Interoperable node integrity verification for confidential machines based on AMD SEV-SNP

Davi Pontes[1], Fernando Silva[1], Anderson Melo[1], Eduardo Falcão[2], Andrey Brito[1]

[1]*Federal University of Campina Grande, Department of Computing and Systems, Campina Grande, Brazil*

[2]*Federal University of Rio Grande do Norte, Department of Computer Engineering and Automation, Natal, Brazil*

Abstract

Confidential virtual machines (CVMs) are cloud providers' most recent security offer, providing confidentiality and integrity features. Although confidentiality protects the machine from the host operating system, firmware, and cloud operators, integrity protection is even more useful, enabling protection for a wider range of security issues. Unfortunately, CVM integrity verification depends on remote attestation protocols, which are not trivial for operators and differ largely among cloud providers. We propose an approach for abstracting CVM attestation that leverages an open-source standard, Cloud Native Foundation's Secure Production Identity Framework for Everyone (SPIFFE). Our approach can integrate smoothly even when applications are unaware of CVMs or the SPIFFE standard. Nevertheless, our implementation inherits SPIFFE flexibility for empowering access control when applications support SPIFFE. In terms of performance, CVMs incur an additional 1.3 s to 21.9 s in boot times (it varies with the cloud environment), a marginal degradation for CPU, RAM, and IO workloads (maximum degradation of 2.6%), and low but not imperceptible degradation for database workloads (between 3.6% to 7.13%). Finally, we provide usability mechanisms and a threat analysis to help users navigate cloud providers'

Email address: {davi.pontes,fernando.silva,anderson.melo}@lsd.ufcg.edu.br, eduardo@dca.ufrn.br, andrey@computacao.ufcg.edu.br (Davi Pontes[1], Fernando Silva[1], Anderson Melo[1], Eduardo Falcão[2], Andrey Brito[1])

Preprint submitted to Journal of Internet Services and Applications, 2023

different CVM implementations and resulting guarantees.

Keywords: Confidential virtual machines, Confidential Computing, Cloud computing, Attestation, Interoperability, AMD SEV-SNP

1. Introduction

Confidential computing aims to improve the security of cloud and other shared infrastructures by removing the trust in infrastructure providers and their human operators. It achieves this level of trust by creating isolated environments, known as Trusted Execution Environments (TEEs), which rely on hardware features for confidentiality and integrity. Agencies such as ITU, CISA, and NIST already recommend TEEs to protect critical services in cloud environments [23, 17].

TEEs provide confidentiality and integrity features [31]. The confidentiality feature protects the data and applications from the host by encrypting the main memory with keys that never leave the processor. Confidentiality is transparently managed by an embedded engine that encrypts and decrypts data as they leave and arrive in the processor. Memory is then protected from other environments (e.g., OSs, processes, hypervisors), trusted or not, that run on the same machine.

In contrast, the most interesting usages of integrity cannot be completely transparent. On the basic level, the processor should manage the integrity of the memory, ensuring that memory pages were not corrupted or replayed (i.e., replaced by previous versions). However, leveraging TEEs to verify that environments have been created according to the user's requirements and without malicious interferences from infrastructure providers or their human operators is much more complex.

As discussed in the next section, TEEs may come in two flavors: enclaves or confidential virtual machines (CVMs) [40]. Enclaves protect a specific service and are part of a regular operating system process. Alternatively, CVMs protect a complete virtual machine, including the operating system, runtime, and services. Both have advantages and disadvantages, but in either case, verifying their integrity requires verifying all components involved, from the version of the firmware in the processor to the integrity of the binary used to create the environment. To make matters worse, providers offer different subsets of functionalities and different interfaces to access verification APIs.

Although complex, verifying trust in software components and services is in high demand. Several security standardization agencies have recommended the Zero Trust Architecture (ZTA) model to compartmentalize components and support verification [18, 36, 11]. ZTA reduces the attack surface of cloud-native applications by requiring fine-grained resource identification as well as authentication, authorization, and encryption on all communication and service accesses. Therefore, identities are the paramount pillar of ZTAs, and issuing them can become more robust when leveraging the security of TEEs' remote attestation.

To combine ZTAs and confidential computing, we leverage projects from the Cloud Native Computing Foundation[1] (CNCF). More specifically, we use an identity standard, SPIFFE, and its reference open-source implementation, SPIFFE Runtime Environment (SPIRE). We propose an extension that provides an attestation mechanism based on AMD SEV-SNP technology [3] for confidential virtual machines. The attestation will issue identities in the form of regular X.509 certificates [12] or JSON Web Tokens (JWTs) [24] that can be used for authentication while abstracting all confidential computing peculiarities of the major public cloud providers and on-premise installations.

The outcome facilitates the proper use of CVMs, verifying whether they are running in AMD SEV-SNP CVMs and verifying the integrity of the execution platform, besides strengthening ZTA deployments. Furthermore, as different cloud providers' adoption of AMD SEV-SNP restricts access to some of the features, we discuss the different trust assumptions and provide usability mechanisms, such as relying on Trusted Platform Module (TPM), to circumvent differences without compromising usability.

In summary, our contributions are the following:

1. We are the first to propose the usage of confidential VMs with the CNCF identity framework. This is implemented as a SPIFFE-compliant extension to the SPIRE framework that enables attestation of CVMs in public cloud and on-premise installations;
2. We propose a new mechanism to combine properties used by different platform attestation plugins;

[1]The Cloud Native Computing Foundation (https://www.cncf.io/) is a hub for open-source, vendor-neutral solutions for cloud-native computing. It hosts popular projects such as Kubernetes and Prometheus.

3. We investigate the threat models for different cloud providers and create mechanisms to abstract from their differences;
4. We evaluate our solution in the major cloud providers currently supporting AMD SEV-SNP while discussing design decisions and implementation results with the open-source community.

We organize the document as follows. The next section presents some background on fundamental technologies, such as Zero Trust, SPIFFE, SPIRE, AMD SEV-SNP, and TPM. Section 3 defines the threat model of our work. Then, Section 4 reviews the current support for AMD SEV-SNP for on-premise and public cloud installations, summarizing their limitations. Section 5 details the plugin design, implementation, and usage. After that, Section 6 explains how to verify the integrity of CVMs. Section 7 evaluates our implementation, including a security analysis detailing the attacks we consider and how we mitigate them. Finally, Sections 8 and 9 summarize related work and our contributions.

2. Background

Zero Trust is a crucial concept regarding the security of cloud native apps. SPIFFE/SPIRE provides guidelines that help implement ZTA for applications following a microservices architecture. Trusted computing technologies such as AMD SEV-SNP and TPM provide confidentiality and integrity guarantees that strengthen ZTAs. These concepts are discussed next.

2.1. Zero Trust

The Zero Trust (ZT) model is a set of principles and guidelines to secure communications no matter where they occur, granting access to resources per session and based on device, behavioral, and environmental attributes, always providing the least privilege possible, and assessing such access continuously [13, 36].

Issuing and managing identities is a fundamental pillar of implementing all ZT principles. There is no secure and authenticated communication without software identities; they need to be constructed over strong evidence, and finally, they should be renewed periodically following a robust issuance process. Although there is no specific set of activities to implement Zero Trust, some practical approaches, e.g., SPIFFE and SPIRE, follow the aforementioned principles.

2.2. SPIFFE and SPIRE

With the rise in the usage of automated infrastructures and cloud-native computing, the need for provisioning identities for software components has also increased. Software components have dynamic life cycles orchestrated by tools such as Kubernetes (K8S) [25]. Therefore, the provisioning of these identities should also be automated. In addition, tendencies such as ZT, discussed in the previous section, lead to the need for unique identities for each component or microservice.

2.2.1. The SPIFFE Standard

The Secure Production Identity Framework for Everyone (SPIFFE) is a specification [38] that determines how to operationalize software identities in a platform and technology-agnostic way. Its primary purpose is to address the issue of having a last credential unprotected. In other words, the credentials used to retrieve critical configurations and other credentials should be seamlessly provided to the service or node. SPIFFE solves this problem by combining attestation procedures and signed identities.

In the SPIFFE vocabulary, microservices are named workloads. When attested, a workload receives a SPIFFE Verifiable Identity Document (SVID), a cryptographically verifiable object representing the workload's identity. Currently, SVIDs can be X.509 certificates or JWTs. Each SVID carries a SPIFFE ID, a string used to identify workloads. A SPIFFE ID is composed of a domain name, such as *spiffe://example.com/*, followed by a human-friendly name (e.g., *client*), an opaque one (e.g., a hash or unique id), or a combination of both (e.g., organized in a path hierarchy).

Workloads obtain their SVIDs (and corresponding private keys, in the case of X.509 SVIDs) and trust bundles (one or more Certificate Authority (CA) certificates to validate signatures of other workloads) through the Workload API. To avoid the need for workloads to have bootstrap credentials[2], this API does not require authentication. Instead, the API collects information from other sources, such as the operating system, to derive relevant properties that help identify which SVID a workload should receive.

[2]This last credential problem is also known in the SPIFFE community as the bottom-turtle problem, referring to an anecdote in which the world rested on the back of a turtle, which in turn rested on the back of another turtle, and so on [20], see also https://en.wikipedia.org/wiki/Turtles_all_the_way_down.

The process of deriving such properties is named attestation. The properties collected to perform the attestation are named selectors.

In addition to the attestation and delivery of the associated SVID, a SPIFFE implementation also provides the trust bundles. Implementations should also rotate SVIDs according to a provided configuration.

2.2.2. SPIFFE's reference implementation

The SPIFFE Runtime Environment (SPIRE) [38] is the reference implementation of SPIFFE. SPIRE has two main components: the server and the agent. The server is responsible for attesting the agents, delivering the agents' SVIDs, and minting the workloads' SVIDs. The agent's primary role is to attest the workloads and deliver SVIDs that match the attested selectors. SPIRE is an open-source project, and both components follow a plugin-oriented architecture. The most important plugin type for this work is the attestation plugin type, which enables the operator to configure a source for the selectors used for the attestation of a node. Having a plugin type that collects properties related to the integrity and confidentiality features used by workloads greatly simplifies the usage of TEEs.

A SPIRE infrastructure is typically composed of a server and multiple agents. Each agent is deployed on a node, which is a physical or virtual machine. The agent can serve various workloads running on the same node. The agent exposes the Workload API in a Unix Domain Socket where workloads can trigger the process of attestation and receive their SVIDs in return, as determined by the SPIFFE standard. Similarly, the server offers a Node API, where agents request their SVIDs and the SVIDs for the workloads they may eventually host. Finally, the server offers the Registration API. Service operators use the Registration API to define a SPIFFE ID (such as *spiffe://example.com/client*) and which attestation properties the workload needs to satisfy to receive an SVID with that SPIFFE ID (such as requiring that it runs on a specific K8s namespace or atop a specific VM image). An illustration of this setup is presented in Figure 1.

