Confidential Computing for OpenPOWER

Guerney D. H. Hunt*, Ramachandra Pai†, Michael V. Le*, Hani Jamjoom*, Sukadev Bhattiprolu†, Rick Boivie*, Laurent Dufour†, Brad Frey†, Mohit Kapur*, Kenneth A. Goldman*, Ryan Grimm†, Janani Janakirman†, John M. Ludden†, Paul Mackerras†, Cathy May†, Elaine R. Palmer*, Bharata Bhasker Rao†, Lawrence Roy*,‡, William A. Starke†, Jeff Stuecheli†,Enriquillo Valdez*, Wendel Voigt†

*IBM Research †IBM ‡Oregon State University, USA

Abstract

This paper presents Protected Execution Facility (PEF), a virtual machine-based Trusted Execution Environment (TEE) for confidential computing on Power ISA. PEF enables protected secure virtual machines (SVMs). Like other TEEs, PEF verifies the SVM prior to execution. PEF utilizes a Trusted Platform Module (TPM), secure boot, and trusted boot as well as newly introduced architectural changes for Power ISA systems. Exploiting these architectural changes requires new firmware, the Protected Execution Ultravisor. PEF is supported in the latest version of the POWER9 chip. PEF demonstrates that access control for isolation and cryptography for confidentiality is an effective approach to confidential computing. We particularly focus on how our design (i) balances between access control and cryptography, (ii) maximizes the use of existing security components, and (iii) simplifies the management of the SVM life cycle. Finally, we evaluate the performance of SVMs in comparison to normal virtual machines on OpenPOWER systems.

CCS Concepts: • **Security and privacy → Trusted computing**; • **Software and its engineering →** *Virtual machines.*

Keywords: TEE, Trusted Execution Environment, POWER9, confidential computing, secure computing, enclave, Linux, KVM, ultravisor, firmware

ACM Reference Format:
Guerney D. H. Hunt*, Ramachandra Pai†, Michael V. Le*, Hani Jamjoom*, Sukadev Bhattiprolu†, Rick Boivie*, Laurent Dufour†, Brad Frey†, Mohit Kapur*, Kenneth A. Goldman*, Ryan Grimm†, Janani Janakirman†, John M. Ludden†, Paul Mackerras†, Cathy May†, Elaine R. Palmer*, Bharata Bhasker Rao†, Lawrence Roy*,‡, William A. Starke†, Jeff Stuecheli†,Enriquillo Valdez*, Wendel Voigt†. 2021. Confidential Computing for OpenPOWER. In *Sixteenth European Conference on Computer Systems (EuroSys '21), April 26–28, 2021, Online, United Kingdom.* ACM, New York, NY, USA, 17 pages. https://doi.org/10.1145/3447786.3456243

EuroSys '21, April 26–28, 2021, Online, United Kingdom

ACM ISBN 978-1-4503-8334-9/21/04…$15.00
https://doi.org/10.1145/3447786.3456243

1 Introduction

Confidential computing enables users to compute without exposing their application or data to the operator of the underlying infrastructure. Logically, confidential computing creates a new—orthogonal—privileged domain that is separate from the one controlled by the OS or Hypervisor. This privileged domain, commonly called a Trusted Execution Environment (TEE), is rooted in hardware and firmware; through specific instructions, control is transferred to the TEE. TEEs solve a critical concern inhibiting the adoption of cloud computing, especially for regulated industries (e.g., financial services and healthcare) by enabling cloud users to run their critical applications without fully trusting the operator of the cloud.

