On Attestation Dilution

Charly Castes
EPFL

Abstract

The trust we place in "trusted execution environments" is based solely on the evidence of the attestation. The attestation provides a proof of genuine hardware and a list of software hashes. In theory, the hashes are sufficient to evaluate the correctness and security properties of the system. In practice, however, hashes map to closed-source firmware that cannot be identified or audited by a user of the system.

In this paper we propose a simple new metric, called *attestation dilution*, to guide system design toward a more open and trustworthy infrastructure. The attestation dilution of a system is the number of hashes in the attestation that cannot be verified by a user. Ultimately, we believe that principled system design can lead to undiluted attestations, meaning that the system can be fully audited for providing the advertised security properties.

ACM Reference Format:
Charly Castes. 2026. On Attestation Dilution. In *9th Workshop on System Software for Trusted Execution (SysTEX '26), April 27–30, 2026, Edinburgh, Scotland UK*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3805690.3805728

1 Attestation Dilution

An attestation [8, 17] boils down to a signed list of hashes of all software that is part of the trusted computing base (TCB), plus a nonce for freshness and to defend against replay attacks. Typically, an attestation is a variation of:

$$attestation = sign(hash1|hash2|...|nonce)$$

Such an attestation is emitted by a computer system, and received by a challenger (either a person or a machine) whose goal is to establish trust in the system before initiating a high-stakes operation, such as uploading medical records or downloading a software update. For the challenger to make an informed decision, the hashes must be meaningful to them. In this paper we propose a new concept and metric to reason about this problem.

Definition (Attestation Dilution)
The attestation dilution for a particular system is the number of hashes that cannot be recomputed from source code by the challenger. We say that an attestation is undiluted if its attestation dilution is 0, and that the attestation is diluted otherwise.

Here we propose a definition of attestation dilution that provides the strongest guarantees to the challenger, *i.e.,* that they can themselves *reproduce the hashes*, and hence audit the system. A weaker form of attestation dilution would be that the challenger should be able to *identify the author* of all software components from the hashes, and trust the authors rather than the software. However, this opens the door to a common pitfall, which is to rely on an entity considered adversarial in the threat model to validate the hashes, such as the cloud provider when deploying confidential VMs. For that reason, and despite the challenges of reproducible builds, we stick with the stronger definition in this paper.

At first sight that definition ignores the necessary hardware root of trust responsible for signing the attestation. In practice, the hardware root of trust is implemented in software on a security co-processor, and is therefore itself part of the attestation and potentially counting toward dilution.

1.1 Properties

Let us take a look at the consequences of that definition:

Attestation dilution depends on the challenger: For instance, a hyperscaler building its own infrastructure is likely to have an undiluted attestation when it is itself the challenger, as the server runs custom firmware and system software. However, for a cloud customer of that hyperscaler or a company operating a small on-premise data center, the attestation would be diluted. This is intentional, for the same reason that the threat model is different for a cloud provider than for a VM tenant.

Attestation dilution is not comparable across platforms: The scalar value of attestation dilution is not meant to be compared across platforms (e.g. Arm, Intel, or AMD). Some platforms might require a different number of runtime firmware or boot stages; that does not make one intrinsically more or less secure than another. The only meaningful comparison across platforms is undiluted attestation (attestation dilution of 0) vs diluted attestation (attestation dilution > 0). Yet, the scalar value is valuable to evaluate improvements within a platform.

An incomplete attestation is worse than a diluted attestation: Measuring attestation dilution is only meaningful if all of the TCB is part of the attestation. It is easy to reduce attestation dilution by removing a hash from the attestation, but that would leave the attestation incomplete. An incomplete attestation is worse than a diluted attestation,

SysTEX '26, Edinburgh, Scotland UK

ACM ISBN 979-8-4007-2607-1/2026/04
https://doi.org/10.1145/3805690.3805728

because software can be updated (or compromised) in ways that cannot be observed by the challenger.

1.2 The Case for Attestation Dilution

So what is attestation dilution good for as a metric? We argue that it serves two purposes: to reason about the guarantees that existing systems can provide, and to guide the design of future systems.

Most research efforts in the area of trusted execution environments focus on building secure monitors [11, 14, 19] or runtimes [2, 4, 5], and sometimes even formally verify them [7, 13, 20]. This is a very important and so far fruitful research area, yet it is only one part of the whole story. The reality is that modern systems rely on a plethora of software and firmware components to be able to boot and operate. The lowest firmware layers can amount to millions of lines of code [15], and today even the most secure systems [9] blindly assume the firmware to be correct.