Because the SPIFFE ID is a simple string typically embedded as a non-critical extension in an X.509 certificate for provability, SPIFFE facilitates the interoperation between legacy and SPIFFE-aware workloads. For a legacy workload that trusts the CA associated with the trust domain, the SVID is simply a trusted certificate. In addition, if a workload is aware of SPIFFE, it can look at the SPIFFE extensions during the authorization. In the latter case, the SPIFFE ID abstracts the attestation. For example, if a

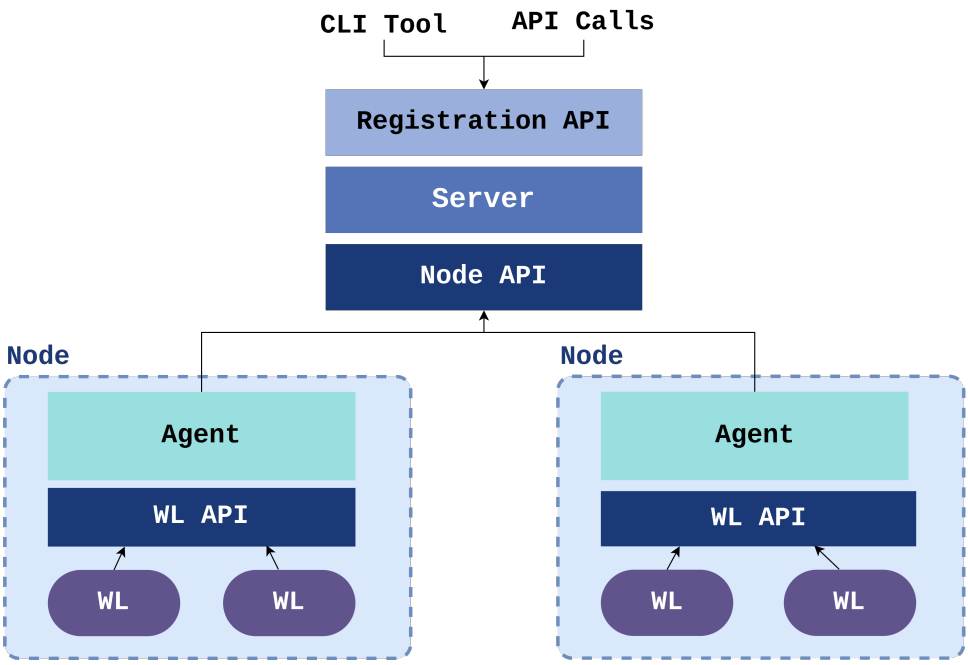

Figure 1: Basic SPIRE setup [37].

workload trusts a SPIFFE ID *spiffe://example.com/client*, it does not need to understand how the attestation was implemented, and the attestation may have different requirements depending on the trust level of the infrastructure (e.g., remote edge nodes may require a stricter attestation then self-managed local ones).

As mentioned above, a server does not autonomously decide which identities to issue and to which nodes or workloads they could be issued. This information needs to be registered on the server via the Registration API. The information related to an identity is called a registration entry and can be assigned to a node or workload. A registration entry comprises a SPIFFE ID, a set of one or more selectors, and, in the case of a workload, the parent ID. The selectors are the information collected during attestation and identify the workload or node. The SPIFFE ID is the exact identity that must be issued to the workload or node whose collected selectors are the same as those specified in the registration entry. The parent ID specifies the SPIFFE ID of a node on which the workload needs to be running to receive the SVID. The parent ID only applies to workloads, and as several nodes can share the

same SPIFFE ID, they are eligible to host the same workloads.

Figure 2 illustrates using the SPIRE Command Line Interface (CLI) tool to create registration entries for a K8s node and a K8s workload. For instance, the selectors used on the node include the K8s cluster name, the K8s namespace, and the K8s service account. For the workload, the selectors used include the parent ID, the K8s workload's namespace, the K8s service account, the pod name, and the pod image.

Node

```
spire-server entry create
    -node
    -spiffeID spiffe://ufcg.edu.br/k8s-agent
    -selector k8s_sat:cluster:k8s-cluster
    -selector k8s_sat:agent_ns:spire
    -selector k8s_sat:agent_sa:spire-agent
```

Workload

```
spire-server entry create
    -parentID spiffe://ufcg.edu.br/k8s-agent
    -spiffeID spiffe://ufcg.edu.br/my-app
    -selector k8s:ns:default
    -selector k8s:sa:default
    -selector k8s:pod-name:my-app
    -selector k8s:pod-image:<container-ImgID>
```

Figure 2: Creation of registration entries.

Figure 3 describes the basic workflow for SPIRE. As shown in the sequence diagram, upon initialization, the agent requests its identity to the server, which initiates node attestation. Attestation involves some message exchange between the server and the agent so that the server can get the node selectors for selecting the appropriate SVID. This communication is mediated by a plugin on the server side and another on the agent side. First, the server uses the selectors to issue an SVID with a SPIFFE ID in a format defined by the plugin. Then, the server checks if any node registration entry associates any received selector with some additional SPIFFE ID; if so, the server issues a new SVID to the agent; then, it delivers that ID along with the SVIDs for workload registration entries associated with that agent. This process involves mutual authentication between server and agent, using the

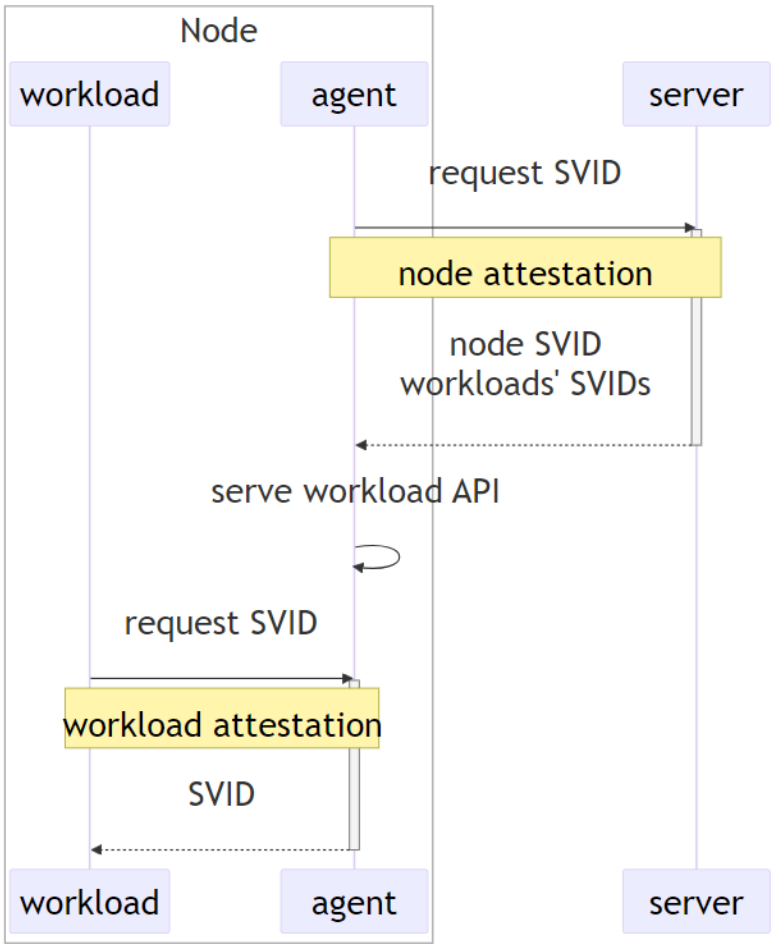

Figure 3: Basic SPIRE workflow.

server public certificate that the agent is pre-configured and the brand new agent SVID. Strategies for node attestation may employ tokens and certificates as selectors (e.g., K8s service tokens, pre-provisioned join tokens, SSH identities, or X.509 certificates). They may also rely on information provided by public clouds, e.g., instance identity documents in Amazon Web Service (AWS), Azure, or Google Cloud Platform (GCP), or use some trusted computing technology such as Trusted Platform Modules [9] or AMD SEV-SNP (as first proposed in this work). Finally, when entirely bootstrapped, the agent listens on the Workload API socket.

When an agent receives an SVID request from a workload on its Workload

API Unix Domain Socket, it will start the workload attestation process. For this, it will use the operating system's capabilities to determine exactly which process opened that connection. After identifying the process, the agent communicates the process ID to the configured attestation plugins, which will collect additional information to build the selector list. Then, if the selectors of the workload match some entry, and the current agent also has the parent ID associated with that entry, an SVID is delivered. Examples of workload selectors are the SHA256 of the application binary (available directly in Linux) and the container image ID (available for both Docker and K8s environments)[3]. When adopting the confidential computing threat model, either the operating system cannot be trusted, and other mechanisms must be used for attestation (as with the plugin for Intel SGX [19]), or the operating system must have been attested (as we do with the proposed plugin).

2.3. AMD SEV-SNP

AMD introduced Secure Encrypted Virtualization (SEV) in 2016. It is enabled by using the Virtual Machine Control Block (VMCB) to specify which memory pages should be encrypted [3]. Such technology is important when we consider a more strict threat model where we want to protect VMs from the cloud admin or bugs from the hypervisor. Confidentiality of workloads running in such CVMs is improved because the memory of VMs is protected using AES cryptography and keys provided, at the hardware level, only to the VMs. This is controlled with an encrypted bit (C-bit) in the CVM page table – 0 is used for unencrypted memory pages and 1 for encrypted pages. Therefore, when the hypervisor attempts to read the memory of a CVM, it retrieves only encrypted data.

In 2017, the AMD extended SEV with the Encrypted State, leading to the AMD SEV-ES [3]. When a SEV VM stops running, either through interruption or some other event, the raw contents of their registers are saved unencrypted in the hypervisor memory. A malicious or compromised hypervisor could read this information or even change it to perform a replay attack. The SEV-ES mitigates such attacks by encrypting the data of registers when a CVM stops, thus diminishing the attack surface.

[3]A full list of node and workload attestation plugins is available at https://github.com/spiffe/spire/tree/main/doc.

In 2020, the SEV-ES was extended with the Secure Nested Paging (SNP) feature to provide memory integrity [3]. In SEV-ES, although a memory region is encrypted, the hypervisor could execute integrity attacks by changing memory values to perform data replay or memory re-mapping, possibly leading to data leakage. Since the software could not assess the memory integrity, it could not avoid these attacks. The SEV-SNP ensures memory integrity by employing a structure called Reverse Map Table to track the owner (Hypervisor or VM) of each memory page and precluding these values from being directly manipulated by software [3].

