contents

List of Figures

List of Tables

Listings

Acronyms

AK	Attestation Key
ARK	AMD Root Key
ASK	AMD SEV Key
ASVK	AMD SEV-VLEK Key
EK	Endorsement Key
GPG	GNU Privacy Guard
HCL	Host Compatibility Layer
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
PCR	Platform Configuration Register
SEV-SNP	Secure Encrypted Virtualization–Secure Nested Paging
SVSM	Secure Virtual Machine Service Modules
TEE	Trusted Execution Environment
TLS	Transport Layer Security
TPM	Trusted Platform Module
UKI	Unified Kernel Image
VCEK	Versioned Chip Endorsement Key
VLEK	Versioned Loaded Endorsement Key
VM	Virtual Machine

1

Introduction

Contents

The increasing sensitivity around how and where data is processed has driven demand for stronger security guarantees than those traditionally available to cloud customers. Organizations handling medical records, financial transactions, or other regulated data must currently rely on advertised regulatory compliance (such as the European Union's General Data Protection Regulation), contractual commitments, audit reports, and institutional reputation to trust that a cloud provider will not observe or mishandle the data entrusted to it. These options are not technical assurances and they offer no mechanism for independent verification by the data owner. Privacy-enhancing technologies built on trusted hardware aim to change this balance by anchoring trust in the physical properties of the processor rather than in the policies of the entity operating it.

Confidential virtual machines, built on hardware such as AMD SEV-SNP [1], allow workloads to execute on public cloud infrastructure while excluding the infrastructure operator from accessing the VM's

memory or execution state. The trust anchor shifts from the cloud provider's policies to the physical properties of the processor and the endorsements of its manufacturer. Remote attestation is the protocol that makes this shift concrete: a verifier collects cryptographically signed evidence of a VM's configuration and compares it against expected values, obtaining an independently verifiable statement about what software the VM is running [2]. Without attestation, a confidential VM provides hardware-level protection but no way for an external party to confirm that the protection is in place or that the intended software was loaded.

In practice, however, attestation for confidential VMs on public cloud infrastructure remains impractical for general-purpose workloads. The three major providers that offer AMD Secure Encrypted Virtualization–Secure Nested Paging (SEV-SNP) (Amazon Web Services, Microsoft Azure, and Google Cloud Platform) expose attestation evidence differently, with varying degrees of transparency about what is measured and how. Verifying that evidence requires knowing exactly what was measured and being able to reproduce the measurement from the same artifact. Existing frameworks such as Keylime [3] and Constellation [4] focus on the remote attestation protocol itself but leave an important question unaddressed: how can the measurements be independently derived while still reflecting the generic workload within the VM? Currently, the practical option for verifiers is either to trust a third party to supply reference measurements or to accept the first observed values as a baseline. Neither approach provides the independent assurance that the confidential VM trust model is designed to support. The gathered evidence should, by itself, let a verifier confirm that the hardware and software configuration of a deployed VM matches its expectations. This thesis presents *Evident*, a framework that closes this gap by enabling the verifying party to independently predict and verify the measurements produced by a confidential VM, without requiring modifications to the deployed workload.

1.1 Motivation

The attestation gap outlined above raises two problems that this thesis addresses.

The first focuses on understanding what current cloud platforms actually guarantee. The three major providers that offer AMD SEV-SNP confidential VMs each configure their own solution differently. They differ in what the AMD *Secure Processor* measures, whether the measured artifacts are available to the verifying party, whether the measurement derivation procedure is documented, and how the virtual Trusted Platform Module (TPM) is implemented and endorsed [5, 6]. These differences are substantial enough that an attestation result on one provider carries a fundamentally different meaning than on another. A verifier who wants to assess the integrity of a confidential VM deployment needs a remote attestation procedure that accounts for the specific security guarantees and limitations of each provider's hypervisor and attestation architecture. No such procedure exists, and no common vocabulary precisely

describes or compares the depth of assurance across providers.

The second problem concerns the absence of tooling for independent measurement derivation and verification. Even where a provider's attestation architecture is well understood, no existing framework enables a verifier to independently derive the expected measurements for a generic workload and verify them against evidence from a running instance. Existing attestation frameworks such as Keylime [3] and Constellation [4] focus on the remote attestation protocol: collecting evidence and comparing it against expected values. They treat the expected values as an external input. Keylime leaves its provenance to the operator. Constellation embeds them in its release artifacts, requiring trust in the platform vendor. In both cases, the verifier cannot confirm, from trusted measurements alone, that the specific workload deployed within the VM is the intended one. No framework spans the full confidential VM lifecycle in a way that lets the verifying party derive measurements directly from the artifacts they built or inspected. Verifiers must either trust a third party to supply reference measurements or accept the first observed values as a baseline. The practical consequence is that even deployments running in measured environments that support independent verification remain effectively unverified, because the existing framework is limited, or the tooling to perform that verification does not exist as a coherent pipeline.

1.2 Contributions

This thesis investigates how the integrity of confidential virtual machines deployed on public cloud infrastructure can be independently verified, and proposes a practical pipeline that enables a verifying party to derive its own expected measurements rather than relying on externally supplied reference values. The thesis makes two primary contributions:

A systematic analysis of public cloud confidential VM attestation. A comparison of AMD SEV-SNP confidential VM support across Amazon Web Services (Elastic Compute Cloud), Microsoft Azure (Azure Virtual Machines), and Google Cloud Platform (Google Compute Engine). The analysis examines the transparency, interpretability, authenticity, reproducibility, and completeness of the measurements produced by both the AMD *Secure Processor* and the virtual TPM on each platform. A software verification depth taxonomy is proposed to structure the comparison, classifying the verification depth achievable for each main software component of the confidential VM, ranging from root-of-trust link authenticity through source code measurement reproducibility. This analysis directly informed the design of the framework introduced below.

A lifecycle management pipeline for independently verifiable confidential VM deployments. A framework design, protocol specification, and architecture that spans the full lifecycle of a confidential virtual machine: image construction, measurement derivation, image upload, deployment, and cross-provider remote attestation. The framework adapts the remote attestation procedures protocol [2] to

account for the heterogeneity of real-world cloud environments. Its distinguishing property is that the verifying party derives expected measurements directly from the artifacts that constitute the VM image, removing the dependency on any third party to supply those measurements. The design is workload-agnostic, requires no modifications to the deployed workload, and is extensible to additional cloud providers and hardware roots of trust. The framework also specifies several extensions that address practical deployment concerns: an image distribution protocol with signed manifests for reproducible verification, attestation certificates that bind instance keys to measured state, and a Confidential Certificate Authority design for scalable attestation through certificate hierarchies.

1.3 Results

This thesis produced the following results:

The *Evident* framework implementation. A concrete implementation of the lifecycle pipeline, built for today's cloud environment landscape. The framework supports independent measurement derivation for both the AMD *Secure Processor* launch digest and the virtual TPM Platform Configuration Register values on Elastic Compute Cloud and Google Compute Engine. The implementation was validated through a confidential inference use case, in which a large language model inference service was built, deployed, attested, and accessed on both providers. Metrics also show that the components of the framework introduce negligible friction to the overall performance and resource consumption of the target workload.

Contributions to open-source tooling. As a partial result of this thesis work, support for Google Compute Engine's launch endorsement derivation was contributed to Virtee's *sev-snp-measure-go* library and *sev-snp-measure* tool. This makes AMD *Secure Processor* launch digest derivation for both Elastic Compute Cloud and Google Compute Engine addressable through a common, established interface.

1.4 Structure of the Document

This thesis is organized as follows:

Chapter 2 provides the technical background necessary for understanding the rest of the thesis. It covers Trusted Execution Environment and AMD Secure Encrypted Virtualization–Secure Nested Paging in particular, the Trusted Platform Module and its role in extending the measurement chain, the boot process of a confidential VM, and the Remote Attestation Procedures architecture. Chapter 3 surveys the related work. It compares the confidential VM offerings of the three major cloud providers, examines on-premises solutions that extend the attestation chain beyond what current public cloud offerings

allow, and reviews existing privacy-preserving systems and frameworks that build on confidential VMs, some of which on top of public cloud provider solutions. Chapter 4 introduces the *Evident* framework. It presents the design principles, the threat model, architecture, and the extensions that address practical deployment concerns. It also discusses the limitations that the framework exposes but cannot resolve from the customer's standpoint. Chapter 5 details the implementation. It documents the technology choices, the construction of VM images using NixOS, the measurement derivation process for both the AMD *Secure Processor* and the virtual TPM, the upload and deployment tooling, the provider-specific remote attestation workflows, and the implemented extensions. Chapter 6 evaluates the framework. It validates the end-to-end workflow through a confidential inference use case, confirms the correctness of the measurement predictions, describes the remote attestation report structure, and characterizes the latency and resource consumption of the attestation components. Finally, Chapter 7 summarizes the thesis contributions and results in further detail and enumerates the directions that are yet to be explored and/or completed.

2

Background

Contents

Cloud computing requires surrendering control over the infrastructure that processes your data. For most workloads, this is an acceptable trade-off: the provider handles operations, the customer gains elasticity, and contractual commitments fill the gap where direct oversight used to be. For workloads that process medical records, financial data, or other sensitive material, the gap is harder to fill. The question is not whether the provider is competent, but whether any contractual or institutional assurance can substitute for the ability to verify what the infrastructure is actually doing with the data entrusted to it.

Self-hosting does not automatically resolve the problem. On-premises infrastructure carries its own costs, its own operational risks, and its own trust dependencies on the people and supply chains that build and maintain it. The practical question is rarely whether *to* trust, but *how* narrowly the set of trusted

components can be scoped for a given deployment, and whether that scoping can be verified rather than assumed.

An increasingly common answer is to anchor trust in hardware. Rather than relying on a provider's policies or audit reports, one relies on the physical properties of the processor and the endorsements of its manufacturer. A hardware root of trust concentrates the assumptions to the correctness of the silicon and its firmware, the integrity of the manufacturing process, and the integrity of the manufacturer's key management. It still requires some trust assumptions, but these are narrower and more stable than the ones they replace, and critically, some can be tested through cryptographic attestation rather than accepted on faith.

This chapter examines the mechanisms that make such a foundation work. Section 2.1 explores Trusted Execution Environments and their guarantees. Section 2.2 explains the Trusted Platform Module and its role co-located with a Trusted Execution Environment. Section 2.3 details the boot process of a confidential VM, and how that process can propagate trust through measurements. Finally, Section 2.4 covers how a relying party can make conclusions about the state of a remote VM. Together, these mechanisms form the technical background for the rest of the dissertation.

2.1 Trusted Execution Environments

A Trusted Execution Environment is a hardware-enforced isolated region of a processor in which code and data are protected against access or modification by any software running outside it. What distinguishes a Trusted Execution Environment (TEE) from software-only isolation mechanisms (such as an operating system's process memory isolation) is precisely that its guarantees are not configured by software and cannot be overridden by it; they are enforced by dedicated hardware structures. This is what characterizes TEEs as trustable elements.

The core capabilities of a TEE follow directly from this premise. First, confidentiality: the contents of a TEE's memory are encrypted such that no external entity, even one with physical access to the machine, can read the plaintext contents. Second, integrity: unauthorized modifications to the TEE's memory are detected and blocked in hardware. Third, and especially relevant for this dissertation, attestation: because the hardware enforces the environment's properties, it is also in a position to generate a cryptographically signed statement describing the TEE's configuration and the software loaded into it. This statement allows a remote party to verify, without trusting additional components, that the environment is correctly configured and that it contains the intended code. Taken together, these capabilities make a TEE independently verifiable, as it exposes sufficient evidence for an external party to assess its state and take a meaningful trust decision.

TEE are most relevant when a workload must execute on infrastructure that the data owner cannot

directly control or inspect. The use of a TEE narrows the set of entities that the data owner must trust, because the cloud provider's software stack can be excluded from the security boundary. The trust anchor shifts to the underlying hardware and its manufacturer.

TEE implementations differ substantially in their granularity and design philosophy. Intel's Software Guard Extensions processor feature operates at the process level: an application developer defines a small, encrypted memory region called an *enclave*, which the processor isolates from all other software, including the operating system [7]. It allows for scenarios with minimal trust models, but the burden of managing enclave memory and designing the trust boundary falls entirely on the application developer. ARM TrustZone partitions the processor into two execution "worlds": a secure world and a normal world; and is widely deployed in mobile and embedded devices, where the threat model emphasizes protecting trusted firmware from untrusted applications rather than protecting workloads from untrusted infrastructure [8]. Intel's Trust Domain Extensions processor feature takes a different approach, operating at the virtual machine level: an entire guest virtual machine can be designated as a trust domain, with the hardware enforcing its isolation from the hypervisor [9]. AMD's Secure Encrypted Virtualization–Secure Nested Paging (SEV-SNP) compatible processors also operate at the virtual machine level, and this is the technology on which this dissertation focuses. It is designed specifically for the cloud context, offering the VM-level TEE primitive (also called confidential VM), with a threat model that explicitly includes a potentially compromised or malicious hypervisor [1].

AMD Secure Encrypted Virtualization–Secure Nested Paging (SEV-SNP) is a hardware technology, built into AMD's *EPYC* processor family, that protects a guest virtual machine's memory from the hypervisor and from any co-located virtual machines on the same physical host. It does so by delegating all memory security responsibilities to a dedicated security coprocessor (also called AMD *Secure Processor*) embedded in the processor die, with its own isolated on-chip private memory for storing sensitive material, such as encryption keys and attestation artifacts [1, 10, 11].

The AMD *Secure Processor* has exclusive authority over encryption and physical memory access across the entire machine. Each confidential virtual machine is assigned a unique secret key at creation time, unknown to the hypervisor, which the coprocessor uses to ensure that the VM's memory pages remain encrypted at rest and decrypted only when accessed within the correct CPU context [1, 11]. Any access from an unauthorized context — whether from the hypervisor or another virtual machine — is guaranteed to be unfruitful by the hardware [1]. The net effect is that the hypervisor retains its operational role, but is entirely excluded from observing or modifying the confidential VM's memory or execution state.

SEV-SNP additionally introduces virtual machine privilege levels, which allow the memory of a single confidential VM to be partitioned into up to four hierarchical privilege tiers [1]. Each virtual CPU and each memory page is assigned to one of these levels, and the hardware enforces that a component at

a given level cannot access the memory of a higher-privileged one. This is particularly useful when the threat model within the VM itself is not flat. Consider, for example, a deployment where a public-facing application and a security-critical operating system component run inside the same confidential VM: assigning them to different privilege levels provides a hardware-anchored guarantee that a compromise of the application cannot reach the operating system's memory pages, without requiring a separate VM for each.

These mechanisms together make a SEV-SNP confidential virtual machine verifiable. They constitute a set of hardware-enforced properties that an external party can examine and assess. Whether the software running inside the confidential VM is also trustworthy is a separate question. The trust anchor is the AMD *Secure Processor* and its firmware; a threat model that accepts AMD's hardware as its root of trust can extend that acceptance to any confidential VM the coprocessor vouches for. One that does not (perhaps because it cannot accept a hardware manufacturer as a trusted third party) will find this chain of trust worthless.

To make the platform's properties verifiable to a remote party, the AMD *Secure Processor* can generate a signed attestation report committing to the virtual machine's configuration and to a measurement of the software loaded into it at launch. Each report must be signed by a key whose authenticity traces back to AMD. In the default configuration, the signing key is the Versioned Chip Endorsement Key (VCEK), a key unique to the physical chip. The VCEK is endorsed by the AMD SEV Key (ASK), which itself is endorsed by the AMD Root Key (ARK), forming a certificate chain rooted at AMD. Cloud providers may instead use a Versioned Loaded Endorsement Key (VLEK), a non-chip-unique signing key that AMD issues in encrypted form. Because the VLEK is not tied to a specific chip, confidential VM migrations between physical hosts are not detectable by the guest. When the encrypted VLEK is loaded into the *Secure Processor*, it is decrypted and used as the report signing key. The VLEK follows a parallel endorsement path: it is endorsed by the AMD SEV-VLEK Key (ASVK), which is itself endorsed by the ARK. In both paths, all endorsing keys are exclusively managed by AMD. How the launch measurement is constructed is covered in Section 2.3, and how a remote party verifies it is covered in Section 2.4.

2.2 Trusted Platform Module

The attestation report generated by the AMD's *Secure Processor* commits to the initial memory contents of a confidential VM at launch time; this process is further detailed in Section 2.3. In practice, it typically covers the VM firmware but omits other important components like the bootloader, the operating system kernel, the early userspace, and all the filesystem's contents. Since all of these other components are not measured, they fall outside the chain of trust anchored in AMD's *Secure Processor*. An adversary with the ability to modify any of those would not be detected by the launch measurement alone.

The Trusted Platform Module provides a mechanism for reliable measurement of software components during the VM boot process, which may be used to gather evidence of the gap left by AMD's launch measurement.

The Trusted Platform Module (TPM) is a trustable cryptographic coprocessor, standardized by the Trusted Computing Group, that provides a hardware-anchored root of trust for general-purpose machines. Unlike the newer AMD *Secure Processor*, which is embedded in the CPU die and scoped to the confidential VM use case, the TPM has historically been a discrete component soldered to the motherboard, operating independently from the main processor and system memory. Its physical isolation from the host's software stack is what differentiates it as a trustable entity: its key material and internal state cannot be read or tampered with through software alone, regardless of what runs on the host.

The TPM's primary contribution to integrity measurement is a set of hardware-protected registers called Platform Configuration Registers (PCRs). Each PCR holds a cryptographic hash and can only be updated through a strictly defined operation called an "extend": the register's current value is concatenated with a new measurement and replaced with the hash of that combination. Formally, $\text{PCR}_i^{(t+1)} = H\left(\text{PCR}_i^{(t)} \parallel m_t\right)$, where PCR_i^t is the i-th PCR value at time t, H is the TPM's hash function and m_t is the new measurement. Because each extension depends on the register's prior state, the resulting value is a cumulative digest of every measurement submitted in chronological order. A PCR value, therefore, cannot be forged to match an expected state without knowing the exact sequence of measurements that produced it.

To expose this record to a remote party, the TPM produces a signed attestation of its current PCR contents called a "quote". The quote is signed by an Attestation Key derived from a secret seed provisioned at manufacturing time and unique to the device. The TPM holds an Endorsement Key that may only be used to prove co-residency of other keys and itself. The TPM manufacturer issues an Endorsement Key (EK) certificate that binds the EK to the manufacturer's identity. A trust model that accepts a certain manufacturer's implementation of the TPM could, by derivation, accept EK-backed proofs of co-residency of the Attestation Key (AK), and AK endorsements to quotes, as they verify both keys live inside the TPM and their use is restricted by the TPM's interface. To ensure quote freshness, the verifier should supply a nonce to be embedded in the quote and verify that it matches.

Deploying physical TPMs in cloud environments presents an immediate scaling problem, since a single chip serves one host. The virtualized TPM addresses this by emulating the TPM interface and behavior in software, giving each VM the appearance of a dedicated device. The trust anchor for that emulated device can be placed in different roots: implementations that maintain the original hardware anchor [12], each virtualized TPM instance derives its identity from the physical TPM through a key hierarchy; in hypervisor-anchored implementations [13, 14], the hypervisor manages each virtualized TPM state and isolation; alternatively, explored further in Section 3.2, the emulated device can be anchored

to AMD's *Secure Processor* and execute inside the guest confidential VM. Regardless of implementation, the EK certificate plays a key role in determining the trustworthiness of the implementation of the virtualized TPM, as it is contingent on the trustworthiness of the entity that issued it.

2.3 Boot Process

The foundation of trust in a boot sequence rests on a simple principle: measure a component before executing it. A measurement is a cryptographic hash of a component's binary contents, extended into a tamper-evident register held by a trustable entity such as AMD's *Secure Processor* or a Trusted Platform Module (TPM). Because that entity controls the register and the extension operation is strictly one-way, no software on the system can overwrite or forge a past measurement. If a component is hashed before it runs, any deviation from the expected binary will produce a different hash, permanently recorded by a hardware component that the software it monitors cannot reach.

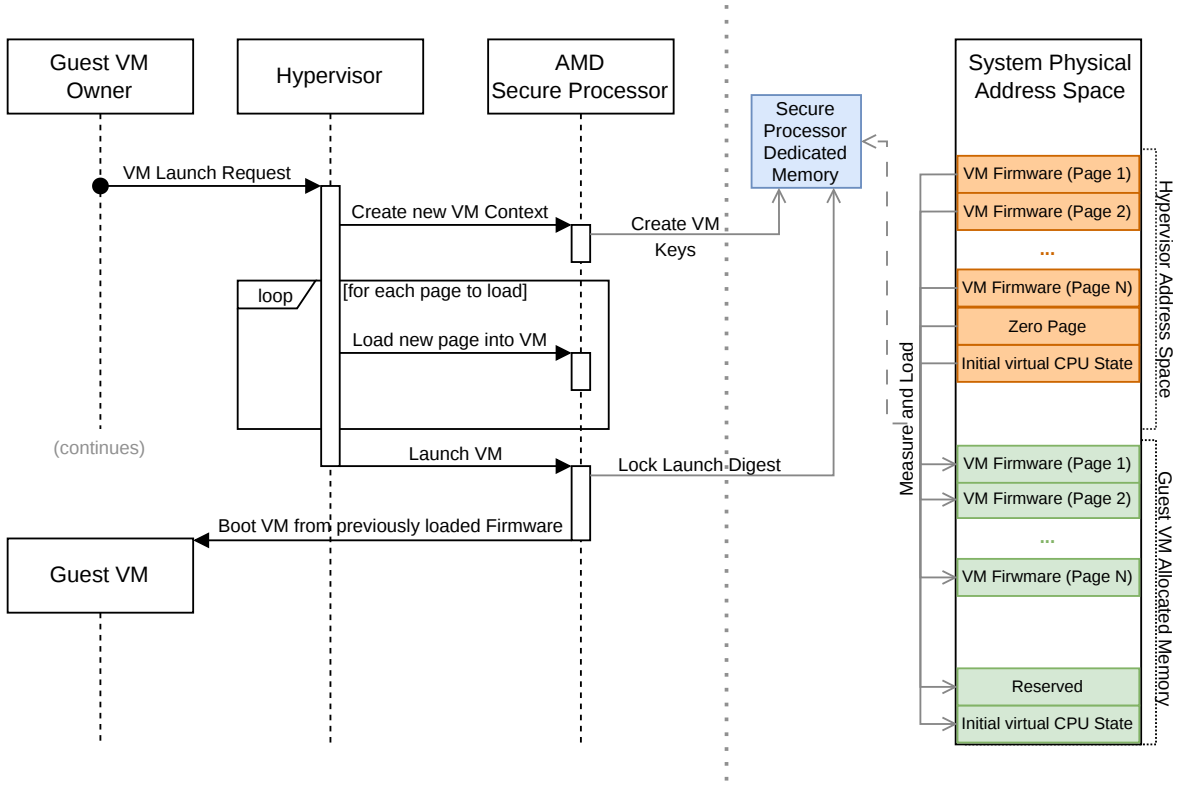
This mechanism carries an inherent bootstrapping problem. The very first component to execute cannot be measured by any predecessor, because there is no predecessor. Physical machines address this by storing the earliest executable code in a small read-only memory whose immutability is enforced by hardware. Virtual machines have no equivalent since the firmware that plays this role is just another file managed by the hypervisor, replaceable before the VM starts. A compromised firmware could self-report a plausible measurement while loading a tampered successor, propagating the compromise silently through every subsequent boot stage. Self-measuring, where the first component extends a hash of its own binary, is unreliable for exactly this reason: a dishonest implementation can extend any value it chooses.

