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

Zen Ishikura*, Sakae Chikara*, Fumiaki Kudoh*, Gen Takahashi*, Hiroki Itoh*, Kuniyasu Suzaki**
* NTT, Inc., ** Institute of Information Security(IISEC)

22nd June 2026

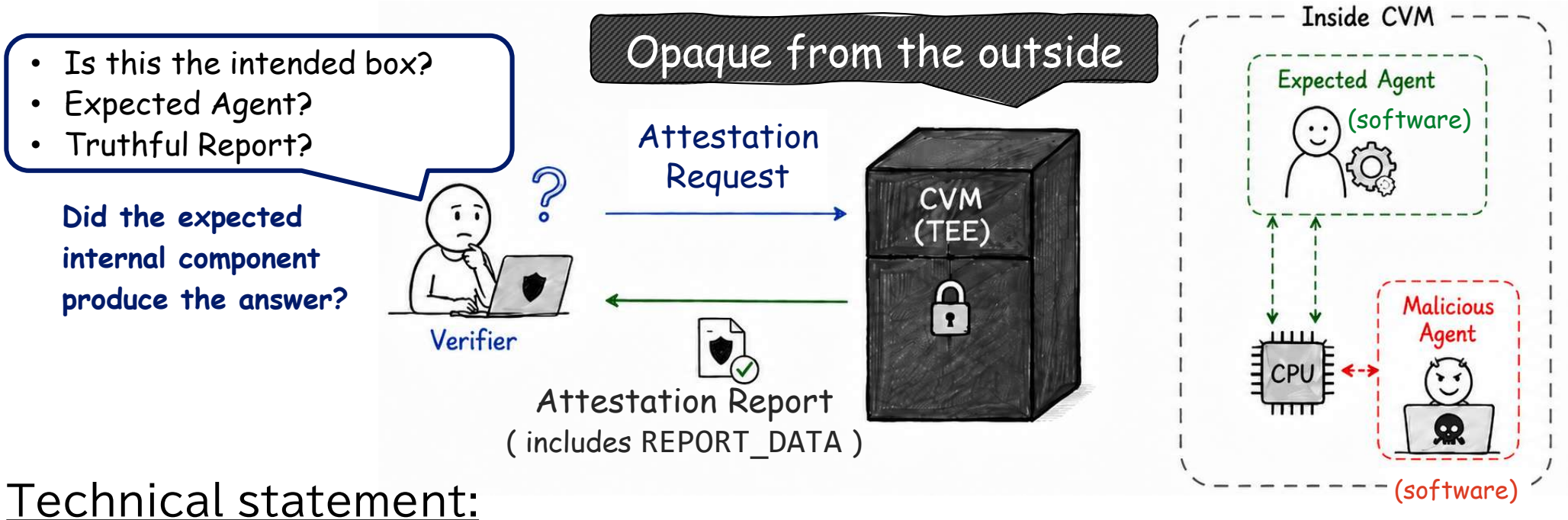
Outline

Background: Confidential Computing is spreading. VM-based TEEs are widely applied because of its high versatility.

- **Problem** — A TEE as a black box — the provenance of guest-chosen `REPORT_DATA` in attestation becomes the key issue
- **Motivation** — Make `REPORT_DATA` more verifiable and broaden its applicability
- **Conventional technologies** — Remote Attestation(RA), aTLS, IMA, ...
- **Proposal**
  - Design direction — Build on existing, proven mechanisms
  - Component & flow — A kernel-resident Pseudo-CA and how reports are constructed
  - Prototyping — Feasibility check and minimal performance measurement
- **Summary**

From the Black-Box Intuition to the Technical Gap

A valid CPU quote can still hide who selected `REPORT_DATA` ; the Verifier must check whether the expected in-CVM Agent constructed the report.



