SEVeriFast: Minimizing the root of trust for fast startup of SEV microVMs

Benjamin Holmes*
MIT CSAIL
Cambridge, MA, USA
Vassar College
Poughkeepsie, NY, USA
bcwh@csail.mit.edu

Jason Waterman
Vassar College
Poughkeepsie, NY, USA
jawaterman@vassar.edu

Dan Williams
Virginia Tech
Blacksburg, VA, USA
djwillia@vt.edu

Abstract

Serverless computing platforms rely on fast container initialization to provide low latency and high throughput for requests. While hardware enforced trusted execution environments (TEEs) have gained popularity, confidential computing has yet to be widely adopted by latency-sensitive platforms due to its additional initialization overhead. We investigate the application of AMD's Secure Encrypted Virtualization (SEV) to microVMs and find that current startup times for confidential VMs are prohibitively slow due to the high cost of establishing a root of trust for each new VM.

We present SEVeriFast, a new bootstrap scheme for SEV VMs that reevaluates current microVM techniques for fast boot, such as eliminating bootstrap stages and bypassing guest kernel decompression. Counter-intuitively, we find that introducing an additional bootstrap component and reintroducing kernel compression optimizes the cold boot performance of SEV microVMs by reducing the cost of measurement on the critical boot path and producing a minimal root of trust. To our knowledge, SEVeriFast is the first work to explore the trade-offs associated with booting confidential microVMs and provide a set of guiding principles as a step toward confidential serverless. We show that SEVeriFast improves cold start performance of SEV VMs over current methods by 86-93%.

ACM Reference Format:
Benjamin Holmes, Jason Waterman, and Dan Williams. 2024. SEVeriFast: Minimizing the root of trust for fast startup of SEV microVMs. In *29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS '24), April 27-May 1, 2024, La Jolla, CA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3620665.3640424

*Work done while at Vassar College

ASPLOS '24, April 27-May 1, 2024, La Jolla, CA, USA

ACM ISBN 979-8-4007-0385-0/24/04.
https://doi.org/10.1145/3620665.3640424

1 Introduction

In today's cloud, clients run applications on remote machines with the assumption that providers will not collect or leak their sensitive data. Trusted execution environments (TEEs) like AMD Secure Encrypted Virtualization (SEV) [10] have become popular because they remove the host from the trust domain and allow secure computation in the cloud by running applications in enclaves where memory is protected [12–14, 19, 41, 42].

Meanwhile, fast VM boot is a hot topic in the effort to reduce latencies of cloud applications [7, 15, 20, 24, 33, 47]. Serverless computing platforms [9, 22, 25, 34] are at the forefront of fast VM boot, leveraging lightweight virtualization environments, or *microVMs* [7], to provide a transient unit of execution with strong isolation, automatic scaling, and low latency. Two key tenants of microVMs that allow them to boot quickly are *booting an uncompressed kernel*, and *removing extra bootstrap components*. To bypass the cost of decompression during boot, microVM monitors directly load an uncompressed Linux kernel, called a *vmlinux*, rather than a compressed *bzImage*. MicroVMs also do not boot with a BIOS and skip legacy bootstrap steps by exploiting the fact that VMs boot on a system that has already been bootstrapped.

However, we find that SEV is incompatible with microVMs due to the high overhead of establishing trust in a new confidential VM. Contrary to microVM principles for fast boot, we find that adding a new bootstrap component and reintroducing kernel compression minimizes boot times under SEV. Establishing a root of trust for a new SEV VM includes encrypting and measuring a VM's initial code/data before entering the guest [6], which we call *pre-encryption*. One or more pre-encryption operations construct a *launch measurement* in the SEV hardware which is used as part of an *attestation report* that is sent to a trusted third party (referred to as the guest owner) during *remote attestation* and proves to the guest owner that their VM was initialized as expected. While vital to establishing trust, pre-encrypting the initial contents of a microVM—the vmlinux—is prohibitively expensive. Pre-encrypting the kernel, compressed or not, results in overheads up to two orders of magnitude greater than state-of-the-art microVM boot times. With this insight, *we design*

a new minimal bootstrap component to replace the kernel and minimize pre-encryption time.

Avoiding pre-encrypting the kernel excludes it from the launch measurement, which means that a guest owner would be unaware of a malicious kernel is running in their VM. Fortunately, *measured direct boot* [36] has been introduced as a way to verify the kernel's integrity in the guest on the CPU rather than the lower powered SEV hardware. Measured direct boot is faster than pre-encryption, but still has a high cost per-byte. We identify a fundamental trade-off between decompression time and measurement time, and observe that *reintroducing kernel compression to a microVM boot with SEV minimizes the overhead of verifying the kernel on the CPU despite the added cost of decompression.*

We propose *SEVeriFast*[1], a new bootstrap design targeted at optimizing the cold boot performance of SEV microVMs. SEVeriFast produces a minimal root of trust using a specialized *boot verifier* that replaces the kernel as the initial boot code in a microVM utilizing *kernel compression* to reduce overhead during measured direct boot, an *out-of-band kernel/initrd hash* to take redundant computation off the critical boot path, and *optimized pre-encryption* to minimize pre-encryption time for critical boot-related data structures.

We have implemented SEVeriFast in the open source AWS Firecracker[2] Linux/KVM-based VMM, a modern hypervisor targeting microVMs and serverless representing the state-of-the-art for fast VM boot times. To our knowledge, we are the first to implement support for SEV guests in a microVM monitor. Furthermore, our boot verifier is a small stand-alone Rust binary that loads an unmodified Linux kernel. By the time of publication, we plan to have released source code for all components of SEVeriFast.

The value proposition of VMMs like Firecracker is in their ability to boot guests quickly to minimize latency and cost on the part of the cloud provider. Our evaluation of SEVeriFast shows that it boots SEV VMs 86%-93% faster than existing approaches.

To summarize, we make the following contributions:

- The first breakdown of costs and fundamental trade-offs associated with the SEV boot process;
- The design and implementation of SEVeriFast, a bootstrap scheme and set of principles to guide the design of an efficient confidential serverless platform; and
- The evaluation of SEVeriFast compared to existing methods of booting SEV guests and state-of-the-art microVMs with the identification of a bottleneck in the SEV hardware that we plan to address as future work.

[1]Pronounced ES-EE-Very-Fast

[2]https://github.com/firecracker-microvm/firecracker/

2 The Boot Process

In this section we outline the choices made by state-of-the-art microVMs for fast boot and explain the current boot process of an SEV guest. In the next section we show that the two are incompatible due to the high latency of booting an SEV guest.

2.1 MicroVMs

As cloud computing has grown in popularity, the field has seen a shift from long-running VMs with full emulation of traditional systems to more fine-grained units of execution. Containers [2, 18, 30] provide lightweight application sandboxes that are isolated from the host, but VMs provide stronger isolation guarantees. Lightweight VMMs have been developed to bridge the gap between the speed of containers and the isolation of VMs [7, 28, 33].

Serverless computing is a cloud computing paradigm in which users submit a short-running, single-purpose function to the cloud provider. In response to a trigger, the function is executed in its own lightweight microVM to provide strong isolation between functions [9]. In this model, cloud data centers benefit from high density and throughput because microVMs have small memory footprints, low CPU demand, and short lifespans. The cost of booting a microVM is not amortized by its runtime as it would be for traditional long-running VMs [35], and depending on the workload, boot time can dominate overall runtime [8, 11, 20, 37, 39, 45]. Moreover, serverless platforms like Amazon Lambda [9], Microsoft Azure Functions [34], and IBM Cloud Functions [25] only charge users for the time their function spends executing, which is more incentive for providers to keep boot times low.

Booting the guest kernel is typically handled by a guest firmware/BIOS and a bootloader like GRUB, but modern VMMs eliminate these steps in favor of *direct kernel boot* to save boot time [7, 27]. For example, a common method of booting Linux is to use a *bzImage*. A bzImage consists of the kernel ELF file, also called *vmlinux*, compressed and appended to a small bootstrap loader program that decompresses and loads the vmlinux. Compression has historically been used to allow the kernel to be loaded into 1MB of real-mode addressable memory while supporting many device drivers and features, and remained popular because the I/O time to read a vmlinux from a hard disk was greater than the time it took to decompress a bzImage in memory. However, when booting a bzImage on a system that has already undergone the bootstrap process, the bootstrap loader becomes redundant. Additionally, the size of the kernel has a smaller impact on microVM boot times because a vmlinux could be read from a ramdisk or held in Linux's buffer cache when thousands of VM instances use the same kernel, which is likely in a serverless use case. Because of this, decompressing the kernel adds unnecessary time to the critical path [7, 24].

