SNPGuard: Remote Attestation of SEV-SNP VMs Using Open Source Tools

Luca Wilke
University of Lübeck, Germany
l.wilke@uni-luebeck.de

Gianluca Scopelliti
Ericsson Security Research, Sweden
KU Leuven, Belgium
gianluca.scopelliti@ericsson.com

***Abstract*—Cloud computing is a ubiquitous solution to handle today's complex computing demands. However, it comes with data privacy concerns, as the cloud service provider has complete access to code and data running on their infrastructure. VM-based Trusted Execution Environments (TEEs) are a promising solution to solve this issue. They provide strong isolation guarantees to lock out the cloud service provider, as well as an attestation mechanism to enable the end user to verify their trustworthiness. Attesting the whole boot chain of a VM is a challenging task that requires modifications to several software components. While there are open source solutions for the individual components, the tooling and documentation for properly integrating them remains scarce. In this paper, we try to fill this gap by elaborating on two common boot workflows and providing open source tooling to perform them with low manual effort. The first workflow assumes that the VM image does only require integrity but not confidentiality, allowing for an uninterrupted boot process. The second workflow covers booting a VM with an encrypted root filesystem, requiring secure provisioning of the decryption key during early boot. While our tooling targets AMD Secure Encrypted Virtualization (SEV) VMs, the concepts also apply to other VM-based TEEs such as Intel Trusted Domain Extensions (TDX).**

1. Introduction

Cloud computing revolutionized the way businesses and individuals utilize computing resources. Instead of owning and maintaining physical servers or data centers, users rent them from large cloud service providers on a pay-as-you-go basis, allowing unprecedented flexibility and scalability. One major concern with shifting computation to the cloud is data privacy. Trusted Execution Environments (TEEs) aim to solve this challenge, allowing to execute software on remote machines without the need to trust their operator. To this end, they provide strong (cryptographic) runtime isolation guarantees as well as an attestation feature to prove their trustworthiness to the end user. Early designs like Intel Security Guard Extensions (SGX) were process-scoped and required software to be rewritten, hindering adoption. More recent TEE designs like AMD Secure Encrypted Virtualization (SEV) and Intel Trust Domain Extensions (TDX) use a different approach by protecting whole Virtual Machines (VMs) to achieve a *lift and shift* solution. These VMs are also known as Confidential Virtual Machines (CVMs). Similar to booting a physical machine, booting a CVM includes several software components and layers. As result, the correct attestation of CVMs is quite complex, requiring changes to several components of the software stack such as guest firmware and kernel. Nonetheless, without proper (remote) attestation the VM owner cannot verify that their CVM has not been tampered with. Cloud providers like Microsoft Azure [18] have solved this challenge, but their solution relies on closed-source, proprietary software. As a result, the VM owner cannot verify these components, requiring them to again trust the cloud service provider, partially defeating the purpose of TEEs.

In this paper, we introduce an easy-to-use, open source tool that tackles this challenge for AMD SEV-Secure Nested Paging (SNP), the latest iteration of SEV. While there already exist several individual components that are open source, to the best of our knowledge there is no unified tool that provides a full workflow from the initial CVM preparation to its deployment, attestation, and secret provisioning. As a result, a high manual effort and some expertise are still required by the VM owner. Our tool solves this issue by integrating these existing components and providing templates for common use cases.

In principle, our ideas should generalize to similar VM-based TEE technologies such as Intel TDX, and in the future we aim at supporting them as well. Our main goal is to provide a solid foundation for future research that wants to either build on top of CVMs or experiment with tweaking the individual components. We also hope that our tool will make CVMs more accessible to developers. While our design is technically similar to Revelio [7], their code does not seem to be publicly available. Additionally, in contrast to libkrun [17], which tries to minimize VM size and boot time, we use the feature-rich reference implementations for the individual components. Finally, Pontes et al. [20] developed an open source integration for SEV-SNP attestation with the SPIFFE framework [23]. SPIFFE aims to standardize secret injection into services. In contrast to our tool, Pontes et al. do not provide tooling for configuring and setting up the SEV-SNP VM. In addition, their dependency on SPIFFE requires a more complex setup, while our tool is focussed on providing building blocks that can be easily set up and customized.

