Binding Application-Layer Provenance to CPU-rooted Attestation in Confidential VMs via a Kernel-resident Pseudo-CA

1st Zen Ishikura
NTT, Inc., Japan

2nd Sakae Chikara
NTT, Inc., Japan

3rd Fumiaki Kudoh
NTT, Inc., Japan

4th Gen Takahashi
NTT, Inc., Japan

5th Hiroki Itoh
NTT, Inc., Japan

6th Kuniyasu Suzaki
Institute of Information Security, Japan

Contact: zen.ishikura@ntt.com

***Abstract*—Confidential Computing (CC) uses Trusted Execution Environments (TEEs) to protect data in use even from highly privileged entities. In distributed CC, multiple Confidential Virtual Machines (CVMs) cooperate over secure channels, so a relying party must judge not only whether a CVM booted on genuine hardware but also whether the intended in-guest application constructed the attestation report. External certificate authorities (CAs) and attested Transport Layer Security (TLS) authenticate endpoints, but they do not show which in-guest component generated the application key or computed guest-chosen `REPORT_DATA` from boot-time evidence. We propose a boot-time provenance-binding method for CVMs. A kernel-resident Pseudo-CA acts as a measured trust anchor: it generates boot-scoped key material, records key and event metadata in Linux IMA (Integrity Measurement Architecture) evidence, certifies a user-space Agent key, and binds this lineage to PCR (Platform Configuration Register) values, launch measurement, nonce, and measurement list evidence. The Agent hashes an extended evidence bundle into the 64-byte (512-bit) `REPORT_DATA` field, obtains a CPU quote, and returns the quote with Pseudo-CA-signed evidence. A Verifier checks quote validity, evidence consistency, key delegation, and boot-order constraints. Our claim is limited to boot-time lineage and identification of the report-construction entity; post-boot compromise, full runtime correctness, and side-channel resistance are out of scope. A preliminary prototype confirms that the proposed evidence acquisition path can be implemented on our current CVM stack.**

***Index Terms*—Trusted Execution Environment, Confidential VM, Remote Attestation, provenance, trust chain**

I. Introduction

Confidential Computing (CC) isolates sensitive computation from highly privileged external entities through Trusted Execution Environments (TEEs). Confidential Virtual Machines (CVMs), namely virtual-machine-based TEEs, are attractive for cloud deployment because they protect relatively unmodified operating systems and middleware. AMD SEV-SNP (Secure Encrypted Virtualization – Secure Nested Paging) is a representative VM-based TEE with guest-attestation support [1].

Our target setting is distributed CC, where multiple CVMs are connected through secure channels across servers, sites, or data centers. A relying party must then judge not only whether a remote CVM booted on genuine hardware, but also whether the intended in-guest application terminated the channel and assembled the attestation report.

A natural question is whether conventional certificate chains or attested TLS are enough. They are useful but incomplete here: an external CA can certify a communication key, and attested TLS can bind a channel to a TEE, but neither explains which in-guest component generated the key, constructed guest-chosen `REPORT_DATA`, or derived the report from boot-time evidence. Our goal is not to replace TLS, but to add provenance constraints to the attestation report.

Threat model. We assume that the hardware root of trust is correct, but we do not trust platform-side privileges once compromised. The attacker has obtained platform-provider privileges, such as the host operating system, hypervisor, or resource-management tooling, or is an equally powerful malicious insider. The attack goal is to modify the deployment or evidence generation path so that a CVM connects to an unintended peer or presents a report assembled by an unintended in-guest application. We focus on boot-time lineage and identification of the report-construction entity; post-boot compromise, full runtime correctness, and side channels remain out of scope.

We address this setting by placing a kernel-resident Pseudo-CA in the CVM and using it as a measured trust anchor. Depending on deployment, its code is covered by launch measurement or by IMA/PCR evidence for a loaded module. It records boot-generated key and event metadata in Linux IMA, and its signatures provide the provenance basis for user-space report construction. In this paper, *key and event metadata* denotes public verification keys, certifications or signatures, signing-request records, and event labels recorded in measured evidence; it excludes private keys. The Agent constructs and sends the report, but its authority derives from the Agent key certified by the Pseudo-CA and evidence that the Verifier can check.