Modern VMMs bypass decompression and redundant bootstrap steps by directly loading a vmlinux, leading to faster boot times. Modern VMMs also exploit the fact that they are running on an already bootstrapped system to do the following on the guest's behalf prior to boot:

1. Load the kernel ELF in one operation to the location in guest memory where it will run
2. Set up important data structures for Linux to boot (Linux boot_params, page tables, boot stack, etc.)
3. Skip the transition from real-mode to long-mode and enter the guest at the kernel's 64-bit entry point

Finally, Firecracker distributes microVM Linux guest kernel configurations to decrease both the size and startup time of a vmlinux, which is key to Firecracker's low startup times.

2.2 Secure Encrypted Virtualization (SEV)

Under the typical cloud computing model, the hypervisor has complete control over VM memory. Therefore, the guest owner has to trust that the hypervisor is benevolent and free from bugs that could lead to information leaks. SEV is a hardware extension on AMD EPYC CPUs that protects data from privileged attackers in secure enclaves by encrypting the physical memory backing a VM. The extensions consist of an encryption engine embedded in the memory controller that transparently encrypts/decrypts data when given the correct key and the Platform Security Processor (PSP), a small low-powered ARM core integrated within the SoC that handles key generation/management, establishing trust during VM launch, and generating attestation reports used for remote attestation.

More recently, extensions to SEV have been introduced in the form of SEV-ES [5] (Encrypted State) and SEV-SNP (Secure Nested Page Tables), where SEV-SNP is a superset of all three versions. SEV-ES ensures that guest register state is encrypted when it is saved during a world-switch. SEV-SNP [1] introduced a new system-wide structure called the Reverse Map Table (RMP), which holds page mappings from system physical addresses to guest physical addresses, and tracks ownership of guest pages to prevent the host from writing to encrypted guest pages. When a write to guest memory occurs, the RMP is checked after normal address translation to ensure that the access came from the correct owner. When the guest runs, it must first validate each page it intends to use for encrypted memory. This is done via the `pvalidate` instruction which sets the valid bit in the RMP entry corresponding to the guest address being validated. `pvalidate` can only be executed from within an SEV guest and is the only way to validate a guest mapping. If the hypervisor changes a mapping, the hardware clears the valid bit in the corresponding RMP entry. Then, if the guest tries to access that page, a new hardware exception called the VMM Communication Exception (#VC) is generated to be caught by the guest indicating that a mapping has been tampered

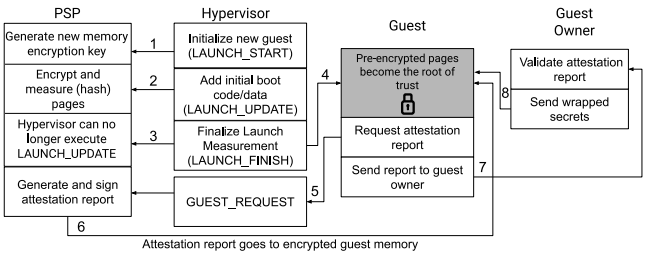

Figure 1. SEV boot overview

with.[3] These changes ensure that secrets cannot leak through register state, and guests are protected from replay attacks, data corruption, memory aliasing, and memory re-mapping by a malicious host. Since SEV-SNP enforces memory integrity and is the latest version of SEV, all experiments in this paper are run with SEV-SNP.

2.3 Trust Model

SEV allows the guest to remove the host from the trust domain by preventing writes to memory from entities outside a VM and ensuring that reads resolve to encrypted data. We assume that an attacker has control over the entire host software stack, i.e., the host operating system, VMM/hypervisor, firmware, and control software, and has physical access to the machine. We trust AMD hardware (the PSP), the firmware running on the PSP, and any software the guest owner provides to run in their VM, provided that it is proven to have not been tampered with through remote attestation. Because the AMD hardware is responsible for protecting guest memory, we trust that data in encrypted memory cannot be leaked to the host.

In a cloud setting, we assume that providing the host with a plain text kernel, kernel cmdline, and initrd does not lead to leaking confidential data because none of these components will contain secrets, and remote attestation proves to the guest owner that none of these components have been tampered with. Any secrets the guest owner needs will be provided after establishing a secure channel via successful remote attestation.

SEV does not provide any protection against denial-of-service (DoS) attacks; it guarantees that secrets are not readable outside a guest, and that encrypted guest memory cannot be modified by the host. Architectural side channel attacks, e.g., prime+probe based attacks, page access tracking, and performance counter tracking, are out of scope. While encrypting the code running in a guest prevents attackers from knowing what applications are being run, encryption alone does not prevent fingerprinting. AMD notes that future iterations of SEV may have increased protections against side channel attacks [1].

[3]The RMP check does not need to happen during a read because guest-owned pages are encrypted.

2.4 Launching an SEV Guest

The steps to securely boot an SEV guest differ from the traditional VM boot sequence in two main ways:

a) Guest code/data is encrypted to protect it from the host.
b) A VMs initial state is measured to form a root of trust.

The SEV API [6] details a series of launch commands that are issued by the hypervisor to the PSP during each boot sequence to provide the security guarantees from (a) and (b). As shown in Figure 1, the `LAUNCH_START` command initializes the SEV platform and allocates a new memory encryption key for a guest (1). The `LAUNCH_UPDATE_DATA` command tells the PSP to hash a region of guest memory (containing initial code/data) to include in the launch measurement, then encrypt it with the new guest's key to protect it from the host (2). For the rest of the paper, we call this process *pre-encryption*.

The hypervisor executes `LAUNCH_FINISH` to finalize the launch flow which transitions the guest to a state where the hypervisor can no longer execute `LAUNCH_UPDATE_DATA` (3). This prevents the hypervisor from encrypting any additional guest memory after an attestation report has already been requested and verified. Pre-encrypting some portion of guest memory is necessary for a secure boot with SEV because it adds the guest's initial state to the root of trust. The VMM then enters the guest at code inside the root of trust (4). This code first validates guest pages with `pvalidate` and initializes page tables with the enCryption bit (C-bit) set in every entry that corresponds to an encrypted page. The C-bit is carried through to the encryption engine during address translation and determines whether the data at the resolved address will be encrypted/decrypted on its way to the CPU. Once page tables are initialized and memory is validated, the guest has full control over protecting memory from the host, and it can continue to boot securely.

Once the guest runs, it requests an *attestation report* from the PSP that contains the launch measurement that is signed by a chip-unique key. Before any secrets are provided to an SEV guest, the attestation report must be verified by the guest owner to establish a secure channel. Attestation is performed after the guest kernel boots via the following steps shown in Figure 1:

a) Guest requests attestation report from PSP (5)
b) PSP places attestation report directly in encrypted guest memory (6)
c) Guest sends attestation report to guest owner (7)
d) Guest owner validates report and sends secrets to guest (8)

Since secrets should not be provided before attestation, the code responsible for attestation must be plain text, and thus vulnerable to the host. Attestation code should be placed in encrypted guest memory and reflected in the launch measurement to ensure that it is protected and was not tampered with. Thus, we assume all attestation code enters the guest via an initrd that is provided to the VMM as plain text and mounted in encrypted memory.

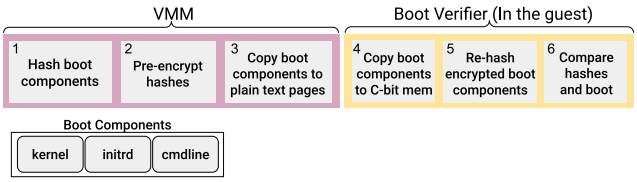

Figure 2. Measured Direct Boot

2.5 Current State of SEV Boot

The mainstream hypervisor and guest support for booting an SEV guest is split between QEMU and the EDKII project's Open Virtual Machine Firmware (OVMF)[4]. Under pressure from microVM capable VMMs like Firecracker, QEMU has recently added support for direct boot instead of emulating bare-metal boot. With a similar motivation, support for *measured direct boot* [36] was added to QEMU/OVMF to allow SEV guests to bypass GRUB.

With measured direct boot, the user provides the VMM with a plain text guest kernel, initial ramdisk (initrd), and kernel command line. Each component is then given to the guest which lays them out in memory and boots the kernel directly. However, this means that the boot components are not in guest memory until after the guest starts running, so they would not be reflected by the launch measurement. To prevent the host from loading malicious components without the guest owner's knowledge, the VMM and the guest must do the following, depicted in Figure 2:

1. The VMM hashes the boot components (kernel, initrd, and kernel command line).
2. The VMM pre-encrypts the hashes which will include them in the final launch measurement so they can be verified by the guest. Encrypting them also protects them from the VMM because the VMM cannot write to encrypted pages
3. The VMM transfers boot components to the guest through shared (plain text) pages.
4. The guest protects components by copying them from the shared region to an encrypted region.
5. The guest re-hashes the encrypted components and compares against the pre-encrypted hashes.
6. If the hashes match, the guest loads the protected (encrypted) components.