In confidential VMs, AMD's *Secure Processor* solves this issue. The boot process unfolds in two distinct phases. In the pre-boot phase, AMD's *Secure Processor* measures every memory page loaded into the VM before the processor executes a single instruction inside it. In the boot phase, once the VM is running, successive boot components measure one another and extend those measurements into a virtual TPM. The two phases form a contiguous chain: AMD's *Secure Processor* covers the part that no software component can reliably cover, and the TPM carries the chain forward through the rest of the software stack.

The pre-boot phase begins when the hypervisor requests AMD's *Secure Processor* to instantiate a new confidential VM. The *Secure Processor* creates a guest context in its own reserved memory and initializes a launch digest to zero [15], as seen in Figure 2.1. This digest will accumulate a cryptographic record of every memory page loaded into the VM before it starts.

The hypervisor then populates the VM's memory one page at a time. For each page, the *Secure Processor* encrypts it with the VM's unique memory key and extends the launch digest by hashing the

Figure 2.1: AMD SEV-SNP confidential VM pre-boot measurement and launch



current digest together with the page contents and its metadata [15], illustrated in Figure 2.1. That metadata encodes the page type, its target guest physical address, and the privilege level under which it will be accessible [15]. Including the physical address in the measurement binds the initial memory layout to the digest; therefore, reordering pages or placing them at different addresses produces a different final value.

It is the hypervisor, not the guest, that decides what goes into the VM's initial memory. The AMD *Secure Processor* does not independently choose the VM's content; it faithfully records what the hypervisor presented. The security guarantee is that the hypervisor cannot misrepresent what it loaded: any deviation from what the verifier expects will produce a different digest that attestation will expose.

Once all pages are loaded, the hypervisor issues a finalization instruction. The AMD *Secure Processor* locks the launch digest, stores it in its isolated memory, and makes it available for inclusion in attestation reports [15], also observed *3* in Figure 2.1. Execution inside the VM begins only after this point. The firmware, typically the first binary to run, has already been measured in full before it executes a single instruction, closing the bootstrapping gap that software-only approaches cannot close.

Once the VM is running, the boot measurement phase begins. The firmware sets up access to the virtual TPM and measures the bootloader before transferring control to it; the bootloader measures

the kernel before transferring control to it; and so on. Each component measures its successor and extends that measurement into a TPM Platform Configuration Register before handing over execution. If an unexpected component appears at any stage, its measurement will differ from the expected value, leaving a permanent record in the TPM state [16, 17].

The specifics of this phase vary considerably across firmware implementations, bootloaders, and operating systems. Which components are measured, which Platform Configuration Register (PCR) receives each measurement, and the granularity of those measurements are all implementation-defined. Since the firmware implementation is often platform-agnostic (does not assume it is running on a confidential VM), measurements of the firmware may be covered both in the *Secure Processor*'s launch digest and the TPM's PCRs. This is redundant, but non-malicious.

After the boot finalizes, a verifying party may fetch the gathered measurements. However, ensuring that the measurements had a trustworthy source is necessary but not sufficient to trust an observed boot sequence. Section 2.4 covers how the verifying party can ensure there is a reliable link between the gathered evidence and the trustable components, and also how exactly the collected evidence can be translated into security guarantees.

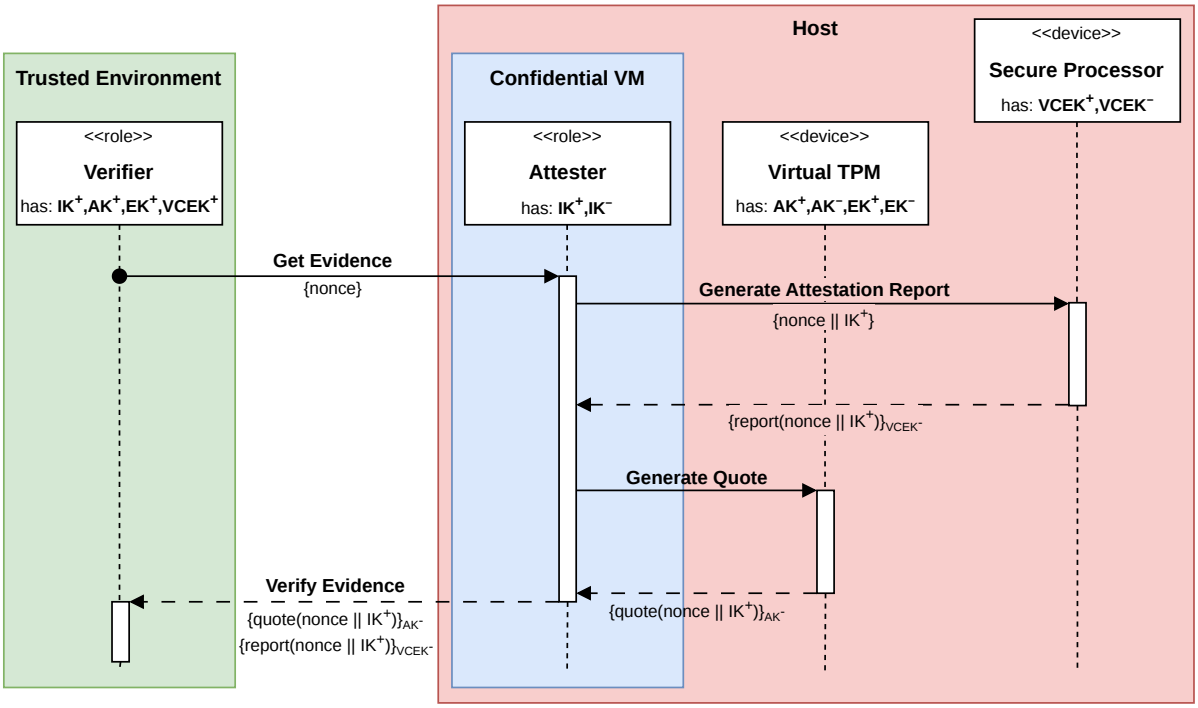
2.4 Remote Attestation

Remote attestation is a protocol that enables a *verifier* to assess the trustworthiness and integrity of a remote computing platform. The Internet Engineering Task Force's "Remote Attestation Procedures" architecture [2], defined in the Request for Comments number 9334, establishes the generalized conceptual framework for the attestation process across heterogeneous computing environments.

This architecture delineates three primary entities: the *Attester* (the platform generating *evidence* of its state), the *Verifier* (the entity evaluating attestation *evidence*), and the *Relying Party* (the consumer of attestation results). Reliable evidence is a collection of measurements whose origin is exclusively attributable to a trusted entity, such that no untrusted component could have fabricated or tampered with them, and which accurately and completely depicts the measured component's state and integrity. As seen in Section 2.3, this reliability is achieved through a measurement chain in which each component measures the next component *before* transferring control, thereby transferring trust from the root of trust. The root of trust component cryptographically signs the collected measurements, providing a tamper-resistant, non-repudiable endorsement of the results. The verifier validates the trusted component's signature before verifying the measurements themselves, either by comparing them against reference trusted measurement values or by reproducing measurements from the trusted measurable components.

In Figure 2.2, a remote attestation sequence for both AMD's *Secure Processor* and virtual TPM

Figure 2.2: Confidential VM remote attestation sequence



evidence is illustrated. Here, and throughout this thesis, the $+$ and $-$ symbols respectively denote public and private parts of the preceding key pair. Public cloud infrastructures are currently aiming to support platforms that execute these procedures, where the *Secure Processor* endorses the correct and secure configuration of the confidential VM and initial memory contents, and the virtual TPM endorses the VM boot process. Both pieces of evidence require verification to be deemed valid and meaningful. To verify the AMD *Secure Processor* evidence, the *Verifier* needs to check (1) if the provided nonce and public Instance Key (IK) are included in the report and covered by the cryptographic signature; (2) if the Versioned Chip Endorsement Key (VCEK) signature has not been tampered with; (3) if a VCEK certificate endorsed by the AMD SEV Key (ASK) exists and is valid; and (4) if a ASK certificate endorsed by the AMD Root Key (ARK) exists and is valid. The process is analogous for Versioned Loaded Endorsement Key (VLEK) (as an alternative to VCEK), using AMD SEV-VLEK Key (ASVK) instead of ASK. The public key certificates for ASK, ASVK, and ARK are available online[1]. The launch digest included in the attestation report is (ideally) reproducible by simulating the page-loading process, given the binary contents of the initial VM memory. This requires transparent documentation on how the hypervisor loaded the initial memory for the running confidential VM. Similarly, to verify the virtual TPM evidence, the *Verifier* needs to check (1) if the provided nonce and public Instance Key are included in the virtual TPM quote and covered by the cryptographic signature of the Attestation Key (AK); (2) if

[1] `https://www.amd.com/en/developer/sev.html`

the AK signature has not been tampered with; (3) if a AK is co-resident with the EK; and (4) if the EK is endorsed by the entity who manages it (sometimes certificates are used, depending on implementation). Additionally, PCR values are validated at this stage, with some PCRs directly compared against known-good values, while others require full PCR recomposition using the individual measurement log. In the latter, individual measurements are verifiable by themselves, but, typically due to non-deterministic sequencing of the measurements, the PCR is not directly verifiable.

By including the public Instance Key (IK) in the *Secure Processor*'s attestation report and the virtual TPM's quote, the key pair is bound to the state of the confidential VM. If the measurements included are not trustworthy, then the key can be disregarded, as the system should not be used in that case. Conversely, if the measurements are trustworthy, it would be impossible for the attester to include a malicious actor's public key instead, as this would be inconsistent with the expected behavior of the measured software.

Remote attestation in real-world scenarios faces significant challenges due to the complexity of establishing trusted measurement chains across multiple participating entities. The absence of universally accepted reference measurement values aggravates these difficulties, while achieving measurement reproducibility requires extensive transparency from all measurable components. Even when a trusted and reproducible measurement chain is established, selecting a reliable root of trust remains critical for securely maintaining and endorsing attestation measurements.

Trust relationships in secure computing environments exist on a spectrum rather than as binary choices, requiring relying parties to evaluate components based on the relative strength of their security guarantees. For instance, while both AMD's *Secure Processor* and hypervisor-anchored virtual TPM implementations may be deemed trustworthy within a deployment, their attestations carry different weights due to fundamental differences in their security foundations. AMD's *Secure Processor* provides hardware-rooted isolation and other security features that diminish the attack vectors, whereas hypervisor-anchored virtual TPM implementations rely on hypervisor isolation and software-based operations. This graduated trust model enables relying parties to accept attestations from multiple sources, weighting them based on their assurance levels and specific threat models.

Furthermore, for better understanding and comparison, this thesis proposes a software verification depth taxonomy to clarify the different trust levels that may be established for a given software component, each with distinct and incremental security guarantees, based on the *attestation levels* proposed in [5].

L1: *Root of Trust Link Authenticity* At the foundational level, verifiers validate evidence freshness, cryptographic signatures, certificate chains anchored at a trusted certificate authority, and measurement coverage of the specific component. This evidence is endorsed by the root of trust, which ensures the integrity of its contents. However, this level provides no semantic interpretation of the

measurements themselves.

L2: *Reference-Based Measurement Matching* The second level involves comparing reported measurements against known reference values (often referred to as golden measurements). This verification requires access to reference databases or the ability to independently compute expected measurements from known measurable components. At this level, semantic interpretation is limited to what the trusted reference value source can provide. Trust is extended to the entities providing reference values.

L3: *Binary Measurement Reproducibility* At the third level, verifiers possess the actual binary artifacts whose measurements are attested. Verification involves recomputing hashes or other cryptographic measurements locally and comparing them with the reported values. If successful, the verifier has assurance that the binary artifacts in hand are the ones measured remotely and possibly executed (depending on the previously measured trusted component). This provides a larger auditable surface and greater accountability for the binary authors if these binary artifacts were cryptographically endorsed by them.

L4: *Source Code Measurement Reproducibility* The fourth level involves access to source code and build environments, offering the maximum auditable surface and enabling independent compilation and measurement computation. This approach requires comprehensive toolchain verification and reproducible build processes. Trust assumptions are minimized to the compiler, build environment, and measurements.

Summary

This chapter examined the hardware and system-level mechanisms that enable the use of hardware as a root of trust in cloud environments. AMD SEV-SNP protects a guest Virtual Machine's memory from the hypervisor and co-located VMs through a dedicated security coprocessor, which encrypts VM memory with a key unknown to the hypervisor and can generate cryptographically signed attestation reports describing the VM's configuration and launch contents. The Trusted Platform Module extends the measurement chain beyond the firmware by accumulating measurements in hardware-protected registers through a one-way extend operation.

The boot process unfolds in two phases: a pre-boot phase where the *Secure Processor* measures every memory page before the VM executes its first instruction, closing the bootstrapping problem that software-only approaches cannot solve; and a boot phase where each component measures its successor before transferring control, extending those measurements into the virtual TPM. Remote attestation enables a remote verifier to assess the integrity of a confidential VM by evaluating evidence from both sources. To structure this evaluation, a software verification depth taxonomy was proposed, defining four

levels of assurance ranging from root-of-trust link authenticity up to source code measurement reproducibility. Together, these mechanisms form the technical foundation for the trust analysis and framework design presented in the following chapters.

3

Related Work

Contents

This chapter offers a comprehensive overview of the current landscape concerning the use and attestation of confidential virtual machines VMs. It is organized as follows: Section Section 3.1 clarifies, analyzes, and compares the configurations available from major cloud service providers, highlighting existing industry limitations and the threat models associated with each provider. Keeping these limitations in mind, this chapter explores two fundamentally different questions. Section Section 3.2 discusses how these limitations could be addressed, assuming access to the host system. In contrast, Section Section 3.3 examines what can still be achieved even when the host system is not accessible.

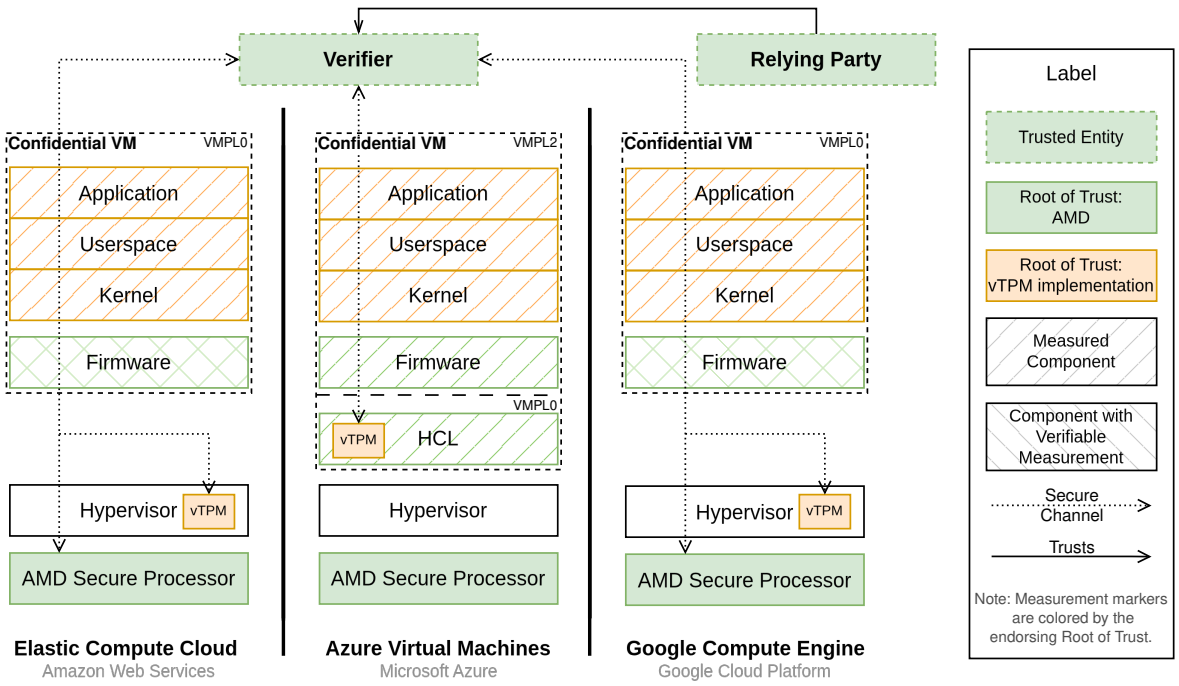
3.1 Confidential Computing Offering to the General Public

This section examines the public cloud offering of AMD SEV-SNP confidential virtual machines from Amazon Web Services, Microsoft Azure, and Google Cloud Platform. The discussion is limited to virtual

machine offerings that are directly relevant to the trust and attestation model introduced in Chapter 2. Other confidential computing primitives, such as process-level enclaves, alternative processor technologies, bare-metal deployments, or managed confidential container platforms, are outside the scope of this section because they expose different roots of trust and different operational assumptions.

The comparison is organized around the properties that most directly affect the meaning of an attestation result. Among these, reproducibility of measurements is the central evaluation lens, because it determines whether a verifier can independently interpret evidence or must instead accept third-party-issued claims about what was measured. Figure 3.1 summarizes the entities, the components, the relationships and interactions involved in Remote Attestation Procedures. Table 3.1, within cloud deployment confidential Virtual Machine (VM) configurations, shows the different trust anchors for each measured software component.

Figure 3.1: Component measurement chain at public cloud providers



All three providers offer AMD SEV-SNP-based confidential VMs and present a similar high-level security promise: memory confidentiality and integrity against privileged software outside the guest. In all three cases, customers may instantiate a machine from a provider-supplied image or upload their own image [18–20]. From the perspective of this dissertation, provider-supplied images weaken the purpose of confidential VMs, because the customer must trust the provider to supply the advertised image. Uploading a custom image built in a trusted environment is therefore the stronger starting point across all platforms, although it still does not prevent tampering before instantiation. A further common

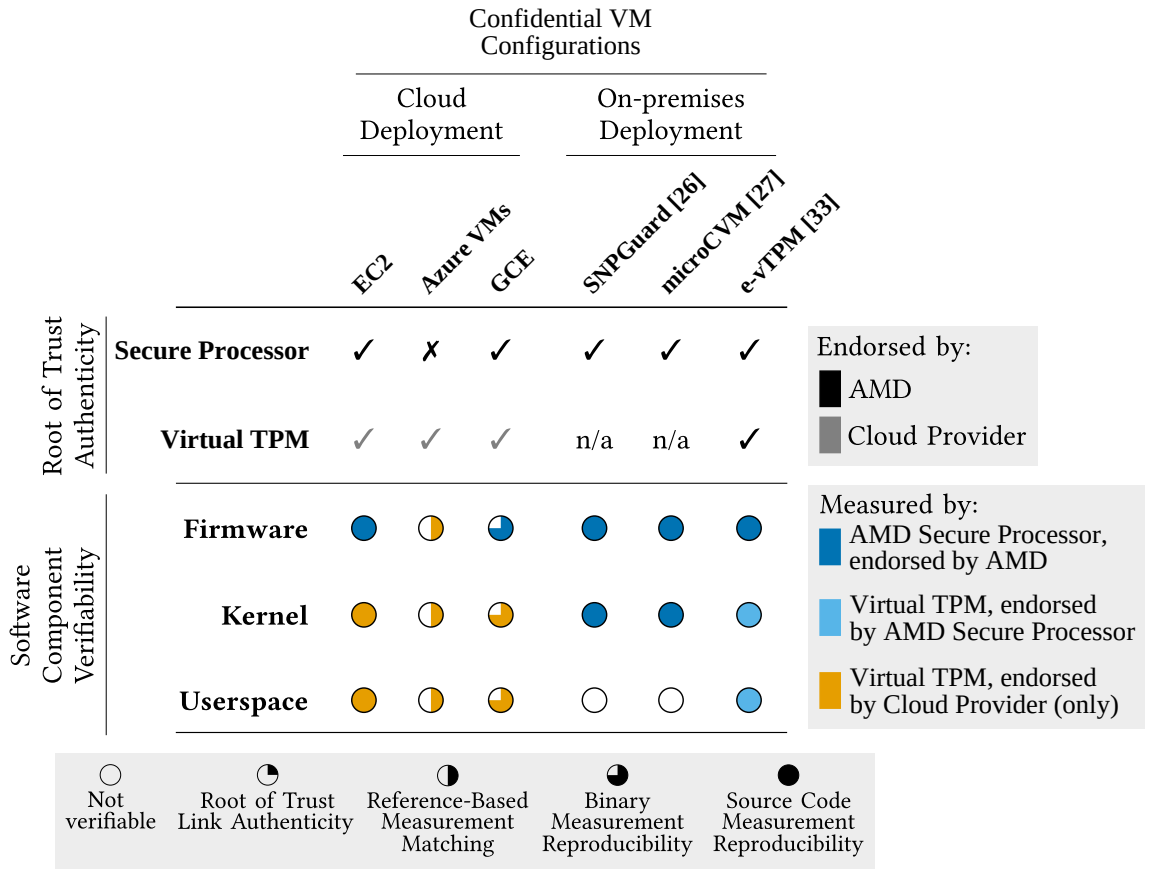
limitation is that the firmware is not part of the customer-supplied image and cannot be customized by the customer, which leaves a critical early boot component under provider control in all three offerings.

The first major point of divergence concerns firmware measurement transparency. In (Amazon Web Services') Elastic Compute Cloud, the AMD *Secure Processor* measurement covers the firmware, and the provider publishes both the firmware source and the measurement procedure (as source code) to make the resulting launch measurement reproducible in practice [21]. This is a strong property because the verifier can know both what was measured and how the measurement was derived. In (Google Cloud Platform's) Google Compute Engine, the measured scope is similarly well defined, covering the firmware, and the endorsed firmware binary can be obtained through Google's public interfaces, which makes the measured content accessible to the verifying party. Google has also made progress on disclosing the measurement derivation process, although the currently available implementation still requires adaptation before it becomes a convenient customer-facing workflow [22]. Azure Virtual Machines differ fundamentally at this point. The launch measurement includes the firmware and a special component called Host Compatibility Layer, which runs before the firmware, at a separate, higher virtual machine privilege level, isolated from the lower ones. The Host Compatibility Layer (HCL) is proprietary, the measurement methodology is opaque, and the measured contents are not available to verifier, therefore, the verifying party cannot reproduce the AMD *Secure Processor* launch measurement. In practice, this reduces the measurement to a provider-defined value whose interpretation depends on prior trust in Azure [6].

These differences have direct consequences for verification depth. For Elastic Compute Cloud, the documented measured contents and disclosed derivation procedure make it possible to move beyond basic signature validation and reach source code measurement reproducibility, L4 as defined in Section 2.4. Google Compute Engine achieves L3, as the firmware's source code is not disclosed, but the measured binary contents are. Azure Virtual Machines, by contrast, does not disclose any necessary artifacts to reproduce and verify the measurement. Additionally, the guest can only fetch a single AMD *Secure Processor*'s attestation report previously generated at launch, without any reliable proof of freshness. Consequently, the HCL and firmware components on Azure Virtual Machines do not qualify to any verification depth level, as the link to the root of trust is compromised.

The second major point of divergence concerns the virtual TPM used to continue measurement beyond launch. In Elastic Compute Cloud and Google Compute Engine, the virtual TPM is implemented at the hypervisor level and remains closed-source [23]. As a result, its state and quotes are controlled by provider-managed software outside the confidential VM. In Azure Virtual Machines, the situation is different in architecture, because the virtual TPM is provided through the HCL, which runs inside the guest context and benefits from hardware-enforced isolation from the remaining VM components through virtual machine privilege levels, while also relying on AMD SEV-SNP to isolate it from the hypervisor [24].

Table 3.1: Root of trust authenticity and component verifiability summary for different confidential VM configurations



From a design perspective, this is the strongest of the three architectures for excluding the cloud provider from the trusted computing base. However, because the HCL implementation and its measurement procedures remain closed, that architectural advantage does not by itself produce independently verifiable evidence. In other words, Azure Virtual Machines appears closest to the right structure, but the absence of transparency prevents the verifier from converting that structure into a strong assurance result [6].