In addition to memory encryption and integrity control, the AMD Secure Processor (AMD-SP) measures all the memory pages injected during the creation of the CVMs. This measurement enables the production of attestation reports, which include a hash of the VM construction process and a set of relevant configurations of the physical processor (e.g., firmware version, hyperthreading, debug mode). Later, processes inside the CVMs can request attestation reports to the AMD-SP. Third parties can then use this report to remotely attest that a VM is running in AMD SEV-SNP confidential mode because the reports are signed by a key internal to the AMD-SP and not accessible by software, allowing the third party to be sure a legitimate AMD-SP generated the report.

Reports can be obtained directly from the SNP device exposed to the guest or through a vTPM. Cloud providers adopt one of these strategies at their will. To assure the vTPM integrity, some providers leverage the AMD SEV-SNP Virtual Machine Privilege Levels (VMPLs) feature. VMPLs are used to restrict access to memory pages depending on the level a process is running. In this context, VMPLs are typically used to execute a Linux Secure VM Service Module (SVSM) with a vTPM in the most privileged level, the VMPL0, inaccessible from the host, but also from the guest, which executes at VMPL1. The SVSM-vTPM is measured during boot and reflected on the SNP launch measurement. Therefore, SEV-SNP with SVSM-vTPM deployments can leverage the security features of vTPMs to enhance CVMs' integrity verification, including the measurement of the OS kernel and, when applicable, *initrd*.

2.4. Trusted Platform Module

The Trusted Platform Module (TPM) [9] is a low-cost chip physically attached to the motherboard of computers. It serves as a root of trust for a platform and is commonly used to store secrets, verify platform integrity,

and identify platforms. In this sense, it is possible to use TPMs via SPIRE to facilitate the implementation of ZTAs that use TPMs as a source of identification and also to increase the level of security for platform integrity verification, especially for AMD SEV-SNP nodes, by measuring software loaded after the virtual machine has been created.

During its production, each TPM receives an Endorsement Key (EK), which is a signed asymmetric key provided by a Certification Authority (CA). The private part of the EK never leaves the TPM but can be used to identify a TPM and, consequently, the machine. This can be done because the public part of the key can be provided to a third party, which could verify its integrity and authenticity using the CA. After certifying that the public key corresponds to a private key within a TPM, inaccessible to any software component or user, the third party could propose a challenge that involves encrypting a secret unknown to the TPM. If the TPM successfully solves the challenge and presents the secret in plain text, then the challenger is convinced of the authenticity of the TPM and can use the public part of the EK to identify it.

A TPM has 24 registers called Platform Configuration Registers (PCRs). The PCRs can store hashes of measurements of vital components for the machine's execution since its boot, i.e., firmware, software, and configuration files. However, each measurement is not directly assigned to a PCR. To include a measurement, the TPM performs an extended hash operation on the last value of the PCR. Therefore, due to the high difficulty of tampering with a hardware-based TPM, it can be used as a root of trust to verify the environment's integrity. For this purpose, after the machine boots, the TPM can generate reports with the PCR values, signing them with its private key so that a remote third party can verify the report's integrity.

In the boot process, the firmware and all configurations used in the boot are extended into PCRs 0-7. In addition, PCR 9 plays a vital role in this work because it stores the measurement of the *initrd*, which is an initial root filesystem bound to the kernel, mounted prior as part of the two-stage boot process. It allows to customize the boot process easily, and set up the boot, permitting, for example, the filesystem's integrity verification, disk encryption, or network configuration. We use the *initrd* to perform remote attestation with the aid of SPIRE and the AMD SEV-SNP node attestor. According to the result of this attestation, the machine will receive the disk decryption keys to proceed with the boot. Therefore, using the TPM to measure *initrd* ensures the keys are not leaked. The details of verifying the

integrity of CVMs are presented in Section 6.

3. Threat Model

The first premise of the threat model for a SPIRE infrastructure is that the SPIRE server operator is trustworthy. This trust in the server is essential because a malicious server could use its CA certificate and keys to sign arbitrary SVIDs. For the same reason, we assume that the server component is free of bugs and runs on a trusted machine. This machine could be a physical machine under the organization's administration but could also run in a TEE in some public cloud.

On the other hand, the agent is not trustworthy a priori but may become trusted after the node attestation. Depending on the node attestation protocol applied, a malicious operator could take control of the agent and try different approaches to obtain SVIDs of workloads running in other nodes. Such an operator could adapt the agent's code, the code of node and workload attestors, and change agents' settings on the configuration file. Unlike the server, which is isolated and hardened, the nodes with agents run user-provided workloads, and these can attack the local agents.

As with AMD SEV-SNP's threat model, we assume a powerful attacker with privileges to access and manipulate all software layers of the infrastructure, including the operating systems, hypervisors, and cloud platforms where workloads are running. The AMD SEV-SNP threat model also does not consider rollback attacks on storage, side-channel attacks, or denial of service (DoS) attacks. Rollback attacks on storage do not affect our plugin. Nevertheless, they may affect applications running in the VMs, and they should be mitigated as in regular workloads (e.g., using internal or external monotonic counters). Mitigating side-channel attacks is orthogonal to our approach and should be done through patches in the operating system or *firmware* [27]. However, our approach does support conditioning SPIFFE IDs to specific *firmware* versions. Therefore, we assume that our user, the owner of the application, can determine what is an untainted operating environment (e.g., operating system, firmware, and hardware Trusted Computing Base (TCB)) by using reference measurements that have been validated by a security expert or organization. Thus, our goal is to enable the application owner to be protected by changes that the cloud provider or its operators can apply. Ensuring that end-users of applications understand the guarantees provided by CVMs is out of the scope of this work.

DoS attacks can happen, for example, if a malicious hypervisor refuses to run the guest. Consequently, this kind of attack is easily detectable and out of the scope of our work.

Under SEV-SNP, the AMD System-On-Chip hardware, the AMD-SP, and the VM itself are treated as fully trusted, while all other CPU software components, PCI devices, and operators are treated as untrusted. This includes the BIOS on the host system, the hypervisor, device drivers, and other VMs. For instance, we consider a model where the hypervisor is benign but may be vulnerable. It may not be actively trying to compromise CVMs, but it may be exploited to do so, likewise other untrusted components.

Physical TPMs' threat model is similar to the SEV-SNP. Both are trusted computing technologies and can be remotely attested to verify the state of untrusted components. Nevertheless, TPMs run virtually (vTPMs) in cloud providers, and could be tampered with. The strategy to detect such integrity violations is by anchoring it to the SEV-SNP root of trust. Hence, its threat model will be equivalent to the SEV-SNP if the vTPM is measured by the AMD-SP – which varies in different environments and cloud providers.

4. AMD SEV-SNP & Cloud Providers

environment	**measurement**		**vTPM**		**report**	**disk encryption control**
	components	reprod.	impl.	measured		
AWS	VMF	✓	Nitro	✗	SNP device	custom initrd, vTPM
Azure	VMF, SVSM	✗	SVSM-vTPM	✓	vTPM	vTPM
GCP	VMF	✗	IBM SWTPM fork	✗	SNP device	custom initrd, vTPM
on-premise	VMF, initrd, kernel	✓	custom	✗	SNP device	custom initrd, vTPM
	VMF, SVSM	✓	SVSM-vTPM	✓	vTPM	custom initrd, vTPM

Table 1: AMD SEV-SNP support in different deployments.

The SEV-SNP features are supplied in different forms for on-premise and the major public cloud providers. When attesting SEV-SNP CVMs, essential security properties must be verified. Next, we present these properties, detail why they are relevant, and then summarize the current support in cloud providers.

4.1. Security Properties and Mechanisms

For the AMD SEV-SNP attestation, we consider six relevant security properties/mechanisms: (*i*) the components included in the launch measurement reported by the AMD-SP; (*ii*) the reproducibility of such a launch

measurement, which may hide code for which the user has no access; (*iii*) the availability and customization of vTPMs for CVMs, which can be used for additional measurement; (*iv*) the inclusion of the vTPM in the launch measurement; (*v*) the mechanism to supply SEV-SNP attestation reports; and (*vi*) the mechanism to provide disk decryption keys.

It is important to know the components measured into the launch measurement because they compose the TCB. Understanding the TCB is vital because it is possible to rely on some of these trusted components to perform extra security and validation steps. For instance, CVMS must operate over encrypted disks, which naturally need to be decrypted during initialization. Thus, the decryption key must be handed in by a third-party verifier only after approval of the launch measurement. On the CVM side, we must place a hook to request the decryption key, which will cause the verifier to initiate the remote attestation procedure. Notice that to accomplish this task securely, this hook must be placed in a component within the TCB; otherwise, it would not be possible to check if the requester is a CVM and if it has the expected trusted TCB.

Reproducing such a measurement is essential to securely close the remote attestation loop. While the steps to reproduce the measurement are clear, some environments do not provide the source code for some components, precluding the offline computation of the launch measurement, i.e., discovering the measurement before launching a CVM.

An important security feature for CVMs is the availability of vTPMs. The launch measurement included in the SEV-SNP attestation report is only computed once during CVM initialization. Therefore, it does not track the execution platform state after boot, including executing processes in the OS. Thus, if some attacker somehow obtains access to the CVM after initialization, it is impossible to detect by leveraging the standard SEV-SNP remote attestation. vTPMs can measure a binary used to start a process in the OS and relevant configuration files. Then, while SEV-SNP provides a means to verify the TCB state with reports generated at boot time, vTPMs can complement it with dynamic runtime reports. The issue here is establishing trust in a vTPM, given that it runs as software and thus could also be adulterated. For this reason, we also investigated which environments the vTPM is measured and could be securely verified.

The SEV-SNP attestation report may be obtained from a SEV-SNP guest device, exposed in the CVM, or from a vTPM if the SVSM is enabled on the host. From the SPIRE Node Attestor implementation perspective, it

can obtain the SEV-SNP report either from the SEV-SNP device or the vTPM. Similarly, it verifies its integrity and chain of trust in both approaches. We must recall that reports obtained from the SEV-SNP device are freshly generated each time they are requested. However, for the SVSM-vTPM approach, the report is generated during boot and inserted in the vTPM's memory. Thus, in this approach, whenever a CVM requests the SEV-SNP report from the vTPM, it will always obtain the same report. Although the launch measurement remains the same in both approaches, some other values could change and are not reflected in reports obtained from SVSM-vTPM.