Currently, there are several commercially available architectures supporting TEEs. They include Intel Secure Guard Extensions (SGX), AMD Secure Encrypted Virtualization (SEV), ARM TrustZone, and IBM Z Secure Execution. While there has been significant progress towards making TEEs widely available, we observe three key challenges that hinder their broad adoption. First is balancing between isolation and confidentiality on server class processors. Existing server-class processors link their isolation and confidentiality, potentially making the use of TEEs a more expensive operation. Second is reusing existing security technologies. Although all TEEs reuse cryptographic algorithms, current TEE implementations rely on the introduction of many new security components. These new components require either large changes to applications (e.g., Intel SGX) or greater trust in the new components (e.g., AMD SEV). Third, all TEEs must address managing the life cycle of the secured entities (both virtual machines and processes), especially in cloud deployments. Life cycle management of secure entities that need to run inside the TEE is cumbersome. Such complexities arise from how TEEs are attested for authenticity, how secrets are provisioned, and how TEEs are integrated into the typical build-deploy cloud methodology.

This paper introduces the design and implementation of a TEE for Power ISA, called Protected Execution Facility (PEF). Power ISA is the architecture for IBM's POWER[1] family of processors. We provide a detailed overview of the major design changes made in the latest version of the POWER9 chip, firmware, and operating system. We particularly focus on the design aspects that helped address the challenges outlined above. We also focus on how to create a TEE with a simple, straightforward and, most importantly, open design that carefully partitions the responsibilities between hardware and firmware to reduce bugs and simplify fixes.

PEF provides a virtual machine (VM) based TEE exploitable by members of the OpenPOWER foundation, a consortium of vendors who build systems based on POWER processors. Earlier research [12] indicated that it is easier to secure the VM-to-hypervisor interface than the process-to-OS interface. PEF introduces the Protected Execution Ultravisor (or Ultravisor), new firmware that manages hardware-enforced access control. The firmware runs in ultravisor state which is more privileged than hypervisor state and has exclusive control over security-sensitive features. This design partitions the security responsibility between hardware and cooperating firmware. This approach also creates a smaller Trusted Computing Base (TCB) than traditional virtualized environments.

Our approach does not currently link the isolation (or access control) with confidentiality and integrity. This enables us to embed sensitive information in the secure VM (SVM) at creation. Our access control approach allows us to protect the hypervisor and all normal virtual machines (NVMs) from each of the SVMs as well as protecting each SVM from other SVMs, normal VMs, and the hypervisor. This bidirectional protection is absent for some TEEs [13, 14, 65].

In the context of cloud integration, managing the life cycle of an SVM running in our TEE is more complex than managing the life cycle of an NVM. Our approach allows the creator of the SVM to incorporate information that enables the SVM to execute on the target system without requiring a runtime attestation with the processor vendor. Our use of encryption based integrity and confidentiality enables secrets to be incorporated into the SVM. The ability to directly embed secrets in the SVM simplifies the protocol exchange required to start the SVM.

In this paper we present the design and performance evaluation of a TEE for OpenPOWER based on access control for isolation and cryptography for confidentiality and integrity. We demonstrate how existing security technologies can integrate into a TEE design that balances between isolation and cryptography. The design is simpler because it exploits the Trusted Platform Module (TPM) [7, 28], secure and trusted boot, and existing features of Linux. If previously existing errors are discovered, we can leverage the corrections created in the wider community. In a similar fashion, we also describe how leveraging existing security technologies and minimizing the introduction of new ones can ease the integration into cloud infrastructures. Finally, we show excellent performance for computation running inside of the TEE with modest overhead for I/O operations.

Section 2 of this paper discusses the design objectives of PEF. Section 3 describes the key design and implementation of PEF and how we address the challenges outlined above. Section 4 presents an evaluation of our approach. Section 5 discusses the limitations and extensions of our current approach. Section 6 reviews related work and Section 7 concludes the paper.

2 Security Model and Design Goals

Users of a confidential computing platform must be assured that the system their computation runs on is *legitimate* and their computation is *valid*. The target system is legitimate if it is produced by a trusted vendor and has no unauthorized modifications. The computation is valid if it has not been modified by an unauthorized party and cannot be executed on an unauthorized system. In addition, as the name implies, users of confidential computing want to know that the information within their computation remains confidential, implying that no unauthorized party can gain access to or extract information from their computation while it is at rest, being moved from one computer to another, or executing on a target platform.