The contributions are threefold. First, we identify why

a simple TLS/CA composition leaves an application-layer provenance gap inside a CVM. Second, we design a Pseudo-CA-rooted Agent model in which a measured kernel component delegates report construction to a user-space Agent. Third, we define an evidence format and verification procedure that use the limited 64-byte (512-bit) `REPORT_DATA` field to bind CPU quote contents to Linux IMA measurements, PCR values, launch measurement, nonce, and Pseudo-CA and Agent public verification keys.

II. Background and Related Work

The closest related line of work is attested TLS. RA-TLS by Knauth *et al.* integrates Remote Attestation (RA) into TLS so that the channel endpoint can be tied to a TEE during connection establishment [2]. Weinhold *et al.* later argued that certificate-based TLS assurances and attestation-based assurances should coexist as additive guarantees [3]. The more recent *Identity Crisis in Confidential Computing* analysis shows that ambiguous notions of identity in attested TLS can lead to vulnerabilities [4]. Our concern is analogous but shifted inside the CVM: even after an attested endpoint is established, the identity of the in-CVM entity that generated the key and assembled `REPORT_DATA` can remain ambiguous. We therefore complement attested TLS rather than replace it.

AMD SEV-SNP allows a guest to include 64 bytes of guest-chosen data in an attestation report, for example a public key or digest bound to the quote [5]. The size is sufficient for a cryptographic digest, but not for structured evidence. Linux IMA records measurements into a runtime measurement list and supports extensible templates such as `ima-buf`, which can carry key and event metadata in tamper-evident logs [6]. These properties make IMA a natural carrier for boot-time provenance checked against a digest committed in the CPU quote. Stronger post-boot guarantees, such as runtime monitoring or periodic re-attestation, remain complementary.

III. Problem Statement

We assume that the boot state of a CVM up to the kernel is already covered by the existing CPU-rooted chain of trust, represented by the CPU quote and launch measurement. We also assume that secure channels can be established using conventional TLS or attested TLS. These guarantees do not by themselves identify the in-guest application that assembled `REPORT_DATA`, nor do they provide an externally checkable provenance path from boot-time evidence to the application-layer data included in the report.

A straightforward baseline is to combine TLS certification with remote attestation: an application presents a certificate to authenticate a channel endpoint, while attestation proves the integrity of the underlying TEE. This baseline is easy to explain and should be considered first, but it leaves two gaps in distributed CC, where each CVM hosts a user-space Agent that terminates a secure channel and constructs attestation reports for a peer or the Verifier. We do not claim that CVMs should be tied to a physical host; the target is instead to identify, for a given boot instance, which in-guest entity produced the evidence consumed by the relying party.

The two main gaps, shown in Fig. 1, are the following.

Gap 1: Lack of binding between key generation and boot-time measurements. When certificates are provisioned by an external CA, the certified key is not guaranteed to be generated under the measured boot state of the CVM. Even if attestation proves the integrity of the CVM, it does not ensure that the key used by the application originates from that measured state.

Gap 2: Ambiguity of the report-construction entity. In existing attested TLS designs, key generation and attestation are not tightly coupled. It can therefore remain unclear whether the intended in-guest component generated the application key and constructed the report.

This ambiguity is amplified by `REPORT_DATA`, which allows arbitrary in-guest applications to include application-defined values. The challenge is therefore to bind the report-construction entity and the application-layer evidence path to CPU-rooted attestation without changing the ordinary CVM execution model. In other words, the CPU quote authenticates the bytes placed in `REPORT_DATA`, but it does not authenticate the in-guest process that selected those bytes. The missing link is therefore not quote integrity, but provenance from measured boot-time events to the application-layer value committed in the quote. Our objective is narrow: the Verifier should reject a report unless the key material, measurement evidence, and `REPORT_DATA` digest form a consistent lineage rooted in measured boot-time events.