The remainder of this paper is organized as follows: Sect. 2 defines the attacker scenario and gives background on AMD SEV-SNP. In Sect. 3 and Sect. 4 we describe our tool and give implementation details. In Sect. 5 we discuss remaining problems and future work, and in Sect. 6 we summarize our paper.

2. AMD SEV-SNP in a Nutshell

SEV-SNP [2], [3] is a TEE that protects the confidentiality and integrity of whole VMs against an attacker with root privileges and physical access to the machine, enabling to run SEV-protected VMs without trusting the infrastructure provider and virtualization layers such as the hypervisor.

The root-of-trust of an AMD CPU is the AMD Secure Processor (SP) (formerly known as PSP), a coprocessor that hosts the SEV-SNP firmware and performs security-critical operations like generating and managing memory encryption keys. Each protected VM is assigned a unique AES-128 key that is used to encrypt its data before it is written to the RAM. Inside the CPU, an access-right based system is used to protect data in the guest VM. In addition, the Reverse Map Table (RMP) prevents the hypervisor or other tenants from writing to VM memory, ensuring integrity. Moreover, SEV-SNP introduced Virtual Machine Privilege Levels (VMPLs), an additional set of hardware enforced privilege levels than can be used by the VM. This enables use cases like running a virtual TPM (vTPM) inside a secure VM, on a higher privilege layer than the rest of the software stack.

The AMD SP contains a chip-unique signing key called Chip Endorsement Key (CEK), derived from secrets stored in chip's fuses. This key is used to derive other keys such as the Versioned Chip Endorsement Key (VCEK), used to sign attestation reports. The VCEK also depends on the version number of the SP firmware and the CPU microcode. This allows a verifier to attest that the platform where a secure VM is running has an up-to-date software stack. A public key infrastructure is used to certify the VCEK from AMD's root certificates, ensuring the authenticity of attestation reports. Recently, the SP firmware started to support an alternative to the VCEK called Versioned Loaded Endorsement Key (VLEK), which identifies a specific cloud provider. When requesting an attestation report, the VCEK and VLEK can be used interchangeably.

3. Design

In this section we describe the high level design of our tool. Implementation details will be given in Sect. 4.

We designed our tool with two main goals in mind:

1) *Integrity of the VM boot chain.* It is paramount that we ensure that the deployed VM is in a good, known state. This is typically done via remote attestation, although confidential VMs have additional challenges because their boot process follows multiple stages. Normally, the launch measurement included in the SEV-SNP attestation report only covers the guest firmware, hence there is a need to extend this measurement to later stages such as kernel, initial RAM disk (*initramfs*), and eventually the root filesystem.
2) *Secret provisioning.* A typical confidential workflow involves injecting some kind of secrets into the deployed TEE instance, e.g., private keys, sensitive data such as personal identifiable information, or intellectual property such as AI models. Our tool supports two modes of secret provisioning, i.e., at runtime and at rest. In the first case, the VM initially does not contain any secrets, which are only injected after boot via a secure channel established upon remote attestation. In the second case, the VM's root filesystem requires confidentiality and is thus encrypted before transmission to the hypervisor. As a result, we need to enable provisioning the disk encryption key during early boot.

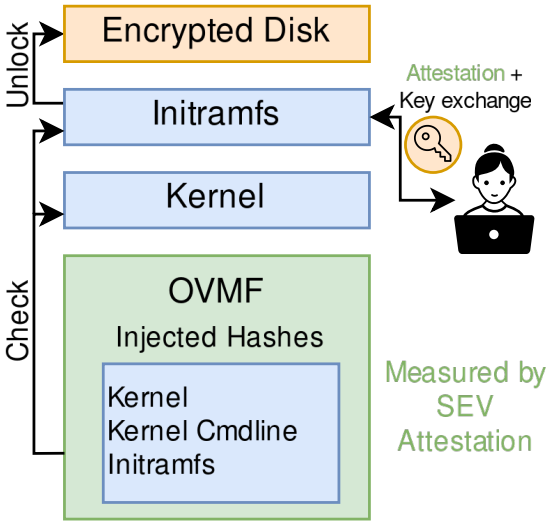

Figure 1: Trust chain of the software stack running inside the VM that shows how to use the SEV measurement as a trust anchor for software-based measurements. The control flow is bottom to top.

