CloudVaults: Integrating Trust Extensions into System Integrity Verification for Cloud-based Environments

Benjamin Larsen, Heini Bergsson Debes, Thanassis Giannetsos

Cyber Security, Department of Applied Mathematics and Computer Science,
Technical University of Denmark, Kgs. Lyngby, 2800, Denmark
{benlar,heib,atgi}@dtu.dk

Abstract. Container-based virtualization technologies are rapidly evolving, as an integral part of cloud-based environments, while introducing several new challenges regarding security, resilience, and configuration correctness. In this paper, we present CloudVaults for coping with these challenges: a multi-level security verification framework for supporting trust aware service graph chains with verifiable evidence on the integrity assurance and correctness of the comprised containers. It is a first step towards a new line of security mechanisms that enables the provision of Configuration Integrity Verification (CIV), during both load- and runtime, by providing fine-grained measurements in supporting container trust decisions, thus, allowing for a much more effective verification towards building a global picture of the entire service graph integrity. We additionally provide and benchmark an open-source implementation of the enhanced attestation schemes and benchmark their performance.

Keywords: Cloud-Based Environments · Container-based Microservices · Configuration Integrity Verification · Privacy-Oriented Attestation

1 Introduction

The vision of cloud-based Smart Connectivity Networks (SCNs), comprising multiple edge and fog computing node deployments, is seen today as a key enabler for evolving sectors like automotive, smart factories, smart grids, or healthcare [8, 25]. At the same time, their number is expected to increase significantly with the advent of new mixed-criticality services. To this end, the cloud community is already embracing recent well-known technologies, like Network Functions Virtualization (NFV) and Mobile Edge Computing (MEC) [21] intelligent orchestration. These frameworks are based on the unrestrainable "softwarization" process, which will transform physical infrastructures into distributed data centers with advanced virtualization and software-driven capabilities. They are considered the two key enablers for intelligent edge computing and the cloud to operate in tandem; Virtual Functions ($\mathcal{V}$Fs) allow for flexibly customizing cloud-based networks to the needs and peculiarities of mixed-criticality applications and expose them as Service Graph Chains (SGCs) and network slices.

Furthermore, with the advent of the Internet of Things (IoT), we have just begun reaping the benefits of this evolution that, however, also brings several new challenges (or rather makes old unsolved challenges urgent to be tackled

with); with *security*, *resilience* and *configuration correctness* being some of the major concerns at both logical extremes of a network. While virtualization offers some security advantages (such as isolation and sandboxing), it has issues such as insecure production system configuration, vulnerabilities inside the images, and vulnerabilities directly linked to various container-based technologies (e.g., Docker, LXC, rkt) [7]. The primary existing mechanisms to alleviate such issues leverage the concept of trusted computing [4,5,7,18,23], which addresses the need of verifiable evidence about a system and the integrity of its trusted computing base and, to this end, related specifications provide the foundational concepts such as *measured boot* and *remote attestation*. A key component in building such trusted computing systems is a highly secure anchor (either software- or hardware-based) that serves as a Root-of-Trust (RoT) towards providing cryptographic functions, measuring and reporting the behavior of running software, and storing data securely. Prominent examples include Trusted Execution Environments (TEEs, e.g., TrustZone) [22] and Trusted Platform Modules (TPMs) [13].

Despite recent intensive research efforts towards trust aware containers [7,18], none of the existing mechanisms are sufficient to deal with the security challenges of container-based $\mathcal{V}$Fs. Firstly, there is the perceived aspect of the incompleteness of integrity measurements or guarantees, due to the traditional focus of trusted computing on the system boot time or, at most, the load-time of applications, without covering system integrity beyond these stages, during system execution, which is especially crucial for high-availability cloud-based environments. After a $\mathcal{V}\mathsf{F}$ is deployed, the integrity of its loaded components is ignored. Indeed, while a containerized $\mathcal{V}\mathsf{F}$ should work correctly (as constructed by the orchestrator) just after it has been deployed, it could start behaving unexpectedly (e.g., modify data in an unauthorized way) if it receives a malformed input from a corrupted module acting on the $\mathcal{V}\mathsf{F}$. The assurance that a $\mathcal{V}\mathsf{F}$ works correctly after loading is known as *load-time integrity*, while *run-time integrity* refers to the whole process life-cycle. Secondly, it is imperative to ensure the privacy of the $\mathcal{V}\mathsf{F}$ (and the underlying host) configuration. One overarching theme of building trust for containers is to leverage IBM's Integrity Measurement Architecture (IMA) [23] that measures the integrity of a designated platform. Since IMA measures all components and records them into a single log, each verifier ($\mathcal{V}\mathsf{rf}$) with access to this log (when validating the integrity of a $\mathcal{V}\mathsf{F}$), will also get all configuration information of the prover ($\mathcal{P}\mathsf{rv}$). Adversaries benefit from such artifacts and are capable of stealing the information of other users' containers.

Compounding these issues, sets the challenge ahead: *Can we identify sufficient Configuration Integrity Verification (CIV) schemes that can capture the chains-of-trust, needed for the correct execution of a* $\mathcal{V}\mathsf{F}$ *during both load- and run-time, and that allow for inter- and intra-*$\mathcal{V}\mathsf{F}$ *attestation without disclosing any information that can infer identifiable characteristics about individual* $\mathcal{V}\mathsf{F}$ *configurations?* Solutions will, in turn, enable the provision of adequate trust models for assessing the trustworthiness and soundness of the overall SGC.

Contributions: In this paper, we design and implement CloudVaults, a security verification framework for supporting privacy- and trust-aware SGCs, in

lightweight cloud-based environments, with verifiable evidence of the integrity and correctness of the $\mathcal{VF}$s. The solution can be either applied separately to each deployed $\mathcal{VF}$, equipped with a virtual TPM (vTPM) security anchor, or the entire SGC, and it enables CIV of the constructed container(s). Key features provided, that extend the state-of-the-art, include the: (*i*) possibility to distinguish which container is compromised, (*ii*) the possibility for low-level fine-grained tracing capability (Attestation by Quote), and (*iii*) Secure Zero Touch Provisioning (S-ZTP) capability which allows for inter- and intra-$\mathcal{VF}$ attestation without disclosing any $\mathcal{VF}$ configuration information (Attestation by Proof). Our proposed solution is scalable, (partially) decentralized, and capable of withstanding even a prolonged siege by a pre-determined attacker as the system can dynamically adapt to its security and trust state. We make an open-source reference implementation of all CloudVaults schemes and protocols and benchmark their performance for $\mathcal{VF}$s that are equipped with either a software- or hardware-TPM[1]. Such an implementation can be used as a basis for further development of enhanced attestation schemes using TPM 2.0 and comparative benchmarking. It should also lower the barrier of entry for other researchers who want to explore TPM-based Configuration Integrity Verification solutions. Overall, our approach is viable for remedying limitations of existing attestation techniques; nonetheless, there is still a need to overcome other open issues towards a holistic end-to-end security approach.

2 Towards Trust-Aware Service Graph Chains (SGCs)

Leveraging cryptographic techniques and Trusted Components ($\mathcal{TC}$s) towards protecting and proving the authenticity and integrity of computing platforms has been extensively researched. Both *integrity* and *authenticity* are two indispensable enablers of trust. Whereas integrity provides evidence about correctness, authenticity provides evidence of provenance. There are two possible avenues towards achieving configuration integrity: either make the configurations themselves immutable or make the hashes of the configurations immutable. The latter approach follows the Trusted Computing Group's (TCG) open integrity standards [24], which recommends the utilization of hardware TPMs for storing an accumulated hash over its Platform Configuration Registers (PCRs). TPMs also inherently provide indisputable evidence of authenticity in the form of signatures over data using securely stored keys (Section 2.1).

As aforementioned, IMA accumulates measurements in a TPM. It extends the principle of measured boot, where components are measured in the order in which they are loaded into the Operating System (OS) with the use of Linux OS kernel functionality. By default, IMA measures the load-time integrity of user-space applications and files read by the root user during runtime. It is based on the Binary-Based Attestation (BBA) scheme proposed by TCG, where measurements and attestation consider hashes of binaries. However, even the smallest change in a binary dramatically changes its hash, making IMA measurements susceptible to grow unwieldy as the number of measured objects increases. Furthermore, the temporal order in which files are accessed, or applications are

[1] The CloudVaults C reference implementation can be found at [16].

loaded, can be highly unpredictable, making it difficult to verify the accumulated measurements. The inherent disadvantage of BBA paradigms is the disclosure of the platform's software and hardware configuration, which is a legitimate privacy concern since an intermediate adversary $\mathcal{A}\mathsf{dv}$ (or a malicious $\mathcal{V}\mathsf{rf}$) can use this information to infer identifiable characteristics about the platform.

Further, the variety and mutability of software and their configurations make it difficult to evaluate the platform's configuration integrity [4] during runtime. Several architectures extend upon the IMA-BBA paradigm [7, 18] to provide integrity verification. DIVE [7] and Container-IMA [18] both incorporate IMA for virtualized Docker containers to enable orchestrators (remotely) determine the runtime integrity of containers in cloud-based environments. DIVE distills the measurements to only present configuration information related to containers of interest, while Container-IMA proposes *xor-ing* measurements belonging to distinct containers with container secrets to preserve their privacy. Irrespectively, both solutions necessitate the exchange of some identifiable information.