IV. Proposed Method

A. Overview and Evidence Path

As shown on the right side of Fig. 1, the proposal consists of three elements. The **Pseudo-CA**, placed in the CVM kernel, generates its own key pair at boot time, self-signs its verification key, and certifies a boot-scoped Agent verification key. It emits key and event metadata into the IMA log, for example through `ima-buf` records, so that verification is based on kernel-measured events rather than self-declared user-space data. When built into the kernel image, the Pseudo-CA is covered by the SEV-SNP launch measurement; as a loadable module, its loading is captured by Secure Boot and/or Measured Boot and reflected in IMA/PCR evidence.

This design closes Gap 1 by making key generation part of the boot-measured evidence path. The Pseudo-CA private key remains within the kernel-side trust anchor for the boot instance, while only key and event metadata are exposed. Thus, the key generation event and corresponding public key become measured claims rather than self-declared user-space data.

The **Agent**, running in CVM user space, collects the IMA measurement list, PCR values, launch measurement, nonce, and Pseudo-CA and Agent public verification keys after boot. In SEV-SNP, `REPORT_DATA` is limited to 64 bytes (512 bits). To accommodate larger evidence, the Agent hashes an extended evidence bundle E into a fixed-size digest $D = H(E)$ and embeds D into `REPORT_DATA`. The bundle includes

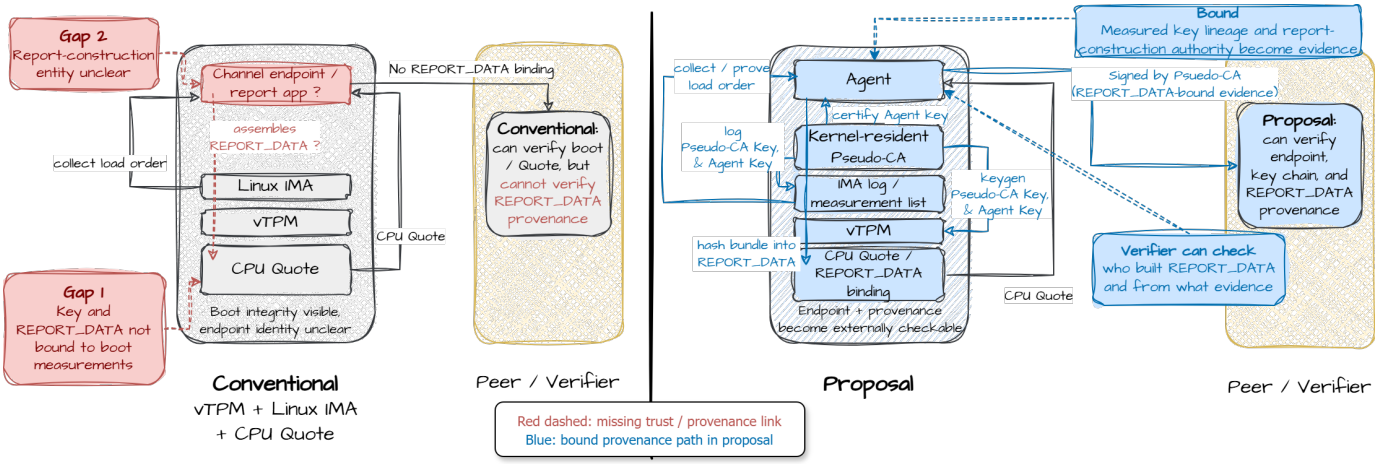

Fig. 1. Combined view of the conventional trust boundary and the proposed extension. On the left, conventional vTPM (virtual Trusted Platform Module), Linux IMA (Integrity Measurement Architecture), and a CPU quote provide boot-time evidence up to the CVM kernel. However, two gaps remain: **Gap 1**, the application key and the `REPORT_DATA` value are not externally bound to boot-time measurements; and **Gap 2**, the report-construction entity, namely the intended endpoint/report application, cannot be identified. On the right, the proposed kernel-resident Pseudo-CA certifies the Agent key, records key and event metadata into the IMA log and measurement list, and enables the Verifier to check both the report-generation endpoint and the provenance path from boot-time evidence to `REPORT_DATA`.

nonce, key and event metadata, and key certifications, and is transmitted alongside the quote. Thus, `REPORT_DATA` becomes a cryptographic commitment to structured provenance evidence. The Agent then submits a signing request over E to the Pseudo-CA. The Pseudo-CA signs the bundle submitted by the Agent, making both user-space report construction and kernel-side approval externally verifiable.