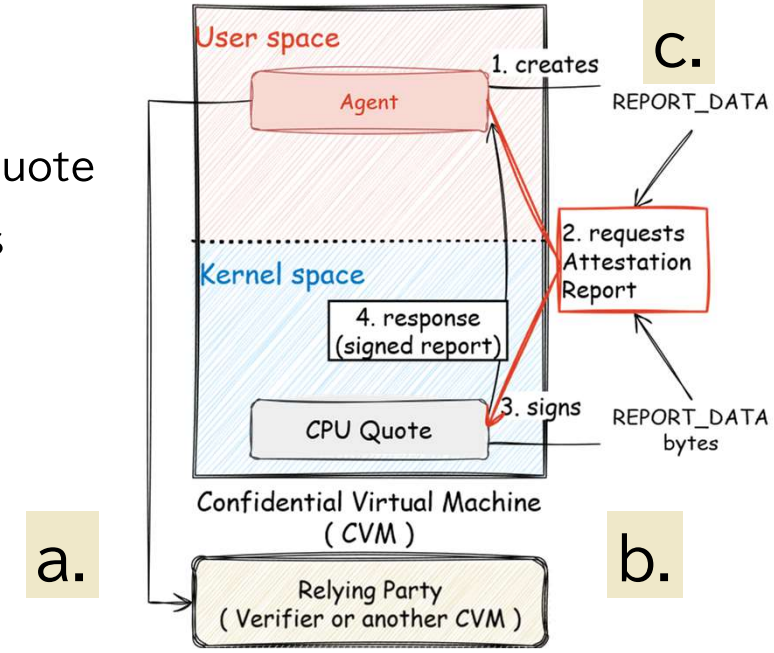
Technical statement:

A CPU quote authenticates `REPORT_DATA` bytes, but not authenticate the in-guest component that made those bytes meaningful.

Problem Statement:
"Why a valid quote is not enough?"

A valid CPU quote proves that REPORT_DATA was signed; it does not prove why the Verifier should trust the path that selected it.

1. Agent populates `REPORT_DATA`
2. Agent requests CPU quote
3. CPU signs quote signs over `REPORT_DATA`
4. CPU responses (to the request 2.)
5. Agent sends report package to Verifier



a. A verifier receives a report and a CPU quote.

b. The CPU quote proves that some `REPORT_DATA` bytes were signed. (→ 3.)

c. But `REPORT_DATA` is guest-chosen.(→ 1.)

<u>Remaining question :</u>

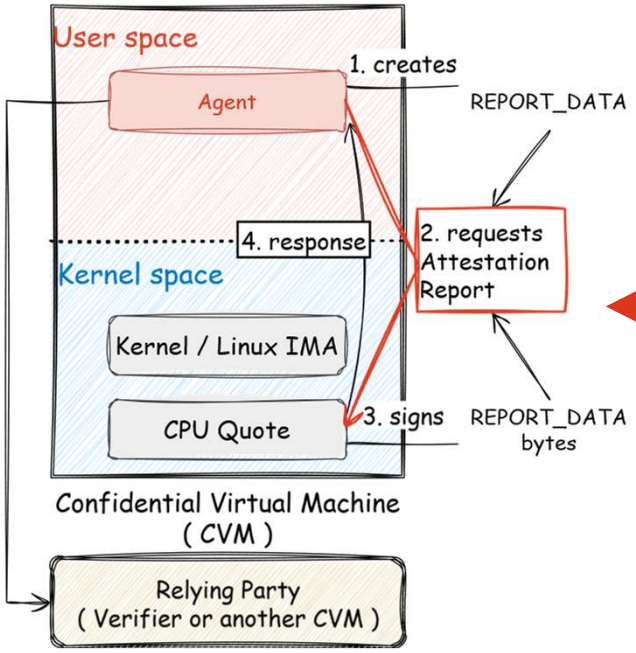
Which in-guest component selected those bytes, and why should the Verifier trust that construction path?

Our Motivation

Existing mechanisms tell us important facts, but they still leave one question open: who made the report meaningful?

What we already get

- CPU quote
  signs `REPORT_DATA`
- Secure Boot / IMA
  records boot-time measurements
- TLS / aTLS
  authenticates a channel endpoint



The missing link

- Is the value correct?
  `REPORT_DATA` is guest-chosen
- Who made `REPORT_DATA` meaningful?
  intended "Agent" assembled the report?