In the same line of research, Property-Based Attestation (PBA) [4,5] schemes map the platform configurations to attestable properties in order to avoid the disclosure of the host configurations altogether. Attesting properties has the advantage that different platforms, with different components, may have different configurations but still yield the same fulfillment of properties. In particular, PBA gives more flexibility for handling system patches and updates [5], but with the deficiency of detail. [26] presents a PBA-BBA hybrid to the cloud environment where an attestation proxy mediates attestation requests between the prover ($\mathcal{P}\mathsf{rv}$) platform and $\mathcal{V}\mathsf{rf}$, such that only the proxy can be aware of the correct configurations of a $\mathcal{P}\mathsf{rv}$. It then presents to $\mathcal{V}\mathsf{rf}$ only the security property of $\mathcal{P}\mathsf{rv}$ as the attestation proof, thus, preventing exposure of platform configuration information. The inherent limitation of PBA, however, is that it is only applicable to specific properties (which require accurate identification) and is not directly transferable to reflect changes of mutable configurations.

2.1 Solidifying the $\mathcal{V}\mathsf{F}$s: Inter-Trustability of Service Function Slices

A combination of these concepts is of great interest to the secure composability of SGCs, encompassing a broad array of mixed-criticality services and applications. In particular, CloudVaults strives to enable orchestration of heterogeneous $\mathcal{V}\mathsf{F}$s containing mutable configurations by leveraging the profoundness of BBA while retaining the privacy-centered approach of PBA. In what follows, we elaborate on the inherent functionalities of a TPM that are leveraged by CloudVaults.

Monotonic Counters for Trusted Measurements. Internally, each TPM has several PCRs that can be used for recording irreversible measurements through accumulation, e.g., extending PCR slot i with measurement m, the TPM accumulates: $PCR_i = hash(PCR_i||m)$. This is an indispensable property towards the creation of strong and transitive CoT. For instance, to enforce and regulate trustworthiness of the system boot sequence we can require that all components measure and verify their successors by the following recurrence construct [22]: $I_0 = true; I_{i+1} = I_i \wedge V_i(L_{i+1})$, where $i \leq n \wedge n \in \mathbb{N}^*$, I_i denotes the integrity of layer i and V_i is the corresponding verification function which

compares the hash of its successor with a trusted reference value. For example, as in [6], let us assume that we require the boot sequence: $seq\langle sinit, BL(m), OS(m), VS(m), VM(vf), APP(vf)\rangle$, where *sinit* is the value that the PCR is reset to. If we know that the sequence will yield PCR extensions with the values $v_1, \ldots, v_n$, and all components extend PCR j, then we will trust the chain *if and only if (iff)* $\text{PCR}_j = hash(\ldots(hash(sinit||hash(v_1))||hash(v_2))\ldots||hash(v_n))$.

Attestation & Policy-Based Sealing/Binding. Attestation can be either *local* or *remote*. Local attestation is based on Attestation Keys (AKs), which are asymmetric key pairs $AK = \{AK_{pub}, AK_{priv}\}$. To perform local attestation, we enforce usage restrictions (authorization policies) onto AK_{priv}, such as requiring that PCRs must be in a certain state to permit signing operations, e.g., PCR_j (from the example above) actually reflects the accumulation of $v_1, \ldots, v_n$. Thus, using AK_{priv} to sign a nonce chosen by $\mathcal{V}\mathsf{rf}$ provides indisputable evidence that the machine state is correct. Remote attestation is delegating the verification of PCR_j to $\mathcal{V}\mathsf{rf}$, through TPM quotes comprising a signed data structure of the nonce and the contents of a specified choice of PCRs, which $\mathcal{V}\mathsf{rf}$ verifies against trusted reference values. Regardless of the attestation method, $\mathcal{P}\mathsf{rv}$ must also prove authenticity to $\mathcal{V}\mathsf{rf}$. The TPM contains several key hierarchies, but authenticity is founded specifically in the *endorsement hierarchy*. The root endorsement seed, from which Endorsement Keys (EKs) are generated, passes irrefutable evidence to the EK in a transitive manner. The credibility of the seed, and hence loaded EKs, is usually based on the trustworthiness of the Original Equipment Provider (OEP), which during manufacturing signs, loads, and later vouches that the seed corresponds to a valid TPM [25].

3 System and ($\mathcal{A}\mathsf{dv}$)ersarial Model

System Model. The considered system (Figure 1) is composed of a virtualized network infrastructure in which the application orchestrator ($\mathcal{O}\mathsf{rc}$) spawns and governs a set of heterogeneous $\mathcal{V}\mathsf{F}$ instances, as part of dedicated service graph chains. Each graph is composed of the ordered set of $\mathcal{V}\mathsf{F}$s that the service runs to manage better the correct execution of the onboarded (safety-critical) application workloads and guarantee its offered attributes (e.g., reliability, availability, performance). As aforementioned, state-of-the-art software engineering trends are based on the $\mathcal{V}\mathsf{F}$ microservice concept for achieving high scalability and adequate agility levels [14]. In our model, we assume the integration of lightweight virtualization techniques, namely *containerization* [2], where applications are decomposed into a mesh of cloud-native containerized $\mathcal{V}\mathsf{F}$s, each one with specific and "small-scope"-stateless processing objectives, packaged on independent virtual execution environments equipped with highly secure anchors (i.e., vTPMs) that serve as our RoTs. Each deployed $\mathcal{V}\mathsf{F}$ contains workload configurations, such as its software image, platform configuration information, and other binaries (see Definition 1), which are measured and securely accumulated into the PCRs of the loaded vTPM.

Definition 1 (Configurations). *The configuration set of a $\mathcal{V}\mathsf{F}$ encompasses all objects (blobs of binary data) accessible through unique file identifiers.*

Table 1: Notation used

Symbol	Description
$\mathcal{VF}$	A Virtual Function $\mathcal{VF}$
$\mathcal{A}\mathsf{dv}$	An adversary resident in a $\mathcal{VF}$
$\mathcal{TC}$	Trusted Component
EK	Endorsement Key containing a public and private part (EK_{pub} and EK_{priv}) and a protected symmetric key used for encrypting child keys (EK^{sk}_{priv})
AK	Attestation Key
σ	Cryptographic signature
KH	Key handle to a loaded key in the $\mathcal{TC}$
$\mathcal{I}$	Selection of PCR identifiers
n	Randomly generated nonce
S	Internal session digest in $\mathcal{TC}$
$A_{\texttt{tmp}}$	Key template
$h_{\texttt{Conf}}{}^{\dagger}$	Expected configuration (PCR hash)
$h_{\texttt{Pol}}$	Policy digest based on $h_{\texttt{Conf}}$
$h_{\texttt{Create}}$	Key creation hash, w. $\mathcal{TC}$ state and parent key
$T^{\dagger}$	Creation ticket proving origin of creation hash
A_{cert}	Key creation certificate
$Q_{\texttt{Cert}}$	Quote Certificate
$h_{\beta}{}^{\dagger}$	Hash of a binary

† We further use a prime to denote a reference, e.g., $h'_{\texttt{Conf}}$ is a calculated reference to the actual hash of the PCR contents.

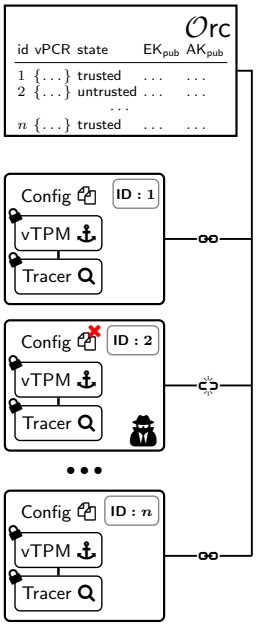

Fig. 1: Orchestration of Segregated $\mathcal{VF}$s

More formally, the $\mathcal{O}\mathsf{rc}$ maintains a Service Forwarding Graph ($\mathcal{SG}$), of function chains, defined as $\mathcal{SG} = \{s_1, s_2, \ldots, s_n\}$, where $n \in \mathbb{N}^*$. Each service chain comprises a set of deployed $\mathcal{VF}$s, $s_i = \{vf_1, vf_2, \ldots, vf_m\}$, where $m \in \mathbb{N}^*$ and $s_i \in \mathcal{SG}$, deployed over the substrate virtualized network. The ownership of the physical resources, over which the secure deployment and placement of these SGs take place, is not of interest. Each $vf_i \in \left\{\bigcup_{j=1}^{n} \mathcal{SG}_j\right\}$ is defined as a tuple of the initial form: $vf_i = (id, vPCR, state, EK_{pub}, AK_{pub})$, where id is the unique $\mathcal{VF}$ identifier, $vPCR$ refers to an artificial set of PCRs that reflect the obligatory policy (measurements) that must be enforced in the actual PCRs of the target $\mathcal{VF}$, *state* denotes whether the $\mathcal{VF}$ is considered trusted or not (policy-conformant), EK_{pub} and AK_{pub} are the public parts of the EK and AK of the vTPM that is uniquely associated with vf_i.

In addition, each $\mathcal{VF}$ is equipped with a Runtime Tracer ($\mathcal{T}\mathsf{rce}$) for recording the current state of the loaded software binary data (during both boot-up time and system execution) to be then securely accumulated into the PCRs of the hosted vTPM. Tracing techniques are used to collect statistical information, performance analysis, dynamic kernel or application debug information, and general system audits. In dynamic tracing, this can take place without the need for recompilation or reboot. In the context of CloudVaults, a detailed dynamic tracing of the kernel shared libraries, low-level code, etc., and an in-depth investigation of the $\mathcal{VF}$'s configuration is performed to detect any cheating attempts

or integrity violations. Such a $\mathcal{T}$rce can be realized either as: (*i*) a static binary analyzer for extracting hashed binary data measurements (i.e., digests) [1], or (*ii*) a general, lightweight tracer with kernel-based code monitoring capabilities based on the use of "execution hooks" (e.g., extended Berkeley Filters) [15].