The **Verifier**, outside the CVM, checks the quote and evidence bundle against the expected software configuration, nonce, and boot parameters. It asks not only whether the CVM is genuine, but also which entity built the report and whether that entity's key and evidence follow the intended boot-time lineage. Because launch measurement is CPU-measured rather than guest-supplied, the kernel-side trust anchor can be assessed independently of the user-space Agent. This closes Gap 2 by making report-construction authority explicit and externally checkable.

The evidence bundle is self-contained but not self-trusting: the Verifier recomputes the `REPORT_DATA` digest, replays measurement evidence against PCR values, and checks that the Pseudo-CA and Agent keys appear in the expected measured events.

B. Report Structure and Verification

Our current format supports an *initial report* generated after startup and an *additional report* generated later. The initial report contains the CPU attestation report, the digest bound into `REPORT_DATA`, the Pseudo-CA public key and self-signature, the measurement list, PCR values, and key and event metadata. The additional report extends this with the Agent public key, signing events triggered by the Agent, and continuity information showing that the same lineage generated both reports. The initial report is used to establish the boot-instance root of provenance, namely that the Pseudo-CA key and related events originate from measured boot-time evidence. The additional report is used to bind a later Agent-submitted evidence bundle to that already established root. The additional report carries a signature associated with the certified Agent key over the CPU quote and the extended evidence bundle, followed by the Pseudo-CA signature over the same material. Through this process, the kernel-resident Pseudo-CA provides a verifiable basis that the data processed by the user-space Agent is tied to the measured boot-time lineage.

The verification procedure closes the two gaps in the same order as the problem statement. For Gap 1, the Verifier checks that key generation events and public verification keys are present in measured evidence and consistent with PCR values and launch measurement. For Gap 2, it checks that the signature associated with the certified Agent key covers the CPU quote and the extended evidence bundle, that the Pseudo-CA signed the same material submitted by the Agent, that the Agent key is certified by the Pseudo-CA, and that the digest in `REPORT_DATA` corresponds to the evidence bundle assembled by that Agent.

Verification consists of five checks: (1) verify the SEV-SNP quote and recomputed `REPORT_DATA` digest; (2) confirm from the measurement list that the Pseudo-CA key, Agent key, and relevant events appear in the expected order; (3) verify the Pseudo-CA self-signature, certification of the Agent key, and the Agent-key and Pseudo-CA signatures over the quote-and-evidence material; (4) confirm that nonce, launch measurement, PCR values, and reference values are mutually consistent; and (5) if an additional report is present, confirm continuity between the reports. These checks bind the report-construction entity, boot-time provenance, and CPU quote.

V. Preliminary Evidence Acquisition and Evaluation Outlook

We implemented the evidence acquisition path needed by the proposal. The prototype retrieved the IMA measurement list and PCR index 10, generated `REPORT_DATA` from nonce, public key hashes, and hashed extended evidence, obtained a CPU attestation report, requested signatures from the Pseudo-CA kernel API, and emitted initial/additional report files. This is a feasibility check of the evidence path, not a claim that implementation alone is surprising.

The prototype also revealed a practical ordering issue: if PCR values are read through a command that itself changes the measurement list, PCR replay and measurement list verification can diverge, so direct file access or careful acquisition order was necessary. For the scope of this short paper, the normal case should demonstrate quote verification, key chain verification, measurement list matching, and nonce matching. Negative tests should include Agent replacement, omission of key and event metadata, measurement list tampering, and nonce mismatch, each mapped to the detection step. Overhead should report report generation time, verification time, and report size increase.