This distinction is important because the virtual TPM is the mechanism that could extend the trust chain from firmware into the operating system kernel and userspace software. If the link from the virtual TPM to the *Secure Processor* cannot be proven, its quotes do not provide the same assurance as the AMD-rooted attestation reports. For Elastic Compute Cloud and Google Compute Engine, trusting the virtual TPM, therefore, requires including the provider's closed hypervisor-level TPM implementation into the trusted computing base. For Azure Virtual Machines, a similar conclusion follows for a different reason. The virtual TPM is better placed architecturally, but because its implementation is closed and its measurement chain cannot be independently reproduced, meaningful reliance on its quotes still requires trust in Azure's statements about the HCL and its endorsement path.

Across all three providers, the AMD *Secure Processor* is the only independent hardware root of trust available. Its report can establish the authenticity of the confidential VM configuration and its initial measured contents, but it does not, by itself, provide a measurement chain for later boot components. None of the studied platforms extends the AMD *Secure Processor*-rooted measurement directly through the full boot chain, and none allows the customer to customize the provider-controlled firmware that would be needed to change this situation. As a consequence, the verifier cannot derive trustworthy claims about the operating system kernel or the application stack solely from the AMD *Secure Processor*. This is the central practical limitation of current public offerings.

The verification depth level of a given component is based on its own verification depth level, but also on its preceding component's verification depth level. A compromised firmware, for example, may measure a correct kernel, but instead load a compromised, unmeasured kernel. For that reason, even if the kernel, by itself, would be verifiable at L4, the practical verification depth level will never be higher than the preceding component's counterpart — in this case, the firmware. This is why, in Table 3.1, components' verifiability depth levels are maximized by the firmware's.

The providers also differ in how attestation evidence is exposed to the verifying party. In Elastic Compute Cloud, attestation reports are signed using a Versioned Loaded Endorsement Key (VLEK), rather than the chip-specific Versioned Chip Endorsement Key (VCEK). As covered in Section 2.4, both are rooted in AMD Root Key, but VLEK allows the provider to hide the identity of the underlying chip, which should not have any security implications. In Azure Virtual Machines, direct access to the AMD *Secure Processor* device interface is blocked. Instead, an attestation report is generated once at launch and stored within the HCL. Although the report remains cryptographically verifiable, this design weakens freshness because the verifier cannot request a new report with a verifier-chosen nonce. This results in a narrower assurance statement than one would obtain from an on-demand report, since an otherwise valid report may be replayed after the fact. This issue makes the verifying party unaware whether the fetched attestation report is from the target VM. Furthermore, this issue makes Azure the only provider's solution where the verifier cannot conclude that memory confidentiality and integrity protections are in place.

Operationally, confidential VM support is usually offered as a low- or no-cost[1] enablement option for selected instance families. Elastic Compute Cloud and Google Compute Engine offer a relatively limited geographical choice for confidential VM deployment[2].

The current public offerings support two configurations with different security assumptions. In the

[1] Pricing information may be found at `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html#snp-pricing`, `https://azure.microsoft.com/en-us/pricing/details/virtual-machines/linux/#pricing`, and `https://cloud.google.com/confidential-computing/confidential-vm/pricing` for Elastic Compute Cloud, Azure Virtual Machines and Google Compute Engine, respectively.

[2] Information regarding geographical limitations may be found in `https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html#snp-requirements`, `https://azure.microsoft.com/en-GB/explore/global-infrastructure/products-by-region/table`, and `https://docs.cloud.google.com/compute/docs/regions-zones#available` for Elastic Compute Cloud, Azure Virtual Machines and Google Compute Engine, respectively.

first, the relying party deems the cloud provider as untrustworthy and relies solely on the AMD *Secure Processor* as a trust anchor. Under that model, the relying party obtains strong assurance about the confidential VM's initial measured state, but no assurance about the software loaded after firmware. In this variant, confidential VMs hosted by Elastic Compute Cloud and Google Compute Engine are more verifiable than the ones under Azure Virtual Machines. In the second trust model, the relying party also accepts the provider-managed virtual TPM as a trust anchor. This possibly extends attestation coverage to later boot components and provides a more useful and complete integrity guarantee on the VM's contents, but it does so by reintroducing cloud-provider-managed components into the trusted computing base. The result is an equilibrium point that has a stronger operational security posture against VM content-tampering attacks, yet a weaker realization of the original confidential VM objective, which is to reduce or remove the cloud provider from the trusted computing base.

Among the three providers, none currently has a public offering that fully resolves the tension between attestation depth and provider exclusion. Elastic Compute Cloud currently offers the most verifiable launch measurements for AMD *Secure Processor*, and Google Compute Engine follows closely, except for the lack of source code transparency for the firmware. Azure Virtual Machines currently provide the least verifiable measurements, but its architectural direction is important. Moreover, *OpenHCL* [25], an open-source implementation of the HCL role, is in active development and aims to substitute the HCL in confidential VMs. This change, together with transparent launch measurement derivation and fresh AMD-rooted attestation report generation, could potentially turn Azure's solution into the most deeply verifiable one across all VM components. For the moment, however, all three providers leave a substantial part of the boot chain outside a fully independent and reproducible trust path. Section 3.2 explores different solutions that could be made possible if customers could freely configure the confidential VM, which is also the case with on-premise solutions. Section 3.3 covers existing solutions compatible with trust models that accept the provider's implementation of the virtual TPMs as a trusted component.

3.2 On-Premises Confidential Computing Solutions

The solutions discussed in this section address a limitation identified in Section 3.1: current public confidential VMs offerings stop the independently verifiable chain of trust too early in the boot process. Several academic and industrial proposals attempt to extend that chain without requiring the relying party to trust a provider-managed virtual TPM implementation. Their common feature, however, is that they configure the hypervisor behavior, launch configuration, or firmware control that customers cannot replicate in the covered public cloud environments. For that reason, these proposals are best understood as on-premises solutions, or as design directions that cloud providers could adopt, rather than as mechanisms that users can reproduce today on Amazon Web Services, Microsoft Azure, or Google

Cloud Platform.

The comparison again uses reproducibility of measurements as its primary lens, because the central question is whether the verifier can independently determine what was measured and how that measurement entered the attestation chain. This framing recalls the verification-depth taxonomy introduced in Section 2.4, where stronger assurance requires more authentic evidence and depends on the ability to interpret, reproduce, and audit the measured artifacts. Table 3.1 summarizes this difference by comparing the verification depth that can be achieved in current cloud solutions, assuming the provider's virtual TPM implementation is trusted, with the depth reached by the on-premises proposals examined below.

These proposals pursue the common objective of excluding the cloud provider from the trusted computing base while preserving meaningful attestation beyond the earliest launch stage. They do so in different ways. Some avoid the virtual TPM entirely and instead enlarge the set of components measured by the AMD *Secure Processor*. Others keep a virtual TPM, but relocate it to an execution context that is itself measured by the AMD *Secure Processor* and isolated from both the hypervisor and the guest operating system.

SNPGuard [26] not only binds the firmware to the AMD *Secure Processor* measurement (as cloud provider solutions do), but also the Linux kernel, kernel command-line arguments, and the initialization process — enough boot components that enable a completely verifiable minimal operating system with limited capabilities. It requires the hypervisor to hardcode the measurements of these components in the firmware's binary, which are then loaded into the confidential VM and, therefore, measured by AMD's *Secure Processor*. The firmware then performs a custom boot procedure where it loads each component, measures it, and compares the measurement with the binary-included one. This solution is feasible since the verifying party has access to the correct firmware and the extra measured components, so they can reproduce the expected measurement. If the attestation report has the expected measurement, the guest owner knows that the correct firmware was loaded and the correct extra components' measurements were inserted. Because the firmware is correct, it will compare the extra components' measurements before transferring execution to them; otherwise, the firmware would abort the boot process. Upon a successful boot process, and the inclusion of the expected measurement in the AMD-backed attestation report, the verifier has attested the firmware, kernel, kernel command-line arguments, and the initialization process, with a single hardware root of trust — the AMD *Secure Processor*. This solution does not use a TPM or virtual *TPM* device.

Decentriq[3] documented another way to completely mitigate all the hypervisor's unmeasured inputs on the boot process [27]. Instead of deploying full, traditional confidential VMs, they figured that microVMs (popularized by the Firecracker project[4], introduced by A. Agache *et al.* [28]) had one interesting

[3] `https://www.decentriq.com/`
[4] `https://firecracker-microvm.github.io/`

implementation consequence: hypervisor isolation; therefore, the hypervisor's inputs were minimal compared to those of traditional VMs. According to M. Misono *et al.* [29], the hypervisor boot input types on traditional confidential VMs are summarized as firmware code, firmware variables, device tables, device drivers, operating system loader, kernel code, and kernel command-line parameters, which should be reliably measured by AMD *Secure Processor* or a TPM. With micro confidential VMs (microCVMs), the *Secure Processor* initially loads and measures a complete package, containing the firmware, initialization process, kernel, kernel command-line arguments, and processor and memory layouts. Given that all the components used are open source, the measurement becomes fully reproducible. Although the microCVMs solution makes hypervisor inputs easier to measure, they lack operating system flexibility, support for hardware device passthrough, and live-migration capabilities [30]. MicroVMs were designed and implemented to host Function-as-a-Service workloads, assuming guests were ephemeral and short-lived. Therefore, they might not currently be fit to sustain many use cases out of the box when turned into microCVMs.

As seen in Azure's confidential VM implementation, virtual machine privilege levels may be used to offer an environment for privileged code modules to execute, isolated from both the hypervisor and the guest operating system. To address potential implementation disparities, AMD has published the Secure Virtual Machine Service Modules (SVSM) specification [31] for standard communication between the guest operating system and the collection of privileged services. The SVSM enables, among other things, the secure and isolated execution of a virtual TPM accessible to all guest processes. For the TPM measurements to remain relevant, the virtual TPM software and the surrounding environment need to be measured by the AMD *Secure Processor* at launch. This is the recommended behavior in the specification, as it indicates that the entire SVSM should be measured, including the virtual TPM [31]. With the SVSM, additional services may be developed independently, as the interface is already specified. Since the guest operating system only needs to use the SVSM interface, remote attestation procedures may be further standardized and broadly used without requiring many changes in both the provider and user-managed components. A popular ongoing project, CoCoNut-SVSM [32], is actively moving forward into building services and standardizing remote attestation across different platforms.

V. Narayanan *et al.* [33] achieved a complete virtual TPM-focused implementation of the SVSM, where they execute a virtual TPM with ephemeral state (e-vTPM) inside the highest virtual machine privilege level, isolated from the guest operating system running in lower privilege levels, with no other additional services. As recommended by the SVSM specification, their solution effectively links the virtual TPM trust anchor to AMD *Secure Processor* by measuring all SVSM components upon boot. Unfortunately, this design is not reproducible in current public cloud environments since the virtual machine privilege levels are static and defined by the hypervisor. As seen in Section 3.1, a customer is also unable to use customized firmware, include a SVSM to precede the firmware at launch, or modify initial

memory measurement sequences.

3.3 Privacy-Preserving Systems and Services

The previous sections examined the attestation properties of public cloud confidential VM offerings and the on-premises proposals that extend the measurement chain beyond firmware. This section turns to the systems and frameworks that build on top of these offerings, aiming to make confidential VMs usable for privacy-sensitive workloads. The comparison uses reproducibility of measurements and verification depth as its central lens, consistent with the preceding sections. An attestation system is only as strong as the independence of its reference values and the transparency of its verification logic.

As established in Section 3.1, the virtual TPM is the only mechanism currently available to extend attestation beyond the firmware in public cloud environments. Trusting its implementation is a bounded assumption about one specific component, not a full endorsement of the cloud provider. The systems reviewed in this section that operate on public cloud infrastructure all depend on this assumption, whether they acknowledge it or not. The relevant question is how transparently each system communicates the resulting trust model, and how effectively it uses the available evidence.

Constellation. Edgeless Systems[5] developed Constellation [4], a platform for deploying Kubernetes[6] clusters inside confidential VMs on public cloud infrastructure, using AMD SEV-SNP and Intel Trusted Domain Extensions. The platform built a custom operating system based on Fedora[7], packaged it with an Unified Kernel Image, and used a read-only root filesystem protected by *dm-verity* for integrity verification. Attestation evidence was integrated into the TLS handshake through a mechanism called attested TLS, where endorsed artifacts were exchanged during connection establishment rather than through a separate protocol. Constellation has since been deprecated and replaced by Contrast [34], discussed later in this section.

Constellation is the most directly comparable system to the work presented in this thesis, as it targeted the same operational context: confidential VMs deployed on public cloud providers. Its attestation properties, however, fell short of what the underlying hardware could support. The platform did not independently reproduce AMD *Secure Processor* launch measurements. Its documentation acknowledged that measurement reproducibility was limited, but did not make explicit what verification depth was actually achieved or what trust assumptions the user implicitly accepted with each use. Through analysis of the project's source code, the verification depth levels attainable by Constellation were classified as shown in Table 3.2, revealing significant gaps relative to what the current cloud infrastructure makes possible.

[5] `https://www.edgeless.systems/`
[6] `https://kubernetes.io/`
[7] `https://fedoraproject.org/`

Like most systems in this space, Constellation designed the verifier as a consumer of externally provided expected measurements rather than as a party that derives its own. The expected measurements were embedded in the platform's release artifacts, and the user had to trust Edgeless Systems to have produced them honestly. This is a meaningful trust assumption that the platform did not clarify. Furthermore, Constellation provided no tooling for generic VM image construction or measurement prediction. Its build system produced a fixed operating system image tailored to Kubernetes workloads, and the user had no support path to substitute a different workload without also adopting the platform's orchestration layer.

Keylime. Keylime [3] is an open-source framework developed at MIT Lincoln Laboratory that automates remote attestation using TPM devices. It was originally designed for hardware TPMs on physical machines, but its architecture generalizes to any system exposing the TPM 2.0 interface, including cloud instances with virtual TPMs. The framework operates through an agent on each attested machine that collects measurements from the TPM, a verifier that evaluates those measurements against predefined policies, and a registrar that manages enrollment of key credentials.

Keylime provides a production-ready attestation pipeline, but when applied to confidential VMs on public cloud infrastructure, several gaps emerge. It does not recognize or process AMD *Secure Processor* attestation reports, so any assurance it provides rests entirely on the virtual TPM. A verifier using Keylime in a cloud environment has no mechanism to independently verify the confidential VM's launch measurement or to confirm that the virtual TPM is itself anchored to the hardware root of trust. More fundamentally, Keylime does not provide a mechanism for deriving expected measurements from the source artifacts. The verifier must obtain reference measurement values from an external source, and the framework offers no way to assess whether that source is honest. If the reference values are supplied by the same party that controls the VM image, the verification becomes circular. Finally, Keylime does not address the lifecycle stages that precede attestation: it assumes instances are already running and provisioned with its agent, offering no tooling for image construction, measurement prediction, or deployment management.

Bare-metal and application-specific systems. Several systems operate on bare-metal confidential VM instances rather than the publicly accessible confidential VMs offered by cloud providers. This distinction matters because bare-metal deployments allow control over firmware, virtual machine privilege levels, and the hypervisor configuration, removing several of the constraints that shape the attestation problem on public cloud infrastructure. These systems are worth examining for the techniques they introduce, but they operate on fundamentally different ground from the public cloud context that this thesis addresses.

Contrast [34], also developed by Edgeless Systems as Constellation's successor, deploys confidential containers on Kubernetes using bare-metal AMD SEV-SNP and Intel Trusted Domain Extensions

instances. Each workload pod runs inside its own confidential VM, and a centralized *Coordinator* verifies hardware attestation reports against a manifest of expected policy hashes. Contrast does not use virtual TPMs; it relies directly on CPU-level attestation reports. Its reference values and pod-VM images are reproducibly built using *Nix* and *mkosi* (tools will be covered in Section 5.2), and users can rebuild from source to verify the embedded measurements. This represents one of the stronger reproducibility approaches among current systems. However, because Contrast targets bare-metal Kubernetes deployments, its attestation model does not contend with the provider-controlled firmware and virtual TPM constraints that define the public cloud setting. The recommended production deployments use bare-metal nodes on providers that offer them, with managed cloud support available only in preview.

Trustee [35] is the client-side attestation evidence verification and secret delivery infrastructure for the Confidential Containers project, a Cloud Native Computing Foundation initiative. It provides the machinery for verifying that a workload runs inside a genuine Trusted Execution Environment (TEE) before releasing secrets to it, following the Remote Attestation Procedures model. Trustee supports the broadest range of hardware platforms among the surveyed systems, including AMD SEV-SNP, Intel Trusted Domain Extensions, Intel Software Guard Extensions, ARM Confidential Compute Architecture, and others. Like Contrast, Trustee is designed primarily for bare-metal and self-managed infrastructure, where the operator controls the full stack from firmware through orchestration.

Trustee's architecture separates attestation verification from reference value provisioning. A Key Broker Service handles secret delivery, an Attestation Service verifies hardware evidence against policies, and a Reference Value Provider Service stores pre-provisioned expected measurements. This separation is clean, but it exposes the same limitation seen in Keylime: Trustee does not independently reproduce measurements. It compares TEE evidence against reference values that the workload operator must supply, and the quality of the attestation result depends entirely on the provenance of those values. The Reference Value Provider Service supports provenance formats that can provide cryptographic assurance about reference value origins, but it does not address the prior question of how those values should be derived from the actual measured artifacts. Trustee's default installation ships with empty reference values, meaning attestation will fail until an operator manually populates them. This design is architecturally intentional, as Trustee's generic infrastructure is meant to serve any workload, but it pushes the hardest problem in confidential VM attestation entirely to the operator.

Confer[8], created by M. Marlinspike, applies confidential VMs to private artificial intelligence inference. It runs open-source large language models inside Intel Trusted Domain Extensions-based confidential VMs with NVIDIA encrypted-memory graphics cards, embedding attestation evidence directly into the Noise [36] protocol handshake that establishes encrypted client sessions. The VM image is reproducibly built with *Nix* and *mkosi*, and signed measurements are published to a Sigstore[9] transparency log for

[8] `https://github.com/conferlabs/confer-image`
[9] `https://docs.sigstore.dev/`

public verification. Confer achieves L4 verification depth for the core system components (kernel, early initialization process, root filesystem), but does not require users to independently verify those measurements or rely on others to do so. The client application performs a shallower verification automatically during the handshake, which favors usability over transparency: the user trusts that the client correctly evaluated the evidence without an independent path to assess what was verified. Additionally, documentation left the origin of the model's weights unaddressed and its corresponding attestation unclear. As a single-application hosted service rather than a general-purpose framework, Confer illustrates how confidential VM attestation can be integrated into a consumer-facing product, but it does not address the generic deployment and verification problem.

Table 3.2: Summary of practical trust models given the verifiability levels on confidential VM deployments, trusting on AMD *Secure Processor* and the virtual TPM implementation

		Keylime [3]			Constellation [4]			Thesis Target		
		AWS	**Azure**	**GCP**	**AWS**	**Azure**	**GCP**	**AWS**	**Azure**	**GCP**
Secure Processor		–	–	–	✓	✗	✓	✓	–	✓
Virtual TPM		✓	✓	✓	✓	✓	✓	✓	–	✓
Firmware	**L1**	●	●	●	●	●	●	●	n/a	●
	L2	●	●	●	●	●	●	●	n/a	●
	L3	○	○	○	○	○	○	●	n/a	●
	L4	○	○	○	○	○	○	●	n/a	○

✓: Verifiable authenticity ✗: Non-verifiable authenticity –: Not used
○ : Not verifiable
● : Verifiable with virtual TPM root of trust
● : Verifiable with AMD *Secure Processor* root of trust

AWS: Elastic Compute Cloud, a product from Amazon Web Services
Azure: Azure Virtual Machines, a product from Microsoft Azure
GCP: Google Compute Engine, a product from Google Cloud Platform

Proprietary Solutions

Several commercial products operate in the confidential computing space, including *Anjuna*[10], *BeeKeeperAI*[11], *Decentriq*[12], and Opaque[13], among others. Each deploys workloads inside confidential VMs on public cloud infrastructure and advertises security guarantees around data confidentiality and attestation. Because these solutions are closed-source, an independent assessment of their attestation depth is not possible. Their security claims are subject to the same infrastructure limitations documented throughout this chapter, but unlike the open-source alternatives, they cannot be audited to determine whether those limitations are addressed or merely obscured. The closed nature of these solutions represents a redistribution of the trusted computing base rather than a reduction: the user must trust the solution vendor in addition to the cloud provider, without the ability to verify the vendor's claims independently.

Summary

This chapter examined the attestation landscape for AMD SEV-SNP confidential VMs across three dimensions: public cloud provider offerings, on-premises research proposals, and existing systems and frameworks. Table 3.1 summarizes the analysis conclusions on root of trust authenticity and software component verifiability. Table 3.2 compares the concretization of this thesis's software verification level for the firmware and the most comparable systems.

Among public cloud providers, Elastic Compute Cloud currently offers the most independently verifiable launch measurements, reaching source-code reproducibility (L4) for the firmware. Google Compute Engine follows at binary reproducibility (L3). Azure Virtual Machines does not disclose the artifacts or methodology needed to reproduce the launch measurement, and restricts the guest to a single pregenerated attestation report without verifier-controlled freshness. All three providers rely on closed-source virtual TPM implementations to extend measurements beyond the firmware, and none allows the customer to substitute the firmware or virtual TPM. The result is a choice between trusting only the AMD *Secure Processor* (strong assurance limited to firmware) or additionally trusting the provider's virtual TPM implementation.

On-premises proposals such as SNPGuard [26], *Decentriq*'s micro confidential VMs [27], and the ephemeral virtual TPMs proposed by V. Narayanan *et al.* [33] demonstrate that the hardware-rooted trust chain can be extended through the full boot stack, but all require firmware or hypervisor-specific configuration that public cloud customers do not have.

The surveyed systems and frameworks focus on the remote attestation protocol, providing infrastruc-

[10] `https://www.anjuna.io/`
[11] `https://www.beekeeperai.com/`
[12] `https://www.decentriq.com/`
[13] `https://opaque.co/`

ture for collecting evidence and comparing it against expected values. With the exception of Contrast [34] and Confer, which achieve reproducible builds for their own specific images, there is no focus on how expected measurements should be derived from source artifacts for arbitrary workloads. Trustee [35] and Keylime [3] treat reference values as an external input whose provenance is the operator's responsibility. Constellation embedded reference values in its release artifacts, requiring trust in the platform vendor. The verification depth that each system achieves is therefore limited by the absence of tooling to independently predict measurements from source artifacts for a generic workload. A verifier using any of these systems for a custom deployment must either trust a third party's reference values or accept the measurements observed on first deployment as the baseline, neither of which provides independent assurance that the confidential VM trust model is designed to support.

4

Evident Framework

Contents

This chapter introduces *Evident*, a framework designed to automate the trust-enabling operations surrounding the lifecycle of a confidential VM on public cloud infrastructure. Its purpose is to make confidential VMs a verifiable platform for arbitrary workloads, without requiring modifications to the workload itself. The design approach is different from the alternatives surveyed in Section 3.3 when dealing with expected measurements. In *Evident*, the verifying party derives its own expected measurements directly from the artifacts that constitute the VM image, rather than consuming reference values supplied by a third party. Keylime [3], Constellation [4], Trustee [35], Contrast [34], and Confer all design the verifier as a consumer to externally provided measurements, implying a trust assumption that the framework

is designed to eliminate. The framework is workload-agnostic and cloud-agnostic at the architectural level, using custom-built VM images as its necessary primitive. Provider-specific and hardware-specific behavior is confined to well-defined modules, so support for additional platforms can be added without altering the core design.