For the rest of the paper, we call the process of copying boot components to encrypted memory, re-hashing them, and comparing the new hashes to the pre-encrypted hashes *boot verification*. We call the unit that carries out this process the *boot verifier*.

[4] https://github.com/tianocore/edk2

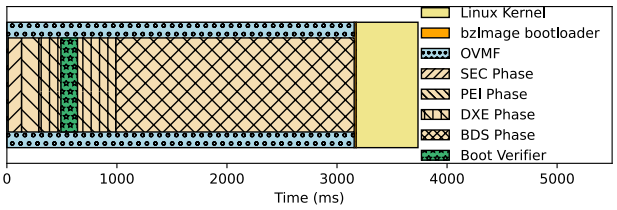

Figure 3. Breaking down the OVMF boot process with SEV-SNP shows that the boot verifier is a small portion of overall boot time.

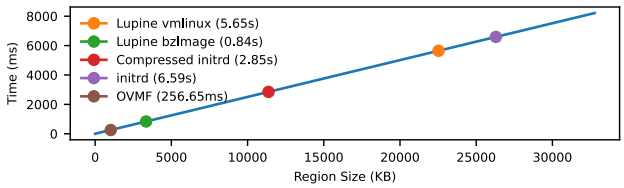

Figure 4. Even with the smallest possible candidates for initial boot code, pre-encryption with SEV-SNP is prohibitively expensive.

2.6 Security Implications of Measured Direct Boot

Measured direct boot relies on a key insight: *the hash of a component can be pre-encrypted instead of the component itself and used to check its integrity.* Although the boot components themselves are not included in the root of trust, using measured direct boot does not compromise the integrity of an SEV guest for two reasons that we explain below.

Secret-free Construction. Each component is generic and open source; none are built with secrets. The kernel is compiled from an unmodified source tree. The initrd includes the open-source kernel module, scripts, and set of standard command line tools needed for attestation. Keys used to wrap secrets are generated in encrypted guest memory at attestation time so they are not present in the plain text initrd and protected from the host. Any secret data needed after attestation, like the key to an encrypted disk, can be provided over the network via the secure channel established during attestation.

Protection from the Host. There are three ways that the host could attempt to make the guest load compromised components.

1. The host could give the guest malicious boot components after the hashes of the correct components are pre-encrypted. The boot verifier detects this when it checks the hashes.
2. The host could pre-encrypt hashes of the malicious components so the boot verifier would trust them. The guest owner will detect this in the attestation report because the launch digest will not match the expected digest.
3. Finally, the host could load a malicious boot verifier that does not check hashes. The guest owner also detects this in the attestation report because the boot verifier must be pre-encrypted and the launch digest will not match the expected digest.

3 MicroVM Boot is Incompatible with SEV

In this section we show that current state of SEV boot is incompatible with microVMs due to the high cost of measurement while establishing a root of trust on the critical boot path.

3.1 Large firmware is not suited for microVMs

OVMF's goal is to provide UEFI support for virtual machines. Because of this, it conforms to UEFI standards and supports features that are unnecessary when booting a microVM including support for drivers that emulate hardware devices, running a UEFI shell, and executing EFI programs. It is Platform Initialization (PI) [43] compliant, so it supports the six boot phases outlined by the UEFI specifications. Supporting these boot phases results in redundant and unnecessary computation since the VMM is a process running on a system that has already undergone this bootstrap.

Figure 3 breaks down these phases during an SEV-SNP boot and shows that OVMF's runtime is over 3 seconds. For reference, on our system (as shown in Section 6), booting the standard AWS microVM configured Linux kernel with Firecracker without SEV takes about 40ms. The main insight here is that the only portion needed for SEV is the "Boot Verifier" where the boot components (kernel, initrd, and kernel command line) are checked for integrity. Furthermore, the smallest supported build of OVMF is 1MB which is large enough for pre-encryption to be expensive, adding an extra 256.65ms to the overall boot time of an SEV guest, which is many times slower than the boot time of a standard microVM. A successful SEV boot does not depend on any UEFI support, and coupling SEV bootstrap with UEFI bootstrap clashes with the microVM theme of eliminating bootstrap steps for performance gains.

3.2 Direct boot is incompatible with SEV

The initial code that runs in an SEV guest must be pre-encrypted to include it in the root of trust so the guest has a trusted entry point. Because OVMF is incompatible with microVMs, we next explore the possibility of bypassing the guest firmware and adapting direct boot for SEV. In a typical direct boot, the VMM parses the vmlinux and loads each

ELF segment to its location to run in guest memory. The VMM also loads an uncompressed initrd into guest memory. However, with SEV the initial boot code—in this case, the kernel and initrd—must be pre-encrypted. Figure 4 shows that pre-encryption time grows linearly with size, leading us to choose the smallest Linux kernel configurations for our attempt at direct boot with SEV. We chose the `lupine-base` configuration from Lupine Linux [31] for our kernel as it represents a lower bound on the size of a kernel capable of running general-purpose applications.

Taking the typical direct boot approach using our Lupine `vmlinux` (23MB), pre-encryption takes 5.65 seconds on average, which is two orders of magnitude greater than a reference non-SEV boot of the Lupine kernel which takes less than 40ms. By reintroducing compression we can reduce the size of the kernel/initrd in hopes of better pre-encryption performance. However, pre-encrypting the Lupine `bzImage` takes 840ms on average, and pre-encrypting the compressed initrd (12M) still takes 2.85s, an increase over our reference microVM boot time of two orders of magnitude. *Because pre-encryption is costly even with the smallest components, we need a new initial boot component to minimize the size of the root of trust.*

3.3 Measured direct boot favors kernel compression

As described above, measured direct boot can alternatively be used to pre-encrypt a hash of the kernel/initrd instead of the components themselves. Measured direct boot is faster than pre-encryption because the CPU handles measurement rather than the less powerful PSP, but still costly since we pay twice per byte: once for the copy from plain text to encrypted memory, and again for the second hash. Because these operations both depend on size, *minimizing kernel size is critical to fast boot.* Copying and hashing a bzImage is cheaper than copying and hashing a vmlinux due to compression. However, this reintroduces decompression to the critical path of a microVM boot. Figure 5 shows the cost of compression during a measured direct boot. There are two main takeaways from Figure 5:

1. Regardless of kernel size, the most efficient way to do measured direct boot with Linux is to use a bzImage compressed with LZ4.
2. It is faster to leave the initrd uncompressed because the CPIO archive already needs to be unpacked and adding decompression adds unnecessary overhead.

We collected measurements using kernel configurations from Lupine Linux [31], the AWS configuration shipped with Firecracker, and a typical Ubuntu configuration to represent small, medium, and large kernels, respectively. Section 6 discusses kernel configuration choice in more detail. The initrd contains only the functionality to perform remote attestation and its size does not depend on the kernel configuration.

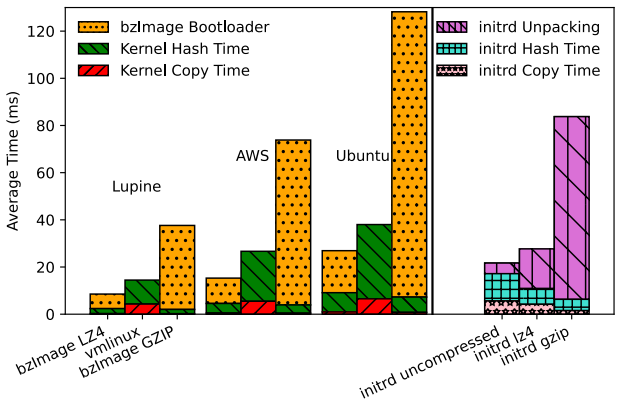

Figure 5. The overhead of measured direct boot steps for the kernel and initrd shows that the most efficient way to boot is with an LZ4 compressed kernel and an uncompressed initrd

4 SEVeriFast

Figure 6 provides an overview of SEVeriFast's design. SEVeriFast is based on the overarching principle that the amount of data measured during boot should be minimized to achieve fast secure boot with SEV. SEVeriFast is made up of the following 4 components, highlighted in Figure 6:

1. A minimal boot verifier to limit the size of the root of trust;
2. An optimized pre-encryption component that minimizes pre-encryption time for important data structures;
3. An out-of-band kernel/initrd hashing component that reduces measurement on the critical boot path; and
4. An LZ4 kernel compression component that minimizes time spent during measured direct boot.

SEVeriFast's goal is to optimize the cold start performance of SEV guests. Although warm start is an active area of research [16, 17, 20, 37, 38, 44], cold start invocations make up a significant portion of overall serverless function invocations [39]. We discuss our decision to focus on cold start and the challenges associated with warm start for SEV in Section 7.