Assumed Platform. Our design assumes that the platform could be a dedicated system, a shared system, or part of a cloud infrastructure. Similar to other confidential computing platforms, our design assumes that the user has a trusted (uncompromised) platform where his/her secure computation is created. The trusted platform could be independent from the platform where the secure computation will run.

Threat Model. Our threat model assumes that the adversary has limited physical access to the computing platform. By limited physical access, we exclude from the threat model physically modifying, probing, or monitoring the system. Our objective is that when the adversary is an operator or administrator of the platform, has compromised the hypervisor, or is a user of the computing platform, he/she cannot extract any useful information from SVMs. We allow the attacker with limited physical access to do anything to the system that a legitimate trusted party would be able to do, even though the purpose might be nefarious. An attacker may attempt to modify any firmware or software component of the computing platform including the hypervisor to accomplish his goals. We exclude RF-type side channels from our threat model. While the prevention of side channels is critical in the overall confidential computing space, it would require a separate and dedicated analysis and evaluation. Instead, in this paper, we focus our discussion on the design,

[1] POWER stands for Performance Optimization With Enhanced RISC.

operational challenges, and performance characteristics of our platform.

Goals. We targeted the following goals through a combination of software and hardware mechanisms. The design should prevent exposing potentially sensitive state from SVMs to the hypervisor or other SVMs and NVMs on the platform. It should also allow users to verify that the underlying TEE is valid, a critical step in enabling confidential computing. Finally, the amount of new firmware should be as small as possible to increase the likelihood that properties of the Ultravisor could be formally proven, potentially resulting in a higher security classification.

PEF does not provide protection for I/O operations from the confidential computation. PEF assumes that the confidential computation will encrypt any data that it wishes to pass through an untrusted hypervisor. For example, we are assuming that Transport Layer Security (TLS) will be used to protect all network communications. Similarly, we chose to rely on Linux encrypted file systems. As of this writing, there is no widely-accepted Linux encrypted file system that protects against all possible hypervisor attacks. Nevertheless, our design provides a basis upon which I/O protection can be achieved.

Our goal is to support confidential computing on a general-purpose computing platform, namely OpenPOWER. This creates additional constraints not present for the excellent work demonstrated in clean slate[2] academic designs [19, 25, 53, 55, 57, 68, 71]. Specifically, all hardware and firmware changes had to fit within the existing product development cycles. The design needed to minimize the impact of introduced changes to existing customers, in particular to NVMs. We chose to reuse the Trusted Platform Module (TPM), TPM Support Services (TSS), secure and trusted boot, and Linux encrypted file systems. This choice minimizes the introduction of new technology, thereby, reducing the likelihood of introducing new errors. Our initial target is a Linux based hypervisor on OpenPOWER hardware.[3] We strove to minimize the proposed Linux changes so that they could be more easily adopted by the open source community.

3 Protected Execution Facility (PEF)

All TEEs provide isolation, confidentiality, and integrity protections. *Isolation* is defined as separating computation from other computations including support software, *confidentiality* as keeping the contents of the TEE secret, and *integrity* as detecting and/or preventing attempts to tamper with and/or modify the TEE. In Subsection 3.1, we present our approach for providing isolation, confidentiality and integrity. In Subsection 3.2, we describe how we verify the integrity of the hardware and SVMs, and discuss our exploitation of existing technologies. In Subsection 3.3, we explore how this approach addresses life cycle management of SVMs. Finally, in Subsection 3.4, we present an approach for integrating PEF into cloud infrastructures.