The chapter begins with the threat model under which the framework's attestation results are meaningful (Section 4.1, then identifies the principal technical challenges that shaped the design (Section 4.2). Section 4.3 presents the framework's architecture, its two components, and the protocol that connects them. Section 4.4 covers the design of the image assembly and deployment stages, including the image distribution extension for endorsed, reproducible VM image packages. Section 4.5 specifies the attestation protocol and the attestation certificate mechanism that binds instance keys to measured state, enabling secure communication channels anchored in attestation results. Section 4.6 describes a scalability extension built on hierarchical certificate authorities, each running inside an attested confidential VM, that allows attestation to scale beyond individual verification of every instance.

4.1 Threat Model

The *Evident* framework operates under a threat model that reflects the confidential VM deployment scenario on public cloud infrastructure and defines the assumptions under which the framework's attestation results are meaningful.

Adversary model. The hypervisor may be adversarial: it may attempt to observe, modify, or replay the guest's memory and execution state. Co-located VMs may also be adversarial and may attempt side-channel or direct memory access against other guests. The cloud provider, as the operator of the hypervisor and the surrounding infrastructure, is not assumed to be benign. Administrative access to the host, the storage backend, and the network fabric are all considered available to the adversary. The network is assumed to be susceptible to attacks on both confidentiality and integrity.

Specification-level assumptions. The threat model assumes that the hardware and cryptographic specifications underlying confidential VM are sound. The memory encryption and integrity protections specified by AMD SEV-SNP are assumed to function as documented. The TPM 2.0 specification is assumed to correctly protect the confidentiality and integrity of its internal state. These are assumptions about the correctness of the technology specifications.

Trusted components. The threat model accepts AMD SEV-SNP as a trustworthy implementation of confidential VM mechanisms, as specified in AMD's SEV-SNP whitepaper [1]. This means accepting that AMD's hardware behaves as specified, that its firmware is correct, that its manufacturing and key provisioning processes have not been compromised, and that its certificate authority faithfully endorses AMD *Secure Processors* exclusively. The AMD *Secure Processor* is therefore the primary trust an-

chor. Its attestation reports are meaningful, and the launch endorsement it endorses is accepted as an accurate record of the VM's initial memory contents.

The threat model also accepts the cloud provider's virtual TPM implementation as a trustworthy realization of the TPM 2.0 specification, resistant to attacks on its integrity and confidentiality by any actor, including the cloud provider. This is a bounded assumption about one specific component, not a general assumption on the cloud provider's behavior. It means accepting that the virtual TPM correctly manages the confidentiality and integrity of its private keys, preserves the integrity of the measurements held in its PCRs, and behaves according to the TPM specification. With this assumption, components such as the bootloader, kernel, and filesystem, may be measured and the full boot sequence of a confidential VM verified.

The *Evident*'s attestation results should be interpreted taking these two assumptions. If the AMD *Secure Processor* assumption is removed, no attestation report is meaningful, and the framework provides no security guarantees. If only the virtual TPM assumption is removed, the attestation conclusions are limited to the VM firmware integrity. The framework is designed for the case where both assumptions hold.

Additionally, attacks that subvert the silicon design, compromise the manufacturing supply chain, or break the underlying cryptographic primitives are outside the scope.

In-guest threats. The threat model does not consider attacks executed by components that are legitimately part of the VM image. It does consider the possibility that malicious components are included unexpectedly, for example, through tampering during image handling on the cloud provider's premises. Detecting such tampering is one of the purposes of the remote attestation workflow.

Relying party responsibility. The threat model defines the assumptions under which *Evident* operates, but it does not mandate what any particular relying party must accept. For instance, a relying party that does not accept the virtual TPM implementation may still use the framework and draw conclusions from the AMD *Secure Processor* evidence alone, understanding that the resulting assurance covers only the firmware. A relying party that does not accept AMD's hardware at all will find no actionable output from the framework. The success or failure of a remote attestation, in the sense of whether the relying party acts on it, ultimately depends on whether the relying party's own judgment aligns with the framework's threat model. The *Evident* Client collects and reports all available evidence unconditionally; how that evidence is interpreted is the relying party's decision.

Evidence collection beyond the threat model. A relying party that does not fully accept the framework's threat model may still find the collected evidence useful. Virtual TPM measurements, even when the relying party is uncertain about the TPM implementation's trustworthiness, can serve as a practical differentiator between otherwise equivalent deployments: a deployment where all TPM-reported measurements align with expected values is at minimum consistent with a correctly booted system, and a

mismatch signals a problem regardless of whether the TPM is fully trusted. The evidence may also serve third parties with different threat models, such as data owners or regulatory bodies who accept the virtual TPM assumption even if the VM operator does not. Finally, trust assumptions change over time. A provider may later publish independently verifiable proof that its virtual TPM operations were performed correctly, at which point historically gathered evidence becomes meaningful to parties who previously withheld judgment. That option is preserved by collecting the evidence now. The cost of gathering and processing is explored in Chapter 6.

4.2 Challenges

The *Evident* framework was designed in a scenario where several practical obstacles must be overcome before attestation can produce meaningful results. This section identifies the most significant challenges that shaped the framework's design. Chapter 5 addresses each of them in detail.

Reproducing AMD *Secure Processor* launch measurements. As established in Chapter 2, reproducing the launch digest requires knowing both the measured contents and the exact sequence in which memory pages were loaded into the confidential VM. The availability of these two pieces of information varies substantially across providers. Some providers publish enough artifacts and documentation for the measurement to be independently derived; others disclose the measured binary but not the full derivation procedure; and one discloses neither. The framework must accommodate this spectrum, producing reproducible predictions where the inputs exist and clearly reporting where they do not.

Reproducing virtual TPM Platform Configuration Register values. Reproducing a Platform Configuration Register (PCR) value requires knowing which components were measured, in what order, and into which register each measurement was extended. The TPM specification recommends particular PCR assignments for particular components, but these recommendations leave room for implementation variance. The same component may be measured more than once across different PCRs, and a single PCR may accumulate measurements from multiple components. Combined with the absence of strict standards and the variability introduced by different firmware, bootloader, and operating system combinations, PCR reproducibility becomes a significant implementation challenge. The framework addresses this by constraining the build environment to produce deterministic, reproducible VM images whose boot-time measurements can be predicted before deployment.

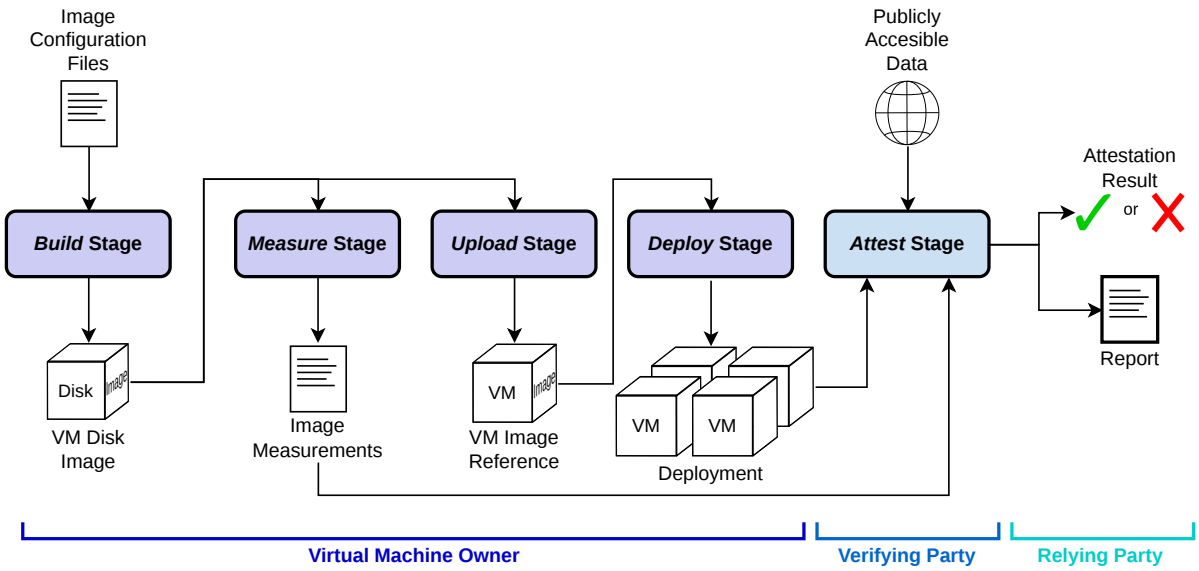
Filesystem immutability as a trust boundary reduction. The VM image, once uploaded to a cloud provider, is under the provider's control. If the root filesystem were writeable, a malicious provider could inject files into the image before launch, and such modifications might not be captured by the virtual TPM's boot measurements. The framework must therefore ensure that the filesystem state is fully determined at build time and that any post-build modification is detectable through the measurement

chain, so that only the virtual TPM implementation is trusted, rather than also the provider's handling of the VM image.

4.3 Architecture

This section describes the framework's component structure, the protocol that connects its two components, and the design priorities that guided each. The lifecycle stages and extensions introduced in subsequent sections all build on the architecture presented here.

Figure 4.1: *Evident* confidential VM lifecycle management stages



The framework addresses five lifecycle stages, illustrated in Figure 4.1: the *build* stage packages the workload inside a VM image; the *measure* stage analyzes the image and derives the expected measurements; the *upload* stage prepares the image for instantiation on a supported cloud provider; the *deploy* stage instantiates confidential VMs across one or more providers; and the *attest* stage verifies actual measurements against the expected ones.

The framework has two main components: the *Evident* Server and the *Evident* Client. The *Evident* Server runs within the confidential VM, where it supports remote attestation by collecting evidence from trusted components (the AMD *Secure Processor* and the virtual TPM). The *Evident* Client runs on the environments of the interested parties and is organized in several stages: *build*, *measure*, *upload*, *deploy*, and *attest*. Users decide which stages they use based on their role. Each stage has well-defined inputs and outputs that are meaningful even outside of the framework's context, making each stage implementation replaceable, provided that it supplies the expected inputs to the dependent stages.

Whenever possible, the implementation delegates functionality to established and auditable tools and libraries. This choice reduces self-containment, but may increase user adoption by reducing the amount of additional software that must be trusted.

Evident is designed to not require any workload modification. The framework operates exclusively on the VM image and its customization; it does not instrument, intercept, or depend on the behavior of the included workload. The VM customization is fully declarative and auditable. Users are expected to take the VM image configuration and use it as a minimal base onto which new additions, specifically the workload, can be merged. These VM image configuration files also include the installation of the *Evident* Server.

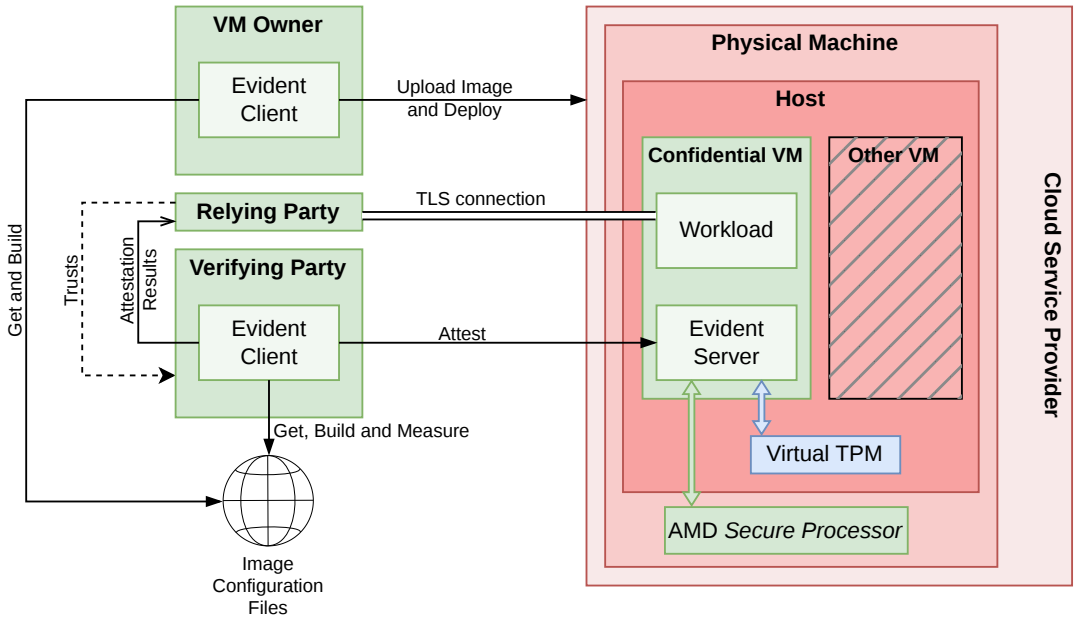
The *Evident* Client and Server communicate through a well-defined protocol, independent of either component's implementation. The protocol is the contract that both sides must conform to. A framework user who does not trust the provided client or server implementation can reimplement either side against the protocol specification without loss of interoperability. The framework may therefore be understood as a set of protocols specifying how processes communicate, accompanied by a proposed implementation that follows those protocols. If a trust model does not accept the proposed implementation, the framework may still be useful for its protocol specification alone.

The Server component runs inside the confidential VM alongside workloads that may handle sensitive data. Since the Server is not the critical process, its design prioritizes vulnerability protection and resource usage reduction, aiming to minimize both the attack surface and the performance overhead. The Client component runs in the interested party's environment and is responsible for critical attestation logic. Its trustworthiness depends on the user's trust model, specifically on the user's confidence in the Client's implementation. Source code readability, clarity of logic, and auditability were therefore prioritized in its design. The Client is also responsible for presenting the conclusions of the remote attestation process clearly.

Figure 4.2 illustrates the intended architecture. The VM Owner uses the *Evident* Client to build, measure, upload, and deploy the confidential VMs. Once launched, the *Evident* Server listens for remote attestation requests and, upon receiving them, collects evidence from both the AMD *Secure Processor* and the virtual TPM. This evidence contains relevant measurements that allow the verifier, using the *Evident* Client, to compare them against the expected values. The evidence is cryptographically signed by a key traceable to the respective root of trust. The verifier uses the *Evident* Client to first build and measure the expected VM image independently. With these measurements, the *Evident* Client performs remote attestation of the target confidential VM and reports the observed state alongside a self-signed instance certificate. The relying party relies on the verifier's conclusions and decides whether to use the deployed workload based on its trust model.

The framework is generic and applicable to various use cases; Chapter 6 demonstrates one in detail.

Figure 4.2: *Evident* Framework context view



Real-world deployments, however, often require capabilities beyond the core pipeline. Sections 4.4 to 4.6 introduce the lifecycle stage designs and optional extensions that address these needs. All extensions are independent of the core pipeline functionality.

4.4 Image Assembly and Deployment

A confidential VM is prepared from the build to the deploy stage. The first produces the image, and the last deploys it to cloud infrastructure. This section covers the design of both stages and an image distribution extension that sits between them.

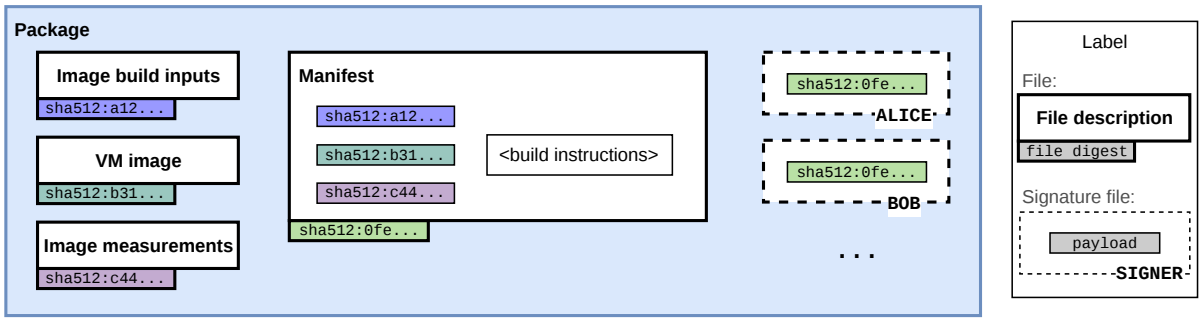
The build stage takes a declarative configuration and produces a bootable VM disk image, a single artifact that all subsequent stages operate on. The measure stage derives expected measurements from it, the upload stage transfers it to a cloud provider, and the attest stage verifies that what is running matches what was built. The properties of the image, therefore, determine the upper bound on what the framework can guarantee. The design constraints that the image must satisfy, including deterministic boot measurements, an immutable root filesystem, and a self-contained boot binary, are documented in Chapter 5, where the implementation choices that realize them are also described.

The upload stage transfers the built image to a cloud provider's storage and registers it as a launchable machine image. The deploy stage then instantiates confidential VMs from the uploaded image. Deployment must follow each provider's API to request confidential VM enablement and ensure the

instance is correctly configured with the hardware protections active. Cloud providers expose this as an enablement option for selected instance families; the implementation is responsible for requesting it through the appropriate provider interface. The *Evident* Client's deploy stage can also be configured to automatically perform remote attestation on freshly instantiated instances before returning control to the user. This simplifies interaction and avoids repetition, since a single deploy invocation may trigger multiple confidential VM instantiations, while the attest stage attests one instance at a time.

Beyond the core build-upload-deploy path, the framework provides an image distribution extension that combines the build and measure stages into a single endorsement workflow. It is directed at image distributors who need a way to endorse a VM image build for consumption by other parties. Instead of publishing only the built VM image, the distributor uses the *Evident* Client's *package* mode, which builds the VM image, measures it, and signs a manifest that describes the build instructions, build environment, the expected measurements, and the digest of the resulting image, as illustrated in Figure 4.3.

Figure 4.3: Image distribution package



Target users would then decide how to proceed based on their trust model. If they trust the image distributor to package the VM image honestly, they may use the VM image and the expected measurements directly. Otherwise, the manifest contains enough information to reproduce the build independently and use the resulting outputs instead. In practice, the build process may be long and computationally expensive, so this reliable distribution of VM images allows trusting users to skip the build stage entirely.

The VM image build process would be considered reproducible if, every time the image build inputs were used to follow the build instructions, regardless of the environment, the resulting VM image was bit-for-bit identical. Consequently, a reproducible build process causes the package to become independently verifiable: the locally built VM image and its measurements should match the endorsed ones bit-for-bit. To support this verification, the package can be used as input to a *reproduce* execution mode, which interprets the included manifest, builds and measures the VM image as instructed, and compares the results. The packaged VM image measurements may be independently verified by using the Client's measure stage, as the measurement derivation is fully deterministic.

4.5 Attestation Protocol and Certificates

Remote attestation establishes that a confidential VM is running the intended workload, but by itself it does not guarantee that subsequent network connections will reach the same instance. This section describes how the framework binds attestation results to cryptographic identity.

The core pipeline assumes that the relying party trusts the verifier. Once the verifier collects and verifies evidence from the target confidential VM, the relying party assumes (under a trust model that permits it) that the workload is now safe to use and potentially to provide sensitive data to. The gap is that remote attestation proves a confidential VM running the intended workload exists, but does not prove that future accesses to the same network address will reach the same confidential VM.

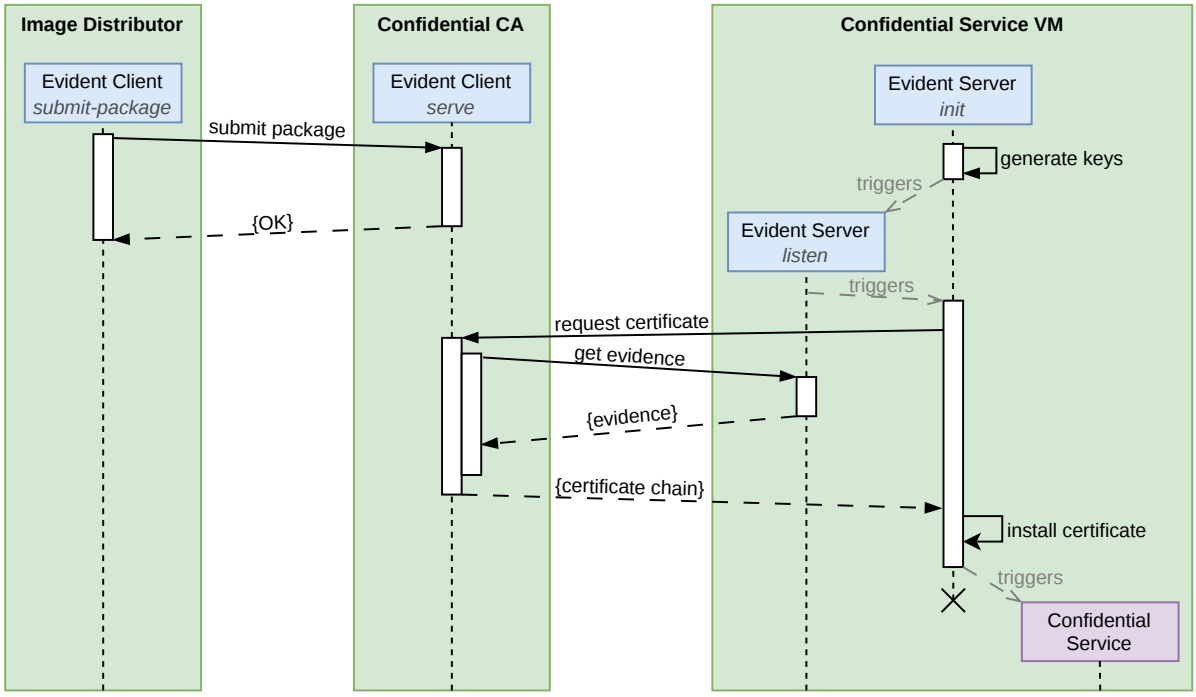
To address this issue, as proposed by the AMD SEV-SNP whitepaper [1], the VM generates an asymmetric key pair upon launch (*instance keys*) and binds the public key to its measured state. The binding works by including the public instance key in the user data field of the evidence generation request, so that it becomes covered by the root-of-trust signing key's signature. A VM that, inside the endorsed report, presents both the expected measurements and the identification of that key pair, lets the verifying party conclude that the instance keys belong to the attested VM and that their confidentiality and integrity are protected by the confidential VM mechanisms and the measured software that handles them. This mechanism exposes the public instance key, which can then be used to establish a secure communication channel with the attested instance, with strong guarantees that the remote endpoint is the verified VM and not a potentially malicious substitute. If it were a substitute, the measurements would not match the expected ones.

To make this procedure more practical, the framework's remote attestation protocol expects the *Evident* Server to present a certificate signing request to the verifier, as illustrated in Figure 4.4. Upon successful attestation, the verifier signs this request with a verifier-owned private key, acting as a certificate authority. The relying party still needs to trust the verifying party, but can now be confident that it is communicating with the attested workload, provided that the workload uses the verifier-issued certificate to establish the communication channel.