4.2. Current Support

These relevant aspects are summarized in Table 1 for the different AMD SEV-SNP environments considered. For the measurement, we focus only on the guest context, which includes all memory pages loaded in the CVM. For instance, the possible components are VM firmware (VMF), *initrd*, kernel, and SVSM. Then, we detail the reproducibility of this measurement, which implies giving the user access to the VMF and SVSM source code. For the vTPM, we specify which implementation is used and if it is enabled by SVSM and, consequently, measured into the launch measurement. Finally, we provide information about how the SEV-SNP report is provided and how disk encryption can be enforced.

From Table 1, we can observe that the AWS and GCP deployments are similar regarding the SEV-SNP measurement. Although in both providers the SEV-SNP TCB includes the firmware, currently, we can only reproduce the launch measurement for AWS CVMs since the code of the VM firmware is open and available at AWS github [2]. One drawback common to AWS and GCP is that there is no instruction on verifying the integrity of the vTPM. There is no information about whether the vTPM is measured, where this measurement reflects, and the exact source code used for the vTPM. Thus, although it is possible to obtain vTPM attestation reports, it is impossible to attest the vTPM integrity.

On the other hand, the Azure and on-premise deployments employ SVSM-enabled vTPMs. In this strategy, the vTPM is included in the SVSM binary, and the SVSM is measured into the CVM launch measurement. Thus, anyone possessing the SVSM source code can compute the expected launch measurement of the CVM beforehand. However, to this date, the Azure SVSM used to instantiate CVMs is not publicly available. Therefore, the launch measurement for the SVSM approach is only completely verifiable for on-premise

installations, allowing the proper usage of runtime vTPM reports in addition to the SEV-SNPs' reports. In this work, for the SVSM-enabled on-premise CVMs, we use the open-source SVSM-vTPM implementation [32] available on github [14].

CVMs with disk encryption must obtain the decryption keys before operating. This key could be stored in a secure Key Management Service (KMS), in the SPIRE Server itself, or be sealed to the vTPM [32]. As mentioned above, one challenge in performing this task with vTPMs is that no public cloud provides a means to verify the integrity of the vTPM. Azure is the one that gets closer by using SVSM-vTPMs, which could be verifiable if the SVSM-vTPM source code was available. In contrast, tools such as SPIRE can be used to perform remote attestation and issue SVIDs that can be used to retrieve the disk key from a KMS. This can be done by customizing the *initrd* to include the SPIRE agent. However, no public cloud provider includes the measurement of the *initrd* into the SEV-SNP report. The *initrd* can be measured into vTPM's PCR 9, but again, we fall back to the same problem of being unable to verify the vTPM integrity.

In summary, the current status is that (*i*) it is possible to securely use AWS CVMs if user-controlled disk encryption is not mandatory[4], and (*ii*) on-premise installations can be fully verified by remote parties – the SVSM approach should be employed when vTPMs are needed. The current main drawbacks of each cloud provider can be summarized as follows:

- **AWS**: the NitroTPM source code and SVSM-enabled deployment are not available;
- **Azure**: the CVM firmware (including the SVSM) source code is not available;
- **GCP**: the CVM firmware, SVSM-enabled deployment and vTPM source codes are unavailable.

5. Plugin design and implementation

To allow the implementation of different attestation strategies, SPIRE operates on the concept of employing plugins to extend its capabilities. If a

[4]Note that all cloud providers offer basic disk encryption. In this case, the data is not unencrypted in the physical disks, but the decryption keys are managed by the cloud provider.

new attestation form is desired, a new plugin can be implemented to leverage some root of trust to attest the node. When attested, the plugin will generate selectors to identify the node, and this information is used to issue specific SVIDs. In this section, we will discuss our implementation of the AMD SEV-SNP node attestor and a hybrid attestation plugin that allows the combination of different attestation plugins into one.

5.1. AMD SEV-SNP Node Attestor

To build a Chain of Trust (CoT) between the AMD-SP and the CVM that remote parties can validate, the AMD-SP provides a protected path through which the CVM can request attestation reports on their behalf. The attestation report aims at allowing the CVM to prove to third parties that it runs confidentially on an AMD server with SEV-SNP enabled and has the expected configurations.

The attestation report is signed by the private part of an endorsement key (EK) located inside the AMD-SP and is not accessible by software. This EK can be a Versioned Chip Endorsement Key (VCEK) or a Versioned Loaded Endorsement Key (VLEK). The VCEK is a signing key derived from chip-unique secrets and a TCB version. While the VLEK is derived from a seed maintained by the AMD Key Derivation Service (KDS) and can be loaded into the processor. Both can be used to sign the attestation reports and guarantee the authenticity of it. The public part of the EK can be used to verify the signature, and its authenticity can be verified against the AMD certificate chain. Therefore, it is possible to guarantee that the report was generated inside an AMD-SP, as it is the only entity with access to the EK private key. In addition, the requester can include 512 bits of arbitrary data as a nonce[5] in the attestation report. With the verification of signatures and nonce, a remote party can be sure the report is fresh and authentic, generated by a genuine AMD-SP.

We implemented a node attestation plugin that executes the process of verifying the confidentiality of a VM in the context of SPIRE. There are two strategies to perform the remote attestation depending on the environment: (*i*) non-SVSM CVMs and (*ii*) SVSM CVMs. Figure 4 presents the node attestation workflow for non-SVSM CVMs, while Figure 5 describes the workflow for SVSM CVMs.

[5]The nonce is some information (e.g., a random number) provided by the attestation challenger to guarantee the freshness of the report.

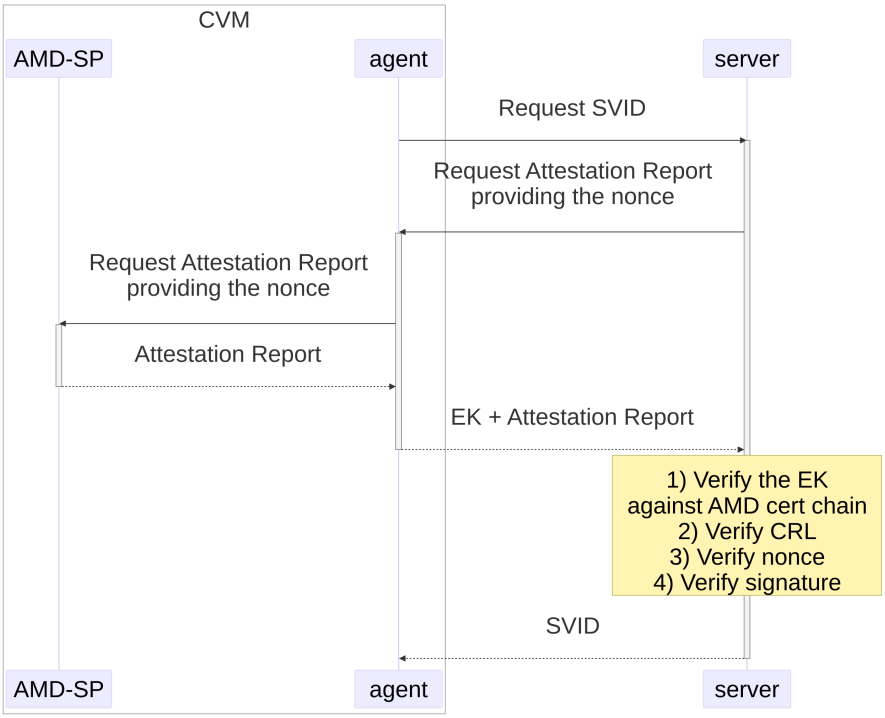

Figure 4: Attestation of Non-SVSM CVMs with SPIRE.

In the first strategy, which does not use vTPM, the agent uses the AMD-SP to generate an attestation report with a nonce provided by the server, and the server verifies the EK certificate chain, if the EK is included on the Certificate Revocation List (CRL) provided by AMD, the nonce, and the signature, as described in Figure 4.

Figure 5 highlights the attestation workflow for an SVSM CVM. For this workflow, the agent uses the vTPM to retrieve the SNP attestation report, which is saved on the vTPM's memory. To ensure that the report is fresh, we used the vTPM quote with the nonce provided by the server. The server verifies the authenticity of the quote by using the hash of the vTPM's public key, which is included in the 512 bits of arbitrary data in the SNP report. The agent sends the AMD EK, the attestation report, the vTPM quote, and the vTPM's public key to the server. The server then verifies the AMD EK against the AMD certificate chain and CRL, the SNP report signature, the hash of the vTPM's public key, the vTPM quote signature, and the nonce.

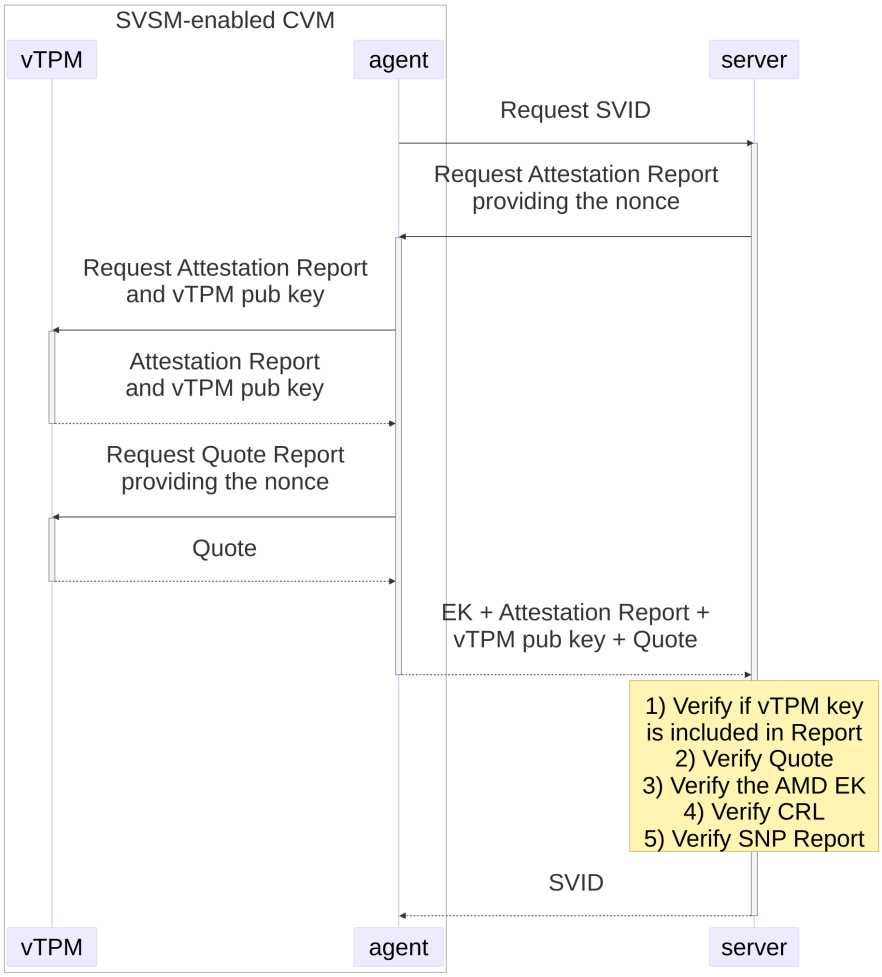

Figure 5: Attestation of SVSM-enabled CVMs with SPIRE.