Goal: Making the provenance of the attestation report externally verifiable

reduces ambiguity in evidence and helps to answer deeper than endpoint authentication

"This report was constructed through this measured in-CVM path"

Existing building blocks / obvious composition

CA, TLS, aTLS(attested TLS), CPU quote, vTPM, and IMA(Integrity Measurement Architecture) are necessary building blocks, but they do not fully identify the report-construction path.

Need to know	CA / TLS	aTLS / RA	CPU quote	vTPM / IMA	Our method
Authenticate channel key	✓	✓	X	X	Complements
Authenticate TEE/CVM endpoint	X	✓	✓	X	Complements
Use boot-time measurements	X	~	✓	✓	Uses
Show the key originates from this boot instance	~	✓	~	~	✓
Identify who assembled `REPORT_DATA`	X	X	X	~	✓

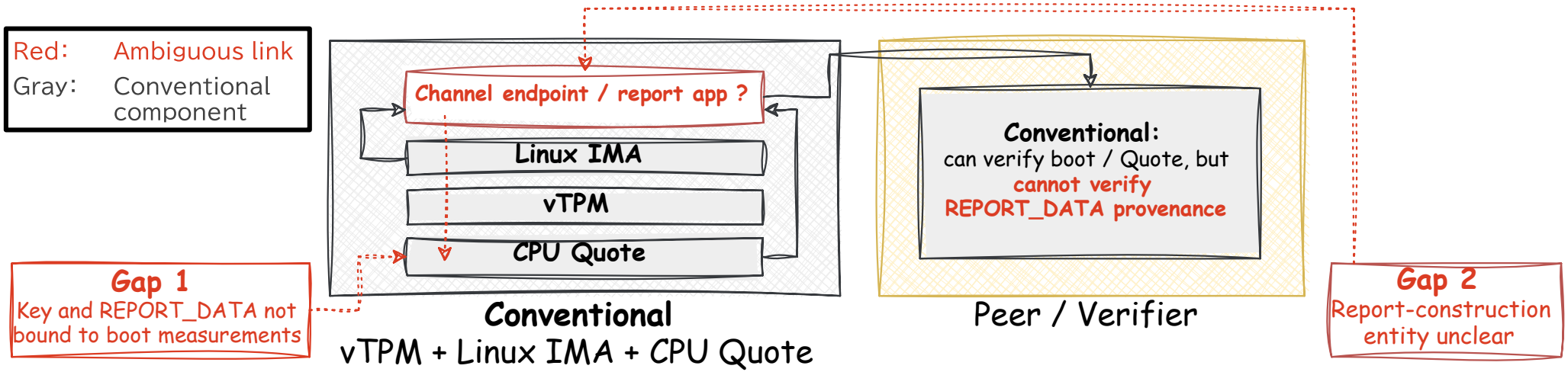
Useful baseline: CA, RA, CPU quote, and vTPM already provide important building blocks for channel, endpoint, and boot-time assurance.

Still missing: They still do not fully tell us
① whether the key originated in this boot instance
② who actually assembled `REPORT_DATA` (guest-chosen field).

Legend: ✓ direct support ~ partial support x not provided

Two gaps inside the CVM

The remaining gaps are inside the CVM: boot-instance binding and report-construction identity.



Gap 1: key and evidence binding

The application key, evidence bundle, and `REPORT_DATA` digest are not yet shown to originate from the same measured boot instance.

Gap 2: report-construction identity

Even after endpoint attestation, it can remain unclear which in-guest component assembled the report.

➡ **Objective : Accept a report only if** the key material, evidence bundle, `REPORT_DATA` digest, and CPU quote form **one measured lineage.**

Design Direction

We bring the user-space Agent into the verifiable evidence path without trusting the Agent as a standalone root.

1. Bring the user-space Agent into the verifiable evidence path.
2. Bind the Agent's report-construction role to measured boot-time evidence.
3. Because the Agent starts later in user space, it cannot be the root of the initial trust chain.
   Therefore, the root anchor must exist at the kernel boundary or earlier.
   A kernel-resident component is a practical design point.