4.6 Attestation Certificate Issuance Scalability

The attestation certificate mechanism described in Section 4.5 requires the verifier to run the *Evident* Client and perform the full attestation protocol for each instance. When dealing with a large number of confidential VMs, performing remote attestation individually can become cumbersome or too computationally demanding. The problem is sharper when the relying party interacts with confidential VMs through a web browser or any other environment where the *Evident* Client is not present. The attestation protocol is not embedded in browser behavior, and there is no standardized mechanism for a browser

Figure 4.4: *Evident* attestation certificate issuance

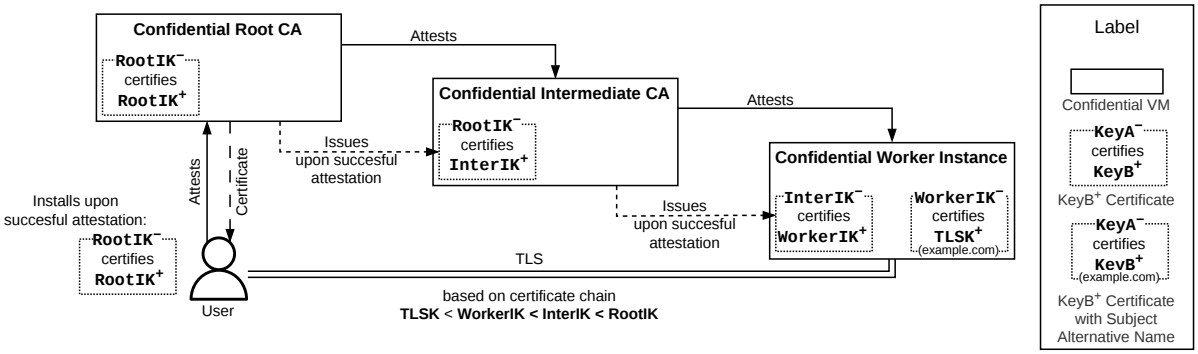


to perform the evidence collection and verification steps that the framework requires. Requesting every user to run the client locally before interacting with a service is not a viable model at scale.

The framework addresses this through a dedicated instance design: a Confidential Certificate Authority. A Confidential Certificate Authority is a confidential VM whose workload is the *Evident* Client running in a dedicated serving mode. In this mode, the Client listens on a given port for incoming attestation requests from other confidential VMs. The Confidential Certificate Authority (CA) must be reachable via a well-known endpoint and deployed before any of the confidential VMs it will attest. When a confidential VM starts up and is configured to request a certificate, it contacts the Confidential CA, which performs remote attestation on the requesting instance. If the result is successful, the Confidential CA issues a certificate for the requesting instance.

As Figure 4.5 illustrates, users interact with the deployed services by first attesting the Confidential Root Certificate Authority using the *Evident* Client. Once the Confidential CA is verified to be running the expected *Evident* Client workload in a correctly measured confidential VM, any certificate it issues carries an indirect trust endorsement, since it was produced by a verified attestation authority running in a verified environment. A browser or other client that trusts the Confidential CA's certificate can then interact with any leaf confidential VM that presents a certificate rooted to it, without performing the full attestation protocol independently. For users or systems that require direct assurance, the leaf confidential VM remains directly attestable through the standard *Evident* Client workflow.

Figure 4.5: Hierarchical certificate issuance with Confidential Certificate Authorities



Each confidential VM instance, at startup, generates an asymmetric key pair (instance keys) and a self-signed certificate, held exclusively in memory. If the instance is configured to request a certificate from a Confidential CA, it sends a certificate signing request upon startup and, upon successful attestation, replaces the self-signed certificate with the issued one. If the instance's service requires a Transport Layer Security (TLS) connection, the instance key is used to issue a certificate for a separate key pair (in Figure 4.5, the TLS Key Pair) that includes the address as the Subject Alternative Name for browser integration. All key material and certificates exist only in volatile memory. A restart destroys them, and the instance must re-request a certificate from the Confidential CA on the next boot.

The design is composable. A Confidential CA is itself a confidential VM and can be configured to obtain its own certificate from a higher-level Confidential CA before it begins issuing certificates to others. This layering allows the structure to scale. A single root Confidential CA can issue certificates to one or more intermediate Confidential CAs, each of which can issue certificates to an indefinite number of leaf instances. Users may attest the root Confidential CA or any intermediate one, because trust is established through remote attestation rather than solely through hierarchical indirect trust. The number of layers and the fan-out at each level are deployment choices. The trust chain from the root to any leaf is preserved regardless of depth, because each certificate in the chain was issued by an entity that was itself attested before it began issuing.

For the Confidential CA to perform remote attestation on a requesting instance, it must know what measurements to expect. Loading these expectations statically at image build time would make the set of attestable images fixed and require redeployment whenever a new image is introduced. The framework addresses this through integration with the image distribution mechanism described in Section 4.4. The Confidential CA is loaded at build time with a set of trusted package submitter identities, each identified by a public key. After an image distributor packages a VM image using the *Evident* Client's package mode, they can submit the resulting package to the Confidential CA's endpoint, as seen before in Figure 4.4. The serving Client validates each package before registering its measurements: it verifies the manifest signature against the pre-loaded trusted keys and confirms that the declared ex-

pected measurements are correctly bound to the submitted artifacts. A package that fails either check is rejected, and its measurements are never stored. When a confidential VM subsequently requests attestation, the Client compares the presented evidence against all previously accepted measurement sets. A match against any of them constitutes a successful attestation, and the requesting instance receives a certificate. This design allows the set of attestable images to grow dynamically without redeploying the Confidential CA, while still keeping the trust boundary well-defined: only packages endorsed by a submitter whose key was trusted at build time can influence which images the authority will accept.

4.7 Discussion

This section examines limitations and open questions that the *Evident* framework exposes but does not fully resolve, either because they depend on decisions outside of the customer's control or because they fall outside the scope of this thesis.

Trust in cloud providers' virtual TPM implementations. The virtual TPM implementations offered by Elastic Compute Cloud and Google Compute Engine are closed-source and execute at the hypervisor level. Stating that a trust model accepts the provider's virtual TPM implementation is a narrower and more precise claim than stating that the trust model accepts the provider in general. When the broader statement is adopted, most of the verification procedures introduced by the Evident framework become redundant, as does the use of confidential VMs themselves. This thesis assumes that cloud deployment is a requirement for the workload in question and seeks the minimal trust model that supports it. If the virtual TPM implementation is not trusted, there is currently no mechanism to reliably assert the integrity of the VM contents beyond the firmware, and therefore no working solution to fully attest the state of the workload. If the virtual TPM implementation is trusted, the integrity of the VM contents becomes verifiable without expanding the trust boundary to cover anything beyond that specific component. A compatible trust model could, for example, assume the virtual TPM implementation to be correct while simultaneously assuming that an adversary has unrestricted access to the VM image; even under those conditions, the framework's attestation would detect any modification to the image contents.

Azure Virtual Machines present a structurally different situation. The virtual TPM executes inside the guest context with hardware-enforced isolation from the host, which is architecturally stronger than the hypervisor-level placement used by the other providers and consistent with the design proposed by V. Narayanan *et al.* [33]. However, since the virtual TPM implementation running on Azure confidential VMs is closed-source, it could contain vulnerabilities or intentional backdoors that allow unexpected access to the TPM's internal state. Compatible trust models must therefore still include a correct virtual TPM implementation as an explicit assumption, even on Azure. More broadly, none of the three studied providers allows the customer to customize the firmware or substitute the virtual TPM implementation,

which means that the architecturally optimal design cannot be reproduced or independently verified by the customer in any current public cloud offering.

Elastic Compute Cloud TPM key certification. As established in Chapter 2, an endorsement key certificate issued by the TPM manufacturer is necessary for a remote party to trust that the endorsement key pair resides inside a genuine TPM. Elastic Compute Cloud service does not provide such a certificate. Instead, it offers an access-controlled interface that returns the endorsement key's public key for a given VM instance. By comparing this value against the public key retrieved directly from the TPM interface, the VM owner can confirm that Elastic Compute Cloud endorses the key pair as belonging to their TPM implementation. This provides a functionally similar guarantee for the VM owner, but it has significant consequences for third-party attestation. A third party cannot use the same interface in the same way and therefore has no independent basis for concluding that the endorsement key resides inside a TPM rather than being an arbitrary key pair. This limitation can only be resolved by Amazon Web Services, either by issuing an unrepudiable endorsement of the TPM-resident endorsement key or by exposing a public-facing interface for retrieving it, as implemented by Google Compute Engine, seen in Chapter 3.

Azure attestation report freshness. As discussed in Chapter 3, Azure confidential VMs block direct access to the AMD *Secure Processor* device interface. A single attestation report is generated at launch and stored within the paravisor. The verifier cannot request a new report with a fresh, unpredictable nonce, which means freshness cannot be established. It is indistinguishable, from the verifier's perspective, whether the attestation report was genuinely generated by the *Secure Processor* for the running VM or was side-loaded into the paravisor from another source. The only mitigation would be on-demand report generation from the AMD *Secure Processor* with a verifier-supplied nonce, which would require changes to the paravisor that only Azure can make.

Disk encryption. The *Evident* framework uses a read-only filesystem paired with volatile-memory-backed filesystems mounted on directories that require write access to properly boot the virtual machine. This design assumes that the data is written to memory at runtime without swap and that the read-only files are not sensitive. When these assumptions do not hold, additional mechanisms are needed. If the workload must write sensitive data after boot (that does not fit in memory), an encrypted partition could be created empty at boot time and populated after successful attestation. If the workload must read sensitive files that should not be available before attestation succeeds, those files could reside on an encrypted partition that remains unmounted until the verifier provides the decryption key upon successful remote attestation, following the approach proposed by L. Wilke *et al.* [26]. Both scenarios are outside the scope of the current implementation but represent valuable directions for future work.

Summary

This chapter introduced the *Evident* framework, a system designed to automate the trust-enabling operations surrounding the lifecycle of confidential VMs in public cloud environments. The framework addresses the gaps identified in Chapter 3: the absence of standard tooling for independently predicting, reproducing, and verifying the measurements that the AMD *Secure Processor* and the virtual TPM produce for a given deployment. *Evident* is organized around five lifecycle stages (build, measure, upload, deploy, and attest), operates on custom-built VM images without requiring workload modifications, and is designed to be both workload-agnostic and cloud-agnostic at the architectural level.

The chapter established the threat model under which the framework operates, where the hypervisor, co-located VMs, and the cloud provider are all considered potentially adversarial, while the hardware specifications and cryptographic primitives are assumed sound. The framework collects all available evidence unconditionally, leaving the trust model to determine how that evidence is interpreted rather than whether it is gathered.

Several extensions were described that build on the core pipeline without altering it: automatic attestation at deploy time, a signed image distribution mechanism with reproducibility verification, attestation certificates that bind instance keys to measured state, and a scalable attestation mode that supports hierarchical certificate issuance through Confidential Certificate Authorities. The chapter also identified the principal challenges that shaped the implementation, including measurement reproducibility across providers, Platform Configuration Register value prediction for the virtual TPM, and the use of immutable filesystems to narrow the trust boundary. Finally, the discussion examined limitations that the framework exposes but cannot resolve from the customer's standpoint, including the closed nature of all three providers' virtual TPM implementations, the absence of TPM key certification on Elastic Compute Cloud, the lack of attestation report freshness on Azure, and the deferral of disk encryption to future work. Chapter 5 details how each of these challenges was addressed in the implementation.

5

Implementation

Contents

The previous chapter described *Evident* as a design comprising lifecycle stages, a threat model, and the protocol that binds its client and server components. That description was deliberately independent of any specific technology, so that each component could be reimplemented without loss of interoperability. This chapter documents a concrete implementation of that design, covering the engineering constraints that shaped technology choices and the practical obstacles that had to be resolved before the framework could produce meaningful attestation results on real infrastructure.

The implementation[1] supports AMD SEV-SNP as the hardware root of trust and targets two cloud providers: Elastic Compute Cloud (from Amazon Web Services) and Google Compute Engine (from Google Cloud Platform). Azure Virtual Machines are not supported at this time. Where the two supported providers diverge the most is in how they expose attestation evidence or endorse TPM keys,

[1] Available at `https://gitlab.com/dpss-inesc-id/achilles-cvm`

those differences are implemented in minimal modules, used within a shared pipeline. The most consequential of these differences is how Attestation Key trust is established, which is covered in detail in Section 5.4.

Section 5.1 describes the technology stack and the rationale behind each choice. Section 5.2 covers the construction of VM images: the use of *NixOS* for declarative, reproducible builds, the Unified Kernel Image for deterministic boot measurement, the immutable root filesystem, and the upload process. Section 5.3 addresses how expected measurements are derived before deployment for the virtual TPM PCR values, including a contribution to the open-source *sev-snp-measure* project that enabled Google Compute Engine support. Section 5.4 details the remote attestation workflow, first the general four-phase pipeline, then the provider-specific implementations.

5.1 Technology Stack

As established in Section 4.3, the framework contemplates two components with distinct responsibilities: a client that orchestrates the full lifecycle and performs all verification logic, and a server that collects evidence from the AMD *Secure Processor* and the virtual TPM on behalf of the client. The server does not evaluate evidence or make trust decisions. This section documents the technology choices for each component and the protocol that connects them.

The client is implemented in *Go*[2]. The primary motivation is readability and auditability. As established in Chapter 4, the client is the critical component whose correctness must be assessed by the relying party. *Go*'s explicit error handling, minimal language surface, and straightforward control flow make the source code easier to review by someone unfamiliar with the codebase. Its static compilation produces a single binary with no runtime dependencies, which simplifies distribution and eliminates the need for a managed runtime on the verifier's machine. The tradeoff is that *Go* packages its own runtime that, together with the complete set of self-contained dependencies, makes the resulting binaries heavier compared to alternatives. In practice, this was an acceptable cost for a tool whose primary audience values transparency over performance and space optimization.

The server is implemented in *Rust*[3]. Chapter 4 established that the server should prioritize vulnerability protection and minimal resource usage, since it runs alongside potentially sensitive workloads. *Rust*'s memory safety guarantees, enforced at compile time without garbage collection, constitute a major advantage when compared to alternatives. Its low runtime overhead and small binary size address the second priority. The tradeoff is a steeper learning curve and longer compilation times, neither of which affects the end user.

[2] `https://go.dev/`
[3] `https://rust-lang.org/`

The client and server communicate over *gRPC*[4], a remote procedure call framework built on HTTP/2 and *Protocol Buffers*. The choice is motivated by the protocol-first design principle from Chapter 4. *Protocol Buffers* provide an explicit, language-neutral service definition that serves as the formal contract between client and server. Code generation from this definition ensures that both sides agree on message structure and available operations without manual coordination. The transport layer uses TLS with a self-signed certificate generated by the server at launch. This certificate is ephemeral and provides only transport encryption; it does not authenticate the server's identity. Identity authentication is deferred to the attestation process itself.

5.2 Image Construction

The build stage transforms a declarative configuration into a bootable virtual machine image. This becomes the single artifact that all subsequent stages operate on: the measure stage derives the expected measurements from it, the upload stage transfers it to a cloud provider, the deploy stage chooses which instances use it, and the attest stage verifies that what is running matches what was built. The properties of the image, therefore, determine the upper bound on what the framework can guarantee.

NixOS and Reproducibility

The implementation uses *NixOS*[5], a Linux distribution built on the *Nix* [37] package manager, where the entire operating system configuration is specified declaratively in a set of configuration files. A *NixOS* configuration describes the desired final state of the system, including installed packages, enabled services, kernel parameters, filesystem layout, user accounts, and more, and the build system computes the steps needed to realize that state. The framework uses Nix Flakes, a feature that pins every input (package repositories, external dependencies, toolchains) to exact revisions. This pinning is what makes reproducibility achievable, *i.e.*, given the same flake and the same inputs, the build process should produce an identical output.

Several properties make *NixOS* a strong fit for the framework's objectives. Its package repository, *Nixpkgs*[6], is the largest and most actively maintained collection of packaged software available for any Linux distribution, reducing the likelihood that a user's workload or needed dependency will require manual packaging. Every package and its dependencies are addressed by a cryptographic hash of their inputs, making it straightforward to trace exactly what is included in the final image. The system is hardened by default, with a minimal set of enabled services and conservative default configurations.

[4] `https://grpc.io/`
[5] `https://nixos.org/`
[6] `https://github.com/NixOS/nixpkgs`

The framework provides the *NixOS* configuration as a template. Users adapt it to their needs by adding their workload, enabling services, and adjusting system parameters. This flexibility comes at a cost: a user can produce a configuration that the measure stage cannot correctly predict, for instance, by altering kernel configuration parameters or filesystem layout in ways that change PCR values without updating the measurement derivation accordingly. Configuration lines that affect measurement are clearly marked in the template, and preserving them is the user's responsibility.

Alternative build systems were considered. *mkosi*[7] produces declarative image definitions but relies on distribution package managers whose output is not bit-for-bit reproducible without *Nix* underneath; layering both tools adds complexity without clear benefit. *Yocto*[8] builds everything from source, which provides strong traceability but is extremely resource-intensive and time-consuming for full operating system images, and its learning curve is steep for users who simply want to deploy a workload. *Nix* is configurable to disable cached binaries when building, effectively also building every component from source. This feature is available for interested users.

The image configuration declares a Unified Kernel Image (UKI), a single portable executable binary that bundles the Linux kernel, the *initrd* (initialization process), and the kernel command-line into one file. With direct kernel boot, also declared in the minimal image configuration, the firmware loads this single binary and measures it as a unit. This design eliminates the ambiguity that arises when the kernel, *initrd*, and command-line are separate files that could be independently substituted. With a UKI, a single measurement covers all components that determine early boot behavior.

Immutable Filesystem

The root filesystem uses *erofs*[9], a read-only compressed filesystem. At build time, a *dm-verity*[10] hash tree is computed over the entire *erofs* partition. The root hash of this tree is embedded in the kernel command-line, which is itself part of the UKI. When the kernel mounts the root filesystem, *dm-verity* verifies every block read against this hash tree. Any modification to the filesystem contents, whether made before or after deployment, will produce a hash mismatch and cause the read to fail.

This design, proposed by M. Sanft [38], closes a trust boundary identified in Section 4.2. Because the VM image is under the cloud provider's control between upload and instantiation, a write-permissive filesystem would allow the provider to inject or modify files without reliable detection by the virtual TPM's boot measurements. With *erofs* and *dm-verity*, the integrity of the entire filesystem is reduced to the integrity of the root hash, and the root hash is measured as part of the kernel command-line inside the UKI.

[7] `https://mkosi.systemd.io/`
[8] `https://www.yoctoproject.org/`
[9] `https://docs.kernel.org/filesystems/erofs.html`
[10] `https://docs.kernel.org/admin-guide/device-mapper/verity.html`

Directories that require write access for the operating system to function properly, such as `/tmp`, `/var`, `/run`, and `/etc`, are mounted as *tmpfs*[11] (volatile, memory-backed filesystems). Their contents exist only in RAM and are lost on reboot. Critically, `/nix/store`, the directory containing all executable binaries and libraries, is part of the read-only *erofs* partition. It is not possible to install applications that were not packaged with the VM image at build time.

Server and Workload Integration

The *Evident* Server is included in the image as a *NixOS* module and launched as a *systemd*[12] unit at startup, following the same mechanism used for any other system service. Workloads are integrated identically: the user declares them in the *NixOS* configuration, and they are packaged into the image and started by *systemd*. There is no instrumentation, interception, or modification of the workload by the framework.

The VM image is in *raw* format, chosen because it is compatible with all target platforms: Elastic Compute Cloud, Google Compute Engine, and Azure Virtual Machines.

Upload

Image upload is handled by *Uplosi*[13], a tool designed to upload raw disk images to cloud providers and register them as launchable machine images. Declarative configuration files dictate the tool's behavior, and an example configuration for each supported provider is included in the *Evident* repository, so that users need only supply their use-case specifics. Alternatively, users may upload images using the provider's own command-line interface (the *awscli*[14], *gcloud*[15], or *az*[16]), as the raw image format is compatible with each provider's import instructions.

5.3 Measurement Derivation

The measure stage takes a built VM image and produces the set of expected measurements that the attest stage will later compare against evidence collected from a running confidential VM. Two independent measurement sources must be predicted: the AMD *Secure Processor*'s launch digest, and the virtual TPM Platform Configuration Registers values.

[11] `https://docs.kernel.org/filesystems/tmpfs.html`
[12] `https://systemd.io/`
[13] `https://github.com/edgelesssys/uplosi`
[14] `https://aws.amazon.com/cli/`
[15] `https://cloud.google.com/sdk/gcloud`
[16] `https://learn.microsoft.com/en-us/cli/azure/`

AMD Secure Processor Launch Digest

Reproducing the launch digest requires knowing both the binary contents loaded into the confidential VM's initial memory and the exact page-loading sequence used by the hypervisor, as described in Chapter 2. The availability of this information differs across providers, as covered in Chapter 3.

For Elastic Compute Cloud, the measurement derivation uses *Virtee*'s[17] *sev-snp-measure-go*[18] library. Amazon publishes the firmware source code, and the measurement procedure is presented in this library, making the launch measurement fully reproducible. The tool simulates the page-loading sequence that the hypervisor performs, extending the launch digest with each page's content and metadata in the same order as the *Secure Processor* would. The result is a predicted launch digest that can be compared against the one reported in the attestation report.

For Google Compute Engine, the derivation process is described in Google's *gce-tcb-verifier*[19] tool. Google publishes the firmware binary through public interfaces, but not its source code. However, no library interface could take the firmware binary as input and independently derive the expected measurement. As a partial result of this thesis work, support for the Google Compute Engine's launch measurement derivation has been included in the *sev-snp-measure-go* library, and its user-facing tool *sev-snp-measure*[20], making both providers addressable through a common interface.

For Azure Virtual Machines, the launch measurement derivation process is not public. The measured contents (firmware and HCL) are not available to the verifying party, and the measurement methodology is opaque. The framework documents this limitation and does not produce a predicted launch digest for Azure.

As seen in Chapter 2, the launch digest also depends on the virtual cores assigned to the confidential VM, since each virtual core needs an initial state configuration page, measured by the AMD *Secure Processor* on loading. This cannot be predicted at the measure stage; therefore, either the launch digest derivation is done directly in the attest stage, or the measure stage preemptively maps common virtual core counts to their respective expected measurement for the given firmware binary. In the current *Evident* Client implementation, the first option was chosen.

Platform Configuration Register Prediction

The virtual TPM extends measurements into Platform Configuration Registers (PCRs) during the boot process. The framework predicts three specific PCRs, chosen because they cover the components whose integrity the framework can guarantee.

[17] `https://virtee.io/`
[18] `https://github.com/virtee/sev-snp-measure-go`
[19] `https://github.com/google/gce-tcb-verifier`
[20] `https://github.com/virtee/sev-snp-measure`

PCR 4 records the measurement of the boot binary loaded by the firmware. In the case of *Evident* VM images, this is the UKI, since Linux direct boot causes the firmware to measure and load it directly. The individual components of the UKI, including the kernel image, the *initrd*, and the kernel command-line (which contains the *dm-verity* root hash), are already captured in this register. The first component to take control within the UKI is the *systemd-stub*[21], which also measures the rest of the UKI's individual sections, extending them to PCR 11. PCR 12 records overrides or additions to the components measured by PCR 11, applied through UKI extension mechanisms, processed and measured by the *systemd-stub*. Because *Evident* images do not use UKI extensions, PCR 12 is expected to be zero (its initial state, never extended), and the framework hardcodes this expectation as a verification that no unexpected extensions were applied.

Other PCRs, such as those recording device tables or Secure Boot[22] artifacts, are not relevant to the security guarantees the framework provides. Secure Boot is a firmware-level mechanism that verifies cryptographic signatures on boot binaries before executing them, preventing the loading of unsigned or tampered code. *Evident* does not use Secure Boot because the mechanism is enforced by the firmware, which is not under the customer's control, and the only security guarantee drawn from it is that the expected UKI was booted. The framework instead assumes that relying parties perform remote attestation before offering sensitive information to the confidential VM. If remote attestation succeeds, the relying party may conclude the expected UKI was booted, achieving similar security guarantees.

Derivation Tooling

To extract the UKI from the raw disk image, the framework uses the *godiskfs*[23] library, a Go library for reading disk image partition tables and filesystems. This avoids the alternative, *systemd-dissect*[24], which requires root permissions to achieve the same results. The *godiskfs* library reads the partition table, locates the relevant system partition, and extracts the UKI binary without mounting the image.