To achieve the above goals, our tool uses a two-stage approach. The first stage covers the first part of the VM boot process and includes guest firmware, kernel, initramfs and kernel command line parameters. Here, we assume that these components do not contain any secrets and come from known (e.g., open source) code. Thus, we only need to ensure their integrity (Goal 1). The second stage consists of the VM's root filesystem, which may or may not contain secrets (Goal 2). Here, integrity is always mandatory, but confidentiality is optional. To ensure the integrity of the first stage, we use the SEV-SNP isolation and attestation features. To achieve integrity and confidentiality for the second stage, we use the disk protection techniques supported by the Linux kernel.

3.1. First Stage

The very first piece of code that is executed after starting the SEV-SNP VM is the firmware, which has to be SEV-SNP-aware in order to correctly set up the VM, e.g., by allocating memory pages and configuring page bits. To achieve this, we use the SEV-SNP-enabled version of the OVMF [1] UEFI implementation.

Normally, the launch measurement in the SEV-SNP attestation report only includes the memory content of the OVMF firmware and the initial register state of the VM's virtual CPUs. However, there is an SNP-enabled OVMF firmware version that supports extending the measurement to include the kernel, the initramfs and the kernel command-line parameters, which are then loaded using Direct Linux Boot [21]. Fig. 1 shows an overview of the mechanism. In short, before loading OVMF into memory,

the hypervisor measures those components and adds the corresponding hashes in a special section of the OVMF binary. Afterwards, the "hot-patched" version of OVMF is loaded into memory, making the injected hash values for the kernel, initramfs and kernel command line part of the SEV-SNP launch measurement. At runtime, before transferring the execution to the Linux kernel, OVMF re-computes the measurements of each component and compares them against the stored reference value, failing to boot if they do not match. As a result, we can use the remote attestation features of SEV-SNP to prove the integrity and authenticity of the first stage to the VM owner.

With SEV-SNP, the code inside the VM is responsible for requesting the attestation report. The report is signed by the AMD SP whose public key can be verified using a publicly available certificate chain. Thus, it is safe to transmit the attestation report over an unprotected channel. To prevent an attacker from replaying an old attestation report, we make use of the "Guest Data" field, which allows the VM to include 64 bytes of arbitrary data into the signed attestation report, to include a nonce sent by the VM owner. By checking that the report contains the expected nonce, the VM owner can verify that they received a fresh report.

3.2. Second Stage

With the first stage, we can ensure the integrity of the guest VM up to the so-called *early userspace*, i.e., where the initial RAM disk is mounted into memory and some initial configuration of the VM is done before passing control to the "real" root filesystem. However, as SEV-SNP does not provide protection of code and data at rest, we need other mechanisms to ensure the integrity and (optionally) confidentiality of persistent disks, which can otherwise be easily accessed by the hypervisor.

Depending on whether the root filesystem requires confidentiality, our tool offers two different approaches to secret provisioning (Goal 2). If the root filesystem does not contain any secret data, the VM can boot uninterrupted and we can later verify that the VM is in a good state via attestation. Secrets may still be injected after the VM has fully booted. If the root filesystem should remain secret, we need to provision it in encrypted form. As a result, the VM cannot boot past the first stage without having provisioned the decryption key. Therefore, we necessarily need to perform both attestation and secret provisioning *during* early boot. Below, we describe the two approaches in more detail.