4.1 Minimal Boot Verifier

Our design relies on the key insight from Section 2.6 that the root of trust should be as small as possible. The kernel and initrd are too large to be pre-encrypted, so the optimal way to boot an SEV guest is to add each component's *hash* to the root of trust instead and load the kernel/initrd via measured direct boot. With this in mind, we designed a minimal boot verifier to include in the root of trust as the initial guest code with the sole purpose of verifying the kernel and initrd.

As discussed in Section 2, microVMs reduce boot time over traditional VMs by removing unnecessary driver support and

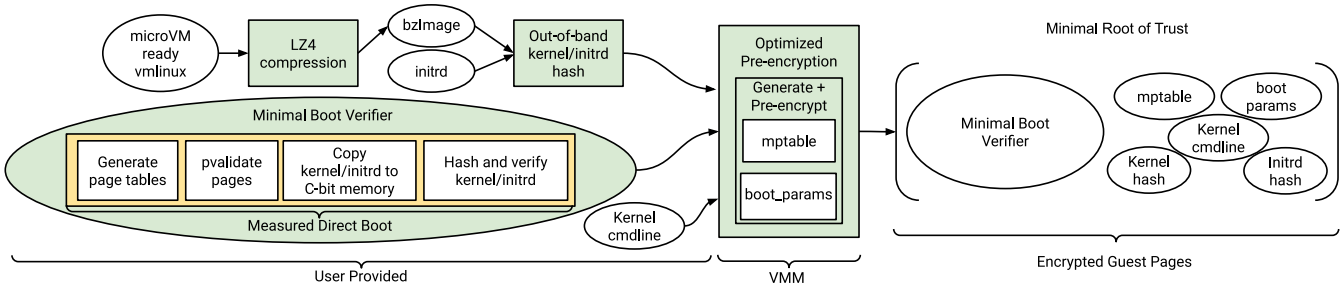

Figure 6. SEVeriFast's components (shown in green) produce a minimal root of trust to optimize the boot time of SEV guests

Data Structure	Purpose	Struct Size	Code Size	Pre-encrypt or Generate?
mptable	CPU config	284B + 20B/CPU	4KB	Pre-encrypt
cmdline	Kernel args	155B	N/A	Pre-encrypt
boot_params	System info	4KB	5KB	Pre-encrypt
page tables	Paging in guest	4KB	2.4KB	Generate

Figure 7. We pre-encrypt the data structures the kernel needs to boot when the code required to generate them is larger than the structure itself.

redundant bootstrap from a BIOS or bootloader. SEVeriFast follows this principle, and we choose to only include the functionality required to securely load the kernel/initrd in our boot verifier. Figure 6 shows that the boot verifier simply initializes protected memory by setting the C-bit in all page table entries and executing `pvalidate` on every page in guest memory, then performs measured direct boot with the process outlined in Figure 2. Because we minimize the functionality of the boot verifier, we have a standalone binary that is about 13KB that we include in the root of trust. In addition to reducing the amount of trusted code in the guest, our small boot verifier helps minimize the size of the root of trust thus minimizing the cost of pre-encryption.

4.2 Optimized Pre-encryption

During a typical microVM boot, the VMM generates data structures that the kernel needs and loads them into guest memory. When the guest kernel boots, it accesses these data structures in pages with the C-bit set so, for integrity and compatibility with Linux, they must be encrypted.

We can choose to pre-encrypt these data structures if the VMM already generates them, but this increases the size of the root of trust and adds pre-encryption time. The alternative is for the boot verifier to generate the data structures after launch in C-bit memory to implicitly encrypt them. However, our boot verifier is designed to be small, and adding functionality to generate data structures will increase its size and in turn increase pre-encryption time. *Because our goal is to minimize the size of the root of trust, we pre-encrypt a data structure only when the code that generates it is larger than the structure itself.*

Figure 7 shows the data structures the kernel needs at boot time, the purpose of each data structure, their sizes, and whether we pre-encrypt or generate them. We pre-encrypt the `mptable` because it only spans 304 bytes for a VM configured with a single CPU, plus 20 bytes for each added CPU, while the code required to generate it is around 4k. The boot_params span a 4k page, and the code required to generate it is about 5k so we pre-encrypt it as well. We pre-encrypt the kernel command line for two reasons. While the command line has a maximum size of 4K, the default command line provided by Firecracker is only 155B. The cmdline is also passed to the VMM by the client, so it cannot be generated by the VMM or the guest. The only other way to get the cmdline to the guest would be to hash and verify it in the boot verifier, but because the command line is small, avoiding pre-encryption does not lead to significant performance gains. Finally, we choose to generate page tables in the boot verifier rather than in the VMM because the page tables span 4K in memory (1GB identity mapped with 2MB pages) and removing the code required to generate them from our boot verifier only saves 2.4KB.

Pre-encrypting more than just a single binary blob adds complexity to computing the expected launch measurement, but we remedy that by including a tool with SEVeriFast to generate a SHA256 digest of each pre-encrypted component. To generate the digest, the tool needs the guest kernel hash, the initrd hash, the boot verifier, and a Firecracker VM configuration file. This way it can hash the command line, generate the mptable, and boot_params, then hash them as well. The digest can then be used to compare against the digest received with an attestation report during remote attestation.

4.3 Out-of-band Hashing

Because the kernel and initrd are too large to be included in the root of trust, we use measured direct boot to load them. Measured direct boot requires the kernel and initrd to each be hashed twice: once before pre-encryption so the hashes are included in the root of trust, and again in the guest to compare to the original hashes and verify that the correct components were loaded. Hashing the kernel/initrd in the VMM could add up to 23ms of boot time, so we hash them off the critical boot path to save time. Providing the

VMM with pre-computed hashes at boot time does not compromise security because the hashes will be pre-encrypted and reflected in the attestation report. We modify the VMM to take a file containing the kernel hash and initrd hash as extra arguments to remove redundant measurement from the critical path.

4.4 Kernel Compression

Based on our observations from Section 2.6, measured direct boot with an LZ4 compressed bzImage is the fastest way to securely boot the Linux kernel, and a key element of our design is the decision to move away from booting an uncompressed vmlinux like typical microVMs. Importantly, compression has been considered harmful with respect to microVMs because decompression is expensive. In the context of SEV where measurement is more expensive, compression reduces the data being measured during boot and thus the time to establish a root of trust through measured direct boot. Furthermore, booting a bzImage simplifies measured direct boot compared to an optimal measured direct boot with an uncompressed vmlinux, which we describe in more detail in Section 5. We modify the VMM to place the kernel, now a compressed bzImage, at a location in memory known to the boot verifier. The boot verifier also only contains support for loading a bzImage which helps it stay small since loading a bzImage requires less code than parsing and loading the uncompressed kernel ELF.

5 Implementation

We implemented support for launching SEV guests in Firecracker v0.26 because it is the state of the art in terms of microVM boot times, and provides minimal functionality for the guest which helps minimize the complexity of attestation. For example, Cloud Hypervisor[5] boots Linux with PVH and ACPI enabled, neither of which are needed for a microVM boot and would add extra data structures to guest memory that would need to be encrypted. Firecracker is open source and implemented in Rust, and our modifications do not affect the boot process of a non-SEV VM. The code we added to support SEV is mostly contained in its own Rust module, and spans just over 1100 lines of code. We also modified the monitor to take in our boot verifier and kernel/initrd hashes as arguments.

Our boot verifier is also implemented in Rust, based on a fork of `rust-hypervisor-firmware`[6]. Rust hypervisor firmware was already small, but still contained support we did not need: a virtio block device driver, support for reading files from FAT partitions, booting with PVH, executing EFI programs, and PCI drivers. The virtio drivers, file system support, PCI drivers, and support for executing EFI programs were there because the firmware supported booting Linux either through GRUB, which is an EFI program, or by reading the kernel from the guest file system. Since our design relies on the VMM to put the kernel into guest memory, we removed all of this support. The support for PVH boot was there to provide the firmware with information about the VM so it could generate the Linux boot_params. Since Firecracker does not use PVH, we removed PVH support as well. The only functionality that we retained is support for initializing page tables and loading a bzImage. We modified the existing page table support to check if SEV is enabled and find the C-bit position in a page table entry by executing two `cpuid` instructions, and set the C-bit in all entries. We also added support for validating all of guest memory with `pvalidate` before booting the kernel. The bzImage loader was originally intended to load a bzImage from a file, so we modified it to load a bzImage from a memory region instead. We also added the `sha2` Rust crate to the boot verifier to support hashing the kernel/initrd with SHA256. We chose this crate because it uses the x86 SHA instructions which support hardware acceleration of the SHA family.

Uncompressed vmlinux: To compare against a bzImage boot, we also implemented support for booting an uncompressed vmlinux directly. However, loading a vmlinux through measured direct boot is more complex from the perspective of the boot verifier than loading a bzImage. The steps to load a vmlinux through measured direct boot are as follows.