   Placing this function in VMPL0 would complicate communication and report handling, while a kernel-resident component keeps the ordinary CVM model largely unchanged.

Design Choice: A Measured Kernel-Side Anchor

The Pseudo-CA turns key generation and Agent delegation into measured, externally checkable events.

1. Place a Pseudo-CA on the kernel side of the CVM.
2. Cover the component by Secure Boot, Measured Boot, launch measurement, or Linux IMA/PCR evidence.
3. Generate boot-scoped key material.
4. Record public verification keys and signing events as measured evidence.
5. From this measured anchor, the Verifier can check the Agent's delegated role in report construction.

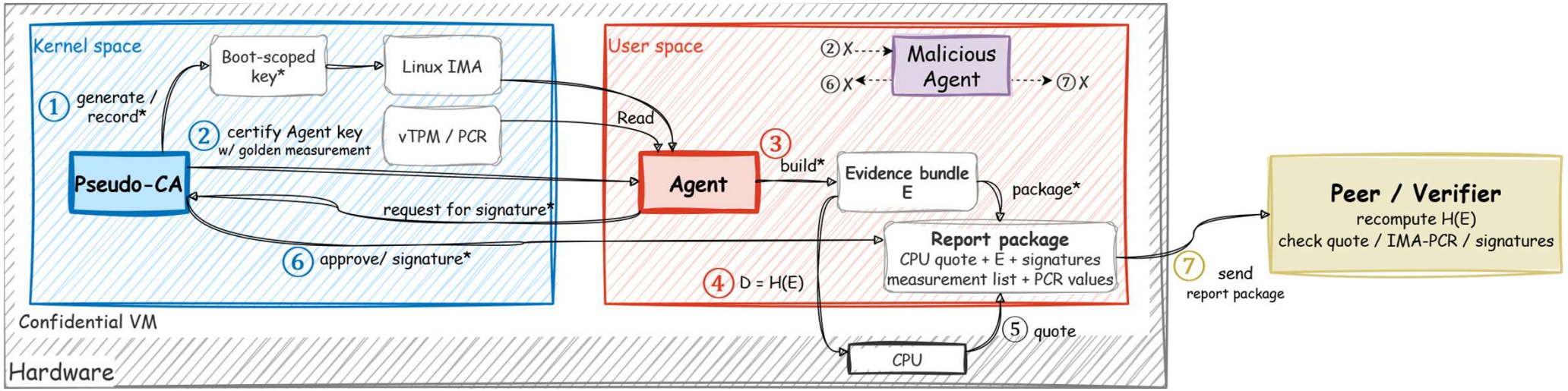
Security Claim and Boundary

This is a boot-time provenance mechanism, not a full runtime protection mechanism.

Category	Security assumption/ boundary
Assumed correct	hardware root of trust
Potentially compromised	host OS, hypervisor, resource-management tooling
In scope	boot-time lineage, report-construction identity
Out of scope	full runtime correctness, arbitrary post-boot compromise, side channels
Attacker goal	unintended peer connection, unintended in-guest report construction

Three roles and one lineage

REPORT_DATA is guest-chosen, so any in-guest app could fill it. The Pseudo-CA certifies only a measured Agent, and the Verifier checks one lineage — so a fake Agent is blocked at signing, or detected at verification.



Pseudo-CA (kernel space)

① measured; generates boot-scoped key material
② certifies the Agent key
⑥ signs Agent-submitted evidence

Agent (user space)

③ collects evidence and builds bundle E
④ places H(E) in REPORT_DATA

Verifier (outside CVM)