This process builds on top of the IMA feature [23] and records measurements of the $\mathcal{V}$F's software binary images of interest (as specified in the deployed security/attestation policy) that reflect its state/integrity: these can span from *hardware-related properties* related to the BIOS/UEFI and kernel information, to *dynamic properties* such as executable code, structured data and temporary application data (e.g., configuration files, file accesses, kernel module loading). When a measurement is extracted (Section 5.2), a register of the TPM accumulates the digest of the captured event data to protect the integrity and constitutes the basis of the subsequent verification of a $\mathcal{V}$F's trusted state: The trust state is the result of the remote attestation functionalities of CloudVaults (Section 5.2), in which the measurements of the software loaded on a $\mathcal{V}$F is verified either locally (Attestation by Proof) or by the $\mathcal{O}$rc (Attestation by Quote) against reference values that characterize known (and, thus, trusted) software configurations.

Definition 2 (Tracer, $\mathcal{T}$rce). *Given an object identifier (see Definition 1), the $\mathcal{T}$rce utility returns (in a secure way) the corresponding object's binary data.*

($\mathcal{A}$dv)ersarial Model. Our in-scope threats include both external attackers who exploit existing vulnerabilities in the $\mathcal{V}$F stack, and insiders such as cloud users and tenant administrators who cause security breaches either by mistake or with malicious intentions. Like most security verification solutions, we trust the $\mathcal{O}$rc responsible for the provision, management, and deployment of the $\mathcal{V}$F Forwarding Graphs. The focus is on detecting threats that can lead to the violation of the specified integrity properties, and attacks by those adversaries who can remove or tamper with dynamic data (e.g., configuration files, logged events, etc.) as these can also be inferred from our integrity analysis: the only cause, a verifier would be aware of, whereby an application can get compromised at run-time is the reading of a malformed datum previously written by a malicious process. Unlike existing schemes, our solution can also cope with adversaries that try to manipulate such dynamic, unstructured data which, together with regular configuration files and network sockets, represent the majority of processes interfaces and, thus, can allow a verifier to determine the integrity and trusted state of a $\mathcal{V}$F with a high degree of confidence.

We assume an $\mathcal{A}$dv that has unrestricted virtual access to the user space of a $\mathcal{V}$F, including oracle access to its attached vTPM. Similar to other attestation architectures, we do not consider availability threats, such as Denial of Service (DoS). Further, the computational capabilities of $\mathcal{A}$dv are restricted to the Dolev-Yao model [9], where $\mathcal{A}$dv cannot break cryptographic primitives (e.g., forging signatures without possessing the correct credentials), but can, nonetheless, perform protocol-level attacks. Note also, that we do not consider a sophisticated $\mathcal{A}$dv that can perform stateless attacks that target a program's control flow [11, 20] where the measurement of a binary can remain unchanged

even though the software's behavior has been altered. In particular, a residential $\mathcal{A}\mathsf{dv}$ has the following **C**apabilities:

C-1 Unrestricted *passive* and *active* oracle access to the attached vTPM: (passive) $\mathcal{A}\mathsf{dv}$ can monitor the exchange of commands and responses between the TSS and the vTPM; (active) $\mathcal{A}\mathsf{dv}$ can unilaterally craft and exchange illicit commands to the vTPM trying to manipulate the CIV process. However, as with any oracle, $\mathcal{A}\mathsf{dv}$ cannot access the underpinnings and secure structures (e.g., PCRs) of the vTPM.

C-2 Unrestricted ability to Create, Read, Update, and Delete (CRUD) $\mathcal{VF}$ software binary configurations (by Definition 1). Note that we do not consider attestation of memory contents (objects without a unique system identifier); thus, we do not audit direct accesses to the disks and the memory.

4 High-Level Security Properties of CIV

In this section, we provide an intuitive description of the security properties our CIV scheme is designed to provide and extract the corresponding axioms (Table 5 in Appendix C) representing the trust properties that must be satisfied by the various components (i.e., $\mathcal{VF}$, vTPM, $\mathcal{O}\mathsf{rc}$) involved in the creation and management of SGCs. Such end-to-end definitions for $\mathcal{VF}$ soundness and security are then analyzed in Section 6, where we prove how CloudVault's design satisfies them in their entirety. Recall that the focus is on trust-aware SGCs with verifiable evidence on the integrity assurance and correctness of the comprised $\mathcal{VF}$s. Verification of the host Virtual Machine (VM), its kernel, and the entire virtualization infrastructure (NFVI) [17] is beyond the scope of this paper.

Trust is evaluated by (securely) measuring the state (and configuration behavior) of a $\mathcal{VF}$ at any given point in time, and then comparing the measured state with the reference (expected) state. Note that the vTPM component of a $\mathcal{VF}$ is the trusted element that generates the signatures, certified attestation keys, and quoted PCR values in conjunction with the (potentially untrusted) host. Thus, the **P**roperties that must be achieved are:

P-1 **$\mathcal{VF}$ Configuration Correctness.** Both load-time and run-time configurations of a $\mathcal{VF}$ (by Definition 1) must adhere to the attestation policies issued by $\mathcal{O}\mathsf{rc}$ (thus, ensuring *load-time* and *run-time* $\mathcal{VF}$ *integrity*).

P-2 **SGC Trustworthiness.** It must, at all times, be possible for the $\mathcal{O}\mathsf{rc}$ to determine the trustworthiness of the entire SGC. An SGC is trusted if all $\mathcal{VF}$s are attested correctly and have shown verifiable evidence that their configurations comply with the enforced attestation policies. This transfers and extends the sound statements on the configuration security properties of single $\mathcal{VF}$s (by **P-1**) to statements on the security properties of hierarchical compositions of $\mathcal{VF}$s and SGCs.

P-3 **Attestation Key Protection.** To retain trust in a $\mathcal{VF}$, despite mutable configurations, it must be possible to deploy, during run-time, new (certified) AKs that reflect updates to the configuration policies deployed (i.e., an updated set of vPCRs about the $\mathcal{VF}$'s expected configuration). The $\mathcal{VF}$, by leveraging the vTPM, must securely create AKs such that $\mathrm{AK}_{\mathrm{priv}}$ is *never*

leaked to an $\mathcal{A}\mathsf{dv}$ so that she cannot forge valid CIV messages (thus, ensuring *unforgeability*), and must present verifiable evidence that the created AK is "bound" to the newly deployed attestation/configuration policies.

P-4 Immutability. The measurement process must be immutable, such that $\mathcal{T}\mathsf{rce}$ (by Definition 1) always returns the correct (actual) measurements.

P-5 Liveness & Controlled Invocation. It is assumed that attestation inquiries reach the local $\mathcal{V}\mathsf{F}$ attestation agent and that the agent responds with an attestation response within a specified time limit. If a $\mathcal{V}\mathsf{F}$ fails to respond within the specified time limit, this can be considered as evidence of compromise and, thus, the $\mathcal{V}\mathsf{F}$ is deemed as untrusted.

Besides the aforementioned core security properties, in some settings, $\mathcal{P}\mathsf{rv}$ might need to authenticate $\mathcal{V}\mathsf{rf}'s$ integrity verification requests in order to mitigate potential DoS attacks; e.g., an $\mathcal{A}\mathsf{dv}$ impersonating the $\mathcal{O}\mathsf{rc}$ might send a "bogus" configuration policy to a $\mathcal{V}\mathsf{F}$ representing an incorrect (reference) state. This functionality can be easily provided (and verified) by CloudVaults: In a case where the $\mathcal{O}\mathsf{rc}$ acts as the $\mathcal{V}\mathsf{rf}$, the respective request (reflecting either a new policy digest or an update measurements request - Section 5), can be signed with its (trusted) certificate so that the target $\mathcal{V}\mathsf{F}$ can verify its authenticity (note that $\mathcal{V}\mathsf{F}$s are employed with $\mathcal{O}\mathsf{rc}$'s certificate when constructed and deployed over the substrate network). On the other hand, for achieving inter-trusted $\mathcal{V}\mathsf{F}$ communication, where a $\mathcal{V}\mathsf{F}$ (acting as the $\mathcal{V}\mathsf{rf}$) tries to attest the correctness of another $\mathcal{V}\mathsf{F}$, handling a potential forged request will not have any impact on the state measured by the $\mathcal{P}\mathsf{rv}$ since this cannot result in the update of the configuration policy (can only be initiated by the $\mathcal{O}\mathsf{rc}$). Such a malevolent act will impose some additional performance overhead due to the verification process that will be performed. However, this is negligible as will be seen in Section 7.

Our work provides the missing fine-grained details of the already standardized IMA [23] and fills the perceived gaps of *dynamic* and *runtime* remote attestation and configuration integrity verification in a complex software stack as the one met in emerging virtualized environments; from the trusted launch and configuration to the runtime attestation of low-level configuration properties about a $\mathcal{V}\mathsf{F}$'s integrity and correctness. In Section 6, we provide game-based models for our enhanced CIV scheme satisfying all the above properties.