Integrity-only Workflow The main idea here is that we do not want to "pause" the boot process, but instead allow the VM to boot without any intervention, and only later, whenever needed, attest it and provision secrets. To achieve this goal, we leverage the `dm-verity` utility [14] to ensure the integrity of the root filesystem. In short, the filesystem is divided into blocks of a predefined size, and each of them is measured using a hash function (e.g., SHA-256). All hashes are then stored in a Merkle tree on a separate data disk, leaving the original root filesystem unmodified. The root of the Merkle tree, called *root hash*, guarantees the integrity of the whole filesystem and needs to be stored separately. When the root filesystem is

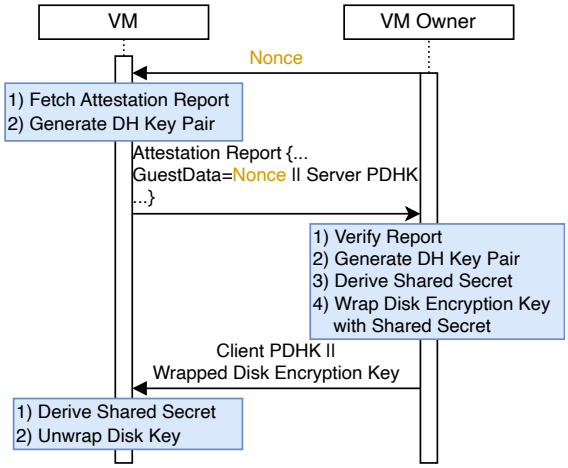

Figure 2: Protocol that uses the Guest Data field in the attestation report to securely send a disk encryption key to the VM. DH stands for Diffie-Hellman and PDHK refers to the public part of the DH key.

mounted, during early userspace, `dm-verity` takes as input the data disk and root hash and verifies the integrity the filesystem. Besides, integrity is checked at each read, resulting in a kernel panic if a violation is discovered.

With this approach, the integrity of the root filesystem can be reduced to the integrity of the root hash, which has to be provisioned to the guest VM without tampering, but does not require confidentiality. We solve this problem by adding the hash as a kernel parameter. As the first stage already ensures the integrity of the kernel command-line parameters, this approach naturally extends the VM's integrity guarantees to the root filesystem. This can be verified, at any moment, by the usual remote attestation procedure.

One caveat of `dm-verity` is that the root filesystem is mounted as read-only, which may not be acceptable for all applications. Besides, there are partitions in the filesystem that need to be read-write for the VM to function properly, e.g., `/home`, `/tmp` and `/var`. We solve this problem by creating one `tmpfs` filesystem [15] for each partition that needs to be read-write. These filesystems will only reside in memory, thus their integrity and confidentiality are preserved at runtime by SEV-SNP. This process is done in the initramfs where, after opening and verifying the `dm-verity` filesystem, files are copied to the `tmpfs` partitions.

Clearly, secrets such as cryptographic keys should be deleted from the root filesystem when choosing this workflow, as there is no confidentiality at rest. Our tool helps with this process by automatically deleting all SSH keys from `/etc/ssh` when preparing the `dm-verity` filesystem. Those keys are then regenerated in early userspace and stored in the `tmpfs` partitions, such that the guest owner can safely connect to the CVM at runtime. Afterwards, they can retrieve the attestation report to verify its boot chain.

Integrity+Confidentiality Workflow For this workflow, the VM owner encrypts the root filesystem with `dm-crypt` [12] before sending it to the untrusted hypervisor. `dm-crypt` can also be configured to ensure

integrity protection by leveraging `dm-integrity` [13] under the hood. To unlock the encrypted disk for the second stage, the first stage needs access to the encryption key. Thus, we need to build an encrypted channel between the VM owner and the VM to transfer the key. To achieve this, we again leverage the "Guest Data" field to perform a Diffie-Hellman (DH) key exchange, as depicted in Fig. 2. The VM owner starts the process by requesting the attestation report, using a nonce to ensure freshness. Next, the VM generates an ephemeral DH key and includes the public part in the "Guest Data" field, together with the nonce. After verifying the attestation report, the VM owner uses their ephemeral DH key pair to derive a shared secret and uses it to encrypt the disk encryption key using an authenticated encryption scheme. Finally, the VM owner sends both the encrypted disk encryption key and the public part of their DH key to the VM. While the VM authenticates to the guest owner by providing a valid attestation report, the VM owner implicitly proves their authenticity to the VM by providing the correct disk encryption key.