Table 2 presents the main selectors of our plugin[6]. The chip ID is a value associated with the AMD-SP and thus never changes. The EK can identify a cloud provider (in the case of VLEK, as the same VLEK can be loaded on multiple chips) or identify a specific chip (in the case of VCEK). The launch measurement is computed over the initial memory content of the CVM. Finally, the policy is defined by the CVM owner to impose restrictions

[6]To see all the selectors, check https://github.com/ufcg-lsd/spire-amd-sev-snp-node-attestor

on the host executing the CVM.

Selector	Description
amd_sev_snp:chip_id	SHA512 of the host chip ID
amd_sev_snp:signing_key_hash	SHA512 of the public EK
amd_sev_snp:measurement	CVM's launch measurement generated by the AMD-SP
amd_sev_snp:policy:single_socket	CVM can be activated on one or multiple sockets
amd_sev_snp:policy:debug	Allow or disallow debugging
amd_sev_snp:policy:migrate_ma	Allow or disallow the association with a migration agent
amd_sev_snp:policy:smt	Allow or disallow simultaneous multi-threading
amd_sev_snp:policy:abi_major	Minimum ABI major version required for the CVM to run
amd_sev_snp:policy:abi_minor	Minimum ABI minor version required for the CVM to run

Table 2: SEV-SNP node attestor selectors.

The SPIFFE ID produced by the SEV-SNP attestor, i.e., the SPIFFE ID appearing in the first SVID issued by the server, has the following format: spiffe://<trust-domain>/spire/agent/amd_sev_snp/chip_id/<chip_id>/measurement/<measurement>/report_id/<report_id>.

5.2. Hybrid Node Attestor

Because of the heterogeneous nature of modern infrastructures, we may want to use different characteristics and capabilities to identify a node. However, SPIRE does not allow combining different node attestors to identify a unique node. For instance, to combine different public cloud attestors with AMD SEV-SNP, we should implement new plugins that use both attestation strategies for both cloud providers. This gives rise to the need to implement an exponential number of plugins to combine different attestation capabilities, leading to code duplication.

To solve this problem, we designed a plugin that reuses the node attestation process of other SPIRE plugins and inherits the selectors generated by them with the prefix *hybrid*. This enables the combination of different attestation strategies to form the identity of a unique agent without needing to rewrite any attestation procedure. The hybrid plugin configuration is straightforward because it inherits the settings of the components plugins.

Table 3 presents an example of selectors when combining AMD SEV-SNP and AWS IID (Instance Identity Document) node attestors through the hybrid plugin.

Selector
hybrid:amd_sev_snp:chip_id
hybrid:amd_sev_snp:signing_key_hash
hybrid:amd_sev_snp:measurement
hybrid:aws_iid:image:id
hybrid:aws_iid:az

Table 3: hybrid node attestor selectors.

The SPIFFE ID produced by the hybrid attestor has the following format: spiffe://<trust-domain>/spire/agent/hybrid/<uuid>.

In the following section, we describe how the hybrid plugin can be used to enhance the trust in CVMs.

6. Node Integrity Verification

The key field in the SNP attestation report is the **launch measurement**. The launch measurement is the result of an HMAC over (but not restricted to) the following values: AMD-SP *firmware* version, CVM's policy, and CVM's launch digest (for other values, see [4]). The launch digest is a hash generated by the AMD-SP, including all plaintext data imported by the hypervisor into the CVM's memory at launch. This launch digest is important to measure critical components such as the CVM's *firmware*, *kernel*, *initrd*, and SVSM-vTPM binary. For SVSM-CVMs, the kernel and *initrd* are not measured in the launch measurement, but they are measured by the vTPM. However, since the SVSM-vTPM is measured into the launch measurement, consequently the *kernel* and *initrd* can be securely verified through vTPM PCRs.

A standard CVM does not guarantee the protection of data at rest and the integrity of the runtime environment, which includes tools used to execute the workloads, such as the SPIRE agent, the container runtime, and the orchestrator (K8s). One alternative to both issues is encrypting sensitive data and tools and only decrypting them within the CVM and verifying the integrity of the CVM's filesystem using the kernel capabilities, given that the kernel can be measured.

The *initrd* can be instrumented to perform remote attestation with the aid of a SPIRE agent and our proposed plugin. Upon successful attestation, a simple workload named "fetch key" (FetchKey WL) can use its freshly obtained SVID to retrieve the decryption key from a Key Management Service (KMS). For simplicity, within the context of this work, we have implemented a basic KMS. However, an alternative solution such as Hashicorp Vault [22] or VMware Secrets Manager [39] could also be used to handle the decryption keys. The KMS contains the decryption keys, so it must run on a trusted environment (internal or external to the cloud provider).

We used the Linux Unified Key Setup (LUKS) to encrypt and decrypt partitions. To assure the filesystem's integrity, we used *dm-verity*, a Linux kernel module that provides transparent integrity checking using a hash digest of the device blocks. Figure 6 presents a sequence diagram detailing the steps to boot a CVM with disk encryption and filesystem integrity checking.

Before describing the workflow to boot a CVM, it is important to explain the KMS functioning. In the KMS, every SPIFFE ID is associated with a decryption key that it is authorized to retrieve. However, defining the SPIFFE ID of the workload (FetchKey WL) used to obtain the decryption key requires executing some tasks and comprehending the concept and operation of SPIRE registration entries. First, the *initrd* should be repackaged to include the SPIRE agent and the FetchKey WL. Notice that this impacts the measurement of the *initrd* that can be used to provide the SPIFFE ID to the agent. Also, the FetchKey WL will only receive the specific SVID if it runs in a CVM that is in a trusted state (with the expected kernel, *initrd* and *firmware*). This is done by leveraging the parent node ID feature of SPIRE. Any change to critical components, such as the *firmware*, the kernel, the *initrd* itself, and the agent (which is in the *initrd*), would reflect in the agent's SPIFFE ID. This would preclude the FetchKey WL from receiving the expected SVID since there would be no node registration entry with a SPIFFE ID corresponding to the adulterated environment. Since the FetchKey WL registration entry depends on the agent SPIFFE ID, this task

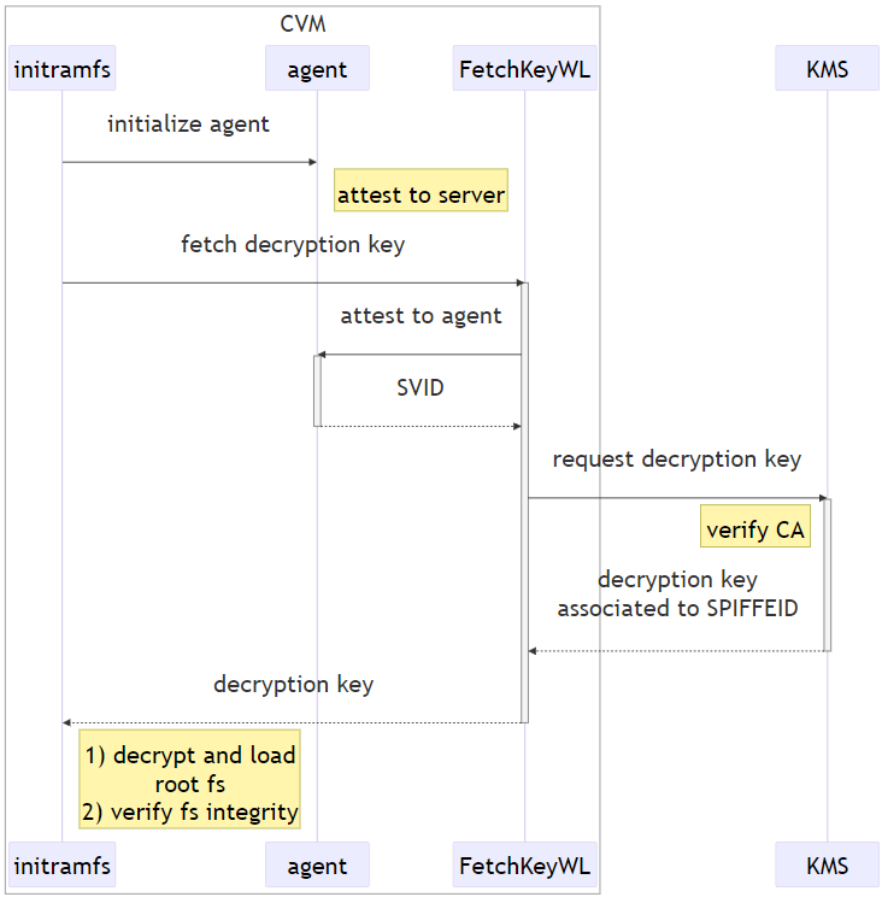

Figure 6: Boot of a CVM with an encrypted disk.

can only be performed after the CVM owner customizes the *initrd*. After creating the registration entries for the node and workload, the CVM owner can register in the KMS the disk decryption key associated with the workload SPIFFE ID.

The boot CVM workflow starts with the *initrd* initializing the agent. The agent attests to the SPIRE server as described in Figures 4 and 5. Recall that on-premise SVSM-enabled CVMs' launch measurement is not affected by the *initrd*. Because it is essential to measure the *initrd*, on-premise SVSM-enabled CVMs should use the hybrid plugin to combine the AMD SEV-SNP and TPM node attestors so that the latter can verify the *initrd* integrity

through TPM PCRs[7]. Then, the *initrd* initializes the FetchKey WL, which attests to the agent and receives its SVID. Considering the critical components are genuine, and the FetchKey WL obtained the expected SVID, the FetchKey WL will request the decryption key to the KMS. The KMS checks the SVID against the trusted CA and relies on the SPIFFE ID to decide whether that CVM/workload can receive some decryption key. Finally, the *initrd* uses this key to load the encrypted partition containing the root file system. After this, the *initrd* calls *dm-verity* module to assure disk integrity. Any attempt to change the encrypted disk would corrupt it and preclude the CVM from initializing.