The reason attestations are diluted today has more to do with the historical evolution of the computing industry than fundamental limitations. CPU vendors and OEMs[1] must provide low-level firmware to operate the hardware, such as monitoring sensors, controlling power distribution and frequency scaling, or managing proprietary devices and extensions. It is often assumed that low-level software requires high privileges, *but that is not the case*. We argue that low-level software should either be open for all to inspect, or be de-privileged and removed from the TCB. We propose to measure attestation dilution as a metric to evaluate progress toward that goal of a secure and transparent infrastructure.

2 Example: Attestation on Arm CCA

Arm CCA [13] is Arm's solution for confidential VMs. The attestation on Arm CCA contains 13 hashes: 9 software and 4 configuration hashes [3]. Table 1 lists the hashes by category.

Of the four software hashes on the application processor (the CPU cores running the VMs), only the RMM[2] is typically known to the challenger. The `AP_BL1` and `AP_BL2` are boot loaders, while `AP_BL31` is the EL3 runtime firmware, which executes with higher privileges than the RMM, and therefore has full access to the confidential VMs' memory.

In addition, the attestation includes software running on two auxiliary cores on the SoC, for which the software is typically closed source despite being part of the TCB. The runtime security engine is a security co-processor that is responsible for producing the attestation itself, while the system control processor manages power and reset. This is not an Arm-specific problem; AMD and Intel both rely on similar co-processors (the PSP and ME, respectively).

Ignoring configuration hashes, only 1 of the 9 software hashes is known to the challenger, yielding an attestation dilution of 8 for a typical Arm CCA deployment from the perspective of confidential VM tenants.

Table 1. Hashes in Arm CCA attestation

Category	Hashes
Application Processor (AP)	`AP_BL1`, `AP_BL2`, `AP_BL31`, `RMM`
Runtime Security Engine (RSE)	`RSE_BL1_2`, `RSE_BL2`, `RSE_S`
System Control Processor (SCP)	`SCP_BL1`, `SCP_BL2`
Configuration	`HW_CONFIG`, `FW_CONFIG`, `TB_FW_CONFIG`, `SOC_FW_CONFIG`

3 Reducing Attestation Dilution

This raises the question: how can we reduce the attestation dilution to zero? We argue that reducing attestation dilution is a system design problem: systems need to disentangle the CPU and OEM vendors' proprietary logic from the security-critical components.

In a system with undiluted attestation, all low-level system software has been either made source-available to the challenger, or removed from the TCB. Recent research efforts have started to sketch such a possibility: firmware can be efficiently sandboxed [10], or run unmodified in user-space with a virtual firmware monitor [6]. For instance, running the `AP_BL31` firmware in user-space with an open-source virtual firmware monitor would reduce Arm CCA's attestation dilution by one. Similarly, the emergence of memory protection mechanisms in low-power processors, such as RISC-V's PMP [18], has opened the door to the design of secure embedded operating systems [12, 16]. Such systems can isolate proprietary logic from attestation capabilities, as we have seen in OpenTitan [1], and could be used as `SCP_BL2` or `RSE_S` on Arm CCA platforms. As it now becomes apparent, reducing attestation dilution to zero will require holistic hardware-software co-design at the scale of the SoC.

Achieving undiluted attestation will ultimately require industry actors' buy-in, but it is up to the research community to address their concerns and chart the path ahead — for instance through prototypes on open RISC-V platforms. By proposing to measure attestation dilution we hope to challenge the current expectations from attestation, and to inspire new research directions toward fully transparent and trustworthy computer systems.

4 Conclusion

Trust in computer systems is rooted in the attestation, yet the attestation often contains hashes of software that cannot be inspected for correctness. We propose to introduce a new metric, called *attestation dilution*, that measures the number of hashes in an attestation that cannot be recomputed by the challenger from available source code. We argue that through system design, the attestation dilution can be brought down, with the eventual goal of completely undiluted attestation, and thus a truly trustworthy computing infrastructure.

[1] Original Equipment Manufacturers, such as Dell, HPE, or Supermicro.
[2] Realm Management Monitor

References

[1] OpenTitan: Open source silicon root of trust. https://opentitan.org/. Accessed: 2025-02-19.