1. The VMM copies the vmlinux into guest memory.
2. The boot verifier copies the plain text vmlinux to an encrypted region.
3. The boot verifier parses the vmlinux and copies each ELF segment to its location to run.

Step 3 is an extra copy of the kernel that does not happen when loading a bzImage, and to give a vmlinux the best chance against a bzImage we implemented a protocol to load a vmlinux that avoids this copy.

Specifically, we implemented a version of QEMU's `fw_cfg` device which allows the boot verifier to load the vmlinux directly from the host file system into guest memory. It does this by parsing the kernel ELF in the VMM rather than in the guest, and copying the ELF header, the program headers, and the loadable segments into guest memory, rather than the entire ELF. This way, the new steps for loading a vmlinux through measured direct boot are:

1. The VMM copies the ELF header into guest memory.
2. The boot verifier copies the ELF header to encrypted memory and hashes it.
3. The VMM copies the program headers into guest memory.
4. The boot verifier copies the program headers to encrypted memory and hashes them.

[5] https://github.com/cloud-hypervisor/cloud-hypervisor
[6] https://github.com/cloud-hypervisor/rust-hypervisor-firmware

kernel config	vmlinux size	bzImage size
Lupine	23M	3.3M
AWS	43M	7.1M
Ubuntu	61M	15M

Figure 8. Guest kernels used in boot time experiments

5. The VMM copies the loadable segments into guest memory.
6. The boot verifier copies the loadable segments to where they should be loaded in encrypted memory and hashes them.

Because the constituent parts of the kernel ELF are copied separately into guest memory rather than the full vmlinux, they require three separate hashes. However, because the loadable segments can be copied directly from plain text memory to the (encrypted) location that they will run, we avoid the extra copy of the full vmlinux.

6 Evaluation

SEVeriFast is designed to allow microVMs to benefit from the confidentiality offered by SEV by minimizing the additional cost associated with cold boot caused by SEV. Thus, when evaluating SEVeriFast, we answer the following questions:

- Does SEVeriFast minimize SEV-related bootstrap?
- How do SEVeriFast and non-SEV microVM boot compare?
- How does SEVeriFast affect microVM memory usage?
- How do concurrent SEVeriFast cold boots perform?

6.1 Experimental Setup

This section describes the hardware used in our experiments, the Linux version and configurations used, our modifications to the Firecracker VMM, and our testing methodology.

Environment Setup: All experiments were run on a machine with an AMD EPYC 7313P @ 3.0GHz and 128 GB of DDR4 memory @ 3200 MT/s. This machine was running a development 6.1.0-rc4 Linux kernel with the AMD patches to support launching SEV-SNP guests. We additionally patched this version of Linux to avoid an expensive and unnecessary worker thread intended as an iTLB multi-hit mitigation on Intel CPUs[7] that is not needed on AMD CPUs.

Guest Kernel Configurations: Figure 8 summarizes the guest kernel configurations we used in our experiments. Each kernel was compiled from a Linux 6.4 source tree with patches to support booting with SEV-SNP through the non-EFI code path. The patches include executing `pvalidate` on memory regions that are touched in the non-EFI code path but are not touched by the EFI code path in the bzImage bootloader. This was necessary so that once the real kernel boots the memory it expects to be valid/invalid is correct. We chose representative small, medium, and large kernel configurations to highlight the best, typical, and worst cases when booting under SEVeriFast. We include the vmlinux version of each kernel as a comparison against the bzImage to show that our design choice to always use compressed kernels for faster boot times holds regardless of kernel configuration. Each bzImage is compressed with LZ4 in accordance with our design. Each kernel is compiled with SEV support (`CONFIG_AMD_MEM_ENCRYPT`), support for obtaining attestation reports (`CONFIG_SEV_GUEST`), and support for the necessary virtio drivers (`CONFIG_VIRTIO_BLK`, `CONFIG_VIRTIO_NET`) needed to boot in Firecracker. For the sake of equivalent comparison, we use the kernels compiled with SEV support when booting non-SEV guests as well.

The *Lupine* configuration is a version of the `lupine-base` configuration from Lupine Linux[31]. It is compiled from the same Linux tree as the other kernels and does not use the kernel patches used in the Lupine paper. Because the goal of Lupine was to create Linux unikernels, our Lupine kernel represents the smallest single purpose Linux kernel that boots in Firecracker. We use the base Lupine config which is configured without networking support, so we do not include attestation time in our results for the Lupine kernel. If we were to enable networking, the Lupine kernel and the AWS kernel become similar in size, so we include it only as a lower bound to illustrate how SEVeriFast scales with respect to kernel size. The *AWS* configuration is the microVM configuration provided by Firecracker and represents a kernel purpose built to run in Firecracker with minimal driver support that can run general purpose workloads. Our last configuration, *Ubuntu*, is based on the Linux 5.15.0-53-generic configuration that came with our distribution of Ubuntu. Each configuration was updated to the 6.4 kernel by running `make olddefconfig` in the source directory.

Attestation: To emulate remote attestation, we run a local `nginx` server that receives and validates an attestation report. Upon successful validation, the server wraps a secret with the guest's public key and sends it to the guest. We use open source scripts provided by AMD[8] in our attestation server. The cost of attestation on our test machine is about 200ms for all VM configurations. In practice, this cost could be amortized by combining attestation with user-level protocols for database fetches when a serverless function needs to access confidential data. Because attestation would not be necessary until the guest's function runs, we do not consider attestation part of boot, but we include it to show the end-to-end cost of establishing trust with SEV.

[7] https://www.kernel.org/doc/html/latest/admin-guide/hw-vuln/multihit.html

[8] https://github.com/AMDESE/sev-guest

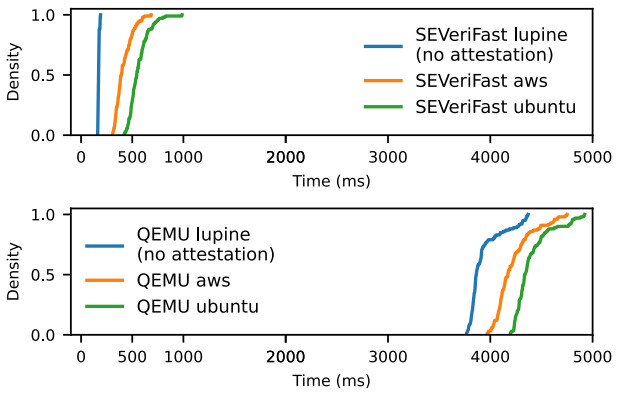

Figure 9. Cumulative distribution of boot times starting SEV-SNP VMs in series with SEVeriFast compared to QEMU/OVMF, measured from the time the VMM is executed until remote attestation is completed

Firecracker Versions: SEVeriFast includes modifications to Firecracker v0.26 that add support for launching SEV, SEV-ES, and SEV-SNP guests with our design choices outlined in Section 4. Our modifications do not make any changes to the non-SEV boot path, other than slightly increasing the size of the Firecracker executable, so we use the same binary to compare SEV boot times to non-SEV boot times. Data labeled 'Firecracker' uses the non-SEV boot path, while data labeled 'SEVeriFast' takes the SEV boot path.

Testing Methodology: To measure overall boot times and the constituent parts of a VM boot we needed to extract timing information from a running guest. To do this in Firecracker, we modified the VMM to attach a debug port device before booting the VM, inspired by the same technique in Cloud Hypervisor.[9] The debug port device listens on port `0x80` for writes from the guest, and records them with timestamps in the Firecracker logs. This way, the boot verifier and guest kernel can simply execute the `outb` instruction at important intervals and timing information will be recorded automatically. We take a similar approach to measuring boot times with QEMU/OVMF with a combination of QEMU logging and `outb` instructions in OVMF. With SEV-ES/SNP guests, executing `outb` generates a VMM communication exception (#VC) because handling an I/O write requires the guest to share register state with the VMM. There are points during early boot where #VC handlers have not been installed yet, so as a workaround SEV-ES/SNP guests leverage writing to the GHCB MSR which is always intercepted by the VMM. We modified Firecracker and QEMU to interpret magic values written to the GHCB MSR as timing events and log them. We use the `perf` utility to record timestamps

[9]https://github.com/cloud-hypervisor/cloud-hypervisor/blob/main/devices/src/legacy/debug_port.rs

	Pre-encryption	Firmware/Boot Verification
QEMU Ubuntu	287.80ms	3239.71ms
QEMU AWS	287.76ms	3181.40ms
QEMU Lupine	287.91ms	3168.53ms
SEVeriFast Ubuntu	8.19ms	32.96ms
SEVeriFast AWS	8.22ms	24.73ms
SEVeriFast Lupine	8.07ms	20.36ms

Figure 10. Boot time breakdown of SEVeriFast vs. QEMU. SEVeriFast reduces average pre-encryption time by 97% and average firmware runtime by 98%.

of debug port writes from OVMF, and the QEMU logs to capture events that occur inside the VMM.