Once the UKI is extracted, *systemd-pcrlock*[25] is used to derive the expected PCR values. The tool operates by parsing the UKI's portable executable binary structure, identifying each embedded section (kernel, *initrd*, command-line, and others), and computing the measurement that the firmware would extend into the corresponding PCR for each section. The tool outputs a description of every expected individual measurement to both PCR 4 and 11. The framework consumes this output, and together with the hardcoded PCR 12 value of zero, produces for each PCR its final digest, by simulating the extension operation performed by the TPM. The final PCRs digest, which is the hash of an ordered concatenation of individual final PCR digests, is the actual value included in TPM quotes.

[21] https://www.freedesktop.org/software/systemd/man/latest/systemd-stub.html
[22] https://wiki.archlinux.org/title/Unified_Extensible_Firmware_Interface/Secure_Boot
[23] https://github.com/diskfs/go-diskfs/
[24] https://www.freedesktop.org/software/systemd/man/latest/systemd-dissect.html
[25] https://www.freedesktop.org/software/systemd/man/257/systemd-pcrlock.html

5.4 Remote Attestation Workflows

The attest stage is the core of the framework's verification function. It connects to a running confidential VM, collects evidence from both the AMD *Secure Processor* and the virtual TPM, and evaluates that evidence against the expected measurements produced by the measure stage. The workflow differs across providers due to differences in how attestation evidence is exposed and how TPM key endorsement is handled.

General Workflow

The attest stage follows a four-phase pipeline that applies uniformly across both supported providers. Provider-specific behavior is confined to well-defined points within each phase, described in the subsections that follow.

Evidence collection. The client generates a 64-byte cryptographically random nonce and sends it to the *Evident* Server running inside the target confidential VM. The server uses the nonce to request attestation evidence from both the AMD *Secure Processor* and the virtual TPM, then returns a bundle containing the AMD's attestation report, the TPM's quote, and the instance key. The client parses this bundle into two evidence objects: hardware evidence, covering the *Secure Processor* attestation report, and software evidence, covering the TPM quote. The AMD processor model is inferred from the collected evidence, which determines which certificate chain will be fetched and used in the subsequent verification step.

Hardware evidence verification. The AMD trust anchor certificates, including the AMD Root Key (ARK) and both AMD SEV Key (ASK) and AMD SEV-VLEK Key (ASVK), are fetched from AMD's key distribution service[26]. The attestation report's signature is then certified against the evidence-included Versioned Chip Endorsement Key (VCEK) (or Versioned Loaded Endorsement Key (VLEK)), and the certificate chain is verified. Following signature verification, the report's freshness and key binding are checked. The client recomputes a digest over the nonce, the serialized instance's public key, and provider-specific additional material, then compares against the data field embedded in the attestation report. A mismatch indicates either that the report was generated at a different time or that it was produced for a different instance key, both of which cause attestation to fail.

The firmware binary is obtained through provider-specific means, and the expected launch digest is computed by simulating the page-loading sequence as described in Section 5.3. This expected value is compared against the launch digest present in the attestation report. A mismatch means that the firmware loaded into the confidential VM at boot time differs from the expected binary.

Software evidence verification. The TPM quote signature is verified using a provider-specific At-

[26] `https://kdsintf.amd.com/`

testation Key (AK) trust mechanism detailed in the following subsections. The quote's qualifying data field is then verified to contain the expected freshness binding, computed as a digest over the nonce, the serialized instance's public key, and the provider-specific additional material. Finally, the Platform Configuration Register (PCR) digest embedded in the quote is compared against the expected value. The expected value is derived from all the individual expected PCR measurements obtained from the *Evident* Client's *measure* mode. A mismatch means that the image contents (kernel, filesystem, and others) differ from the previously measured ones. This stage does not provide a gradual evaluation, either all measurements match, or the attestation fails.

Google Compute Engine

The Google Compute Engine workflow verifies the chip-specific VCEK signature and ASK-ARK certificate chain. The AK certificate is retrieved from Google's key distribution service, which provides both intermediate and root certificate authority certificates, allowing the workflow to construct and verify the full chain.

The freshness binding incorporates the nonce, the serialized instance's public key, and the raw bytes of the AK certificate itself. Including the full certificate, rather than only the key material, ties the binding to the AK's issued identity.

Obtaining the endorsed firmware binary is a multi-step process specific to this provider. Google offers a *launch endorsement* blob from a public Google Cloud Storage bucket indexed by its launch digest value, meaning that this endorsement may only be retrieved after gathering the attestation report from the AMD *Secure Processor* and inspecting the included launch digest. The launch endorsement signature is verified by the attached certificate, which is issued by Google's cloud integrity root certificate, available in Google's public key infrastructure. Once the endorsement is established as valid, the client extracts the firmware binary digest and uses it to download the binary itself from a separate Google Cloud Storage location. This binary is then used to derive the expected launch measurement, which should match the one included in the *Secure Processor* attestation report.

A failed measurement comparison in this workflow causes the attestation to fail.

Elastic Compute Cloud

The Elastic Compute Cloud workflow verifies the VLEK signature and ASVK-ARK certificate chain. No certificate for the AK or the EK is available; therefore, the trust in the AK must be established unconventionally.

The workflow uses the TPM make-credential and activate-credential protocol that aims to prove that two objects reside inside the same TPM. This protocol is then used to verify that the EK and the AK are

co-resident. This works by encrypting a randomized secret 64-byte array with a key derived from the EK public key and the AK identifiable parameters. The TPM interface can decrypt the secret with the EK private key and the same AK information. This secret is incorporated with the client-generated 64-byte nonce, the serialized instance's public key, and the serialized AK public key. This compound is used as the user data for the AMD *Secure Processor* attestation report generation and the TPM quote.

If the Elastic Compute Cloud instance identifier is supplied, a provider-specific interface is used to get a provider-endorsed EK, which should match exactly the EK retrieved from the instance directly alongside the evidence. The user must have the necessary permissions to use this interface, as it is not publicly available.

The endorsed firmware binary is retrieved directly through Amazon Web Services' GitHub repository releases. The binary is used to independently derive the launch digest that is matched against the one included in the *Secure Processor* attestation report.

As of this thesis publication, there is an open issue[27] that indicates a possible firmware versioning mismatch between production Elastic Compute Cloud confidential VM instances and the published firmware repository source.

Differences

The most significant structural difference between the two providers concerns how AK trust is established. Google issues an AK certificate, so the client can verify a chain of authority from the AK up to a publicly available root certificate authority. On Elastic Compute Cloud, no such certificate exists; the activate-credential protocol instead achieves a related but weaker guarantee, namely that the AK and the EK reside within the same TPM. The EK authenticity relies on comparison with a value returned by the Amazon Web Services' cloud interface, which is accessible only to the VM owner and not to independent third parties. This has direct consequences for third-party attestation scenarios, where the verifier cannot use the same interface to corroborate the endorsement key's origin.

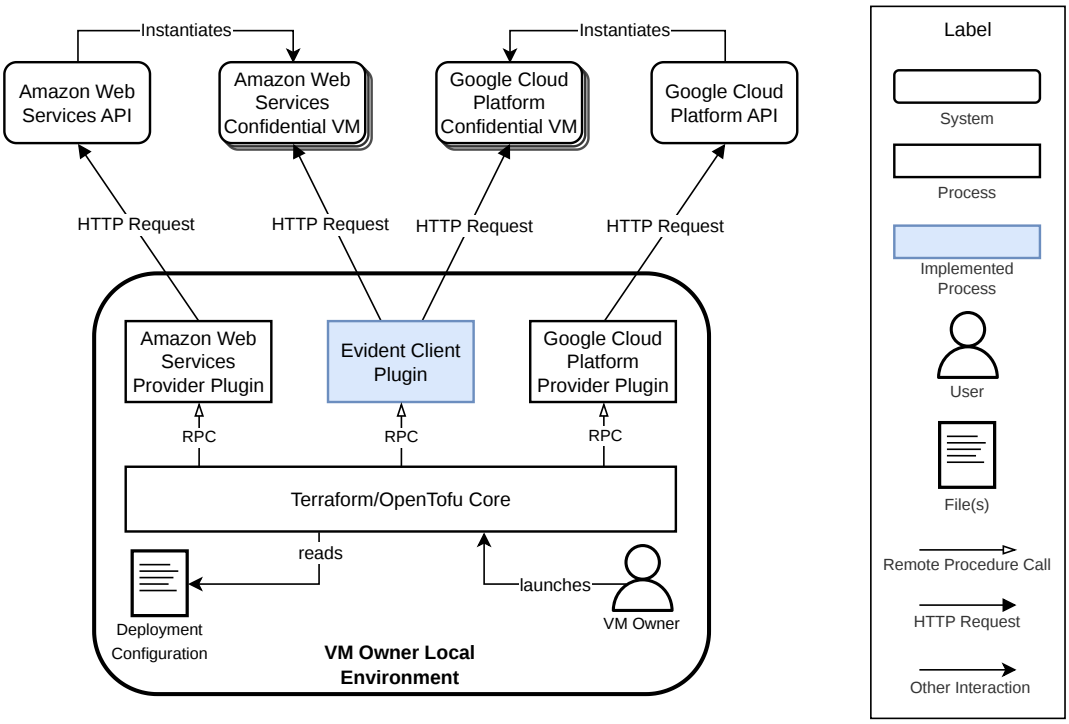
Deploy and Attest Integration

Deployment management uses *Terraform*[28], a declarative infrastructure-as-code tool. The user describes infrastructure state in configuration files, and *Terraform* computes and applies the necessary changes. The configuration files are also compatible with *OpenTofu*[29], an open-source fork with a different license. To integrate attestation into the deployment workflow, the implementation defines a custom *Terraform* resource, using *Terraform*'s plugin architecture illustrated in Figure 5.1. *Terraform* resources

[27] `https://github.com/aws/uefi/issues/19`
[28] `https://www.terraform.io/`
[29] `https://opentofu.org/`

support dependency declarations, so the attestation resource is configured to depend on the completion of the VM resources. Once all VMs are provisioned, the plugin begins polling their endpoints to determine when they are ready to accept attestation requests. When a VM is ready, the plugin performs the remote attestation procedure. If attestation fails for any instance, the error is displayed as the final message of the deployment output for that instance, and the user is responsible for deciding whether to tear down the failed instances.

Figure 5.1: *Terraform*/*OpenTofu* plugin context and architecture



Summary

This chapter documented the implementation of the *Evident* framework across its five lifecycle stages. VM images are built declaratively with Nix, producing a single VM disk image containing a Unified Kernel Image (UKI) that bundles the kernel, *initrd* (initialization process), and kernel command-line into one binary, over a read-only *erofs* root filesystem whose integrity is captured entirely by a *dm-verity* root hash embedded in the UKI. Measurement derivation uses the *sev-snp-measure-go* library to simulate the AMD *Secure Processor*'s page-loading sequence for both Elastic Compute Cloud and Google Compute Engine, with the latter support contributed as part of this thesis work, and *systemd-pcrlock* to predict the virtual TPM PCR values from the extracted UKI. Azure Virtual Machines support is deferred to future work. Upload and deployment stages delegate to established tools (*Uplosi* and *Terraform*), and a

custom *Terraform* provider plugin integrates attestation directly into the deployment workflow.

The remote attestation workflow follows a uniform four-phase pipeline whose provider-specific behavior is confined to well-defined points. The most consequential difference between providers is how the virtual TPM's attestation key is managed: Google Compute Engine issues a certificate with a verifiable chain to a public root, while Elastic Compute Cloud relies on an activate-credential protocol and an access-controlled interface for endorsement key verification, which limits independent third-party attestation. The client and server are implemented in Go and Rust, respectively, chosen to favor auditability on the verification-critical side and memory safety on the workload-adjacent side, communicating over a gRPC protocol that serves as the formal contract between them.

The extensions build on the core pipeline without altering it. Image distribution produces GPG-signed packages containing build instructions, expected measurements, and the VM image, containing all the information for independent reproduction. The Confidential Certificate Authority, a confidential VM running the *Evident* Client in a dedicated mode, issues TLS certificates to other attested instances and accepts new attestable images dynamically through signed package submissions, enabling scalable attestation through composable certificate authority hierarchies.

6

Evaluation

Contents

The previous chapters described the *Evident* framework's design and implementation. This chapter evaluates the framework empirically. The evaluation has three objectives: to demonstrate that the framework produces correct and actionable attestation results for a real workload deployed on public cloud infrastructure, to characterize the latency and resource overhead introduced by the framework's attestation components, and to provide a concrete account of what the remote attestation report communicates to the verifying party.

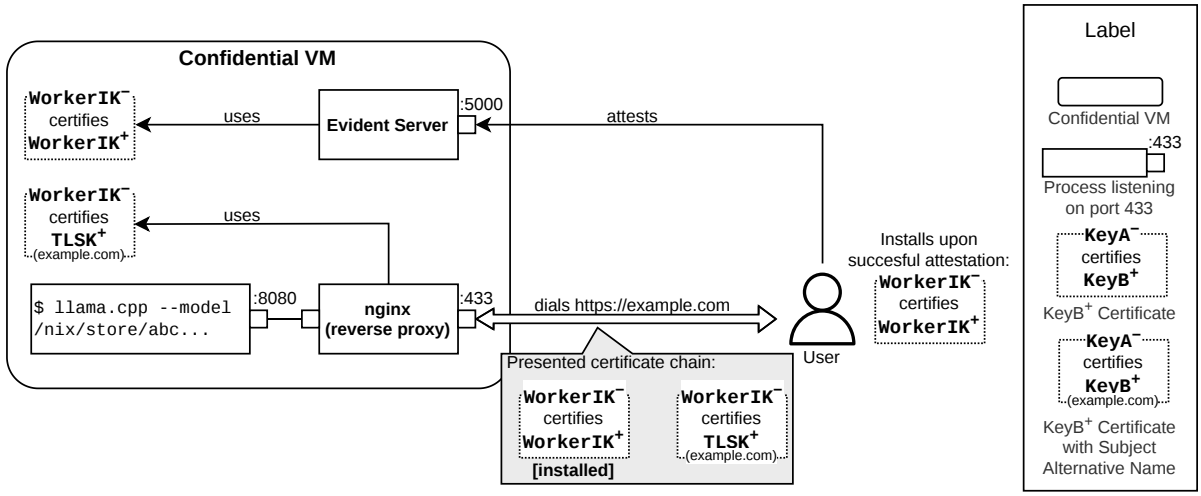
Section 6.1 validates the framework end-to-end through a confidential inference use case, where a large language model inference service is deployed, attested, and accessed over a secure channel bound to the attestation result. Section 6.2 examines the correctness of the framework's measurement predictions by comparing them against the evidence collected from running confidential VMs. Sec-

tion 6.3 describes the remote attestation report produced by the *Evident* Client, its structure, and how the verifying party interprets it. Section 6.4 provides a latency breakdown of the remote attestation workflow from both the client and server perspectives. Section 6.5 characterizes the resource consumption of the *Evident* Server under varying load conditions.

6.1 Use Case: Confidential Inference

To validate the framework's end-to-end functionality, a confidential inference service was built, deployed, attested, and accessed using the *Evident* framework[1]. The service runs a large language model inside a confidential VM, where the model's integrity is verifiable through remote attestation, and all client communication is protected by a certificate bound to the attested instance, as illustrated in Figure 6.1.

Figure 6.1: Confidential inference components



Workload selection

The workload is *llama.cpp*[2], a C/C++ inference engine for large language models. Three inference engines were considered: *llama.cpp*, *vLLM*[3], and *Ollama*[4]. Two criteria guided the selection.

The first is whether the model is present in the measured filesystem at build time. Both *llama.cpp* and *vLLM* expect the quantized model files to reside in the filesystem where the service starts. The model is therefore included in the VM image at build time, becomes part of the read-only *erofs* partition, and is covered by the *dm-verity* hash tree whose root hash is embedded in the kernel command-line,

[1] Available at `https://github.com/joao-hs/evident-llama-cpp`
[2] `https://llama-cpp.com/`
[3] `https://vllm.ai/`
[4] `https://ollama.com/`

as seen in Chapter 5. This means the model's integrity is transitively captured by the Unified Kernel Image (UKI) measurement and verifiable through the TPM's PCRs. *Ollama*, by contrast, is designed to boot minimally and fetch models from a remote registry at runtime. For a use case where the verifying party needs assurance about which model is being served, this is a significant limitation, since this assurance is only indirectly given by the *Ollama* software and registry.

The second is the reproducibility of the packaged binary. *llama.cpp* is written in C and C++ and compiles to a native binary, which is more controllable to reproducible builds than interpreted applications. *vLLM* is a Python application whose runtime behavior depends on the Python interpreter and a large dependency tree, making bit-for-bit reproducibility harder to achieve and verify. Since the use case does not require integration with a larger Python project, the additional complexity of *vLLM* offers no benefit.

These two criteria led to the selection of *llama.cpp*. The model used for this use case was *Meta*'s quantized *Llama 3.1 8B*[5][6]. Additionally, *nginx*[7] was used to terminate TLS and forward the HTTP traffic to the *llama.cpp* process.

Lifecycle walkthrough

The use case demonstrates every stage of the *Evident* lifecycle. The VM image is defined as a *NixOS* configuration that declares *llama.cpp* as a *systemd* service, includes the model weights in the filesystem, and installs the *Evident* Server. The configuration, deployment files, and instructions are published in a separate repository[8].

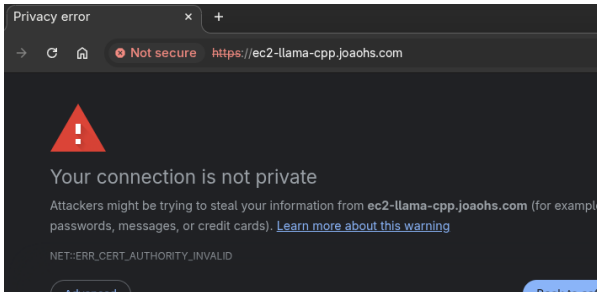
In the *build* stage, the *Evident* Client compiles the *NixOS* configuration into a raw VM image containing the inference service, the model, and the *Evident* Server. In the *measure* stage, the client extracts the UKI from the image and derives the expected PCR values. In the *upload* stage, the image is transferred to Elastic Compute Cloud or Google Compute Engine, and registered as a launchable machine image. In the *deploy* stage, confidential VMs are instantiated from the uploaded image on both providers. In the *attest* stage, the client connects to each running instance, collects evidence from both the AMD *Secure Processor* and the virtual TPM, and evaluates that evidence against the predicted measurements and other trusted artifacts.

Upon successful attestation, the client retrieves the instance's self-signed certificate, whose public key was included in the signed evidence and is therefore bound to the attested measurements by the root-of-trust signature. As pictured in Figure 6.2, the verifying party installs this certificate as a trusted certificate in their local environment. Subsequent HTTPS connections to the inference endpoint use a domain-specific certificate issued by the trusted one for TLS, which guarantees that the remote endpoint

[5] original: `https://huggingface.co/meta-llama/Llama-3.1-8B-Instruct`
[6] quantization: `https://huggingface.co/bartowski/Meta-Llama-3.1-8B-Instruct-GGUF`
[7] `https://nginx.org/`
[8] `https://github.com/joao-hs/evident-llama-cpp`

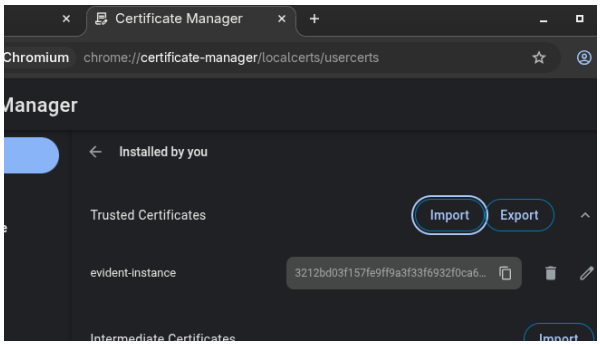
Figure 6.2: Establishing trust in the service's TLS certificate with remote attestation

(a) Before remote attestation: unknown certificate authority

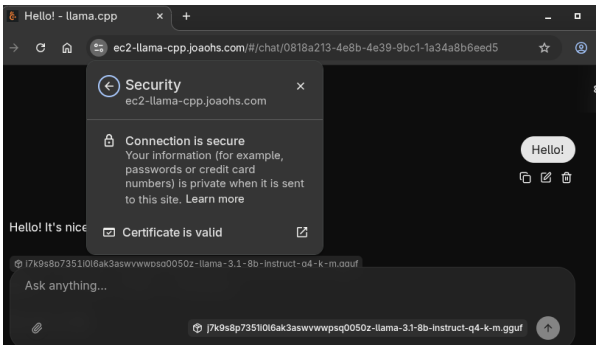


(b) Successful attestation stores the instance root certificate

(c) Install the certificate from the instance in the browser



(d) Trusted TLS session



is the attested confidential VM and not a substitute. The inference service is then accessible through a web browser or any HTTPS client, with the TLS channel anchored in the attestation result.

The use case was successfully completed on both Elastic Compute Cloud and Google Compute Engine. In Google Compute Engine, `n2d-standard-16` images were used in `europe-west3`, while in Elastic Compute Cloud, `m6a.8xlarge` images were used in `eu-west-1`. This use case assumed that the domain name server was correctly configured.

Hierarchical Certificates

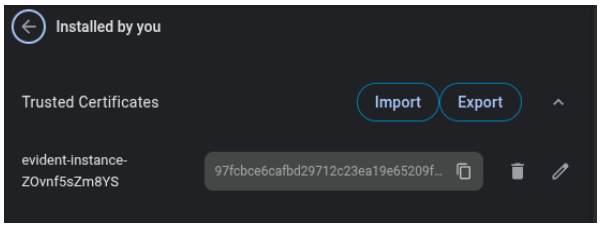
The same use case was also evaluated using the hierarchical certificate configuration instead[9]. The VM owner, deemed as a trusted image distributor for the deployed Confidential Certificate Authorities (CAs), was responsible for deploying the *llama.cpp* loaded confidential VM and submitting the package to an intermediate Confidential CA. The service confidential VM, upon booting, requested a new certificate from the intermediate Confidential CA, which performed the automated remote attestation procedure. All checks were successful, including the matching of the observed software measurements with the previously submitted package. The certificate was then issued, and the worker IK was used to issue the TLS Key certificate, with the Subject Alternative Name `gce-llama-cpp.joaohs.com`.

The user only needed to attest the Confidential Root CA and install its self-signed certificate. The

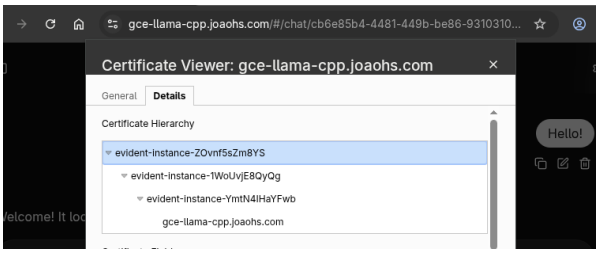
[9] Available at `https://github.com/joao-hs/evident-confidential-certificate-authority`

Figure 6.3: Service's TLS certificate rooted in hierarchical Confidential Certificate Authorities

(a) Installing the Confidential Root CA certificate



(b) Inspecting the presented certificate chain



presented certificate chain, seen in Figure 6.3, started at the TLS Key certificate and went up to the Root IK certificate, which was installed after the attestation. After installing the certificate, no manual step was required.