3.1 Ensuring Isolation, Confidentiality & Integrity

PEF uses hardware-enforced access control for isolation and cryptography for confidentiality and integrity. The design maintains the function and purpose of the hypervisor. Consequently, we introduced a new CPU state, *secure state*. We also introduced firmware to manage the new state, called the *Protected Execution Ultravisor* (or *Ultravisor*). The new firmware manages all security-related hardware functions in the processor/system. The architecture was also changed to restrict security-sensitive operations to ultravisor state. This simple approach has repercussions throughout the architecture as documented in the Request for Change (RFC) [31]. All of the changes are implemented in the latest version of the POWER9 processor [4, 27, 49, 64, 69]; the RFC has been integrated into the POWER architecture document [36]. Systems with the latest POWER9 processor are currently manufactured by IBM and OpenPOWER partners.

Secure Mode. In POWER architectures prior to the introduction of PEF, the Hypervisor (HV) and Problem (PR) bits [4] in the Machine State Register (MSR) defined three mutually exclusive states: problem (for applications), privileged non-hypervisor (for OSs) and hypervisor. Privilege determines what hardware facilities the software is allowed to use or access. The Hypervisor was the highest privileged state, it had full control of all hardware.

There are three primary changes to support PEF. First, we created a new most-privileged state using a newly defined bit in the MSR. Second, we partitioned memory into secure and normal memory. Finally, the system call instruction was modified by adding an additional level to create the ultracall. Ultracalls always go directly to the Ultravisor, and cannot be intercepted by the hypervisor or an OS. Figure 1 illustrates the impact of introducing the new state bit in the MSR. The bit is called the Secure bit, *S*. In normal state, either a VM or the hypervisor is executing; in secure state, either the Ultravisor or an SVM is executing. The Ultravisor controls the value of MSR_S for all processes in the system.

Ultravisor. Maintaining the isolation and security of the computation and associated data is the sole objective of the Ultravisor. System management continues to be the responsibility of the hypervisor. The hypervisor uses ultracalls to continue managing security sensitive facilities. Where required, the Ultravisor confirms that the action requested by the hypervisor will not affect the security of any running

[2] Some of this work uses an open source CPU design that is not yet widely deployed in production systems.

[3] The architecture of PEF can be extended to any POWER9 hypervisor.

[4] The Problem bit is how application state is indicated in POWER architecture. This nomenclature was common when the architecture was created in the 1980's.

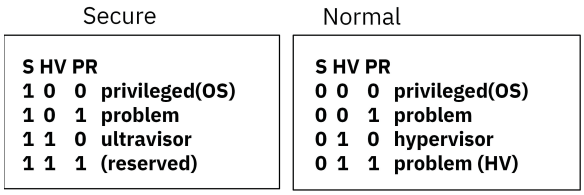

Figure 1. Processing states in the POWER architecture

SVM. In cases where the action could compromise an SVM, the Ultravisor returns without performing the action and does not indicate an error. The Ultravisor does not have a general role in platform management and does not address denial of service attacks.

As indicated in Table 1, the Ultravisor has 20 direct interfaces called ultracalls. The implementation also uses six new hypervisor calls. Three of these calls are related to starting, stopping and aborting SVMs. One is utilized to talk to the TPM. The final two calls support hypervisor dumps of secure memory and can be used for hypervisor paging of secure memory. To improve performance, our approach para-virtualizes the Linux/KVM hypervisor. In addition to the ultracalls and the new hypercalls, the Ultravisor supports the 152 existing hypercalls used by VMs running on top of Linux/KVM. The design requires the H_RANDOM call to be handled by the Ultravisor, but the current implementation reflects this call to the hypervisor, which is a security exposure. However, POWER9 includes a hardware instruction for getting a random number so the hypervisor call should not be utilized by SVMs.

In order to support hypervisor paging of NVMs and hypervisor dump of SVMs, the Ultravisor supports movement of secure page-content to insecure memory and back. It does encryption with integrity using GCM prior to allowing a page to be moved. When a page is accessible to the hypervisor, it is not accessible to the SVM. The Ultravisor removes access from the hypervisor, decrypts the memory, and checks integrity prior to allowing the SVM to access. Since dynamic paging by the hypervisor is not yet supported, this function is only used when the hypervisor dumps the contents of secure memory or the hypervisor tries to read the contents of secure memory.