We consider boot time to be the time between executing the VMM process and executing `init` in the guest. We then break down the overall boot time into four parts: *Firecracker/QEMU*, the time spent in the respective VMM before entering the guest, *Boot Verification*, the time spent running the boot verifier in the guest, *Bootstrap Loader*, the time the bzImage bootstrap loader spends decompressing and loading the vmlinux, and *Linux Boot*, the time between executing the guest kernel and running `init`. We take the average of each constituent part over 100 runs when calculating boot time, and include error bars to show one standard deviation of overall boot time. We boot each kernel five times before collecting data to allow the kernel file to be warm in the buffer cache, then run all subsequent boots sequentially. We assume that a production deployment of Firecracker will use the same kernel per VM instance, and will likely have it resident in memory or cached in some form to save I/O time. Each VM is configured with 1 vCPU and 256MB of memory. All experiments are run with transparent huge pages enabled. In our testing, we found that enabling huge pages brings the time to `pvalidate` guest memory down from over 60ms to a negligible <1ms on average. Enabling huge pages decreases pre-encryption time with base SEV and SEV-ES, but had no effect with SEV-SNP.

6.2 Boot Times

QEMU. Figure 9 compares the overall boot performance between SEVeriFast and QEMU including attestation time to show that our design accomplishes its goal of greatly reducing boot time over the state-of-the-art. SEVeriFast reduces average boot times by 93.8% for the Lupine kernel, 88.5% for the AWS kernel, and 86.1% for the Ubuntu kernel. Figure 10 shows that we save significant time in both pre-encryption and the guest firmware runtime. These savings come from the fact that our standalone boot verifier is small (about 13KB) compared to OVMF. As discussed in Section 4, microVMs do not use a guest firmware or BIOS, and we take advantage of that by only including the necessary functionality in our boot verifier.

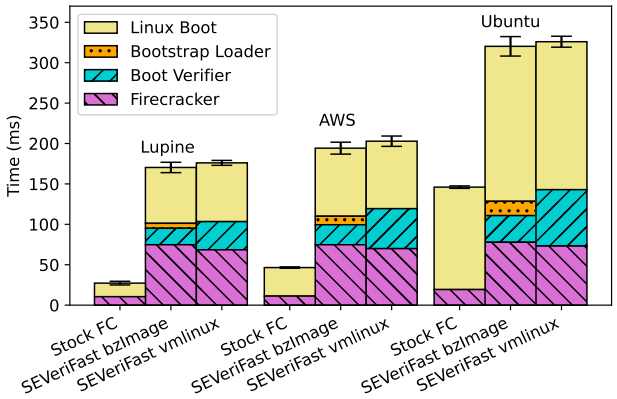

Figure 11. Breakdown of boot times between Stock Firecracker (Stock FC), SEVeriFast with a compressed kernel (SEVeriFast bzImage), and SEVeriFast booting an uncompressed kernel (SEVeriFast vmlinux). Experiments with an uncompressed kernel use our optimized ELF loader from Section 5.

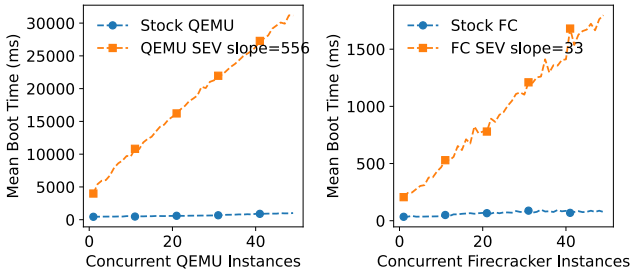

Figure 12. Average boot time of concurrent SEV guests from 1 to 50 concurrent instances.

Stock Firecracker. Because the SEV boot process breaks the assumption that directly booting an uncompressed kernel is the most efficient way to boot, SEVeriFast adds unavoidable overhead compared to a cold boot of a non-SEV microVM. Similarly, there is extra cost in the VMM when launching an SEV guest because KVM needs to initialize the RMP entries mapping guest memory before boot. Figure 11 shows the overhead that SEVeriFast adds to a non-SEV boot in Firecracker, and the overhead of booting a vmlinux with our design. We do not include attestation time in our comparison to stock Firecracker because our modifications to Firecracker do not affect attestation.

The boot time of the AWS kernel with SEVeriFast is about 4x greater than that of stock Firecracker. The two greatest sources of overhead are *Linux Boot* and *Firecracker*. *Linux Boot* takes about 2.3x longer than booting Linux without SEV. We believe this extra overhead comes from the cost of hypercalls and memory accesses in SEV-SNP. When the guest does a world-switch, the #VC handler runs and the guest decides which register state to expose to the host. Similarly, on every write to guest memory an RMP check occurs to ensure the access is coming from the guest. SEVeriFast does not make any changes to the Linux boot process. In addition to pre-encryption, the overhead added to the time spent in Firecracker comes from the other SEV launch commands, and extra communication with KVM to register pages in guest memory that may contain encrypted data. Encrypted pages with identical plain text contents at different physical locations will have different ciphertexts under SEV, so KVM pins guest memory pages during boot, incurring more overhead. The remaining overhead comes from three sources: pre-encryption, boot verification, and the bzImage bootstrap loader. Pre-encryption adds small constant overhead (<9ms) because the boot verifier and boot data structures make up the components that are pre-encrypted which do not depend on the size of the kernel.

A key element of our design is to boot a compressed kernel because the time saved hashing it in software outweighs the extra decompression cost. As shown in Section 2.6 hashing/copying an uncompressed kernel can take twice as long compared to a compressed kernel. This cost is amortized in these results by the higher *Linux Boot* and *Firecracker* costs. If these costs are further optimized, the choice to boot an uncompressed kernel is increasingly important. Moreover, our optimized ELF loader described in Section 5 reduces overhead compared to the naive approach which would show a more significant difference in boot times.

Overall, SEVeriFast provides SEV support to microVMs at speeds that are more comparable to typical microVM boot times than the current method of booting SEV guests in QEMU. The security benefits of SEV do come at a cost, but our design minimizes this cost to preserve the low latency execution of serverless functions.

Concurrent VMs. One of the attractive aspects of microVMs is that their low memory footprints and boot times allow for a high degree of concurrency. Thus, concurrent microVM launch performance with SEVeriFast is an important metric to consider. Figure 12 shows that unfortunately, as the number of concurrent SEV guest launches increases, the average boot time of each VM increases linearly to the point that at just 50 concurrent guests average boot time is around 1800ms. However, if we do the same experiment without SEV, boot times remain almost constant. This is true for both our design and QEMU/OVMF, so we do not believe this is a fault in our design. Rather, we believe these results uncover a fundamental bottleneck in the SEV hardware.

We suspect that this bottleneck comes from the fact that the PSP is a single core, and all launch commands for every new SEV guest must go through it. This essentially serializes the boot process until `LAUNCH_FINISH` is called in each guest each VM is competing for the PSP during launch. By virtue of

our design minimizing pre-encryption times, concurrent VM launches with SEVeriFast significantly outperform QEMU, as they remain below the boot time of a single SEV VM with QEMU even at 50 concurrent guests. However, since average startup time increases linearly with a slope equal to the total time it takes to execute the SEV launch commands, boot times remain expensive for serverless.

Because only long-running SEV guests have been offered by cloud providers where boot time is not a concern, the PSP bottleneck has not been exacerbated like it is in the context of microVMs. We believe that the PSP bottleneck must be solved for SEV to be viable for a serverless use case, and plan to address it in future work. The key challenge here is that attestation must be rooted by the PSP to guarantee integrity. A possible near term approach could be to lessen the burden on the PSP by allowing multiple VMs to share encryption keys, which weakens the trust model, but is also relevant for warm boot, and discussed more in the context of related work (Section 8).

6.3 Memory Footprint

Since it is important for microVMs to maintain small memory footprints, we consider the memory usage of SEV microVMs. First, our modifications to Firecracker only increase the binary size by about 50K, for a total size of about 4.2MB. We then measured memory usage of a running VM with the same technique used in Firecracker [7], running `pmap` on a running instance and subtracting the Firecracker executable size plus the size of guest memory. SEV microVMs only add about 16K to the total memory usage over a non-SEV guest meaning the number of guests that can run concurrently with our design is roughly the same as the number of stock Firecracker VMs that can run on one machine.

7 Discussion

We discuss the challenges associated with warm start for confidential VMs, and SEVeriFast's application to other TEEs.