6.2 Measurement Correctness

One of the main technical claims of the *Evident* framework is that the verifying party can independently derive the expected measurements of a confidential VM before deployment and compare them against the evidence collected from the running instance. This section validates that claim by examining whether the predicted measurements match the actual ones reported by the AMD *Secure Processor* and the virtual TPM across both supported providers.

The measurements were collected during the use case described before in Section 6.1. For each provider, the *Evident* Client's *measure* stage produced the expected PCR values (registers 4, 11, and 12). The *attest* stage then collected the launch digest from AMD *Secure Processor*'s attestation report and the actual PCR values from the virtual TPM's quote. The launch digest is derived independently during the attest stage, using the cloud provider's endorsed firmware binary. Table 6.1 presents the comparison.

Table 6.1: Predicted and reported evidence measurement comparison

		Hardware Evidence (from AMD SEV-SNP)	**Software Evidence** (from virtual TPM)
Amazon Web Services	Predicted	`487266...`	`35ec0a...`
	Reported	`ecb285...`	`35ec0a...`
Google Compute Engine	Predicted	`10f343...`	`1f48f2...`
	Reported	`10f343...`	`1f48f2...`

All predicted measurements matched the corresponding actual values on both providers, except Elastic Compute Cloud's launch digest, due to a recently reported issue[10]. For the AMD *Secure Proces-*

[10] `https://github.com/aws/uefi/issues/19`

sor launch digest, the match confirms that the framework correctly simulates the endorsed firmware's page-loading sequence performed by each provider's hypervisor and that the firmware binary used for prediction corresponds to the one loaded into the confidential VM. For PCR 4, the match confirms that the UKI binary extracted from the image at the *measure* stage is the same binary that the firmware measured and loaded at boot. For PCR 11, the match confirms that the individual sections of the UKI (kernel, *initrd*, kernel command-line containing the *dm-verity* root hash, and others) were measured by the *systemd-stub* as predicted. For PCR 12, the value remains zero on both providers, confirming that no unexpected UKI extensions were applied during boot.

These results were obtained with a specific VM image, specific provider firmware versions, and specific instance configurations. A change in any of these inputs would produce different expected measurements. The framework does not claim that predictions are correct in general; it instead claims that, given the correct inputs, the derivation is deterministic and accurate. The evidence presented here supports that claim for the tested configurations.

6.3 Remote Attestation Report

After each remote attestation, the *Evident* Client produces a report regardless of whether attestation succeeded or failed. The report is a structured record of every verification the framework performed, along with the results and an informative summary of the security guarantees obtained or compromised in each. The verifying party reads the report and applies their own trust model to decide which verifications are relevant and what conclusions to draw. It is *Evident* Client's responsibility to perform all known possible verifications and report their outcomes.

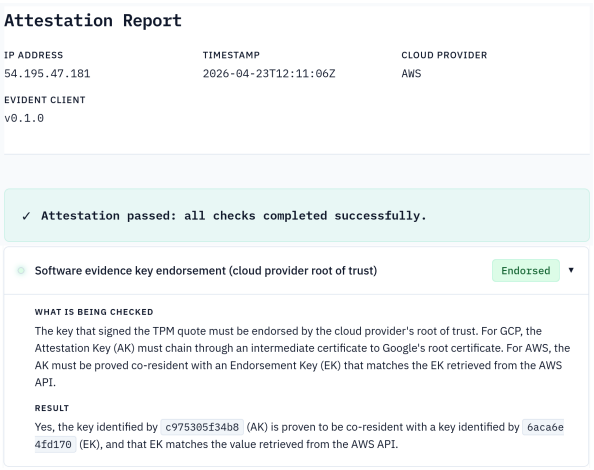
The report is organized as a sequence of checks. Each check corresponds to a specific verification step in the attestation workflow, such as validating the AMD *Secure Processor*'s certificate chain, comparing the launch digest against the expected value, and verifying the TPM quote signature. For each check, the report includes a description of what was verified, the observed values, the expected values (where applicable), and the security guarantee obtained if it passes. This documentation is embedded in the report itself, so the verifying party does not need external references to interpret the results.

A successful attestation means that all checks passed. This does not imply any guarantee beyond what the individual checks cover. If the virtual TPM measurement check passes, it means the TPM's reported values match the expected ones, but whether the TPM implementation itself is trustworthy remains a trust model decision, as discussed in Section 4.1. The report provides the facts that the trust model interprets.

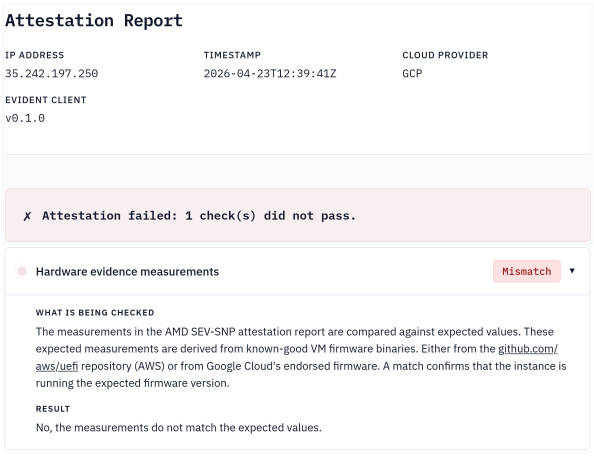
Figure 6.4a shows an excerpt from a report showing that all checks passed, from a confidential VM deployed on Google Compute Engine. Figure 6.4b shows an excerpt of a report where the expected

Figure 6.4: Excerpts from *Evident* remote attestation reports (adapted)

(a) Excerpt from a passed *Evident* remote attestation report



(b) Excerpt from a failed *Evident* remote attestation report



measurements given to the *Evident* Client, intentionally, did not correspond to the attesting target confidential VM image. In the failing case, the report identifies exactly which verification did not match, displays both the expected and observed values, and explains what assurance is no longer available as a consequence. The remaining checks that did pass are still reported and may still be meaningful to a relying party whose trust model does not depend on the failed check.

The full reports for both cases are included in Appendix B.

6.4 Remote Attestation Latency

This section characterizes the execution time of the remote attestation workflow from both the client and server perspectives[11]. The purpose is to identify which phases of the workflow dominate the total latency and whether the framework introduces meaningful overhead beyond the hardware operations it depends on.

Test environment

The *Evident* Client ran on a machine with a 3.5 GHz 6-core AMD Ryzen 5 4500U processor and 16 GB of RAM. The target confidential VMs had 2 virtual CPUs and 8 GB of RAM, using instance type `m6a.large` on Elastic Compute Cloud and instance type `n2d-standard-2` on Google Compute Engine, deployed in regions `eu-west-1` (Ireland) and `europe-west3-a` (Frankfurt), respectively. For each provider, 100 sequential remote attestation workflows were executed against the same running instance. The median and 90th percentile (p90) durations are reported for each task.

[11] Gathered data and scripts to interpret it are available at `https://github.com/joao-hs/evident-evaluation`

Client-side latency

Table 6.2 breaks down the remote attestation workflow as observed by the *Evident* Client. Each row represents a distinct task, and the marked tasks are susceptible to one or more network round-trip times. Task *1* calls the *Evident* Server twice: first, it fetches immediately available information, such as the instance key, and the necessary information to create a TPM credential in the activate-credential protocol (TPM's Attestation Key and Endorsement Key), and other metadata; only then, the AMD *Secure Processor* and TPM evidence are retrieved by calling the *Evident* Server again, which, in turn, calls the device specific interfaces (in Elastic Compute Cloud, the activate-credential protocol is execute here). Task *2* fetches AMD's key distribution service endpoints to retrieve the ARK, the ASK, and the ASVK. Task *3* for Google Compute Engine, gets the AK certificate chain from Google's public key infrastructure. Tasks *4* and *8* verify the signature and certificate chain of the signing key for hardware and software evidence, respectively. Tasks *5* and *9* verify the freshness of the hardware and software evidence, respectively. At these steps, the binding between the measured state and the instance key is also verified. Task *6* retrieves the firmware binary from GitHub releases or Google's Cloud Storage, respectively for Amazon Web Services and Google Cloud Platform. Finally, tasks *7* and *10* derive the expected measurement given the firmware binary (for the hardware evidence) and the expected PCRs file (for the software evidence) that was previously computed directly from the VM image.

Non-network-dependent tasks have near-instant execution times, confirming that the framework's local verification logic (certificate chain verification, digest comparison, PCR recomposition) does not contribute meaningfully to overall latency. The dominant cost is network communication, both with the *Evident* Server and with trusted endpoints that serve certificate chains and endorsement artifacts.

Retrieving evidence from the confidential VM is two to three times slower on Google Compute Engine than on Elastic Compute Cloud. The cause of this difference is examined from the server's perspective in Table 6.3. The total end-to-end remote attestation time, including network delay, is expected to be 1.8 seconds on Elastic Compute Cloud and 4.7 seconds on Google Compute Engine.

The artifacts fetched during tasks *2*, *3*, and *6* are stable and finite across attestation runs for similar instance configurations and could be cached. The measurements reported here do not use caching, so they represent the worst-case latency for each run. The *Terraform* plugin implementation described in Section 5.4 does cache these artifacts when attesting multiple instances using the same provider.

Server-side latency

Table 6.3 shows the same 100 attestation runs from the server's perspective, breaking down the time spent within the *Evident* Server between the moment it receives the client's request and the moment it sends the response.

Table 6.2: *Evident* Client remote attestation workflow timing by task and cloud service provider

#		Amazon Web Services		Google Cloud Platform	
		Median (ms)	p90 (ms)	Median (ms)	p90 (ms)
1	Getting evidence†	364	377	2353	4409
2	Getting trusted certificates from AMD†	722	729	874	1431
3	Getting trusted certificates from Google Cloud Platform†	n/a	n/a	362	707
4	Verifying the signature of the hardware evidence	5	6	5	6
5	Verifying freshness of the hardware evidence	1	1	1	1
6	Getting endorsed artifacts for measurement verification†	670	735	634	1851
7	Verifying measurement of the hardware evidence	27	28	13	15
8	Verifying the signature of the software evidence	2	2	4	4
9	Verifying freshness of the software evidence	1	1	1	1
10	Verifying the measurement of the software evidence	1	1	1	1
	Remote Attestation	**1792**	**1856**	**4740**	**7468**

† The task is susceptible to network delay

Table 6.3: *Evident* Server fetching evidence timing by device and cloud service provider

#		Amazon Web Services		Google Cloud Platform	
		Median (ms)	p90 (ms)	Median (ms)	p90 (ms)
1	Get evidence from AMD *Secure Processor*	14	15	2061	4071
2	Get evidence from virtual TPM	33	36	15	16
	Get evidence	**68**	**77**	**2076**	**4087**

The most significant finding is that retrieving the attestation report from the AMD *Secure Processor* is approximately 30–50 times slower in Google Compute Engine than on Elastic Compute Cloud. This difference accounts for the majority of the overall latency gap between the two providers. Tests were repeated across different regions and instance configurations with similar results. The cause could not

be identified.

In contrast, retrieving the virtual TPM quote is approximately twice as fast in Google Compute Engine as on Elastic Compute Cloud. Since both virtual TPM implementations are proprietary and execute in different contexts (hypervisor-level on both providers), this is likely an implementation difference with no deeper significance, as both are still relatively fast.

6.5 Resource Consumption and Overhead

This section characterizes the resource usage of the *Evident* Server under sustained load. The goal is to determine whether the server, which is expected to run alongside the primary workload, consumes resources that could interfere with the workload's operation.

In practice, the *Evident* Server receives attestation infrequently. A remote attestation is typically performed once per verifying party, and some relying parties may not perform it at all if they trust a verifier who has already done so. The load patterns tested here are deliberately unrealistic: they represent sustained concurrent attestation traffic to establish upper bounds on resource consumption, not to characterize typical behavior.

Test setup

The test environment is the same as described in Section 6.4. For each concurrency level, the specified number of clients continuously requested fresh evidence from the *Evident* Server instance for a sustained period of three minutes. Note that the server process is restricted to a single virtual CPU. Measurements were taken for request latency, throughput, RAM usage, and CPU usage of the server process. CPU usage is reported as a percentage relative to a single core.

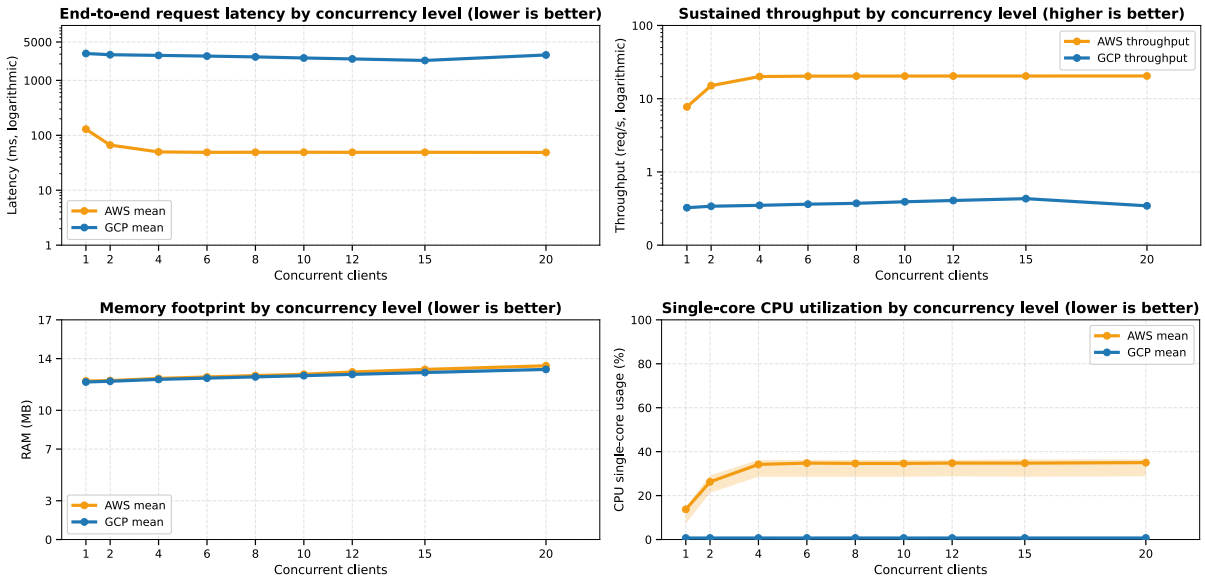
Results

Figure 6.5 presents the results across increasing concurrency levels for both providers.

Latency. On Elastic Compute Cloud, the average latency is approximately 0.050 seconds. On Google Compute Engine, the average latency is approximately 2.500 seconds. The average latency was derived from the throughput.

Throughput. On Elastic Compute Cloud, throughput reaches approximately 20 requests per second at high concurrency. On Google Compute Engine, throughput plateaus at approximately 0.3 requests per second. The rate difference is a direct consequence of the AMD *Secure Processor* report retrieval latency difference identified in Section 6.4.

Figure 6.5: *Evident* Server metrics usage under sustained concurrent load



Memory. RAM usage remains constant across all concurrency levels on both providers, at approximately 15 MB. The server does not allocate additional memory in proportion to the number of concurrent clients.

CPU. On Elastic Compute Cloud, CPU usage rises to approximately 40% of a single core under high concurrency, reflecting the cost of handling multiple concurrent requests and their associated cryptographic operations. On Google Compute Engine, CPU usage remains near 0% regardless of concurrency. This is consistent with the latency analysis: the server spends nearly all its time blocked on the AMD *Secure Processor* device interface, leaving the CPU idle even under load.

Overall. The *Evident* Server is lightweight. Its memory footprint is constant and small, and its CPU usage is bounded well below a single core even under artificial stress. In realistic deployments, where evidence retrieval requests are infrequent, the server's resource consumption is negligible relative to any practical workload. The server is unlikely to interfere with the primary service running alongside it.

Summary

This chapter evaluated the *Evident* framework across three dimensions. The use case validation demonstrated that the framework supports a complete end-to-end workflow for deploying and attesting a real workload, a large language model inference service, on both Elastic Compute Cloud and Google Compute Engine. The measurement correctness results confirmed that the framework's predictions match the actual evidence collected from running confidential VMs across both providers, for the AMD *Secure Processor* launch digest and virtual TPM's PCR values. The remote attestation report was shown to

provide a structured, self-documenting record of every verification performed, with enough information for the verifying party to interpret the results under their own trust model.

The latency analysis revealed that the remote attestation workflow is expected to complete in 1.8 seconds on Elastic Compute Cloud and 4.7 seconds on Google Compute Engine. The difference is dominated by a discrepancy in AMD *Secure Processor* report generation latency between the two providers, a hardware or platform-level difference that the framework cannot influence. The framework's own processing overhead is negligible in both cases. The resource consumption analysis confirmed that the *Evident* Server uses approximately 15 MB of RAM regardless of load, and that its CPU usage stays well within the capacity of a single core. Under realistic conditions, the server's presence alongside the primary workload has no meaningful impact on available resources.

7

Conclusions and Future Work

This thesis addressed the problem of verifying the integrity of confidential Virtual Machines deployed on public cloud infrastructure independently. The main challenge is that current cloud offerings provide the hardware mechanisms for confidential computing, but the tooling to independently predict, reproduce, and verify the measurements produced by those mechanisms is either incomplete or fragmented. Without such tooling, the trust model that confidential VMs are designed to support cannot be fully realized in practice because verifiers must rely on externally supplied reference values or accept first-observed measurements as baselines, neither of which constitutes independent assurance.

The thesis began with a detailed analysis of AMD SEV-SNP confidential VM offerings from Amazon Web Services, Microsoft Azure, and Google Cloud Platform. This analysis confirmed that while all three providers offer memory confidentiality and integrity protections, they differ substantially in how attestation evidence is exposed and in the transparency of the measurement derivation process. Elastic Compute Cloud, in theory, achieves the highest verification depth with its open-sourced firmware, but due to recent issues, the reality falls short, and firmware measurements are not reproducible. Among the three, Google Compute Engine has the highest verification depth, at the time of writing, by providing the deployed firmware binaries for independent verification. Azure Virtual Machines discloses none, and AMD *Secure Processor* evidence has no guarantee of freshness. The virtual TPM implementations frictionlessly hold software measurements; however, no cloud provider offers an independently verifiable

mechanism to attest to the integrity and correctness of the implementation.

The *Evident* framework was designed and implemented to address the identified gaps. Its primary contribution is a shift in how expected measurements are obtained. Instead of consuming reference values from a third party, the verifying party has a convenient path to derive them on its own directly from the artifacts that constitute the VM image. This is made possible by a pipeline that spans the full lifecycle of a confidential VM. In the build stage, a declarative NixOS configuration is compiled into a bootable VM image that uses Unified Kernel Images for deterministic boot measurement and an immutable root filesystem protected by *dm-verity*. In the measurement stage, the image is inspected, and the measurements are derived independently. In the attest stage, evidence is collected from the running instance and evaluated against the predicted measurements, producing a structured report that documents every verification step and its outcome. The framework requires no modification to the deployed workload and is compatible with Elastic Compute Cloud and Google Compute Engine, despite the differences in how these providers expose attestation evidence.

The evaluation demonstrated that the framework produces correct attestation results. The end-to-end workflow was validated through a confidential inference use case, where a large language model inference server was deployed, attested, and accessed over a TLS channel bound to the attestation result. The remote attestation workflow is relatively fast, and the most intensive tasks are network-dependent. The *Evident* Server consumes approximately 15 MB of RAM and minimal CPU, confirming that it can run alongside production workloads without interference.

Beyond the core pipeline, the thesis contributed several extensions that address practical deployment concerns. An image distribution mechanism allows VM image builders to endorse a build alongside its expected measurements and VM image configuration source in a signed manifest, enabling other parties to use the image without repeating the build process or to verify the endorsement independently. A *Terraform* provider plugin integrates attestation into the deploy workflow, allowing VM owners to verify freshly launched instances as part of a single deployment operation. The design of a Confidential Certificate Authority was presented, where attested confidential VMs receive TLS certificates from a certificate authority that succeeds in attesting the receiving one, and that is itself a verified confidential VM, enabling scalable attestation for deployments where individual verification of every instance is impractical. As a contribution to the broader ecosystem, support for Google Compute Engine's launch measurement derivation was added to the open-source sev-snp-measure project, making cross-provider launch digest derivation available via a common, established interface.

The framework also makes explicit the limitations that it cannot resolve from the customer's standpoint. The virtual TPM implementations on all studied providers remain closed-source, and trusting them requires a bounded yet real assumption about a provider-managed component. On Elastic Compute Cloud, the absence of a standard TPM endorsement key certificate limits third-party attestation

scenarios. The framework's response is to collect and report all available evidence unconditionally, so that each verifying party can interpret the results under whatever trust model they adopt.

Several directions remain open for future research and development. The implementation of the Confidential Certificate Authority is unfinished. The web-based client proxy design was presented, but not implemented, which would allow verifiers to have a better experience while interacting with the various framework stages. Image distribution verification automation has yet to be implemented, although all the necessary material is packaged and signed with the manifest. Ideally, the framework would also accommodate workloads that require persistent, confidential data stores within the VM's own filesystem, which could be achieved by pairing disk encryption with an optional overlay filesystem to maintain the measured root filesystem. Azure Virtual Machines are not yet supported by the framework, even though it was designed to be extensible not only in provider options but also in hardware roots of trust. It would be valuable for the framework to monitor runtime integrity to detect vulnerabilities within the trusted system. Finally, reproducible builds would provide greater assurance when distributing image binaries, since anyone could verify and endorse (or denounce) the same resulting binary. Currently, there are no clear guidelines on how to enforce build reproducibility; therefore, it is partially supported on a case-by-case basis.

Bibliography

[1] Advanced Micro Devices, Inc., “AMD SEV-SNP: Strengthening VM isolation with integrity protection and more,” Advanced Micro Devices, Inc., Tech. Rep., 2020, retrieved on March 24th, 2025. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[2] H. Birkholz, N. Smith, D. Thaler, L. Lundblade, and M. Sethi, “Remote attestation procedures architecture,” 2023, rFC 9334. Accessed on May 10th, 2025. [Online]. Available: https://datatracker.ietf.org/doc/rfc9334/

[3] N. Schear, P. T. Cable, T. M. Moyer, B. Richard, and R. Rudd, “Bootstrapping and maintaining trust in the cloud,” in *Proceedings of the 32nd Annual Conference on Computer Security Applications*, ser. ACSAC ’16. New York, NY, USA: Association for Computing Machinery, 2016, pp. 65–77. [Online]. Available: https://doi.org/10.1145/2991079.2991104

[4] Edgeless Systems, “Constellation - The Kubernetes Built for Confidential Computing,” accessed on May 9th, 2025. [Online]. Available: https://www.edgeless.systems/products/constellation

[5] G. Scopelliti, C. Baumann, and J. T. Mühlberg, “Understanding Trust Relationships in Cloud-Based Confidential Computing,” in *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, 2024, pp. 169–176.