The code inside the VM uses the public DH key of the VM owner to derive the shared secret as well, allowing it to decrypt the disk encryption key. Now, we can unlock and transfer control to the root filesystem by executing its *init* script, bringing up the rich userspace environment of the VM. After the initialization of the second stage has completed, the VM owner can, e.g., use SSH to connect to the VM. Since the second stage image was encrypted before it was transferred to the untrusted hypervisor, the VM owner can securely store the private key of the SSH server as well as the list of public keys that are allowed to log in inside the image.

Comparison Both workflows have pros and cons. While the integrity-only workflow does not offer confidentiality at rest, the encrypted workflow necessarily requires users to pause the boot process to get the root filesystem's decryption key. Besides, `dm-verity` might have a better I/O performance compared to `dm-crypt`+`dm-integrity`, but at the same time mounting some partitions as `tmpfs` in the integrity-only workflow would incur some boot latency (for copying files) and increase memory usage. Users should therefore choose the workflow that is best suitable for their use case.

4. Implementation

This section gives implementation details for the design outlined in Sect. 3. For the hypervisor, we use the patched Linux kernel and QEMU versions provided by AMD [1] in order to run SEV-SNP-enabled VMs. For the VM, we use the SEV-SNP target of OVMF [24] as the UEFI, and a self-written client and server for the attestation workflow. In addition, we developed scripts to streamline the whole setup process. Our tool is available at https://github.com/SNPGuard/snp-guard.

Building the first stage image. Our first stage image consists of the OVMF UEFI code plus a Linux kernel, its command-line parameters, and an initramfs. The latter three can be provided either separately or as a Unified Kernel Image (UKI). Both OVMF and the Linux kernel need to be adapted to work with SEV-SNP. Thus, we use AMD's official forks [1] for both. However, most features are also available in the official upstream versions. For OVMF, we need to build the *AmdSev/AmdSevX64.dsc* target instead of the default *OvmfPkgX64.dsc* target used by the build script in the AMD repository. This enables OMVF's support for extending the SEV launch measurement to the kernel (cf. Sect. 3).

Building the initramfs. We require modifications to the initramfs to support integrity and encryption of the root filesystem. The initramfs is a minimal filesystem that contains scripts, kernel modules, applications and configuration files that are required to bootstrap the VM and mount the root filesystem.

To build our custom initramfs, we leverage Docker [6] to create a container image with all the required tools. This approach allows us to easily install components with complex dependencies. Afterwards, we export all layers of our container image into a single folder and convert it to the initramfs format. For the *init* binary inside the initramfs we use a small shell script that calls a series of applications to orchestrate the boot process and eventually mounts the filesystem for the second stage. Next, the *init* script uses *switch_root* [10] to unmount the current initramfs filesystem and jump into the root filesystem using the contained *init* script as the entry point for the execution, taking care of starting the services contained in the second stage. One caveat with this approach is that the second stage will still use the initially booted Linux kernel. We discuss implications and potential solutions in Sect. 5.

If full disk encryption is used, the first stage needs access to the disk encryption key before it can proceed with the boot process. To provision the key, we perform the remote attestation protocol described in Sect. 3.2 which we implemented based on the open source *sev* library developed by the VirTEE community [25].

Building the second stage image. For preparing the root filesystem that is mounted in the second stage we start from an existing Linux filesystem that can be created locally via usual means, such as creating a new VM from scratch and installing a fresh Linux distribution. Afterwards, we provide tools and scripts to extract the filesystem from the VM image, make the necessary adjustments, and generate the corresponding Merkle tree (if `dm-verity` is used) or to create an encrypted copy (if `dm-crypt` is used).