7.1 Cold vs. Warm Start

We acknowledge that adding SEVeriFast to Firecracker increases boot times over non-confidential VMs, but we hope that SEVeriFast's contributions will be used as a basis to investigate warm start. Even with a warm start solution for SEV, SEVeriFast's cold start optimizations are valuable. Prior work shows that cold starts make up a significant portion of total invocations and remain expensive [29, 39]. An obvious approach to warm start is to use keep-alive windows for SEV VMs and allow previously attested state to be reused by the same guest owner. While this approach would be functionally correct, it would result in high memory consumption because pages cannot be deduplicated as they are in other systems [17, 38]. Deduplication is challenging with SEV because pages with identical contents at different physical addresses will have different ciphertext, and if the host changes a guest page mapping the RMP check will fail on the next access to that page. Similarly, loading snapshot images is a challenge in a confidential context because existing solutions rely on pre-fetching [44], lazy loading [20], or sharing deduplicated snapshot blocks [16]. All of these solutions would require communication between the host and the guest during launch and/or runtime to verify pages as they are lazily mapped in to the guest. Verifying these pages would also require the guest to know its own snapshotted state which may not always be feasible.

Plug-in-enclaves [32] exploit the fact that previously attested state like shared libraries can be securely re-used and mapped into a new enclave to improve initialization time of serverless applications. Their design allows previously attested state to be mapped into new serverless instances and shared to eliminate redundant application state for serverless enclaves. Plug-in-enclaves use Intel SGX [26] and require hardware modifications to allow inter-enclave sharing. Sharing previously attested state is a promising avenue toward warm start for SEV guests, and understanding the associated challenges requires future work.

There is significant work to be done toward warm start for confidential microVMs, and we see optimizing cold start as the first step toward a complete confidential serverless system.

7.2 Generalization to other TEEs

SGX. SEVeriFast's design principle to minimize the amount of data to be hashed by the hardware implicitly relies on the fact that SEV does not require every page of guest memory to be measured during boot. SGX 2 [46] allows enclaves to add pages after enclave creation, but each page must be explicitly measured when added to an enclave. SEVeriFast relies on the fact that an SEV guest can simply copy boot components from plain text to encrypted pages without needing to hash pages in hardware. Additionally, SGX has a limited amount of memory reserved for enclave pages, while SEV has no limit on encrypted pages. Therefore SEVeriFast is not trivially applicable to SGX.

TDX. While TDX is built on top of SGX, the launch process for TDX VMs is similar to that of SEV [3]. TDX and SEV both rely on a launch measurement to establish trust with selective measurement of initial pages. We have not yet had the opportunity to explore the costs and trade-offs associated with TDX but because the launch processes for SEV and TDX are quite similar we believe that SEVeriFast's design principles could be applied to TDX as well.

8 Related Work

We discuss other efforts to combine TEEs with serverless and work that improves the performance of applications running in TEEs.

Trusted Serverless. Feng et al. design a system for scalable enclave memory protection based on the open source RISC-V PENGLAI enclave [4] that meets the demands of serverless [21]. Most notably for our design, they identify that measurement calculation (i.e., pre-encryption) is the major cost in enclave initialization. They propose *shadow enclaves* that are pre-warmed and forked to create subsequent enclaves with the insight that previously attested state can be used to bypass the cost of measurement during boot. Shadow enclaves present possible design considerations in approaches to warm boot and addressing the PSP bottleneck from Section 6. Challenges include memory sharing in a TEE with SEV and the implications of sharing keys with multiple VMs.

Gu et al. leverage SGX and Intel Memory Protection Keys (MPK) to provide isolation between multiple distrusted applications running inside a single SGX enclave, called *light-enclaves*, to reduce the TCB within an enclave [23]. They evaluate the execution time of serverless functions within light-enclaves and find similar performance between light-enclaves and launching functions in pre-warmed enclaves with lower resource consumption. They note that light-enclaves allow 16 isolated serverless functions to run in one hardware enclave rather than one function per hardware enclave without MPK, but because SGX is still limited by fixed processor reserved memory and 16 functions per-enclave, SEV remains more flexible for a serverless use case.

Fast Startup for Serverless. REAP [44] makes the observation that multiple invocations of the same function access the same working set of pages, and pre-fetching the working set from disk can greatly reduce boot times of VMs started from snapshots. Catalyzer [20] uses copy-on-write to reuse state from an already running VM to further reduce startup times of similar function invocations. It also introduces a new OS primitive, *sandbox-fork*, to fork new serverless functions from a warm instance. FAASM [40] uses shared memory between functions to reduce initialization times. While sharing memory and reusing state from warm VMs could be applied to SEV guests, it would require multiple VMs to share the same memory encryption key which complicates the trust model.

In-monitor KASLR [24] makes the observation that the security benefits of KASLR can be enjoyed by serverless applications with low overhead by giving the VMM control over randomization. Like other assumptions about direct boot, SEVeriFast breaks in-monitor KASLR. In general, TEEs reduce the benefits of moving computation from the guest to the host.

Performance of TEE Protected Applications. Efforts to improve enclave performance have mainly focused on SGX [12, 13, 32, 41]. SCONE [12] provides a library to secure Docker containers in SGX enclaves to run unmodified Linux applications with low overhead. Graphene-SGX [41] is a fully-featured library OS built to run unmodified user applications in SGX enclaves with minimal overhead.

Similar to our boot verifier, td-shim[10] provides a thin layer between the VMM and the guest workload to initialize an enclave for Intel TDX [3]. Td-shim shares motivation with our work insofar as td-shim is built to load a payload with no firmware dependencies, which one could choose to be a microVM-ready Linux kernel, to minimize overall boot time. However, td-shim measures its payload with the TDX equivalent of pre-encryption which our work has shown to have a large impact on boot time with SEV. Td-shim also supports loading multiple payload types and more functionality, like a heap allocator, building and installing ACPI tables, and an event logger to name a few. There has also been talk about adding support for SEV to td-shim[11]. However, adding generality and quality-of-life features like an allocator leads to a larger binary size and longer pre-encryption times, negatively affecting boot time.

9 Conclusion

Serverless platforms offer low latency, on-demand computation in the cloud but lack support for the confidentiality of TEEs offered by IaaS platforms today. We have identified that the state-of-the-art method of booting SEV guests is incompatible with the low latency expectations of serverless platforms, and designed SEVeriFast to provide efficient cold boot for SEV guests. With SEVeriFast we have demonstrated that optimizing the SEV boot process yields performance comparable to state-of-the-art microVMs. Furthermore, we identified a key bottleneck in the SEV hardware and the challenges related to mitigating it as future work. We believe the SEV-compatible fast cold boot achieved by SEVeriFast will provide an efficient and secure foundation for next generation secure serverless platforms.

References

[1] AMD SEV-SNP: Strengthening VM Isolation with Integrity Protection and More. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf. (Accessed on 2022-15-11).

[2] Docker. http://docs.docker.io/en/latest/.

[3] Intel® Trust Domain extensions. https://www.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf.

[4] Penglai enclave. https://github.com/Penglai-Enclave.

[5] Protecting VM Register State with SEV-ES. https://www.amd.com/content/dam/amd/en/documents/epyc-business-docs/white-papers/Protecting-VM-Register-State-with-SEV-ES.pdf. (Accessed on 2022-15-11).

[10] https://github.com/confidential-containers/td-shim/

[11] https://github.com/confidential-containers/td-shim/issues/135

[6] SEV Secure Nested Paging Firmware ABI Specification. https://www.amd.com/system/files/TechDocs/56860.pdf. (Accessed on 2022-15-11).

[7] Alexandru Agache, Marc Brooker, Alexandra Iordache, Anthony Liguori, Rolf Neugebauer, Phil Piwonka, and Diana-Maria Popa. Firecracker: Lightweight virtualization for serverless applications. In *USENIX Symposium on Networked Systems Design and Implementation*, Santa Clara, CA, February 2020.

[8] Istemi Ekin Akkus, Ruichuan Chen, Ivica Rimac, Manuel Stein, Klaus Satzke, Andre Beck, Paarijaat Aditya, and Volker Hilt. SAND: Towards High-Performance Serverless Computing. In *2018 USENIX Annual Technical Conference (USENIX ATC 18)*, Boston, MA, July 2018. USENIX Association.

[9] Amazon Web Services. Aws lambda. https://aws.amazon.com/lambda/. Accessed on 2022-12-05.

[10] Advanced Micro Devices (AMD). AMD secure encrypted virtualization (SEV). https://developer.amd.com/sev/, Dec 2022.

[11] Lixiang Ao, George Porter, and Geoffrey M. Voelker. FaaSnap: FaaS Made Fast Using Snapshot-Based VMs. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys '22, page 730–746, New York, NY, USA, 2022. Association for Computing Machinery.

[12] Sergei Arnautov, Bohdan Trach, Franz Gregor, Thomas Knauth, Andre Martin, Christian Priebe, Joshua Lind, Divya Muthukumaran, Dan O'Keeffe, Mark L. Stillwell, David Goltzsche, David Eyers, Rüdiger Kapitza, Peter Pietzuch, and Christof Fetzer. SCONE: Secure Linux Containers with Intel SGX. In *Proceedings of the 12th USENIX Conference on Operating Systems Design and Implementation*, OSDI'16, page 689–703, USA, 2016. USENIX Association.