[6] J. Eisoldt, A. Galanou, A. Ruzhanskiy, N. Küchenmeister, Y. Baburkin, T. Dai, I. Gudymenko, S. Köpsell, and R. Kapitza, “SoK: A cloudy view on trust relationships of CVMs – How Confidential Virtual Machines are falling short in Public Cloud,” 2025. [Online]. Available: https://arxiv.org/abs/2503.08256

[7] V. Costan and S. Devadas, “Intel SGX Explained,” Cryptology ePrint Archive, Paper 2016/086, 2016. [Online]. Available: https://eprint.iacr.org/2016/086

[8] S. Pinto and N. Santos, “Demystifying Arm TrustZone: A Comprehensive Survey,” *ACM Comput. Surv.*, vol. 51, no. 6, Jan. 2019. [Online]. Available: https://doi.org/10.1145/3291047

[9] P.-C. Cheng, W. Ozga, E. Valdez, S. Ahmed, Z. Gu, H. Jamjoom, H. Franke, and J. Bottomley, “Intel TDX Demystified: A Top-Down Approach,” 2023. [Online]. Available: https://arxiv.org/abs/2303.15540

[10] Advanced Micro Devices, Inc., “AMD Memory Encryption,” Advanced Micro Devices, Inc., Tech. Rep., 2016, retrieved on April 11th, 2025. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/memory-encryption-white-paper.pdf

[11] ——, “Protecting VM register state with SEV-ES,” Advanced Micro Devices, Inc., Tech. Rep., 2017, retrieved on April 23rd, 2025. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/Protecting-VM-Register-State-with-SEV-ES.pdf

[12] S. Berger, R. Caceres, K. A. Goldman, R. Perez, R. Sailer, and L. van Doorn, “vTPM: Virtualizing the Trusted Platform Module,” in *15th USENIX Security Symposium (USENIX Security 06)*. Vancouver, B.C. Canada: USENIX Association, Jul. 2006. [Online]. Available: https://www.usenix.org/conference/15th-usenix-security-symposium/vtpm-virtualizing-trusted-platform-module

[13] *QEMU vTPM Specification*, QEMU Project, official documentation. Accessed on May 29th, 2025. [Online]. Available: https://qemu-project.gitlab.io/qemu/specs/tpm.html

[14] VMware, “vSphere Virtual TPM (vTPM) Questions and Answers,” VMware, retrieved on May 29th, 2025. [Online]. Available: https://www.vmware.com/docs/vsphere-virtual-tpm-vtpm-questions-answers

[15] Advanced Micro Devices, Inc., *SEV Secure Nested Paging Firmware ABI specification*, Advanced Micro Devices, Inc., May 2025, official specification. Revision 1.58. Retrieved on March 24th, 2025. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/56860.pdf

[16] *TCG PC Client Specific TPM Interface Specification*, Trusted Computing Group, March 2013, official specification. Version 1.3. Retrieved on May 29th, 2025. [Online]. Available: https://trustedcomputinggroup.org/resource/pc-client-work-group-pc-client-specific-tpm-interface-specification-tis/

[17] TianoCore Project, “TCG trusted boot chain in EDKII,” accessed on May 25th, 2025. [Online]. Available: https://tianocore-docs.github.io/edk2-TrustedBootChain/release-1.00/3_TCG_Trusted_Boot_Chain_in_EDKII.html

[18] *Amazon Machine Images - Amazon EC2*, Amazon Web Services, official documentation. Accessed on March 26th, 2025. [Online]. Available: https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html

[19] *Create an Azure virtual machine offer on Azure Marketplace using your own image*, Microsoft Corportation, accessed on March 31st, 2025. [Online]. Available: https://learn.microsoft.com/en-us/partner-center/marketplace-offers/azure-vm-use-own-image

[20] *Manually import boot disks - Compute Engine Documentation*, Google Cloud, accessed on March 31st, 2025. [Online]. Available: https://cloud.google.com/compute/docs/import/import-existing-image

[21] Amazon Web Services, "AWS UEFI firmware for AMD SEV-SNP instances," accessed on March 26th, 2025. [Online]. Available: https://github.com/aws/uefi

[22] Google Cloud, "Verify a Confidential VM instance's firmware," accessed on March 31st, 2025. [Online]. Available: https://cloud.google.com/confidential-computing/confidential-vm/docs/verify-firmware

[23] Amazon Web Services, "The Security Design of the AWS Nitro System," Amazon Web Services, Inc., Tech. Rep., February 2024, retrieved on May 25th, 2025. [Online]. Available: https://docs.aws.amazon.com/pdfs/whitepapers/latest/security-design-of-aws-nitro-system/security-design-of-aws-nitro-system.pdf

[24] Microsoft Corporation, "Virtual TPMs in Azure confidential VMs," accessed on March 28th, 2025. [Online]. Available: https://learn.microsoft.com/en-us/azure/confidential-computing/virtual-tpms-in-azure-confidential-vm

[25] H. Pulapaka and Marysia, "OpenHCL: Evolving Azure's virtualization model," September 2024, microsoft Tech Community, Updated Sep 18, 2024. Accessed on May 26th, 2025. [Online]. Available: https://techcommunity.microsoft.com/blog/windowsosplatform/openhcl-evolving-azure%E2%80%99s-virtualization-model/4248345

[26] L. Wilke and G. Scopelliti, "SNPGuard: Remote Attestation of SEV-SNP VMs Using Open Source Tools," in *2024 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*. Los Alamitos, CA, USA: IEEE Computer Society, Jul. 2024, pp. 193–198. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/EuroSPW61312.2024.00026

[27] Decentriq, "Swiss cheese to cheddar: securing AMD SEV-SNP early boot," July 2022, accessed on March 27th, 2025. [Online]. Available: https://www.decentriq.com/article/swiss-cheese-to-cheddar-securing-amd-sev-snp-early-boot

[28] A. Agache, M. Brooker, A. Iordache, A. Liguori, R. Neugebauer, P. Piwonka, and D.-M. Popa, "Firecracker: Lightweight virtualization for serverless applications," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*. Santa

Clara, CA: USENIX Association, Feb. 2020, pp. 419–434. [Online]. Available: https://www.usenix.org/conference/nsdi20/presentation/agache

[29] M. Misono, D. Stavrakakis, N. Santos, and P. Bhatotia, “Confidential VMs Explained: An Empirical Analysis of AMD SEV-SNP and Intel TDX,” *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 8, no. 3, Dec. 2024. [Online]. Available: https://doi.org/10.1145/3700418

[30] QEMU Project, “microvm virtual platform,” accessed on June 3rd, 2025. [Online]. Available: https://www.qemu.org/docs/master/system/i386/microvm.html

[31] Advanced Micro Devices, “AMD Secure VM Service Module (SVSM) Specification,” AMD, Tech. Rep., July 2023, official specification. Revision 1.00. Retrieved on March 27th, 2025. [Online]. Available: https://www.amd.com/content/dam/amd/en/documents/epyc-technical-docs/specifications/58019.pdf

[32] Coconut SVSM Contributors, “Coconut Secure Virtual Machine Service Module (SVSM),” accessed on March 27th, 2025. [Online]. Available: https://github.com/coconut-svsm/svsm

[33] V. Narayanan, C. Carvalho, A. Ruocco, G. Almasi, J. Bottomley, M. Ye, T. Feldman-Fitzthum, D. Buono, H. Franke, and A. Burtsev, “Remote attestation of confidential VMs using ephemeral vTPMs,” in *Proceedings of the 39th Annual Computer Security Applications Conference*, ser. ACSAC ’23. New York, NY, USA: Association for Computing Machinery, 2023, pp. 732–743. [Online]. Available: https://doi.org/10.1145/3627106.3627112

[34] Edgeless Systems, “Contrast: Confidential containers for kubernetes,” https://github.com/edgelesssys/contrast, 2026.

[35] Confidential Containers Project Authors, “Trustee: Attestation and key management service for confidential containers,” https://github.com/confidential-containers/trustee, 2026.

[36] T. Perrin, “The noise protocol framework,” *noiseprotocol, Protocol Revision*, vol. 34, 2018.

[37] E. Dolstra, “The purely functional software deployment model,” Ph.D. dissertation, Utrecht University, Utrecht, The Netherlands, January 2006. [Online]. Available: http://hdl.handle.net/1874/7540

[38] M. Sanft, “Reproducible and immutable OS images with NixOS,” Talk at All Systems Go! 2024, Berlin, Germany, September 2024. [Online]. Available: https://media.ccc.de/v/all-systems-go-2024-251-reproducible-and-immutable-os-images-with-nixos

Appendix A

Listing A.1 contains a manifest of VM image built with *Evident* package mode. `VERSION` is the version of the manifest itself. The Nix version is pinned in `NIX`, even though other versions of Nix should get the same results; the version is pinned for forward compatibility. `SOURCE` unequivocally identifies the source of the image configuration. `ATTR` is the `variation` argument given to *Evident* Client; it chooses which package of the Nix flake is supposed to be built. `DRV` has a filename followed by the digest of the derivation file; the derivation file captures all the inputs (dependencies pinned to their version) to a given flake build; the derivation must match both in filename (which contains a weaker digest) and in content if a user is attempting to reproduce the manifest. Finally, `OUT` contains the digest of the resulting VM image, and `MOUT` the digest of expected measurements of the same image, derivable with *Evident measure* stage.

Listing A.1: *Evident* VM image packaging manifest

```
1 VERSION 1
2 NIX 2.34.6
3 SOURCE git+github.com/joao-hs/evident-llama-cpp@0c3acdc73b3a[...]
4 ATTR ec2
5 DRV a86rjr5kbhh0qyskrvbh92y3dfb02hq0-image-0.0.1.drv sha512:6
    ↪ f3f5f5019fe[...]
6 OUT sha512:1e01ba1f8629[...]
7 MOUT sha512:71341b52054b[...]
```

B

Appendix B

Attestation Report

IP ADDRESS	TIMESTAMP	CLOUD PROVIDER
54.195.47.181	2026-04-23T12:11:06Z	AWS

EVIDENT CLIENT
v0.1.0

✓ **Attestation passed: all checks completed successfully.**

01 Evidence Integrity

The client retrieves the additional artifacts bundle and the evidence bundle from the Evident server and verifies their structural integrity and digital signatures.

Additional artifacts bundle retrieval — Fetched

WHAT IS BEING CHECKED
The additional artifacts bundle contains supplementary data required for attestation verification. This check confirms whether the client successfully retrieved it from the Evident server.

RESULT
Yes, the additional artifacts bundle was fetched from the Evident server.

Additional artifacts bundle signature — Valid

WHAT IS BEING CHECKED
The additional artifacts bundle is digitally signed by the instance key. This check verifies the signature to confirm the bundle has not been tampered with in transit and originates from the expected instance.

RESULT
Yes, the additional artifacts bundle was correctly signed by the instance key identified by `7fed76541019`.

Additional artifacts bundle contents validity — Valid

WHAT IS BEING CHECKED
Beyond signature validity, the contents of the additional artifacts bundle are inspected for structural correctness and completeness. This ensures all expected artifacts are present and well-formed.

RESULT
Yes, all contents of the additional artifacts bundle are structurally valid and complete.

Evidence bundle retrieval — Fetched

WHAT IS BEING CHECKED
The evidence bundle contains the hardware and software attestation evidence produced by the instance. This check confirms whether the client successfully retrieved it from the Evident server.

RESULT
Yes, the evidence bundle was fetched from the Evident server.

Evidence bundle signature — Valid

WHAT IS BEING CHECKED
The evidence bundle is digitally signed by the instance key. This check verifies the signature to confirm the bundle has not been tampered with in transit and originates from the expected instance.

RESULT
Yes, the evidence bundle was correctly signed by the instance key identified by `7fed76541019`.

Hardware evidence presence — Present

WHAT IS BEING CHECKED
The evidence bundle must include a hardware evidence item — specifically, an AMD SEV-SNP attestation report in the expected format. This check confirms its presence and format correctness.

RESULT
Yes, the evidence bundle includes an AMD SEV-SNP attestation report with the correct format.

Software evidence presence — Present

Attestation Report

IP ADDRESS	TIMESTAMP	CLOUD PROVIDER
35.242.197.250	2026-04-23T12:39:41Z	GCP

EVIDENT CLIENT
v0.1.0

✗ **Attestation failed: 1 check(s) did not pass.**

01 Evidence Integrity

The client retrieves the additional artifacts bundle and the evidence bundle from the Evident server and verifies their structural integrity and digital signatures.

Additional artifacts bundle retrieval — Fetched

WHAT IS BEING CHECKED
The additional artifacts bundle contains supplementary data required for attestation verification. This check confirms whether the client successfully retrieved it from the Evident server.

RESULT
Yes, the additional artifacts bundle was fetched from the Evident server.

Additional artifacts bundle signature — Valid

WHAT IS BEING CHECKED
The additional artifacts bundle is digitally signed by the instance key. This check verifies the signature to confirm the bundle has not been tampered with in transit and originates from the expected instance.

RESULT
Yes, the additional artifacts bundle was correctly signed by the instance key identified by `10c3e9a967ab`.

Additional artifacts bundle contents validity — Valid

WHAT IS BEING CHECKED
Beyond signature validity, the contents of the additional artifacts bundle are inspected for structural correctness and completeness. This ensures all expected artifacts are present and well-formed.

RESULT
Yes, all contents of the additional artifacts bundle are structurally valid and complete.

Evidence bundle retrieval — Fetched

WHAT IS BEING CHECKED
The evidence bundle contains the hardware and software attestation evidence produced by the instance. This check confirms whether the client successfully retrieved it from the Evident server.

RESULT
Yes, the evidence bundle was fetched from the Evident server.

Evidence bundle signature — Valid

WHAT IS BEING CHECKED
The evidence bundle is digitally signed by the instance key. This check verifies the signature to confirm the bundle has not been tampered with in transit and originates from the expected instance.

RESULT
Yes, the evidence bundle was correctly signed by the instance key identified by `10c3e9a967ab`.

Hardware evidence presence — Present

WHAT IS BEING CHECKED
The evidence bundle must include a hardware evidence item — specifically, an AMD SEV-SNP attestation report in the expected format. This check confirms its presence and format correctness.

RESULT
Yes, the evidence bundle includes an AMD SEV-SNP attestation report with the correct format.

Software evidence presence — Present

Comparison Segment 1

WHAT IS BEING CHECKED

The evidence bundle must include a software evidence item — specifically, a TPM quote in the expected format. This check confirms its presence and format correctness.

RESULT

Yes, the evidence bundle includes a TPM quote with the correct format.

Hardware evidence signature — Valid

WHAT IS BEING CHECKED

The AMD SEV-SNP attestation report carries a cryptographic signature. This check verifies that the signature is valid, which confirms the report was produced by a genuine AMD SEV-SNP platform and has not been tampered with.

RESULT

Yes, the AMD SEV-SNP attestation report is correctly signed by a key identified by `2c1d3e61d563`.

Software evidence signature — Valid

WHAT IS BEING CHECKED

The TPM quote carries a cryptographic signature. This check verifies that the signature is valid, which confirms the quote was produced by a genuine TPM and has not been tampered with.

RESULT

Yes, the TPM quote is correctly signed by a key identified by `c975305f34b8`.

AMD SEV-SNP processor model — Milan

WHAT IS BEING CHECKED

The processor model is identified from the AMD SEV-SNP attestation report. This is an informational check — it reports which AMD processor family is running the confidential VM.

RESULT

02 Evidence Trustworthiness

The client verifies that the keys used to sign the hardware and software evidence are endorsed by their respective root of trust, and that each evidence item is fresh (not replayed).

Hardware evidence key endorsement (AMD root of trust) — Endorsed

WHAT IS BEING CHECKED

The key that signed the AMD SEV-SNP attestation report must chain back to AMD's root of trust. The signing key (VCEK or VLEK) must be endorsed by the AMD SEV Key (ASK) or AMD SEV Versioned Key (ASVK), which in turn must be endorsed by the AMD Root Key (ARK). The trusted certificates are retrieved from AMD's publicly available KDS endpoint.

RESULT

Yes, the AMD SEV-SNP attestation report is signed by VLEK identified by `2c1d3e61d563`, endorsed by ASVK identified by `270203010001`, which is endorsed by ARK identified by `ab0203010001`.

Hardware evidence freshness — Fresh

WHAT IS BEING CHECKED

A unique identifier (nonce) generated by the client is included in the attestation request. The same identifier must appear in the AMD SEV-SNP attestation report. A mismatch would indicate that the report is stale or has been replayed from a previous attestation session.

RESULT

Yes, the unique identifier generated by the client matches the identifier included in the AMD SEV-SNP attestation report.

Software evidence key endorsement (cloud provider root of trust) — Endorsed

WHAT IS BEING CHECKED

The key that signed the TPM quote must be endorsed by the cloud provider's root of trust. For GCP, the Attestation Key (AK) must chain through an intermediate certificate to Google's root certificate. For AWS, the AK must be proved co-resident with an Endorsement Key (EK) that matches the EK retrieved from the AWS API.

RESULT

Yes, the key identified by `c975305f34b8` (AK) is proven to be co-resident with a key identified by `6aca6e4fd170` (EK), and that EK matches the value retrieved from the AWS API.

Software evidence freshness — Fresh

WHAT IS BEING CHECKED

The evidence bundle must include a software evidence item — specifically, a TPM quote in the expected format. This check confirms its presence and format correctness.

RESULT

Yes, the evidence bundle includes a TPM quote with the correct format.

Hardware evidence signature — Valid

WHAT IS BEING CHECKED

The AMD SEV-SNP attestation report carries a cryptographic signature. This check verifies that the signature is valid, which confirms the report was produced by a genuine AMD SEV-SNP platform and has not been tampered with.

RESULT

Yes, the AMD SEV-SNP attestation report is correctly signed by a key identified by `0b9a2890c82f`.

Software evidence signature — Skipped

WHAT IS BEING CHECKED

The TPM quote carries a cryptographic signature. This check verifies that the signature is valid, which confirms the quote was produced by a genuine TPM and has not been tampered with.

RESULT

Skipped: Could not be evaluated due to a prior check failure.

AMD SEV-SNP processor model — Milan

WHAT IS BEING CHECKED

The processor model is identified from the AMD SEV-SNP attestation report. This is an informational check — it reports which AMD processor family is running the confidential VM.

RESULT

02 Evidence Trustworthiness

The client verifies that the keys used to sign the hardware and software evidence are endorsed by their respective root of trust, and that each evidence item is fresh (not replayed).

Hardware evidence key endorsement (AMD root of trust) — Endorsed

WHAT IS BEING CHECKED

The key that signed the AMD SEV-SNP attestation report must chain back to AMD's root of trust. The signing key (VCEK or VLEK) must be endorsed by the AMD SEV Key (ASK) or AMD SEV Versioned Key (ASVK), which in turn must be endorsed by the AMD Root Key (ARK). The trusted certificates are retrieved from AMD's publicly available KDS endpoint.

RESULT

Yes, the AMD SEV-SNP attestation report is signed by VCEK identified by `0b9a2890c82f`, endorsed by ASK identified by `270203010001`, which is endorsed by ARK identified by `ab0203010001`.

Hardware evidence freshness — Fresh

WHAT IS BEING CHECKED

A unique identifier (nonce) generated by the client is included in the attestation request. The same identifier must appear in the AMD SEV-SNP attestation report. A mismatch would indicate that the report is stale or has been replayed from a previous attestation session.

RESULT

Yes, the unique identifier generated by the client matches the identifier included in the AMD SEV-SNP attestation report.

Software evidence key endorsement (cloud provider root of trust) — Skipped

WHAT IS BEING CHECKED

The key that signed the TPM quote must be endorsed by the cloud provider's root of trust. For GCP, the Attestation Key (AK) must chain through an intermediate certificate to Google's root certificate. For AWS, the AK must be proved co-resident with an Endorsement Key (EK) that matches the EK retrieved from the AWS API.

RESULT

Skipped: Could not be evaluated due to a prior check failure.

Software evidence freshness — Skipped

Comparison Segment 2

WHAT IS BEING CHECKED

A nonce generated by the client is included in the TPM quote request. The same nonce must appear in the returned TPM quote. A mismatch would indicate that the quote is stale or has been replayed from a previous attestation session.

RESULT

Yes, the nonce generated by the client matches the nonce included in the TPM quote.

03 Measurement Reproducibility

The client compares the measurements embedded in the hardware and software evidence against expected values derived from known-good reference binaries — either provided as input or retrieved at runtime from publicly available endpoints.

Hardware evidence measurements — Mismatch

WHAT IS BEING CHECKED

The measurements in the AMD SEV-SNP attestation report are compared against expected values. These expected measurements are derived from known-good VM firmware binaries. Either from the github.com/aws/uefi repository (AWS) or from Google Cloud's endorsed firmware. A match confirms that the instance is running the expected firmware version.

RESULT

No, this matches the known issue github.com/aws/uefi/issues/19. It may indicate a version mismatch between the published firmware and the production firmware.

Software evidence measurements — Match

WHAT IS BEING CHECKED

The composite digest over TPM PCRs 4, 11, and 12 from the TPM quote is compared against the expected value provided as input to the Evident client. If the expected value was derived from the deployed VM image, a match confirms that the kernel, filesystem, and included services at boot time correspond exactly to the expected software components.

RESULT

Yes, the PCR digest (PCRs 4, 11, 12) matches the expected value provided as input. If derived from the deployed VM image, this confirms the kernel, filesystem, and services at boot time correspond exactly to the expected software components.

04 Bindings

Both hardware and software evidence incorporate the instance key so that it is bound to the endorsed measurements. An instance with different firmware or software at boot time could not produce the same binding.

Instance key bound to hardware evidence — Bound

WHAT IS BEING CHECKED

The instance key is incorporated into the AMD SEV-SNP attestation report's REPORT_DATA field. This binds the key to the hardware-measured boot state: an instance running different firmware could not produce this binding, which prevents key migration to a compromised instance.

RESULT

Yes, the key identified by `7fed76541019` is incorporated into the hardware evidence. An instance with unexpected firmware measured at boot time could not generate the same result.

Instance key bound to software evidence — Bound

WHAT IS BEING CHECKED

The instance key is incorporated into the TPM quote's qualifying data. This binds the key to the software-measured boot state: an instance running different software could not produce this binding, which prevents key migration to a compromised instance.

RESULT

Yes, the key identified by `7fed76541019` is incorporated into the software evidence. An instance with unexpected software measured at boot time could not generate the same result.

WHAT IS BEING CHECKED

A nonce generated by the client is included in the TPM quote request. The same nonce must appear in the returned TPM quote. A mismatch would indicate that the quote is stale or has been replayed from a previous attestation session.

RESULT

Skipped: Could not be evaluated due to a prior check failure.

03 Measurement Reproducibility

The client compares the measurements embedded in the hardware and software evidence against expected values derived from known-good reference binaries — either provided as input or retrieved at runtime from publicly available endpoints.

Hardware evidence measurements — Mismatch

WHAT IS BEING CHECKED

The measurements in the AMD SEV-SNP attestation report are compared against expected values. These expected measurements are derived from known-good VM firmware binaries. Either from the github.com/aws/uefi repository (AWS) or from Google Cloud's endorsed firmware. A match confirms that the instance is running the expected firmware version.

RESULT

No, the measurements do not match the expected values.

Software evidence measurements — Skipped

WHAT IS BEING CHECKED

The composite digest over TPM PCRs 4, 11, and 12 from the TPM quote is compared against the expected value provided as input to the Evident client. If the expected value was derived from the deployed VM image, a match confirms that the kernel, filesystem, and included services at boot time correspond exactly to the expected software components.

RESULT

Skipped: Could not be evaluated due to a prior check failure.

04 Bindings

Both hardware and software evidence incorporate the instance key so that it is bound to the endorsed measurements. An instance with different firmware or software at boot time could not produce the same binding.

Instance key bound to hardware evidence — Bound

WHAT IS BEING CHECKED

The instance key is incorporated into the AMD SEV-SNP attestation report's REPORT_DATA field. This binds the key to the hardware-measured boot state: an instance running different firmware could not produce this binding, which prevents key migration to a compromised instance.

RESULT

Yes, the key identified by `10c3e9a967ab` is incorporated into the hardware evidence. An instance with unexpected firmware measured at boot time could not generate the same result.

Instance key bound to software evidence — Skipped

WHAT IS BEING CHECKED

The instance key is incorporated into the TPM quote's qualifying data. This binds the key to the software-measured boot state: an instance running different software could not produce this binding, which prevents key migration to a compromised instance.

RESULT

Skipped: Could not be evaluated due to a prior check failure.

Comparison Segment 3