5 An Architectural Blueprint Towards Unified CIV

5.1 High-Level Overview

Our schema provides two specific functionalities, *Attestation by Proof* and *Attestation by Quote* (see Figure 2), for enabling the automatic and secure establishment of trust between deployed $\mathcal{V}\mathsf{F}$s of a service graph. The evidence of the integrity state of the service binary images, running inside such containers, is authenticated by the attached vTPM. Key features provided include the: (*i*) possibility to distinguish which container is compromised, (*ii*) the possibility for low-level fine-grained tracing capability (Attestation by Quote), and (*iii*) S-ZTP capability for privacy-preserving attestation (Attestation by Proof). The former is a significant feature because, once a $\mathcal{V}\mathsf{F}$ is compromised, it can be immediately

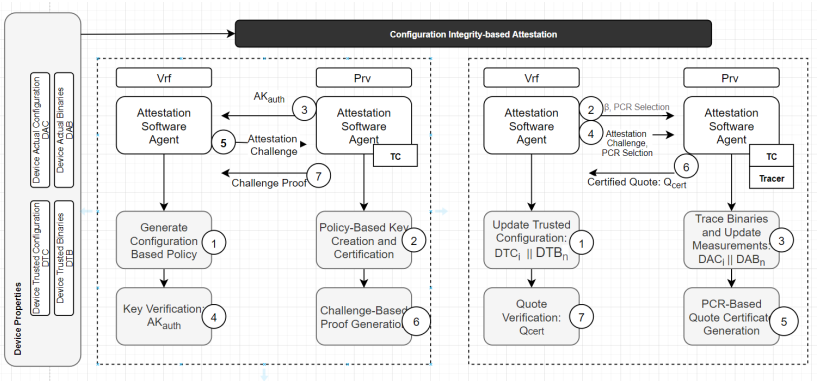

Fig. 2: CloudVaults Work Flow of $\mathcal{V}\mathsf{F}$ Configuration Integrity Verification: Attestation by Proof (**L**eft) and Attestation by Quote (**R**ight).

retracted and replaced by the $\mathcal{O}\mathsf{rc}$ without affecting the entire SGC, thus, catering to efficient SGC management and flexible slicing [3] making CloudVaults viable for practical cloud-based applications. The latter enables the integrity verification of a designated $\mathcal{V}\mathsf{F}$ without conveying other $\mathcal{V}\mathsf{F}$'s information (or unnecessary information of the underlying host) to a remote $\mathcal{V}\mathsf{F}$ (acting as the $\mathcal{V}\mathsf{rf}$), in case of a malicious $\mathcal{V}\mathsf{rf}$ being aware of which components the underlying host and other containers have. This is of paramount importance in emerging smart connectivity networks, leveraging cloud-based capabilities to support safety-critical services with strict security, trust, and privacy requirements [10].

The offered CIV allows to assess and preserve the integrity of the deployed $\mathcal{V}\mathsf{F}$'s Trusted Computing Base (TCB), at load-time and during system execution, by leveraging the capabilities of vTPMs while ensuring predictability of the internal vTPM PCR values regardless of the order of loading of applications/processes (inside the $\mathcal{V}\mathsf{F}$) and reducing performance impact by minimizing the necessary interactions with the host trusted component. It supports complete, configurable attestation that acquires binary signature chains from different unique registers, enabling advanced tracing capabilities to localize areas of compromise. Both schemes rely on the $\mathcal{V}\mathsf{F}$ to access a $\mathcal{T}\mathsf{C}$ (e.g., vTPM) with irreversible PCRs. The privacy-enhanced feature builds on the use of an AK within the $\mathcal{T}\mathsf{C}$ that can only execute a cryptographic operation if a set of PCRs is in particular (trusted) state, inferring the correctness of the component. Integration of our *Enhanced RA* protocols in cloud-based environments, comprising dispersed $\mathcal{V}\mathsf{F}$s, is convenient since the exchange of messages can piggyback conventional TLS protocols. CloudVaults also introduces the concept of *digest lists* for limiting the reporting of measured software only to the case of unknown software (not added to the digest list deployed by the $\mathcal{O}\mathsf{rc}$). This approach ensures predictable PCR values and reduced usage of the $\mathcal{T}\mathsf{C}$ and, consequently, reduces the performance impact.

Following a similar workflow to the most prominent IMA-based architectures, Figure 2 presents the information flow of CloudVaults between a $\mathcal{P}\mathsf{rv}$ and a $\mathcal{V}\mathsf{rf}$: In a nutshell, CloudVaults detects offline and online attacks on mutable files by

verifying their hashed digest with a trusted reference measurement extracted from a corresponding virtual PCR on the $\mathcal{O}$rc. Attestation reports produced by CloudVaults can include as much information as required based on the already defined attestation policies (including the configuration properties to be traced).

Attestation policies must be expressive and enforceable and can be dynamically updated by the $\mathcal{O}$rc. After defining proper policies, the $\mathcal{O}$rc can proceed to periodically (or on-demand) attest to the modeled configuration properties representing the current state of the target $\mathcal{V}$F. A $\mathcal{V}$F is trusted if its state (at that time) matches the (already measured) reference state. As each $\mathcal{V}$F is a software component, its hashed digest defines its state. By comparing the hashed digest (at any given time) to the reference (expected) hashed digest of the $\mathcal{V}$F, provided by the $\mathcal{O}$rc, we can determine the $\mathcal{V}$F trustworthiness.

5.2 CloudVaults Building Blocks

The core of our schemes (Figure 2) is the manageability of mutable configurations throughout the lifespan of a $\mathcal{V}$F and is accomplished by having the $\mathcal{O}$rc mediating any security-critical updates towards the deployed $\mathcal{V}$Fs. Whenever the $\mathcal{O}$rc invokes a periodical or scheduled update, either to determine the trustworthiness of a $\mathcal{V}$F or due to changes in configurations, it proactively determines the update's expected implication by accumulating the artificial vPCR construct of the corresponding $\mathcal{V}$F (**Step 1R**). The $\mathcal{O}$rc requests the $\mathcal{V}$F to similarly accumulate its PCRs to reflect potential changes (**Steps 2R-3R**). This update request contains only the PCR index i that must be updated and a configuration file identifier, $\beta_{\texttt{ID}}$, to measure. Upon receiving such update requests, the $\mathcal{V}$F then invokes the $\mathcal{T}$rce to measure the requested file(s) and subsequently invokes $\mathcal{T}$C to extend PCR i with the new measurement. The simple update protocol is depicted in Figure 7 (see Appendix A). Recall that an $\mathcal{A}$dv must not be able to tamper with the measurement process, or the trustworthiness of the entire process will be compromised (Section 3).

Furthermore, as mentioned in Section 2.1, the privacy-preserving attestation (i.e., local attestation) requires the use of specialized signing keys, called attestation keys (AKs), which can be bound to specific PCR contents, hence making an AK operable *iff* the PCRs reflect the particular PCR state in which the AK is bound to. However, to retain the viability and correctness of such an attestation (despite mutable PCRs), we must create and bind a new attestation key whenever a $\mathcal{V}$F is updated. Since the key creation process is best achieved locally, the $\mathcal{O}$rc requests a $\mathcal{V}$F to create a new attestation key based on the trusted measurements that were artificially accumulated before requesting the $\mathcal{V}$F to update its PCRs. First, the $\mathcal{O}$rc computes an Extended Authorization (EA) policy digest based on the trusted measurements (**Step 1L**), denoted $h_{\texttt{Pol}}$, which reflects the trusted state in which the AK must be bound to. The policy digest is then deployed together with a subset of PCRs, $\mathcal{I}$, to which the policy applies. Upon receiving such a request, the $\mathcal{V}$F is responsible for creating the attestation key on the $\mathcal{T}$C (**Step 2L**). To trust that the policy is actually enforced and that the state of the $\mathcal{V}$F is conformant to the policy, the $\mathcal{V}$F must present indisputable

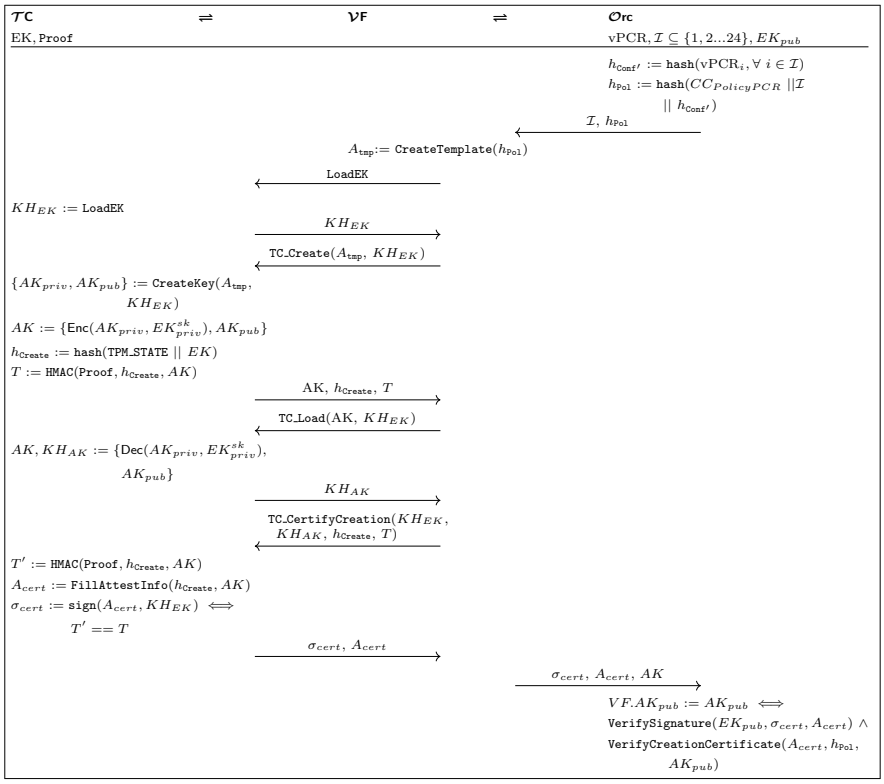

Fig. 3: Create new Attestation Key

evidence towards the: (*i*) creation of AK happened inside the $\mathcal{TC}$, (*ii*) provided policy digest governs the key, and (*iii*) proof originates from a distinct $\mathcal{TC}$.