5. Limitations and Future Work

Flexible boot. One limitation of our *switch_root*-based approach for transitioning from the first stage to the second stage is that it does not change the kernel itself. Thus, we are still executing the publicly known kernel from the first stage. Furthermore, any changes that the second stage VM does to the kernel or initramfs stored on its */boot* partition do not have any effect. One potential solution for this issue is to use the Linux kernel's *kexec* [16] feature, which enables booting into a new Linux kernel from an already running kernel. However, we were not able to use this feature as SEV-SNP support for *kexec* seems to be work in progress [11].

A more principled solution would be to change the first stage image to only include a bootloader like Grub

or systemd-boot, instead of a full Linux kernel. This way, we could properly boot the second stage using its kernel and initramfs. In addition, this decouples the launch measurement from the second stage image as we no longer need to change the first stage if we want to use a different kernel version with our second stage. The main drawback of this approach is that it would result in a quite restricted programming environment for the first stage, which conflicts with our goal of enabling an easy integration of experimental changes. We leave the implementation of this approach for a future version of our tool.

Cloud deployments. Currently, our tool can be used to deploy CVMs on SEV-SNP-enabled hosts where the guest owner can control the boot process and the parameters passed to the CVM at launch. However, cloud providers such as Amazon Web Services (AWS) [4], Microsoft Azure [18], and Google Cloud Platform (GCP) [8] currently only provide limited customization options in their confidential computing offering where, e.g., it is not possible to run a custom guest firmware. Unfortunately, this means that our workflows are not fully supported on public clouds. While we hope that this situation will change in the future, users might still benefit from our tool at present by building individual components (e.g., encrypting the root filesystem).

Virtual TPM. In the future, we are planning to support a vTPM inside the guest VM. In SEV-SNP, this can be achieved by leveraging the SEV-SNP specific VMPL (c.f. Sect. 2) feature to run a module at the highest VMPL privilege level that exposes vTPM functionalities to lower-privileged levels such as the guest OS. Besides, the vTPM can also be used to measure the first stage of boot without requiring to inject kernel measurements into OVMF (cf. Sect. 3.1). Moreover, a vTPM can be combined with the Linux Integrity Measurements Architecture (IMA) [22] to provide runtime attestation of userspace applications, which would enable integrity protection of the second stage with minimal changes (cf. Sect. 3.2). The open source community is actively working on vTPM support as part of the Secure VM Service Module (SVSM) project [5], and a full design has been proposed by Narayanan et al. [19].

Intel TDX. The design in Sect. 3 does not only apply to SEV-SNP but can be generalized to all VM-based TEEs. In particular, it would be worthwhile to extend our tool to also support Intel TDX. Here, the only difference would concern the first stage to build TDX-aware OVMF and kernel binaries. Besides, we could leverage TDX's Runtime Measurement Registers (RTMRs) [9] to perform integrity measurements similarly to a vTPM (possibly combined with Linux IMA).

6. Conclusions

Recent advancements in the confidential computing landscape have highlighted that there is a need to ease the development of TEE-based solutions in order to increase adoption, making this technology accessible even to non-skilled developers. While VM-based TEEs are a significant step in the right direction as they do not require any modifications in user applications, a lot of manual effort is still required to configure and properly attest CVM workloads. With our work, we try to further lower the bar towards TEE adoption by minimizing the manual effort required to provision and attest CVMs, and we hope that our tool can accelerate future TEE research or even be integrated into the supply chain of cloud workloads in the future.

Acknowledgements

This research is partially funded by the Research Fund KU Leuven, by the Cybersecurity Research Program Flanders, by EU H2020 MSCA-ITN action 5GhOSTS under grant agreement no. 814035 and by the BMBF project SASVI.

References

[1] AMD, "AMDSEV Github Repository," https://github.com/AMDESE/AMDSEV/tree/snp-latest, Accessed on March 23, 2024.

[2] ——, "AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More," White Paper, January 2020.

[3] ——, "SEV Secure Nested Paging Firmware ABI Specification," Specification, September 2023, version 1.55.