⑦ checks the lineage and makes a decision

∗ rejects any report not backed by a certified, measured Agent

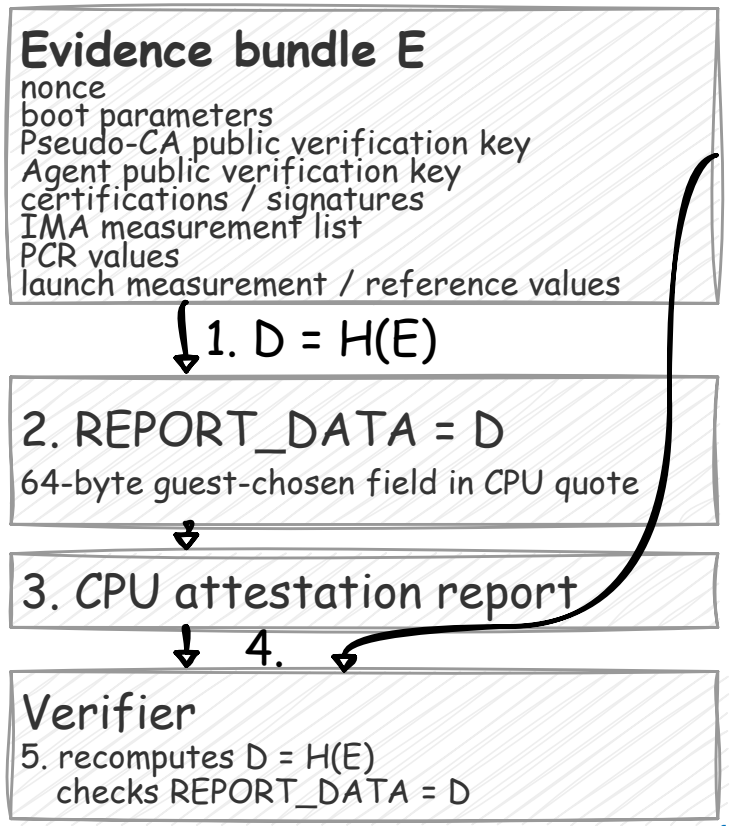
REPORT_DATA binding rule

Kernel residency gives the Pseudo-CA a measured basis that a user-space Agent cannot claim by itself.

REPORT_DATA is only 64 bytes in SEV-SNP, so the full evidence cannot fit there.

Binding rule

1. Compute D = H(E).
2. Place D in REPORT_DATA.
3. Obtain a CPU quote over REPORT_DATA.
4. Send E alongside the quote.
5. The Verifier recomputes D.
   * E is self-contained, but not self-trusting.



Initial report establishes the root; Additional report binds the Agent

Agent activity becomes externally checkable because its key and signing events appear in IMA evidence and are carried into the additional report.

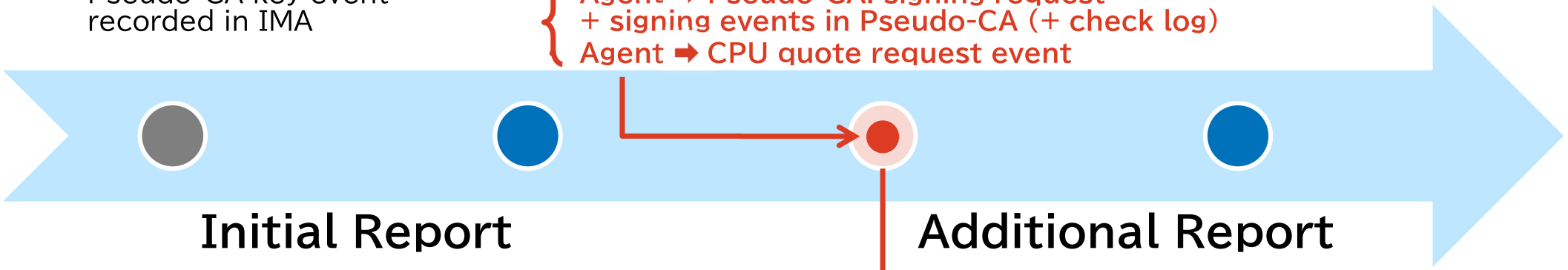
Boot/ startup

- Pseudo-CA key generated
- Pseudo-CA key event recorded in IMA

Agent Activity

Events are captured in IMA measurement list

- Agent ➡ Pseudo-CA: signing request + signing events in Pseudo-CA (+ check log)
- Agent ➡ CPU quote request event