Initrd customization is crucial to assure node integrity and complete confidentiality. For typical SPIRE deployments, where an organization has a single SPIRE server, this task must be performed a single time. It is possible to deploy multiple CVMs with the same *initrd* and customize the root file system with the necessary modules. For each root file system, the admin only needs to associate the SPIFFE ID of the FetchKey WL with a new decryption key and with the root file system metadata at the KMS.

7. Evaluation

There are security and performance perspectives to be evaluated about the proposed plugin. Regarding security, we detail some attacks on the integrity of the CVM to obtain its data or to retrieve SVIDs of confidential workloads. Also, we detail the aspects of our work on different environments such as AWS, GCP, Azure, and on-premise. In terms of performance, we compare our confidential node attestor plugin with a standard non-confidential plugin to attest K8s nodes. In addition, we also compare the attestation time of our plugin in different environments on public and private clouds (on-premise).

7.1. Analysis of the security aspects

Vulnerabilities in specific QEMU and OVMF versions

Some versions of QEMU and Open Virtual Machine Firmware (OVMF) do not measure critical components of the CVM such as *initrd*, kernel, and

[7]Azure SVSM-enabled CVMs are paravirtualized; thus, the *initrd* is not executed and can not be measured by vTPM.

cmdline. This creates vulnerabilities as the launch measurement on the attestation report would not reflect critical changes to the machine software stack. To address this, we used two different strategies for on-premise: a patch applied to QEMU [30] and OVMF [29], ensuring the measurement of these components in the launch measurement, and a strategy also utilizing patches on QEMU [8], OVMF [7], and SVSM [6] to measure the vTPM and utilize it to measure the boot components on its PCRs. As in this second strategy the vTPM binary is measured in the launch measurement, we can rely on the measurements that the vTPM takes from the boot components.

In the context of public cloud providers, we cannot apply the patches to launch the CVM, although, on Azure, the CVMs are launched following the idea of using SVSM-vTPM and measuring it using the launch measurement. Despite Azure having SVSM-vTPM, *initrd* and kernel are not measured by the vTPM because the CVM is paravirtualized. As discussed in Section 4, most public cloud providers do not measure the key components of the boot, so to enhance security in these scenarios, we can use the hybrid plugin to combine different strategies to identify a node.

IO bounce buffer vulnerability

Another issue is bringing data from the disk to the DRAM of the CVM [28]. When a CVM tries to read data from the disk, the host (QEMU) reads this data and writes it in an unprotected region of the DRAM belonging to the CVM. It needs to be unprotected to perform I/O operations, and it is called a software I/O translation buffer (SWIOTLB) or simply a bounce buffer. The CVM would then copy (*memcpy*) this data to its private memory, which is encrypted. However, every raw data in the buffer could be read or adulterated during this process. Therefore, using encrypted volumes or disk encryption approaches where the server operator manages keys would leave data unprotected. We avoid this attack by encrypting the disk with keys provided externally (from the FetchKey WL) and conditioning access to the decryption key on the integrity of the CVM (through the attestation during the execution of *initrd*).

Impersonation attack

In this attack, a non-confidential VM could pretend to be a CVM to obtain the SVIDs for confidential workloads. These SVIDs could be used to access unauthorized resources, e.g., the decryption keys of the root file system. This could be achieved by creating a non-confidential VM between the genuine SPIRE server and the CVM. This VM could have an adulterated

agent requesting an SVID for a CVM (even though it is not a CVM) and an adulterated server that poses as the original server to a genuine CVM. Thus, this intermediary VM would perform a man-in-the-middle (MiTM) attack, and all requests from the genuine server would be forwarded to the CVM. Figure 7 illustrates this attack. This attack requires corrupting the agent and server, which could be achieved with a few simple modifications in the SEV-SNP node attestor. This attack is mitigated by including the configuration of the agent of the CVM in the *initrd*. This change forces the CVM only to trust a SPIRE server with the correct TLS certificates. After that, placing an intermediary VM would require a change in the agent configuration, changing the server address and the associated certificate (the trust bundle). Since these changes impact the launch measurement in the attestation report, a correct server will not provide the SVID for the corrupted agent.

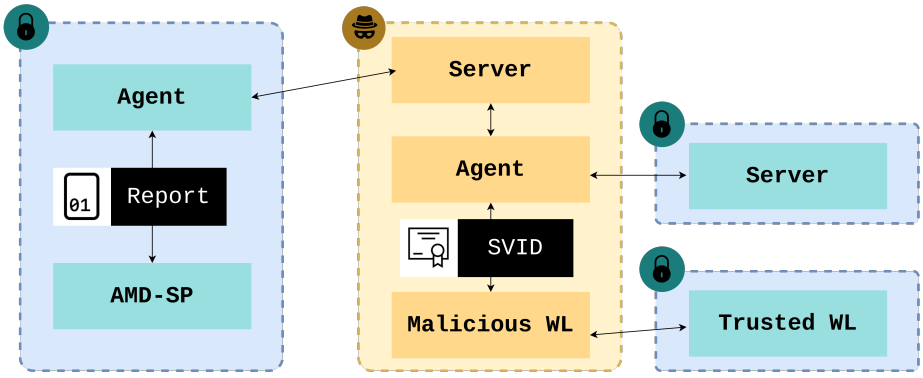

Figure 7: Impersonation attack.

7.2. Performance analysis

The performance of the AMD SEV-SNP node attestor plugin was evaluated by comparing it with the K8s Service Account Token (K8S-SAT) node attestor. We deployed a K8S cluster in two AMD VMs, each with 32 *GB* of RAM and 8 cores. The K8s cluster can execute confidentially by simply enabling the SEV-SNP feature during the initialization of VMs. We configured the SPIRE server always to execute isolated at the master node, and the agent was set up to be deployed always at the worker node. When running in the confidential cluster, the agent is set up with the SEV-SNP node attestor, and for the non-confidential cluster, it is set up with the K8S-SAT. Each

attestation strategy was executed 30 times. To measure the duration of the attestation, we recompiled the SPIRE agent by placing one timestamp before the node attestation procedure and another timestamp after its conclusion.

The node attestation in the non-confidential cluster with the K8S-SAT plugin took, on average, 72.9 ms, with a standard deviation of 2.7 ms. The confidential cluster with the SEV-SNP plugin took an average of 33.5 ms, with a standard deviation of 3.9 ms when the AMD EK is configured in the file system, or an average of 2 s, with a standard deviation of 1.2 ms when retrieving the EK from the AMD-SP. With these values, when we configure the EK on the file system, we notice that the attestation time for AMD SEV-SNP does not significantly differ from other node attestation plugins. For this reason, the following experiments are executed with the EK being retrieved from the file system.

We also experimented with the plugin in different environments to compare their performance. For this, we launched CVMs with SEV-SNP on AWS, Azure, GCP, and on-premise with and without SVSM. For almost all the cases, we launched a CVM with 2 VCPUs and 8 GB of RAM, except for on-premise SVSM CVM. Because of limitations on the vTPM implementation, the On-Premise SVSM CVM was launched with 1 VCPU and 8GB of RAM. Despite the difference of 1 VCPU, it must not represent a significant performance difference since the report retrieval procedure is not CPU-bound. Table 4 shows the mean attestation times for each environment and their standard deviations.

Environment	**Mean Time (s)**	**std dev (s)**
AWS	0.0735	0.0446
GCP	0.0721	0.0315
on-premise (non-SVSM)	0.1533	0.0015
Azure (SVSM)	0.9859	0.2746
on-premise (SVSM)	0.3056	0.1299

Table 4: SEV-SNP node attestation times.

The main finding of this experiment is that the attestation time is marginal for the different environments; for public clouds non-SVSM environments (AWS and GCP), we have very similar attestation times (around 0.07s). For on-premise (non-SVSM), we can see a slight difference due to the latency of

obtaining the CRL from AMD's web server. For on-premise (SVSM), the time increase in comparison to non-SVSM environments happens mainly because of the TPM's Attestation Identity Key (AIK) generation, which takes a mean time of $0.13s$ and represents 42.83% of the attestation time. For Azure, despite the fact we do not need to generate the AIK because it is stored on vTPM's memory, we see a more significant difference in the attestation time due to a longer time to obtain the SNP report from vTPM (mean of $0.89s$). This is related to Azure's vTPM implementation.

We can also compare how long it takes for a VM, CVM, and SVSM-enabled CVM to boot and become ready to serve workloads. We want to understand the costs of essential procedures other than attesting the node, such as running the OVMF (to understand the overhead of the SVSM-vTPM), obtaining the disk decryption key (executed by the FetchKeyWL), and opening the encrypted file system and verifying disk integrity. The non-confidential VM was configured with the X.509 node attestation, and the CVMs were configured with the implemented AMD SEV-SNP node attestor. VM and CVMs (with *initrd* customization) were launched 30 times. Figure 8 depicts the OVMF, kernel initialization, FetchKeyWL, and LUKS and *dm-verity* times. Confidence intervals have a confidence level of 95%.

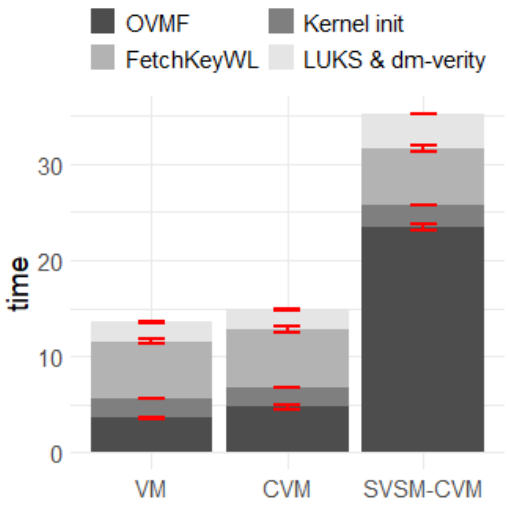

Figure 8: Boot and attestation times for the native and confidential VMs.

The results presented in Figure 8 lead us to the following major findings:

- enabling SEV-SNP adds a marginal mean cost of 1.32 s for the CVM to

become ready to serve workloads, an overhead introduced by AMD-SP operations, such as measuring the contents loaded during boot;

- enabling vTPM through SVSM causes a considerable impact on the time an SVSM-CVM takes to become ready to serve workloads, an average of 20.54 s in comparison with CVMs, an overhead added by the vTPM the measurement operations it performs over firmware and software during boot.