[2] The gramine library os. https://gramineproject.io/, 2020.

[3] Arm. Download and inspect the attestation token. https://learn.arm.com/learning-paths/servers-and-cloud-computing/cca-veraison/attestation-token/, 2024.

[4] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, André Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark Stillwell, David Goltzsche, David M. Eyers, Rüdiger Kapitza, Peter R. Pietzuch, and Christof Fetzer. SCONE: Secure Linux Containers with Intel SGX. In *Proceedings of the 12th Symposium on Operating System Design and Implementation (OSDI)*, pages 689–703, 2016.

[5] Andrew Baumann, Marcus Peinado, and Galen C. Hunt. Shielding Applications from an Untrusted Cloud with Haven. *ACM Trans. Comput. Syst.*, 33(3):8:1–8:26, 2015.

[6] Charly Castes, François Costa, Neelu S. Kalani, Timothy Roscoe, Nate Foster, Thomas Bourgeat, and Edouard Bugnion. The Design and Implementation of a Virtual Firmware Monitor. In *Proceedings of the 31st ACM Symposium on Operating Systems Principles (SOSP)*, pages 85–100, 2025.

[7] Andrew Ferraiuolo, Andrew Baumann, Chris Hawblitzel, and Bryan Parno. Komodo: Using verification to disentangle secure-enclave hardware from software. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, pages 287–305, 2017.

[8] Tal Garfinkel, Ben Pfaff, Jim Chow, Mendel Rosenblum, and Dan Boneh. Terra: a virtual machine-based platform for trusted computing. In *Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP)*, pages 193–206, 2003.

[9] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David A. Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: formal verification of an OS kernel. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*, pages 207–220, 2009.

[10] Mark Kuhne, Stavros Volos, and Shweta Shinde. Dorami: Privilege Separating Security Monitor on RISC-V TEEs. In *Proceedings of the 34th USENIX Security Symposium*, pages 1149–1166, 2025.

[11] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanovic, and Dawn Song. Keystone: an open framework for architecting trusted execution environments. In *Proceedings of the 2020 EuroSys Conference*, pages 38:1–38:16, 2020.

[12] Amit Levy, Bradford Campbell, Branden Ghena, Daniel B. Giffin, Pat Pannuto, Prabal Dutta, and Philip Alexander Levis. Multiprogramming a 64kB Computer Safely and Efficiently. In *Proceedings of the 26th ACM Symposium on Operating Systems Principles (SOSP)*, pages 234–251, 2017.

[13] Xupeng Li, Xuheng Li, Christoffer Dall, Ronghui Gu, Jason Nieh, Yousuf Sait, and Gareth Stockwell. Design and Verification of the Arm Confidential Compute Architecture. In *Proceedings of the 16th Symposium on Operating System Design and Implementation (OSDI)*, pages 465–484, 2022.

[14] Jonathan M. McCune, Yanlin Li, Ning Qu, Zongwei Zhou, Anupam Datta, Virgil D. Gligor, and Adrian Perrig. TrustVisor: Efficient TCB Reduction and Attestation. In *IEEE Symposium on Security and Privacy*, pages 143–158, 2010.

[15] Thomas Meyer-Lehnert. Towards a unified firmware for enzian. B.S. thesis, 2024.

[16] Oxide Computer Company. Hubris: A small open-source debugger-oriented runtime for deeply-embedded systems, 2021.

[17] Trusted Computing Group. Trusted Platform Module (TPM) – ISO/IEC 11889. https://www.iso.org/standard/66510.html, 2015.

[18] Andrew Waterman, Krste Asanović, and John Hauser. The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Document Version 20211203. https://github.com/riscv/riscv-isa-manual, Dec 2021.

[19] Fengzhe Zhang, Jin Chen, Haibo Chen, and Binyu Zang. CloudVisor: retrofitting protection of virtual machines in multi-tenant cloud with nested virtualization. In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP)*, pages 203–216, 2011.

[20] Ziqiao Zhou, Anjali, Weiteng Chen, Sishuai Gong, Chris Hawblitzel, and Weidong Cui. VeriSMo: A Verified Security Module for Confidential VMs. In *Proceedings of the 18th Symposium on Operating System Design and Implementation (OSDI)*, pages 599–614, 2024.