Initial Report

- IMA measurement list and PCR values
  - ✓ **Pseudo-CA public key and self-signature**
- CPU attestation report
- `REPORT_DATA` digest

Establish boot-instance root of provenance

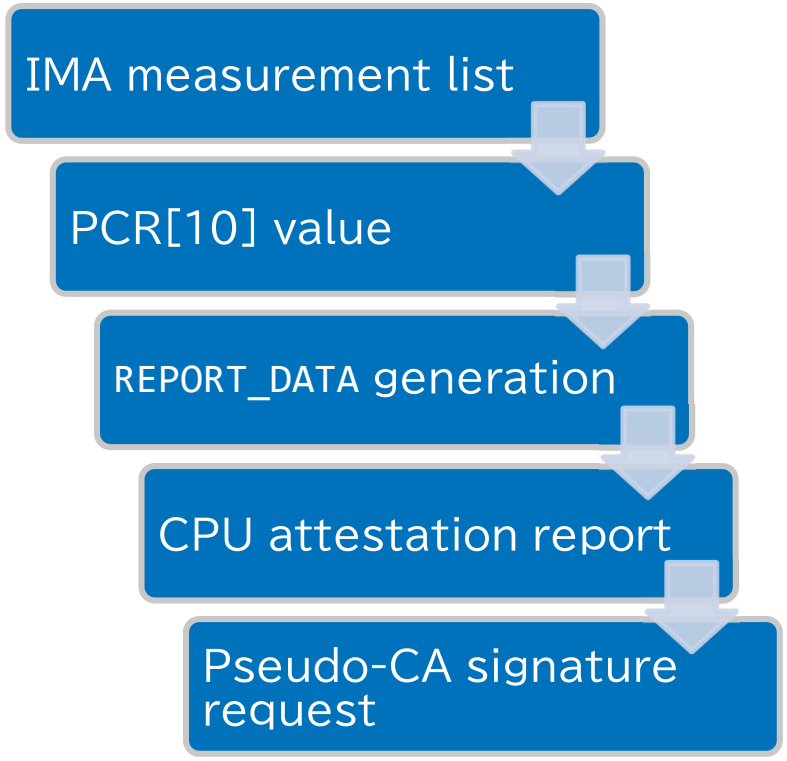
Additional Report

- Events in IMA measurement list
  - ✓ Agent ➡ Pseudo-CA: signing request + signing events in Pseudo-CA
  - ✓ Agent ➡ CPU quote request event
- Continuity with initial report

Binds the Agent activity to the established root

Prototype / Feasibility

The prototype is intended to show feasibility and detectability, not performance superiority.



Item	message
Measurement list	about 2,030 entries
Initial report	retrieved: • measurement list • PCR value • REPORT_DATA • Pseudo-CA signature
Additional report	exercised the path including Agent-triggered signing requests
Preliminary timing	about 192 ms on average over five runs

Note: Values shown as 0 ms mean below 1 ms.

What the Verifier can now ask

The Verifier can now ask not only whether the endpoint is a TEE, but also how the report was constructed inside the CVM.

The CPU quote authenticates `REPORT_DATA` bytes.

The proposed lineage explains where those bytes came from.

- Who built this report?
- From which boot-time evidence?
- Do the key, evidence bundle, `REPORT_DATA` digest, and CPU quote match one measured lineage?

Main contribution: application-layer provenance becomes externally checkable without replacing ordinary secure-channel mechanisms.

Summary

The contribution is a measured lineage from boot-time evidence to application-layer REPORT_DATA.

a. We identified an application-layer **provenance gap**: a CPU quote signs `REPORT_DATA`, **but does not explain who made it meaningful.**

b. We proposed a kernel-resident Pseudo-CA that binds boot-time evidence, Agent delegation, and `REPORT_DATA` **into one measured lineage.**

c. Our prototype confirms that the evidence-acquisition path is feasible; broader negative tests and performance evaluation remain future work.

Innovating a Sustainable Future for People and Planet


NTT