An extra latency of 5.96 s added on all the boot times, is the time the FetchKey WL takes to retrieve its SVID, execute, and fetch the decryption key. Finally, it takes a mean time of 2.03 s to open the LUKS partition and verify its integrity on VM and CVM; for SVSM-CVM, this time is increased to 3.63 s because it has only 1 vCPU due to limitations on SVSM-vTPM implementation. Also, notice that the root file system size does not impact on the time to open the LUKS partition. We measured the time to open LUKS partitions with 10 MB, 1 GB, and 10 GB, and all took the same average time. This happens because the decryption of the data itself is performed during I/O operations on disk blocks. Opening the LUKS partition only creates the virtual block device that points to the encrypted physical block device. All the encryption and decryption are made by *dm-crypt*, a transparent block device encryption subsystem in the Linux kernel.

Finally, we assess the impacts of enabling VM confidentiality with AMD SEV-SNP. We used the *sysbench* unix module to measure the performance of CPU, memory, file I/O, and database I/O, in a VM, CVM, and CVM with LUKS encrypted partition, with 32 GB of RAM and 8 cores on an EPYC 7343 server with 128 GB of RAM and 32 cores. For every resource benchmarking, *sysbench* was configured to execute with 16 threads. The file I/O measurement used the combined random read/write, and the database IO also uses a read/write online transactional processing over 16 tables, each with a size of 10000. We ran the *sysbench* for each resource 30 times. The performance comparison between each resource/VM type is presented with confidence intervals over the 30 executions, with a confidence level of 95%. Figures 9, 10, and 11, present the results for CPU and memory, file I/O, and database I/O benchmarks.

By observing the CPU results, we notice that the performance of CVM and CVM-LUKS are slightly degraded compared to non-confidential VMs. The VMs processed an average of 35517 events per second, while the CVMs

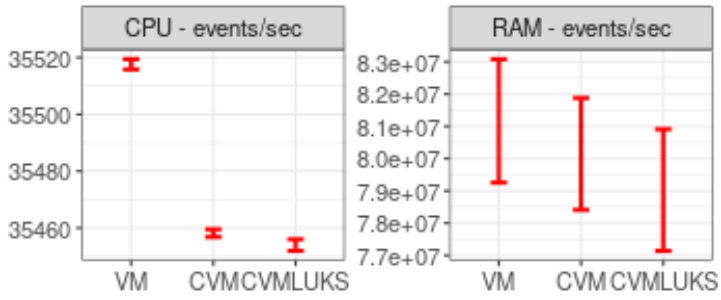

Figure 9: Performance of CPU and RAM over VM, CVM, and CVM with disk encryption.

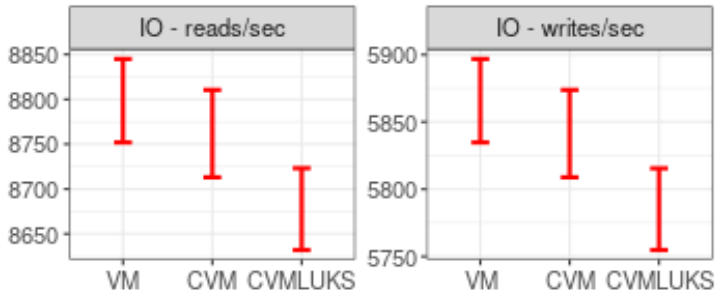

Figure 10: Performance of file I/O over VM, CVM, and CVM with disk encryption.

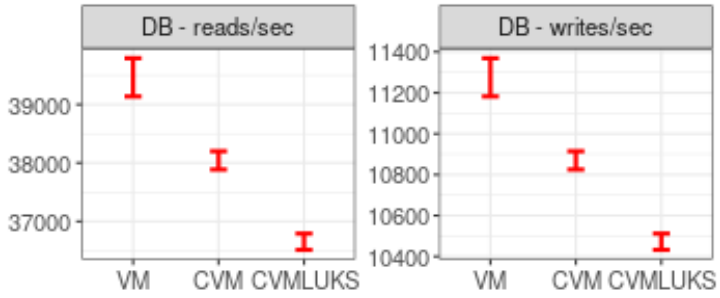

Figure 11: Performance of database I/O over VM, CVM, and CVM with disk encryption.

and CVMs-LUKS processed an average of 35458 and 35453 events per second, respectively. Although the confidence intervals do not overlap, these values represent only a marginal CPU degradation, lower than 1% for both comparisons. It degraded 0.2% between VMs and CVMs (with and without LUKS).

For the memory benchmark, we observed a mean degradation of 1.3% from VMs to CVMs and 2.6% from VMs to CVMs-LUKS. This degradation occurs because of the memory encryption and integrity guarantee of SEV-SNP, which does not significantly impact performance because it is implemented at the hardware level.

The file I/O experiments show a mean degradation of 0.4% between VMs and CVMs for both read and write operations, which does not exhibit a significant difference. However, this small loss between VMs and CVMs may be explained by the path the data traverses from the CVM memory to the disk and vice-versa. Data needs to be copied from the disk to an unencrypted shared memory, then copied to the private encrypted memory, and the reverse path follows the same concept. This overhead imposed on I/O operations can be alleviated with the new SEV technology recently published (March 2023) by AMD, called SEV Trusted IO (SEV-TIO) [5]. SEV-TIO enables trust in devices, thereby eliminating the need to write the data to the bounce buffer before writing to private memory or disk. A mean degradation of 1.4% occurs between VMs and CVMs-LUKS for both read and write operations. Although the performance between VMs and CMVs-LUKS suffers a slightly larger degradation due to disk data encryption and integrity guarantees, it also does not exhibit a significant performance penalty.

Database I/O operations present a low but not imperceptible degradation. For database operations, we observed a mean degradation of 3.6% between VMs and CVMs, and a mean degradation of 7.13% between VMs and CVMs-LUKS for both read and write operations.

8. Related work

This section lists some popular alternatives for generating identities in cloud environments. Next, we discuss some work that also aims to protect the integrity of VMs.

8.1. Identity provisioning

As discussed in Section 2, cloud-native computing requires dynamic provisioning of identities specific to each service or software component. In addition to SPIFFE/SPIRE, there are alternative approaches to this task. In Kubernetes, services can use the Certificate API to generate certificates, Service Account Tokens, or a controller such as Cert-Manager [15]. This would enable the creation of certificates or JWTs recognizable by Kubernetes or

even globally (e.g., using Cert-Manager to issue Let's Encrypt certificates). Nevertheless, such identities are assigned based on the manifest describing the service and can be stolen or changed by the cluster operator, which is incompatible with our threat model.

A more sophisticated approach is Google BeyondProd, an extension of the BeyondCorp model [41]. BeyondProd enables richer forms of attestation, such as combining location and the provenance of the container images. As with SPIFFE/SPIRE, its usage is often associated with mutual authentication in TLS connections. Other products that aim to help the deployment of zero-trust architectures will offer similar solutions to issuing identities and controlling access.

8.2. Protecting the integrity of VMs

Several practical approaches to machine integrity rely on TPMs. Nevertheless, sharing the TPM chip among several machines is not feasible, so virtual TPMs must be used. However, virtual TPMs are susceptible to several attacks [26, 16, 34]. Running virtual TPMs in TEEs is a possibility. TRIGLAV, discussed below, Azure Confidential VMs [10], and AWS EC2 instances with NitroTPM [1] implement this approach. However, TRIGLAV requires Intel SGX, which is unavailable on AMD machines.

TRIGLAV [33] also aims to protect the integrity of VMs against internal attacks. In TRIGLAV, the authors propose a mechanism that uses TPMs to measure the integrity of a host and then uses Intel SGX to host multiple virtual TPMs. The authors then present a mechanism to create a stronger relationship between the VMs and the virtual TPM inside the SGX enclave. Our work contrasts by leveraging AMD SEV-SNP, full disk encryption, and integrity checking to protect the integrity of the VMs at boot. As an advantage, our approach does not depend on a modified kernel. A peculiarity of our approach is that vTPMs are not available in all environments, and when provided, the vTPMs of just a few of these environments can have their integrity securely verified.

In a previous work [35], we discuss the usage of SPIRE for attesting on-premise CVMs using the patched version of QEMU and OVMF. After that, [21] proposed Revelio, a framework to generate certificates for web-facing applications hosted on AMD SEV-SNP VMs. As with our previous work, Revelio uses the patched QEMU and OVMF to measure *initrd*, and then uses *initrd* to verify the filesystem integrity. In contrast to our approach, Revelio introduces the usage of Let's Encrypt to sign certificates (instead

of SPIRE). In this work, we extended our previous plugin to handle public cloud providers and their different approaches to support AMD SEV-SNP. In addition, we also provide a hybrid plugin that enables several node attestors to be combined, which mitigates the limited measurement capability of some cloud providers (e.g., which could not distinguish between two disk images in the same VM flavor) and enables more control for the operator (e.g., enabling additional selectors for conditioning SPIFFE IDs also to properties of the cloud provider, such as the hosting tenants).

Attestation of AMD SEV-SNP in public cloud providers is closer to the approach proposed by [32] for on-premise installations. In this case, the AMD report measures only the CVM firmware, but this firmware includes a vTPM that will be used to measure the rest of the boot process. This approach is based on the Linux Secure VM Service Module, which implements a guest communication interface that can implement services for the guests and, at the same time, be isolated from the guest VM using AMD SEV-SNP Virtual Machine Privilege Levels. We leverage this approach to extend our previous plugin to support SVSM-based vTPMs as an alternative to extended QEMU and OVMF measurement.

While this work provides a node attestation plugin, [19] presents a plugin for attesting workloads that run atop Intel SGX. While attesting workloads running in a TEE brings benefits with its reduced trusted computing base, porting existing services to Intel SGX may require some effort and is likely to reduce performance.

9. Conclusion

This paper presents an interoperable approach to attest AMD SEV-SNP confidential virtual machines in public and private cloud environments. The work was implemented on top of the Cloud-Native Computing Foundation's project for identity provisioning, SPIRE, and, consequently, inherits interoperation capabilities that enable it to introduce confidential computing in modern microservice-based environments seamlessly.

To support the different approaches cloud providers adopt to AMD SEV-SNP machines, we proposed a hybrid plugin that combines the properties of multiple node attestation plugins. For example, suppose a cloud provider does not enable the attestation report to measure the integrity of the guest VM's operating system. In that case, the application owner can now use

the hybrid plugin to specify an additional selector that requires a vTPM to measure the guest's operating system.

Finally, our approach has negligible effects on the initial attestation of VMs and can be extended easily to other cloud providers since (*i*) we consider both approaches used by academics and the major cloud providers (namely, the SVSM-vTPM and the extended OS measurement by the hypervisor and firmware); (*ii*) we detail how the different models and provider approaches impact on the threat model; (*iii*) we share our code with the SPIRE community.