Fig. 3 presents the underpinnings of the protocol for AK creation, where a $\mathcal{VF}$ initiates the process by constructing a "key template" based on the received policy digest. This template dictates the fundamental properties of the key, i.e., whether it is a signing key, decryption key, or both, and whether it is restricted (operates *solely* on TPM-generated objects). The template is passed to $\mathcal{TC}$, which creates an AK as a child key of EK. This process outputs a creation hash $h_{\texttt{Create}}$ and a ticket T, where T is computed with the inclusion of a secret value (`Proof`) known only by $\mathcal{TC}$, which proves that $\mathcal{TC}$ created the AK (**Step 2L**). The ticket is subsequently passed as an argument to the *certifyCreation* functionality of the $\mathcal{TC}$, together with AK, to enable AK's certification using EK, which, due to being restricted, requires such indisputable evidence about the provenance of an object. The certificate and its signature are then sent to the $\mathcal{O}$rc for verification (**Step 3L-4L**). The generated AK is trusted *iff* the signature over the certificate is verified to be authentic, based on the $\mathcal{VF}$'s EK_{pub}. The certificate reflects that the AK was created to require the correct attestation policy to be used for signing operations and that the certificate includes a

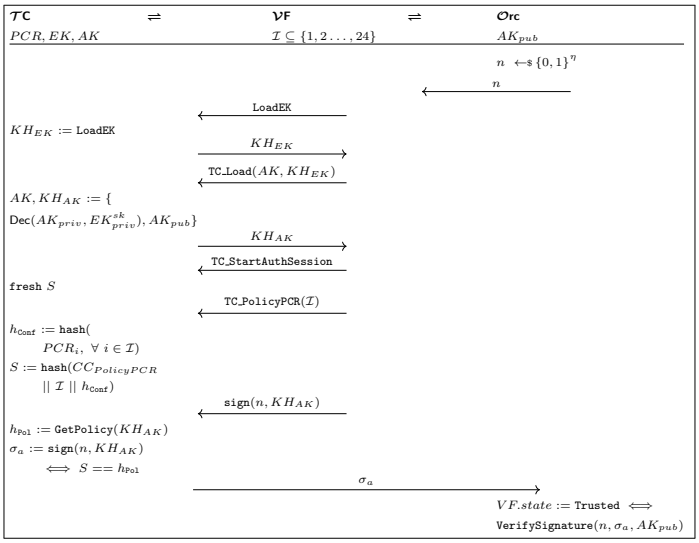

Fig. 4: Attestation by Proof

(public) value called the "magic header" whose presence proves that the signed object was created internally on $\mathcal{TC}$.

Attestation by Quote. The protocol for remote attestation using the TPM quote structure is presented in Fig. 8 (Appendix A). In this protocol, the $\mathcal{O}$rc sends a nonce n (to enforce freshness and prevent replay attacks) and a selection of PCRs to attest, $\mathcal{I}$ (**Step 4R**). The $\mathcal{VF}$ subsequently passes these arguments to $\mathcal{TC}$ which constructs a quote structure comprising the current values of the chosen PCRs, and signs it with its EK (**Step 5R**), which as with AK creation, proves that the quote structure is internal to the $\mathcal{TC}$. The quote certificate and signature are then sent to the $\mathcal{O}$rc (**Step 6R**). The quote and its signature are successfully verified by the $\mathcal{O}$rc (**Step 7R**) *iff* they are valid, and if the PCR values correspond to the artificial reference values (vPCR) managed by the $\mathcal{O}$rc.

Attestation by Proof. In the Attestation by Proof protocol (Fig. 4), $\mathcal{O}$rc *only* sends a fresh nonce n to a $\mathcal{VF}$ (**Step 5L**). If $\mathcal{VF}$ presents $Sign(n, AK_{priv})$ (**Step 6L**), where AK is a fresh and verified AK, then this is indisputable evidence that $\mathcal{VF}$ is in a trusted state (**Step 7L**). Note, that both the Attestation by Proof and Quote can *only* attest to the last known measurement. Thus, both attestation schemes are tightly coupled to run in conjunction with the update of measurements protocol for achieving run-time $\mathcal{VF}$ integrity.

6 Security Analysis

We define four core security games where an $\mathcal{A}$dv (defined in Section 3) tries to manipulate the protocol's building blocks in order to diminish the provided security guarantees on a $\mathcal{VF}$'s soundness and integrity. Our models aim to correctly detect alterations made to configurations (see Definition 1) by intermittently attesting their adherence to the reference values (vPCRs) in $\mathcal{O}$rc. We describe

the circumstances and constraints required for $\mathcal{A}\mathsf{dv}$ to win and show why this, in some cases, is impossible under the current set of assumptions.

Game 1 (Update Measurements).
Notation:
- $\boldsymbol{\beta}_{\mathtt{ID}}$ is a non-empty set of unique file identifiers (see Definition 1) provided by $\mathcal{O}\mathsf{rc}$
- $\boldsymbol{h}_{\boldsymbol{\beta}}$ denotes the $\mathcal{O}\mathsf{rc}$-conformant hash of $\beta_{\mathtt{ID}}$
- $\boldsymbol{h}'_{\boldsymbol{\beta}}$ denotes the actual hash of $\beta_{\mathtt{ID}}$, i.e., $h_\beta = \mathtt{hash}(\mathcal{T}\mathsf{rce}(\beta_{\mathtt{ID}}))$
- $\boldsymbol{i}$ denotes the PCR that must be accumulated with the new measurement(s)
- $\boldsymbol{\phi}$ denotes the $\mathcal{O}\mathsf{rc}$-conformant value of PCR_i after remeasuring $\boldsymbol{\beta}_{\mathtt{ID}}$; $\phi = \mathtt{hash}(\mathrm{PCR}_i || h_\beta)$

The Game:
1. **Goal**: $\mathcal{A}\mathsf{dv}$ conceals non-conformant configurations by deceiving the remeasurement process.
2. **Setup**: $\mathcal{A}\mathsf{dv}$ in $\mathcal{V}\mathsf{F}$ with **C-1** and **C-2**.
3. **Challenge**: An update request, $\mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{upd}}$, containing the pair $\{\beta_{\mathtt{ID}}, i\}$ is received from $\mathcal{O}\mathsf{rc}$ and is accessible by $\mathcal{A}\mathsf{dv}$, where $\beta_{\mathtt{ID}}$ identifies a file object blemished by $\mathcal{A}\mathsf{dv}$.
4. **Response**: N/A.
5. **Win condition**: $\mathcal{A}\mathsf{dv}$ wins the game *iff* $\mathrm{PCR}_i = \phi$, although $h'_\beta \neq h_\beta$.

Game 2 (Create new Attestation Key).
Notation:
- $\mathcal{I}$ identifies a set of PCRs
- $h_{\mathtt{Pol}}$ denotes the policy for using the key
- A_{cert} denotes the creation certificate
- σ_{cert} denotes the signature over A_{cert}
- AK is public part of the key created
- EK_{priv} denotes the secret endorsement key

The Game:
1. **Goal**: $\mathcal{A}\mathsf{dv}$ returns a verifiable key that $\mathcal{A}\mathsf{dv}$ can use at own discretion.
2. **Setup**: $\mathcal{A}\mathsf{dv}$ in $\mathcal{V}\mathsf{F}$ with **C-1** and **C-2**.
3. **Challenge**: $\mathcal{O}\mathsf{rc}$ provides $\mathcal{A}\mathsf{dv}$ with PCR Selection $\mathcal{I}$ and a policy digest $h_{\mathtt{Pol}}$.
4. **Response**: $\mathcal{A}\mathsf{dv}$ responds with creation certificate A_{cert}, signature σ_{cert}, and the certified public key AK, constructed by $\mathcal{A}\mathsf{dv}$ or $\mathcal{T}\mathsf{C}$.
5. **Win condition**: $\mathcal{A}\mathsf{dv}$ wins the game *iff* the key created inside the $\mathcal{T}\mathsf{C}$ contains a different policy digest than provided, while simultaneously providing a valid and verifiable certificate, public key, and signature from the restricted signing key EK_{priv}.

Fig. 5: $\mathcal{A}\mathsf{dv}$ present during updates (Game 1) and AK creation (Game 2).

When the $\mathcal{O}\mathsf{rc}$ requests a $\mathcal{V}\mathsf{F}$ to remeasure specific configurations (Game 1 in Figure 5), an $\mathcal{A}\mathsf{dv}$ must deceive the $\mathcal{O}\mathsf{rc}$ about non-conformant configurations to hide her presence. Such misleading, requires the $\mathcal{A}\mathsf{dv}$ to manipulate the local measuring process to extend the $\mathcal{T}\mathsf{C}$ PCRs with expected (bogus) "*good*" measurements instead of the actual configurations. Obviously, if $\mathcal{T}\mathsf{rce}$ is compromised, or the PCR extension of the (correct) measurement (conducted by $\mathcal{T}\mathsf{rce}$) is disrupted and never reaches $\mathcal{T}\mathsf{C}$ but $\mathcal{A}\mathsf{dv}$ instead manages to feed $\mathcal{T}\mathsf{C}$ with hashes that reflect "bogus" measurements, then $\mathcal{A}\mathsf{dv}$ wins Game 1. However, if this were to be possible, then we could never trust the measurements in $\mathcal{T}\mathsf{C}$. The enforcement of **P-4** and **P-5** on a $\mathcal{V}\mathsf{F}$ overcomes such attacks. **P-4** ensures that $\mathcal{A}\mathsf{dv}$ cannot tamper with the execution of $\mathcal{T}\mathsf{rce}$, and can in practice be achieved using more complicated (and resource-heavy) attestation methods, such as Control Flow Attestation (CFA) [15]. The latter, **P-5**, requires that a $\mathcal{V}\mathsf{F}$ always enforces the LTL invariant given in Eq. (1). This invariant states that when a $\mathcal{V}\mathsf{F}$ receives a measurement update request, $\mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{upd}}$, then the $\mathcal{V}\mathsf{F}$ must not process further requests that create new attestation keys, $\mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{createAK}}$, since they rely on the correctness of the PCRs. Only when $\mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{upd}}$ has been properly processed and the PCRs have been extended with the new measurements as requested by $\mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{upd}}$, then a $\mathcal{V}\mathsf{F}$ may proceed to process $\mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{createAK}}$ requests. Together, these properties guarantees prohibit $\mathcal{A}\mathsf{dv}$ from ever winning Game 1.