[4] AWS, "AMD SEV-SNP," https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/sev-snp.html.

[5] COCONUT, "COCONUT-SVSM Github Repository," https://github.com/coconut-svsm/svsm, Accessed on March 23, 2024.

[6] Docker, "Docker," https://www.docker.com/, Accessed on March 26, 2024.

[7] A. Galanou, K. Bindlish, L. Preibsch, Y.-A. Pignolet, C. Fetzer, and R. Kapitza, "Trustworthy confidential virtual machines for the masses," in *Proceedings of the 24th International Middleware Conference*, ser. Middleware '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 316–328. [Online]. Available: https://doi.org/10.1145/3590140.3629124

[8] Google Cloud, "Confidential Computing," https://cloud.google.com/security/products/confidential-computing.

[9] Intel, "Intel Trust Domain Extensions (Intel TDX) Module Base Architecture Specification," Specification, November 2023, version 348549-003US.

[10] P. Jones, J. Katz, and Z. Karel, "switch_root man page," https://man7.org/linux/man-pages/man8/switch_root.8.html, Accessed on March 22, 2024.

[11] V. Karasulli, "x86/snp: Add kexec support," https://lwn.net/Articles/965773/, Accessed on March 22, 2024.

[12] kernel.org, "dm-crypt," https://www.kernel.org/doc/html/latest/admin-guide/device-mapper/dm-crypt.html, Accessed on March 22, 2024.

[13] ——, "dm-integrity," https://docs.kernel.org/admin-guide/device-mapper/dm-integrity.html, Accessed on March 22, 2024.

[14] ——, "dm-verity," https://docs.kernel.org/admin-guide/device-mapper/verity.html, Accessed on March 22, 2024.

[15] ——, "Tmpfs file system," https://www.kernel.org/doc/html/latest/filesystems/tmpfs.html, Accessed on May 14, 2024.

[16] Linux man pages, "kexec man page," https://linux.die.net/man/8/kexec, Accessed on March 22, 2024.

[17] S. López, "libkrun Github Repository," https://github.com/containers/libkrun.

[18] Microsoft Azure, "About Azure confidential VMs," https://learn.microsoft.com/en-us/azure/confidential-computing/confidential-vm-overview.

[19] V. Narayanan, C. Carvalho, A. Ruocco, G. Almasi, J. Bottomley, M. Ye, T. Feldman-Fitzthum, D. Buono, H. Franke, and A. Burtsev, "Remote attestation of confidential VMs using ephemeral vTPMs," in *Proceedings of the 39th Annual Computer Security Applications Conference*, ser. ACSAC '23. New York, NY, USA: Association for Computing Machinery, 2023, p. 732–743. [Online]. Available: https://doi.org/10.1145/3627106.3627112

[20] D. Pontes, F. Silva, E. D. L. Falcão, and A. Brito, "Attesting AMD SEV-SNP virtual machines with SPIRE," in *12th Latin-American Symposium on Dependable and Secure Computing, LADC 2023, La Paz, Bolivia, October 16-18, 2023*. ACM, 2023, pp. 1–10. [Online]. Available: https://doi.org/10.1145/3615366.3615419

[21] QEMU, "Direct Linux Boot," https://qemu-project.gitlab.io/qemu/system/linuxboot.html, Accessed on March 26, 2024.

[22] R. Sailer, X. Zhang, T. Jaeger, and L. Van Doorn, "Design and implementation of a TCG-based integrity measurement architecture." in *USENIX Security symposium*, vol. 13, no. 2004, 2004, pp. 223–238.

[23] SPIFFE, "Secure Production Identity Framework For Everyone (SPIFFE)," https://github.com/spiffe/spiffe, Accessed on May 7th, 2024.

[24] tianocore, "Open Virtual Machine Firmware (OVMF)," https://github.com/tianocore/edk2, Accessed on May 7th, 2024.

[25] VirTEE, "VirTEE: A community for building virt-based TEEs," https://virtee.io/, Accessed on March 23, 2024.