The following steps of this research include supporting additional confidential computing technologies, such as Intel TDX, and refining selectors to expose slightly different threat models, facilitating the plugin configuration regardless of the platform it is operating on. The long-term goal is to create a generic framework that abstracts the application of confidential computing technologies so that it becomes known and easy to use as other security concepts and technologies.

This manuscript represents an extended version of our previously published paper, 'Attesting AMD SEV-SNP Virtual Machines with SPIRE,' which appeared in the Proceedings of the 12th Latin-American Symposium on Dependable and Secure Computing (LADC '23). In this extended work, we detail the current status of AMD SEV-SNP support across major public cloud providers and elucidate the associated security guarantees. Additionally, we introduce a novel mechanism designed to leverage the capabilities of various platform technologies synergistically, thereby enhancing the security guarantees of AMD SEV-SNP attestation processes within cloud environments.

We thank the members of the Secure and Scalable Identity Provisioning (SSIP) project who participated in discussions that led to this work: Adriane Cardoso and Gustavo Brand from Hewlett Packard Enterprise, and Carlos Filho, Esther Brasileiro, Raniel Dourado, Sabrina Silva, and Vinicius Muniz from the Laboratório de Sistemas Distribuídos (UFCG/LSD). We also thank members of the SPIFFE community for their feedback on the plugin, especially Evan Gilman.

This research was funded by a collaboration between Hewlett Packard Enterprise Brazil and the EMBRAPII unit UFCG-CEEI (Universidade Federal de Campina Grande) with the incentive of the Informatics Law (Law 8.248 from October 23rd, 1991).

AB and EF are the main contributors to the conception of this work.

DP, FS and AM implemented the plugin and performed the experiments. AB and EF are the main writers of this manuscript. All authors read and approved the final manuscript.

The authors declare they have no competing interests.

The current version of the plugin is available at https://github.com/ufcg-lsd/spire-amd-sev-snp-node-attestor.

References

[1] Amazon Web Services (2024a). AWS Nitro System. https://aws.amazon.com/ec2/nitro. Accessed: 2024-03-06.

[2] Amazon Web Services (2024b). AWS UEFI source code for AMD SEV-SNP Confidential VMs. https://github.com/aws/uefi. Accessed: 2024-03-06.

[3] AMD (2020). AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. Technical report.

[4] AMD (2022). SEV Secure Nested Paging Firmware ABI Specification. Technical report.

[5] AMD (2023). AMD SEV-TIO: Trusted I/O for Secure Encrypted Virtualization. Technical report.

[6] AMDESE (2024a). Linux SVSM forked repository to support vTPM. https://github.com/svsm-vtpm/linux-svsm. Accessed: 2024-03-06.

[7] AMDESE (2024b). OVMF forked repository to support SVSM-vTPM. https://github.com/svsm-vtpm/ovmf.git. Accessed: 2024-03-06.

[8] AMDESE (2024c). QEMU forked repository to support SVSM-vTPM. https://github.com/svsm-vtpm/qemu.git. Accessed: 2024-03-06.

[9] Arthur, W. and Challener, D. (2015). *A Practical Guide to TPM 2.0: Using the Trusted Platform Module in the New Age of Security*. A practical guide to TPM 2.0 / Arthur, Will. Apress.

[10] Azure (2024). Azure confidential VMs. https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-vm-overview. Accessed: 2024-03-06.

[11] Biden Jr., J. R. (2021). Improving the nation's cybersecurity. National Archives and Records Administration, College Park, MD, USA, Executive order 14028. https://www.whitehouse.gov/briefing-room/presidential-actions/2021/05/12/executive-order-on-improving-the-nations-cybersecurity/. Accessed: 2023-05-13.

[12] Boeyen, S., Santesson, S., Polk, T., Housley, R., Farrell, S., and Cooper, D. (2008). Internet X.509 public key infrastructure certificate and certificate revocation list (CRL) profile. IETF RFC 5280. https://datatracker.ietf.org/doc/html/rfc5280. Accessed: 2023-11-26.

[13] Campbell, M. (2020). Beyond zero trust: Trust is a vulnerability. *Computer*, 53(10):110–113.

[14] Carvalho, C., Almasi, G., Berrangé, D., Narayanan, V., and Buono, D. (2024). Linux SVSM-Based vTPM implementation Proof-of-Concept. https://github.com/svsm-vtpm/linux-svsm. Accessed: 2024-03-06.

[15] Cert-Manager Community (2024). Cert-Manager: Cloud native certificate management. https://cert-manager.io/. Accessed: 2024-03-06.

[16] Cucurull, J. and Guasch, S. (2014). Virtual TPM for a secure cloud: fallacy or reality?

[17] Cybersecurity and Infrastructure Security Agency (2021). Security guidance for 5G cloud infrastructures - part II: Securely isolate network resources. https://www.cisa.gov/sites/default/files/publications/Security_Guidance_For_5G_Cloud_Infrastructures_Part_II_Updated_508_Compliant.pdf. Accessed: 2023-05-13.

[18] Cybersecurity and Infrastructure Security Agency (2023). Zero trust maturity model. https://www.cisa.gov/zero-trust-maturity-model. Accessed: 2023-05-13.

[19] Falcão, E., Silva, M., Luz, A., and Brito, A. (2022). Supporting confidential workloads in SPIRE. In *2022 IEEE International Conference on Cloud Computing Technology and Science (CloudCom)*, pages 186–193.

[20] Feldman, D., Fox, E., Gilman, E., Haken, I., Kautz, F., Khan, U., Lambrecht, M., Lum, B., Fayó, A. M., Nesterov, E., Vega, A., and Wardrop,

M. (2020). *Solving the Bottom Turtle: a SPIFFE way to establish trust in your infrastructure via universal identity.*

[21] Galanou, A., Bindlish, K., Preibsch, L., Pignolet, Y.-A., Fetzer, C., and Kapitza, R. (2023). Trustworthy confidential virtual machines for the masses. In *Proceedings of the 24th International Middleware Conference*, Middleware '23, page 316–328, New York, NY, USA. Association for Computing Machinery.

[22] HashiCorp (2024). HashiCorp Vault. https://www.vaultproject.io/. Accessed: 2024-03-06.

[23] ITU-T Study Group 17 (2022). Determined new recommendation ITU-T X.1644 (X.SGDC): Security guidelines for distributed cloud. https://www.itu.int/md/T22-SG17-R-0021/en. Accessed: 2023-09-06.

[24] Jones, M. B., Bradley, J., and Sakimura, N. (2008). JSON Web Token (JWT). IETF RFC 7519. https://datatracker.ietf.org/doc/html/rfc7519. Accessed: 2023-11-26.

[25] Kubernetes Community (2024). Kubernetes. https://kubernetes.io/. Accessed: 2024-03-06.

[26] Lauer, H., Sakzad, A., Rudolph, C., and Nepal, S. (2019). Bootstrapping trust in a "trusted" virtualized platform. In *Proceedings of the 1st ACM Workshop on Workshop on Cyber-Security Arms Race*, CYSARM'19, page 11–22, New York, NY, USA. Association for Computing Machinery.

[27] Li, M., Wilke, L., Wichelmann, J., Eisenbarth, T., Teodorescu, R., and Zhang, Y. (2022). A systematic look at ciphertext side channels on AMD SEV-SNP. In *2022 IEEE Symposium on Security and Privacy (SP)*, pages 337–351.

[28] Li, M., Zhang, Y., Lin, Z., and Solihin, Y. (2019). Exploiting unprotected I/O operations in AMD's secure encrypted virtualization. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 1257–1272, Santa Clara, CA. USENIX Association.

[29] Murik, Dov (2023a). OVMF Patch for AMD SEV-SNP VMs. https://listman.redhat.com/archives/edk2-devel-archive/2023-March/059923.html. Accessed: 2024-03-06.

[30] Murik, Dov (2023b). QEMU Patches for AMD SEV-SNP VMs. https://lore.kernel.org/qemu-devel/20230302092347.1988853-1-dovmurik@linux.ibm.com/. Accessed: 2024-03-06.

[31] Ménétrey, J., Göttel, C., Pasin, M., Felber, P., and Schiavoni, V. (2022). An exploratory study of attestation mechanisms for trusted execution environments.

[32] Narayanan, V., Carvalho, C., Ruocco, A., Almasi, G., Bottomley, J., Ye, M., Feldman-Fitzthum, T., Buono, D., Franke, H., and Burtsev, A. (2023). Remote attestation of confidential VMs using ephemeral VTPMs. In *Proceedings of the 39th Annual Computer Security Applications Conference*, ACSAC '23, page 732–743, New York, NY, USA. Association for Computing Machinery.

[33] Ozga, W., Le Quoc, D., and Fetzer, C. (2021). TRIGLAV: Remote attestation of the virtual machine's runtime integrity in public clouds. In *2021 IEEE 14th International Conference on Cloud Computing (CLOUD)*, pages 1–12.

[34] Parno, B. (2008). Bootstrapping trust in a "trusted" platform. In *Proceedings of the 3rd Conference on Hot Topics in Security*, HOTSEC'08, USA. USENIX Association.

[35] Pontes, D., Silva, F., Falcão, E., and Brito, A. (2023). Attesting AMD SEV-SNP virtual machines with SPIRE. In *Proceedings of the 12th Latin-American Symposium on Dependable and Secure Computing*, LADC '23, page 1–10, New York, NY, USA. Association for Computing Machinery.

[36] Rose, S., Borchert, O., Mitchell, S., and Connelly, S. (2020). Zero trust architecture. Technical Report NIST Special Publication, National Institute of Standards and Technology, Gaithersburg, MD.

[37] SPIFFE (2023). SPIRE concepts. https://spiffe.io/docs/latest/spire-about/spire-concepts/.

[38] SPIFFE Community (2023). SPIFFE standards. SPIFFE GitHub. https://github.com/spiffe/spiffe. Accessed: 2023-11-26.

[39] VMware Tanzu (2024). VMware Secrets Manager for Cloud-Native Apps. https://github.com/vmware-tanzu/secrets-manager. Accessed: 2024-03-06.

[40] Wang, W., Song, L., Mei, B., Liu, S., Zhao, S., Yan, S., Wang, X., Meng, D., and Hou, R. (2024). NestedSGX: Bootstrapping trust to enclaves within confidential VMs.

[41] Ward, R. and Beyer, B. (2014). BeyondCorp: A new approach to enterprise security. https://storage.googleapis.com/pub-tools-public-publication-data/pdf/43231.pdf.