$$\mathbf{G} : \Big\{ \Big[\mathit{received}\left(\mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{upd}}\right) \wedge \mathit{process}\left(\mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{upd}}\right) \Big] \rightarrow \Big[\Big(\neg \mathit{process}\left(\mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{createAK}}\right) \Big) \mathbf{U} \; \left(PCR_i = \mathtt{hash}\left(PCR_i, \mathtt{hash}\left(\mathcal{T}\mathsf{rce}(\beta_{\mathtt{ID}})\right)\right)\right) \Big] \Big\}, \text{where } i,\ \beta_{\mathtt{ID}} \in \mathsf{Req}^{\mathcal{O}\mathsf{rc}}_{\mathrm{upd}} \tag{1}$$

In Game 2, an $\mathcal{A}\mathsf{dv}$ tries to exploit the AK creation and certification process in such a way that $\mathcal{O}\mathsf{rc}$ believes that the created key can only be used when the PCR values reflect the correct attestation policy, but $\mathcal{A}\mathsf{dv}$ can use it at her discretion. This win condition is inherently difficult for $\mathcal{A}\mathsf{dv}$ to achieve since the $\mathcal{O}\mathsf{rc}$ requires a fresh and verifiable certificate and a signature over this certificate (Fig. 3). The signature cannot be forged by the $\mathcal{A}\mathsf{dv}$ since it originates from the $\mathcal{TC}$'s secret EK. Furthermore, as described in Section 5.2, the certificate object must be generated by the $\mathcal{TC}$ in order to be signable by the restricted EK, evident through the inclusion of the magic header in the certificate. The only option for the $\mathcal{A}\mathsf{dv}$ is to alter the policy digest during key creation, which will inevitably be discovered by the $\mathcal{O}\mathsf{rc}$, either through the actual policy digest in the returned key or if the hash of the key is not the certified name in the certificate. Since the policy digest is unique and strongly linked to the PCR contents, the magic header, and the fact that the EK is secret and restricted, it is impossible for $\mathcal{A}\mathsf{dv}$ to win this game (under current assumptions).

Game 3 (Attestation by Quote).
Notation:
- σ_a denotes the signature over the certificate
- $Q_{\mathtt{Cert}}$ denotes the quote certificate
- n denotes the challenge (random number)
- EK_{priv} is the secret endorsement key
- $\mathcal{I}$ identifies a set of PCRs.

The Game:

1. **Goal**: $\mathcal{A}\mathsf{dv}$ presents valid signature and certificate with PCR values that hide $\mathcal{A}\mathsf{dv}$ presence.
2. **Setup**: $\mathcal{A}\mathsf{dv}$ in $\mathcal{V}\mathsf{F}$ with **C-1** and **C-2**.
3. **Challenge**: $\mathcal{O}\mathsf{rc}$ challenges $\mathcal{A}\mathsf{dv}$ with n and PCR Selection $\mathcal{I}$.
4. **Response**: $\mathcal{A}\mathsf{dv}$ responds with certificate $Q_{\mathtt{Cert}}$ and signature σ_a, constructed by $\mathcal{A}\mathsf{dv}$ or $\mathcal{TC}$.
5. **Win condition**: $\mathcal{A}\mathsf{dv}$ wins *iff* $\mathcal{O}\mathsf{rc}$ can verify σ_a over $Q_{\mathtt{Cert}}$ (containing n) signed by EK_{priv} and the accumulated digest in $Q_{\mathtt{Cert}}$ matches $\mathcal{O}\mathsf{rc}$'s accumulated digest from vPCR.

Game 4 (Attestation by Proof).
Notation:
- σ_a denotes the challenge signature
- n denotes the challenge (random number)
- AK_{priv} is the private attestation key

The Game:

1. **Goal**: $\mathcal{A}\mathsf{dv}$ provides verifiable signature over challenge, despite of modified binaries.
2. **Setup**: $\mathcal{A}\mathsf{dv}$ in $\mathcal{V}\mathsf{F}$ with **C-1** and **C-2** (see Section 3), and AK has been deployed.
3. **Challenge**: $\mathcal{O}\mathsf{rc}$ (or secondary $\mathcal{V}\mathsf{F}$) challenges $\mathcal{A}\mathsf{dv}$ with n.
4. **Response**: $\mathcal{A}\mathsf{dv}$ responds with σ_a, either constructed by $\mathcal{A}\mathsf{dv}$ or $\mathcal{TC}$.
5. **Win condition**: $\mathcal{A}\mathsf{dv}$ wins the game *iff* $\mathcal{O}\mathsf{rc}$ (or secondary $\mathcal{V}\mathsf{F}$) can verify σ_a being a signature over n signed by AK_{priv}.

Fig. 6: $\mathcal{A}\mathsf{dv}$ present during Attestation by Quote (Game 3) and Proof (Game 4).

In Game 3, an $\mathcal{A}\mathsf{dv}$ tries to falsely convince the $\mathcal{O}\mathsf{rc}$ that binaries have not been manipulated by exploiting either the quoting process or building a fraudulent certificate. The certificate comprises the current PCR values and the nonce from $\mathcal{O}\mathsf{rc}$. Assuming the accumulated PCRs reflect the $\mathcal{A}\mathsf{dv}$'s presence, she can try to tamper with the certificate creation process to reflect a forged PCR digest. Unfortunately for the $\mathcal{A}\mathsf{dv}$, $\mathcal{TC}$ will be reluctant to sign the forged certificate since it did not create it.

Note that the PCRs will only reflect malicious alterations to configurations if an update of the measurements is requested and executed *after* the $\mathcal{A}\mathsf{dv}$ has tampered with the configurations. We denote the time of an update as t_{up}, time of compromise as $t_{\mathcal{A}\mathsf{dv}}$ and time of attestation as t_{att}. Our assumption (that PCRs reflect $\mathcal{A}\mathsf{dv}$'s presence) holds (and $\mathcal{A}\mathsf{dv}$ loses) if $t_{att} > t_{up} > t_{\mathcal{A}\mathsf{dv}}$. If $\mathcal{A}\mathsf{dv}$ can precisely time the manipulation of binaries such that $t_{up} < t_{\mathcal{A}\mathsf{dv}} < t_{att}$, then the PCRs will not reflect her presence and $\mathcal{A}\mathsf{dv}$ will win the game (although will be detected in the next measurements update). This attack is called a Time-of-check to Time-of-use (TOCTOU) [19], which is a disadvantage in the proposed protocol and is discussed in more detail in Appendix B. However, as $t_{up}(n) - t_{up}(n+1) \rightsquigarrow 0$ (approaches 0), the disadvantage

becomes insignificant since $\mathcal{A}\mathsf{dv}$'s time window for malevolent behavior becomes very small, but will have an impact on the overall resource consumption of the system.

Game 4 shows how an $\mathcal{A}\mathsf{dv}$ can try to exploit the signing process in order to provide a valid signature over the challenge n using the issued attestation key while having modified binaries. The overall goal is to convince the verifier that no manipulation of binaries has happened. Recall that usage of the key is bound to particular contents of the PCRs. The PCRs reflect the binary states; hence the $\mathcal{A}\mathsf{dv}$ cannot execute the cryptographic signing operation while having modified the binaries, assuming the registers indeed reflect such modifications. This, of course, is also affected by the TOCTOU attack mentioned earlier, and in this case, with a more severe impact. After every update, a new attestation key has to be deployed, and local reference values have to be updated, taking essential resources from the primary operations of the system. We provide a more thorough analysis of this issue in Appendix B.

7 Experimental Performance Evaluation

Experimental Setup. Our testbed is deployed on a computer equipped with an Intel(R) Core(TM) i7-8665U CPU @ 1.90-2.11GHz running the Windows 10 OS. The main goal of this setup is to evaluate the potential overhead of using a $\mathcal{TC}$ that will, in turn, allow us to assess the overall protocol scalability towards providing verifiable $\mathcal{VF}$ integrity evidence. Therefore, we have opted out of creating a true scale test environment with separate entities, but a single binary file containing all components. To evaluate the performance of CloudVaults, we constructed the protocols and tested them against IBM's software TPM V1628 using the IBM TSS [12] V1.5.0. Each experiment (protocol) is performed 1,000 times. Note that since we rely on a software TPM as the RoT, of a $\mathcal{VF}$, we chose to create an attestation primary-key as an alternative to the EK for key storage, which adds a small overhead each time the AK is used. Also, we chose to use an ECC key as the EK, instead of an RSA-based EK.

Table 2: Timings of CloudVault's protocols (time in ms). Note that the hashing is done without any secure hashing schemes and might be slower in practice.