[13] Andrew Baumann, Marcus Peinado, and Galen Hunt. Shielding Applications from an Untrusted Cloud with Haven. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, page 267–283, USA, 2014. USENIX Association.

[14] Stefan Brenner, Colin Wulf, David Goltzsche, Nico Weichbrodt, Matthias Lorenz, Christof Fetzer, Peter Pietzuch, and Rüdiger Kapitza. SecureKeeper: Confidential ZooKeeper Using Intel SGX. In *Proceedings of the 17th International Middleware Conference*, Middleware '16, New York, NY, USA, 2016. Association for Computing Machinery.

[15] Marc Brooker, Mike Danilov, Chris Greenwood, and Phil Piwonka. On-demand container loading in AWS lambda. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 315–328, Boston, MA, July 2023. USENIX Association.

[16] Marc Brooker, Mike Danilov, Chris Greenwood, and Phil Piwonka. On-demand Container Loading in AWS Lambda. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*, pages 315–328, Boston, MA, July 2023. USENIX Association.

[17] James Cadden, Thomas Unger, Yara Awad, Han Dong, Orran Krieger, and Jonathan Appavoo. SEUSS: skip redundant paths to make serverless fast. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys'20, pages 1–15, New York, NY, USA, 2020. Association for Computing Machinery.

[18] Linux Containers. https://linuxcontainers.org/.

[19] Ankur Dave, Chester Leung, Raluca Ada Popa, Joseph E. Gonzalez, and Ion Stoica. Oblivious Coopetitive Analytics Using Hardware Enclaves. In *Proceedings of the Fifteenth European Conference on Computer Systems*, EuroSys '20, New York, NY, USA, 2020. Association for Computing Machinery.

[20] Dong Du, Tianyi Yu, Yubin Xia, Binyu Zang, Guanglu Yan, Chenggang Qin, Qixuan Wu, and Haibo Chen. Catalyzer: Sub-Millisecond Startup for Serverless Computing with Initialization-Less Booting. ASPLOS '20, page 467–481, 2020.

[21] Erhu Feng, Xu Lu, Dong Du, Bicheng Yang, and Xueqiang Jiang. Scalable Memory Protection in the PENGLAI Enclave. In *15th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, pages 275–294. USENIX Association, July 2021.

[22] Apache Software Foundation. Apache openwhisk: Open source serverless cloud platform. http://openwhisk.apache.org/. Accessed on 2021-01-04.

[23] Jinyu Gu, Bojun Zhu, Mingyu Li, Wentai Li, Yubin Xia, and Haibo Chen. A Hardware-Software Co-design for Efficient Intra-Enclave Isolation. In *31st USENIX Security Symposium (USENIX Security 22)*, pages 3129–3145, Boston, MA, August 2022. USENIX Association.

[24] Benjamin Holmes, Jason Waterman, and Dan Williams. KASLR in the Age of MicroVMs. In *Proceedings of the Seventeenth European Conference on Computer Systems*, 2022.

[25] The International Business Machines Corporation (IBM). IBM Cloud Functions. https://www.ibm.com/cloud/functions. Accessed on 2021-01-04.

[26] Intel. Intel® Software Guard extensions (Intel® SGX). https://www.intel.com/content/www/us/en/developer/videos/intel-software-guard-extensions-sgx.html?wapkw=intel+sgx.

[27] Intel Cloud Hypervisor. https://www.cloudhypervisor.org/. (Accessed on 2023-01-10).

[28] Intel NEMU: Modern Hypervisor for the Cloud. https://github.com/intel/nemu.

[29] Artjom Joosen, Ahmen Hassan, Martin Asenov, Rajkarn Singh, Luke Darlow, Jianfeng Wang, and Adam Barker. How Does It Function? Characterizing Long-term Trends in Production Serverless Workloads. In *Proceedings of the 2023 ACM Symposium on Cloud Computing*, SoCC'23, New York, NY, USA, 2023. Association for Computing Machinery.

[30] Kata Containers: The speed of containers, the security of VMs. https://katacontainers.io/. Accessed on 2022-01-04.

[31] Hsuan-Chi Kuo, Dan Williams, Ricardo Koller, and Sibin Mohan. A Linux in Unikernel Clothing. In *Proceedings of the Fifteenth European Conference on Computer Systems*, Heraklion, Greece, April 2020.

[32] Mingyu Li, Yubin Xia, and Haibo Chen. Confidential Serverless Made Efficient with Plug-In Enclaves. In *2021 ACM/IEEE 48th Annual International Symposium on Computer Architecture (ISCA)*, pages 306–318, 2021.

[33] Filipe Manco, Costin Lupu, Florian Schmidt, Jose Mendes, Simon Kuenzer, Sumit Sati, Kenichi Yasukata, Costin Raiciu, and Felipe Huici. My VM is Lighter (and Safer) than your Container. In *The 26th ACM Symposium on Operating Systems Principles*, Shanghai, China, October 2017.

[34] Microsoft. Azure Functions Serverless Compute. https://azure.microsoft.com/en-us/services/functions/. Accessed on 2021-01-04.

[35] Ming Mao and Marty Humphrey. A Performance Study on the VM Startup Time in the Cloud. In *2012 IEEE Fifth International Conference on Cloud Computing*, pages 423–430, 2012.

[36] Dov Murik and Hubertus Franke. Securing Linux VM boot with AMD SEV measurement. https://kvmforum2021.sched.com/event/ke4h/securing-linux-vm-boot-with-amd-sev-measurement-dov-murik-hubertus-franke-ibm-research.

[37] Edward Oakes, Leon Yang, Dennis Zhou, Kevin Houck, Tyler Harter, Andrea C. Arpaci-Dusseau, and Remzi H. Arpaci-Dusseau. SOCK: Rapid Task Provisioning with Serverless-Optimized Containers. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, page 57–69, USA, 2018. USENIX Association.

[38] Divyanshu Saxena, Tao Ji, Arjun Singhvi, Junaid Khalid, and Aditya Akella. Memory deduplication for serverless computing with Medes. In *Proceedings of the Seventeenth European Conference on Computer Systems*, EuroSys'22, pages 714–729, New York, NY, USA, 2022. Association for Computing Machinery.

[39] Mohammad Shahrad, Rodrigo Fonseca, Í nigo Goiri, Gohar Chaudhry, Paul Batum, Jason Cooke, Eduardo Laureano, Colby Tresness, Mark

Russinovich, and Ricardo Bianchini. Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC'20, USA, 2020. USENIX Association.

[40] Simon Shillaker and Peter Pietzuch. FAASM: Lightweight Isolation for Efficient Stateful Serverless Computing. In *Proceedings of the 2020 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC'20, USA, 2020. USENIX Association.

[41] Chia-Che Tsai, Donald E. Porter, and Mona Vij. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *Proceedings of the 2017 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '17, page 645–658, USA, 2017. USENIX Association.

[42] Chia-Che Tsai, Jeongseok Son, Bhushan Jain, John McAvey, Raluca Ada Popa, and Donald E. Porter. Civet: An Efficient Java Partitioning Framework for Hardware Enclaves. In *Proceedings of the 29th USENIX Conference on Security Symposium*, SEC'20, USA, 2020. USENIX Association.

[43] Unified Extensible Firmware Interface. https://uefi.org/sites/default/files/resources/PI_Spec_1_6.pdf.

[44] Dmitrii Ustiugov, Plamen Petrov, Marios Kogias, Edouard Bugnion, and Boris Grot. Benchmarking, Analysis, and Optimization of Serverless Function Snapshots. In *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, ASPLOS '21, page 559–572, New York, NY, USA, 2021. Association for Computing Machinery.

[45] Liang Wang, Mengyuan Li, Yinqian Zhang, Thomas Ristenpart, and Michael Swift. Peeking behind the Curtains of Serverless Platforms. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, page 133–145, USA, 2018. USENIX Association.

[46] Bin (Cedric) Xing, Mark Shanahan, and Rebekah Leslie-Hurd. Intel® Software Guard Extensions (Intel® SGX) Software Support for Dynamic Memory Allocation inside an Enclave. In *Proceedings of the 5th International Workshop on Hardware and Architectural Support for Security and Privacy (HASP'16)*. Association for Computing Machinery, 2016.

[47] Yanqi Zhang, Íñigo Goiri, Gohar Irfan Chaudhry, Rodrigo Fonseca, Sameh Elnikety, Christina Delimitrou, and Ricardo Bianchini. Faster and cheaper serverless computing on harvested resources. In *Proceedings of the ACM SIGOPS 28th Symposium on Operating Systems Principles*, SOSP '21, page 724–739, New York, NY, USA, 2021. Association for Computing Machinery.