The purpose of this evaluation is not to compare the performance of alternative attestation systems, but to show that the proposed evidence path can be realized and that each intended inconsistency is rejected at a well-defined verification step. In the normal case, the Verifier should accept only when the CPU quote, recomputed `REPORT_DATA` digest, measurement list replay, PCR values, nonce, and Pseudo-CA/Agent key chain are all consistent. In the negative cases, Agent replacement should fail at Agent key certification or event order checking, omission of key and event metadata should fail at evidence-bundle reconstruction, measurement list tampering should fail at PCR replay, and nonce mismatch should fail freshness checking. This evaluation therefore supports feasibility and detectability, while leaving performance at deployment scale and runtime compromise resistance to future work.

VI. Discussion

The scope is boot-time authenticity. If the Agent is compromised after producing evidence, the binding no longer proves runtime correctness; runtime monitoring, policy enforcement, periodic re-attestation, or stronger compartmentalization are complementary. Recent enclave-within-CVM work explores a related direction for stronger isolation inside a CVM [7].

A conventional CA or attested TLS workflow can certify a channel key or TEE endpoint. Our mechanism is needed when the relying party also wants evidence that the key and report were produced by the intended in-CVM entity under measured boot state. A CA-based design could use our evidence as input: the key generation event is measured, the Agent key is certified by the measured Pseudo-CA, and the report is committed through `REPORT_DATA`. We do not tie a CVM to a physical host; we identify a boot instance and the measured events within it.

The Pseudo-CA and Agent keys are boot-scoped. The Pseudo-CA private key is generated and retained in the kernel-side trust anchor, and is used to self-sign its verification key, certify the Agent verification key, and approve evidence submitted by the Agent. The Agent key is a delegated key associated with the user-space Agent and used to bind evidence submitted by the Agent to the report construction path approved by the Pseudo-CA; its private material need not be exposed through the evidence path, and should be retained in protected storage or used through a Pseudo-CA API. The Verifier accepts the Agent public key only when it is certified by the measured Pseudo-CA and appears in measured evidence. TPM-backed protection can further harden private key handling, but the logical binding comes from the measured Pseudo-CA and its key chain.

This safety claim is therefore conditional on the boot-time trust boundary. The proposal does not claim that a private key remains secure after arbitrary kernel compromise, nor that the Agent remains trustworthy after measurement. Instead, it ensures that the Verifier can distinguish evidence produced through the measured Pseudo-CA/Agent lineage from evidence assembled by an unrelated in-guest application or by a key not rooted in the measured boot instance.

VII. Conclusion

We proposed a method that binds application-layer provenance to CPU-rooted attestation in CVMs using a kernel-resident Pseudo-CA, Linux IMA, PCR values, and a structured evidence bundle hashed into `REPORT_DATA`. The method helps relying parties judge which in-guest entity built a report and from what boot-time evidence. Preliminary results indicate that the evidence acquisition path is feasible. Future work includes negative tests, overhead evaluation, composition with attested TLS, and stronger runtime enforcement for Agent identity and private key handling.

References

[1] AMD, "SEV Secure Nested Paging Firmware ABI Specification," Rev. 1.58, 2025.

[2] T. Knauth *et al.*, "Integrating Remote Attestation with Transport Layer Security," arXiv:1801.05863, 2018.

[3] C. Weinhold *et al.*, "Separate but Together: Integrating Remote Attestation into TLS," in *Proc. USENIX ATC*, 2025.

[4] M. U. Sardar, M. Moustafa, and T. Aura, "Identity Crisis in Confidential Computing: Formal Analysis of Attested TLS," in *Proc. ACM ASIA CCS*, 2026, doi: 10.1145/3779208.3785387.

[5] AMD, "SEV-SNP Platform Attestation Using VirTEE/SEV," Rev. 1.2, 2023.

[6] Linux Kernel Documentation, "IMA Template Management Mechanism."

[7] W. Wang *et al.*, "The Road to Trust: Building Enclaves within Confidential VMs," in *Proc. NDSS*, 2025.