Command	Activity	Mean	95% (low)	95% (high)	Description
CreateAK	Prepare	0.01	<0.01	0.01	Compute expected vPCR
	Create	15.92	15.80	16.05	Create AK in $\mathcal{TC}$
	Verify	1.03	1.01	1.05	Verify certificate and key
	Total	16.96	16.81	17.11	
Update	Prepare	<0.01	<0.01	<0.01	Extend vPCR
	Hash/Extend	1.42	1.35	1.49	Hash file(s) and extend PCR
	Total	1.42	1.35	1.49	
Quote	Prepare	0.02	0.01	0.03	Create a nonce
	Quote	8.67	8.56	8.78	Sign PCRs with EK
	Verify	0.83	0.80	0.85	Verify quote and certificate
	Total	9.51	9.37	9.65	
Proof	Prepare	0.01	<0.01	0.02	Create a nonce
	Sign	10.83	10.76	10.89	Sign nonce
	Verify	0.84	0.79	0.88	Verify signature
	Total	11.67	11.56	11.79	

Performance Results. Our experiments (Table 2) highlight the efficiency of our protocols. The entire process of creating an AK takes no more than $\approx$ 17 ms (on average), while including the update of binary measurements still requires less than 20

ms (see Appendix B for more details and a comparison to a HW-based TPM implementation). The enhanced attestation schemes are also efficient (< 12 ms), however, without considering any possible network delay that may be present when communicating the attestation data between the $\mathcal{P}\mathsf{rv}$ and $\mathcal{V}\mathsf{rf}$. With both supporting routines and attestation schemes being extremely lightweight, we can achieve low-cost, rapid attestation capabilities and provide advanced trust assurance services without consuming a lot of computational resources. Such capabilities not only ensure trust from the perspective of the $\mathcal{O}\mathsf{rc}$ but further facilitate bilateral trust assurance (even) between service graphs. As described in Section 6, higher levels of trustworthiness result in more resources being needed. That is why it is imperative for the attestation protocols to be lightweight enough without, however, impeding on their accuracy and correctness. To better demonstrate the achieved effectiveness, we use Eq. (2) (Appendix B) to determine how fast we can detect binary manipulation. In the worst-case scenario, where an $\mathcal{A}\mathsf{dv}$ tampers with a binary just after an update, she will remain undetected for *at most* 293.40 ms, if we utilize as little as 20% of the CPU time.

The ease of operating CloudVault's protocols, including their efficiency, makes the framework highly applicable to be integrated into large-scale networks. While we did not take processing- or network-delay into consideration and only use the AK once, the experiments show that the time of detection is in the order of seconds. In Fig. 10 (Appendix B), we further see that even with 10% utilization, we can still detect a change after ≈ 1 second, making it extremely difficult for an $\mathcal{A}\mathsf{dv}$ to manipulate CloudVaults.

8 Conclusions

In this paper, we proposed CloudVaults, a multi-level security verification framework for supporting trust aware SGCs with verifiable evidence on the integrity assurance and correctness of the comprised containers: from the trusted launch and configuration to the run-time attestation of low-level configuration properties. Based on our analysis, we described how a $\mathcal{V}\mathsf{F}$ achieves privacy-preserving integrity correctness and how to utilize vPCRs for binary data integrity with a virtual-based RoT. Our prototype and the evaluation results demonstrate that our architecture can satisfy the privacy, security, and efficiency requirements. Furthermore, by considering the salient characteristics of remote attestation, we identified several open research challenges. We believe that if these challenges are tackled now while container-based CIV is still at an early stage, this emerging security mechanism can reach its full potential.

9 Acknowledgment

This work was supported by the European Commission, under the ASTRID and RAINBOW projects; Grant Agreements no. 786922 and 871403, respectively.

References

1. Abera, T., et al.: C-FLAT: Control-Flow Attestation for Embedded Systems Software. In: Proceedings of the 2016 ACM SIGSAC CCS Conf. pp. 743–754
2. Bailey, K.A., Smith, S.W.: Trusted virtual containers on demand. In: 5th ACM Workshop on Scalable Trusted Computing. p. 63–72. STC '10 (2010)
3. Beck, M.T., Botero, J.F.: Scalable and Coordinated Allocation of Service Function Chains. Comput. Commun. **102** (2017)
4. Chen, L., et al.: A protocol for property-based attestation. In: 1st ACM workshop on Scalable trusted computing (2006)

5. Chen, L., et al.: Property-based attestation without a trusted third party. In: International Conference on Information Security. pp. 31–46. Springer (2008)
6. Datta, A., et al.: A logic of secure systems and its application to trusted computing. In: 30th IEEE Symposium on S&P. pp. 221–236. IEEE (2009)
7. De Benedictis, M., Lioy, A.: Integrity verification of Docker containers for a lightweight cloud environment. Future Generation Computer Systems **97**, 236–246
8. Dimitriou, T., Giannetsos, T., Chen, L.: REWARDS: Privacy-preserving rewarding and incentive schemes for the smart electricity grid and other loyalty systems. Computer Communications **137**, 1 – 14 (2019)
9. Dolev, D., Yao, A.: On the security of public key protocols. IEEE Transactions on information theory **29**(2), 198–208 (1983)
10. Giannetsos, T., Krontiris, I.: Securing V2X Communications for the Future: Can PKI Systems Offer the Answer? In: 14th Int. ARES Conf (2019)
11. Giannetsos, T., et al.: Arbitrary Code Injection Through Self-propagating Worms in Von Neumann Architecture Devices. Comput. J. **53**(10), 1576–1593
12. Goldman, Ken: IBM's Software TPM 2.0 and TSS, `https://sourceforge.net/projects/ibmswtpm2/,https://sourceforge.net/projects/ibmtpm20tss`
13. Ibrahim, F.A., Hemayed, E.E.: Trusted cloud computing architectures for infrastructure as a service: Survey and systematic literature review. Computers & Security **82**, 196–226 (2019)
14. Jamshidi, P., Pahl, C., Mendonça, N.C., Lewis, J., Tilkov, S.: Microservices: The journey so far and challenges ahead. IEEE Software **35**(3), 24–35 (2018)
15. Koutroumpouchos, N., et al.: Secure edge computing with lightweight control-flow property-based attestation. In: 2019 IEEE Conf. on Network Softwarization
16. Larsen, B., Debes, H.B., Giannetsos, T.: Cloudvaults C implementation. In: https://github.com/astrid-project/Configuration-Integrity-Verification (2020)
17. Lauer, H., et al.: Bootstrapping Trust in a "Trusted" Virtualized Platform. In: 1st ACM Workshop on Workshop on Cyber-Security Arms Race (2019)
18. Luo, W., Shen, Q., Xia, Y., Wu, Z.: Container-IMA: A privacy-preserving Integrity Measurement Architecture for Containers. In: 22nd International Symposium on Research in Attacks, Intrusions and Defenses ({RAID} 2019). pp. 487–500 (2019)
19. Nunes, I.D.O., Jakkamsetti, S., Rattanavipanon, N., Tsudik, G.: On the TOCTOU Problem in Remote Attestation. arXiv preprint arXiv:2005.03873 (2020)
20. Roemer, R., et al.: Return-Oriented Programming: Systems, Languages, and Applications. ACM Trans. Inf. Syst. Secur. **15**(1), 2:1–2:34 (Mar 2012)
21. Sabella, D., et al.: Mobile-Edge Computing Architecture: The role of MEC in the Internet of Things. IEEE Electronics Magazine (2016)
22. Sabt, M., Achemlal, M., Bouabdallah, A.: Trusted execution environment: what it is, and what it is not. In: 2015 IEEE Trustcom. pp. 57–64. IEEE (2015)
23. Sailer, R., et al.: Design and Implementation of a TCG-based Integrity Measurement Architecture. In: USENIX Security symposium. pp. 223–238 (2004)
24. TCG: TCG Guidance for Securing Network Equipment Using TCG Technology Version 1.0 Revision 29 (jan 2018), `https://trustedcomputinggroup.org/wp-content/uploads/TCG_Guidance_for_Securing_NetEq_1_0r29.pdf`
25. Whitefield, J., et al.: Privacy-enhanced capabilities for VANETs using direct anonymous attestation. In: IEEE Vehicular Networking Conference
26. Xin, S., Zhao, Y., Li, Y.: Property-based remote attestation oriented to cloud computing. In: IEEE Conf. on Computational Intl. and Security (2011)

APPENDIX

A Configuration Integrity Verification Sub-Protocols (Extended)

In this section, we present the remaining protocols, *Update Measurements* (Fig. 7) and *Attestation by Quote* (Fig. 8). To initiate the re-measurement process of a $\mathcal{V}\mathsf{F}$, the $\mathcal{O}\mathsf{rc}$ sends an *Update Measurements* request detailing which file object should be re-measured (using $\mathcal{T}\mathsf{rce}$) and into which PCR registers it should be registered. Note that the $\mathcal{O}\mathsf{rc}$ knows what the correct PCR values should be since it also accumulates the artificial PCR registers (vPCRs), as part of the attestation policy, corresponding to the target $\mathcal{V}\mathsf{F}$. To perform a verifiable assessment of the current state of a $\mathcal{V}\mathsf{F}$'s PCRs, the $\mathcal{O}\mathsf{rc}$ sends an *Attestation by Quote* request detailing which PCR registers should be included in the quote, denoted $\mathcal{I}$, and a nonce n to enforce freshness and prevent replay attacks. After the $\mathcal{V}\mathsf{F}$ has securely instructed its $\mathcal{T}\mathsf{C}$ to construct the necessary quote certificate and signature (over the certificate), it forwards them to the $\mathcal{O}\mathsf{rc}$. If the signature over the quote is deemed correct (signed by the $\mathcal{V}\mathsf{F}$'s EK_{priv}), the certificate can the be verified for determining whether it contains the "magic header" (proving that it was generated inside the $\mathcal{T}\mathsf{C}$, as detailed in Section 5.2) and whether the PCR values correspond to the trusted PCR values (vPCR) that were artificially accumulated on the $\mathcal{O}\mathsf{rc}$ when the $\mathcal{V}\mathsf{F}$ was last updated.

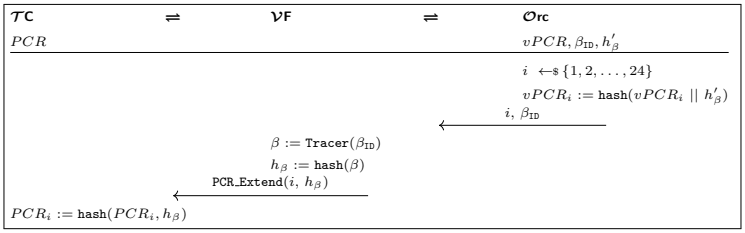

Fig. 7: Update PCR Measurements

B Timings and Benchmarks

Based on our adversarial model (Section 3), we do not consider $\mathcal{A}\mathsf{dv}$ that can perform transient attacks whereby alterations to binaries are only detectable for a short time. Thus, any alterations to binaries by an $\mathcal{A}\mathsf{dv}$ will be detected when the $\mathcal{V}\mathsf{F}$ is re-measured and attested, as shown in Fig. 9. The *advantage* of an $\mathcal{A}\mathsf{dv}$ is defined as the time that she can remain undetected. If, for instance, the *Update of Measurements* and *Attestation by Quote* protocols are executed immediately after the attack, then we will be able to detect any incompliant configurations from the quote structure. However, *Attestation by Proof* will inevitably take longer to complete since the creation of a new AK must occur in-between the *Update of Measurements* and *Attestation by Proof* protocols. Let t_d denote the time of detection (hence, a large t_d is desirable to $\mathcal{A}\mathsf{dv}$), u the time to execute the update routine, c the time to execute the creation of a new AK, a the time to

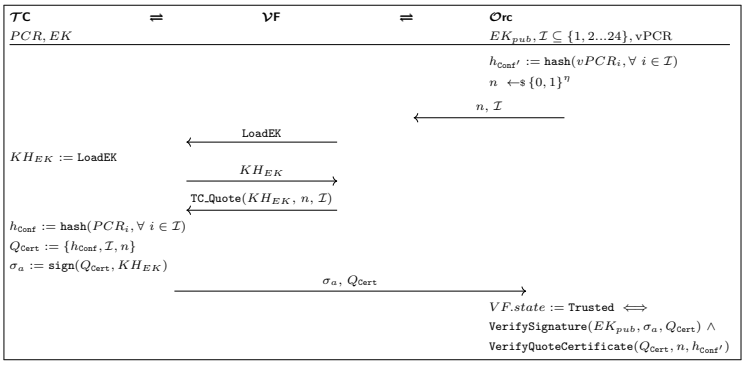

Fig. 8: Attestation by Quote

execute the attestation routine, and n the number of $\mathcal{V}$rf's that, in consecutive order, conduct *Attestation by Proof* on a specific $\mathcal{V}\mathsf{F}$ using a shared and verified AK_{pub} of that $\mathcal{V}\mathsf{F}$, respectively. We further use the variable t_{CPU} to specify the amount of CPU resources allocated to execute these routines, e.g., for 20% utilization we have $t_{CPU} = 0.20$. Using Eq. (2) we calculate the time until the $\mathcal{A}$dv is detected (t_d) (Figure 10). As we can see, the detection time (t_d) increases linearly with n (a) but decreases as we allocate more resources, t_{cpu} (b).

$$t_d = \frac{2c + a(1+n) + u}{t_{CPU}} \tag{2}$$

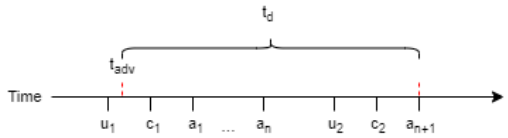

Fig. 9: Visual representation of how long an $\mathcal{A}$dv can go undetected.

Implementation Note. Writing protocols in terms of TPM calls requires reading and understanding the TPM 2.0 specification and this makes TPM development challenging and causes a high-barrier of entry. While the TPM 2.0 specification was designed to be easily maintainable, it is nevertheless challenging to read mainly due to its sheer size. It consists of over 1400 pages split into four parts which not only cover the core specifications, but also numerous errata covering the continuous development of the TPM specification. Therefore, a particular TPM will be based on the core specification and all of the relevant errata which it implements.

HW-based TPM Timings. The timings for executing the individual TPM commands of the CIV protocols are presented in Tables 3 and 4, and are performed using IBM's software (SW) TPM V1628 and the Infineon (HW) TPM 2.0 chip. The mean time is calculated from repeating all experiments 1,000 times for the SW-TPM and 100 times for the HW-TPM. The values reflect the time between executing a command in the IBM TSS [12] V1.5.0 and until receiving

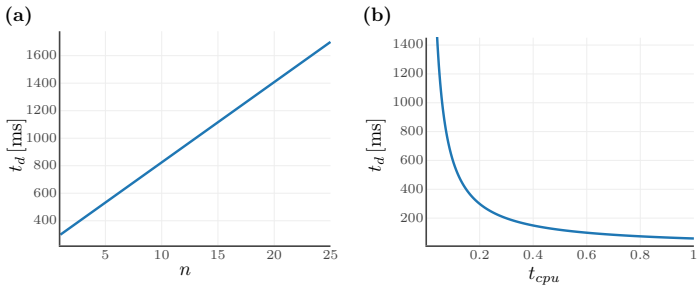

Fig. 10: (a) Changing the number of attestations each key has to do and its impact on the time of detection (20% utilization) and (b) shows how different utilization of resources impact the time of detection with one AK use.

a response. The SW-TPM timings are much faster than those when using the HW-TPM. Even though a hardware TPM has some degree of hardware accelerated cryptography, it still cannot measure itself with a modern CPU, and is not designed to do so. Applying Eq. (2) on the HW-TPM yields a time of detection as $t_d = (2 \cdot 734.89 \text{ ms} + 417.49 \text{ ms}\,(1 + 1) + 6.09 \text{ ms})/0.20 = 11.5$ s, which is indeed larger than that for the SW-TPM. However, these values are somewhat misleading since the host CPU's utilization does not have any effect on the HW-TPM as it executes the operations on the hardware chip itself. By evaluating the TPM command execution and application timings, we can see that the most time-consuming operations are those executed on the TPM, which is why the impact of the CPU utilization using an HW-TPM is significantly lower. Removing this constraint from Eq. (2) gives us $t_d = 2.310$ s (excluding the host times, such as verification, nonce generation, etc.). Additionally, the "create primary key" function is extremely time-consuming, which is why it might be useful to load this AK from NV storage.

C CloudVaults Formal Trust Models

Table 5 presents the axioms (with descriptions) that must be satisfied by the various components (i.e., $\mathcal{VF}$, vTPM, $\mathcal{O}\mathsf{rc}$) involved in the creation and management of SGCs to ensure the CIV security properties (Section 4).

Table 3: Mean time (in ms) of using SW- and HW-TPM for updating measurements and creating a new AK.

Command	SW	HW
Update		
TPM2_PCR_Extend	0.44	6.09
Total	0.44	6.09
Create		
TPM2_CreatePrimary	0.92	238.35
TPM2_Create	0.98	243.75
TPM2_Load	0.44	58.04
TPM2_Load	0.33	59.83
TPM2_CertifyCreation	5.18	123.13
TPM2_FlushContext	1.64	4.10
TPM2_FlushContext	1.60	3.69
TPM2_FlushContext	2.21	4.00
Total	13.3	734.89

Table 4: Mean time (in ms) of using SW- and HW-TPM for Attestation by Quote and Proof.

Command	SW	HW
Quote		
TPM2_CreatePrimary	3.36	244.96
TPM2_Load	1.57	51.69
TPM2_Quote	2.16	112.71
TPM2_FlushContext	0.93	3.71
TPM2_FlushContext	0.89	3.77
Total	8.91	416.84
Proof		
TPM2_CreatePrimary	3.33	241.38
TPM2_Load	1.70	54.17
TPM2_StartAuthSession	1.38	6.72
TPM2_PolicyPCR	0.37	10.71
TPM2_Sign	3.06	95.36
TPM2_FlushContext	0.97	4.12
TPM2_FlushContext	0.91	4.98
Total	11.72	417.44

Table 5: Formalized enablers (axioms) to achieve the CIV security properties.

$$\mathbf{Ax_1}\ \left[\forall vf \in \left\{\sigma | \sigma \in \bigcup_{j=1}^{n} \mathcal{SG}_j\right\}\right]\ \left[\exists!\, vtpm\right] : Trusted_{VTPM}(vtpm) \wedge Bound(vtpm, vf) \wedge HasInv(vtpm, vf.EK_{pub}) \wedge Signed(vtpm.pcr[idx], vtpm.EK_{priv}) \wedge Equals(vtpm.pcr[idx], vf.vpcr[idx]) \equiv Conformant_{config}(vf.vpcr),\ \forall\, idx \in indices(vf.vpcr)$$

(Ax_1 for **P-1**): Any $\mathcal{V}\mathsf{F}$, with a unique and proper vTPM, has conformant configurations *iff* it proves through signing its PCRs with EK_{priv} that its PCRs are conformant to the policy (vPCR) issued by $\mathcal{O}\mathsf{rc}$, where $\text{EK}_{priv} = \mathcal{V}\mathsf{F}.\text{EK}_{pub}^{-1}$ and EK_{pub} is trusted by $\mathcal{O}\mathsf{rc}$.

$$\mathbf{Ax_2}\ \left[\nexists vf \in \left\{\sigma | \sigma \in \bigcup_{j=1}^{n} \mathcal{SG}_j\right\}\right] : \neg Conformant_{config}(vf.vpcr) \equiv Trusted_{SGC}(\mathcal{SG})$$

(Ax_2 for **P-2**): The entire SGC ($\mathcal{SG}$) is considered trusted *iff* all of its $\mathcal{V}\mathsf{F}$s are attested correctly, such that all configurations are conformant to their respective attestation policies.

$$\mathbf{Ax_3}\ \left[\forall vf, vtpm, pol\right] : Trusted_{VTPM}(vtpm) \wedge Bound(vtpm, vf) \wedge Created_{AK}(vtpm.ak, pol) \wedge Signed(vtpm.ak, vtpm.EK_{priv}) \wedge Has_{policy}(vtpm.ak, pol) \equiv Trusted_{AK}(vtpm.ak)$$

(Ax_3 for **P-3**): An attestation key *ak* is trusted *iff* it is created on a trusted vTPM and contains the appropriate policy *pol*, which reflects the appropriate attestation policy (vPCR).