When PEF is enabled, the hardware boot sequence is Hostboot[5] ⇒ OPAL [6] ⇒ Ultravisor ⇒ OPAL ⇒ host operating system. When the POWER9 system boots, OPAL sets a new random password that must be known to access the private key that is used to unlock a lock box. The password is necessary because the TPM is a shared resource. OPAL passes the password to the Ultravisor and forgets it; the password is only known by the TPM and the Ultravisor. This is the basis of the secure communications between the ultravisor and the TPM. A shared TPM requires a TSS [26] in the Ultravisor. The impact on binary code size of this decision is reduced because the Ultravisor TSS is created by a compile time option that eliminates functions not needed by Ultravisor. While the POWER9 is booting the TCB is Hostboot, OPAL, and the Ultravisor. After the Ultravisor is initialized, the TCB is the Ultravisor.

Table 1. Ultravisor interfaces

Interface	number
Ultracalls	20
New hypercalls	6
Existing hypercalls	152

Access Control. The access control mechanism in PEF is based on the assignment of VMs (also called partitions) to security domains [31]. The hypervisor is in one security domain in normal memory, along with all processes that run directly under the hypervisor and all VMs that do not take advantage of PEF. Each of the SVMs is assigned to its own security domain in secure memory. This approach provides bidirectional protection. The SVMs are protected from the hypervisor and one another, and the hypervisor's security domain is protected from all of the SVMs.

One component of integrity is preventing the hypervisor from corrupting the SVM's or Ultravisor's memory. This is achieved with our access control approach by preventing code running in normal state from accessing secure memory.

Secure Memory. Architecturally, secure memory is identified by $\texttt{MEM}_{\texttt{SM}}$, which represents a single bit of information: either memory is secure, $\texttt{MEM}_{\texttt{SM}}\texttt{=1}$ or it is normal $\texttt{MEM}_{\texttt{SM}}\texttt{=0}$. In POWER9, secure memory is implemented using a high order address bit[7], RA(15)[8]. The hardware enforces access control for code by only allowing processes running in secure state to access secure memory. Addresses with RA(15) set are secure; all other addresses are insecure. Any non-secure process or component generating an address with this bit set will cause an exception.

Other components of the platform outside of the processor, e.g., interrupt controllers, PCI bridges, accelerators, etc., were analyzed to determine their security sensitivity. Those subsystems that could not pass the security analysis are prevented from accessing secure memory. If a component was excluded from accessing secure memory, its control remains fully with the hypervisor. Those components that affect the

[5]The first firmware loaded to initialize the hardware.

[6]OpenPOWER Abstraction Layer(OPAL) is the firmware that completes initialization for OpenPOWER. OPAL is also called `skiboot`. This firmware has a component that provides hardware related services to the OS after it is booted.

[7]The amount of memory that will be secure memory is a configuration option that can only be changed by rebooting the machine.

[8]In POWER architecture address bits are labeled left to right, the most significant address bit is bit zero.

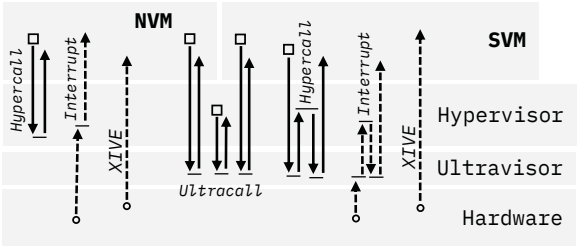

Figure 2. Ultracall, hypercall, and interrupt handling in PEF

security of a VM running in secure state were made accessible only to the Ultravisor.

The Ultravisor controls the Partition Table Control Register (PTCR), which points to the Partition Table, which is in secure memory controlled by the Ultravisor[9]. Each Partition Table Entry points to the partition-scoped page table for the partition. Each partition-scoped page table specifies the mapping to real storage for the partition. The Ultravisor manages the Partition Table entries and partition-scoped page tables for SVMs. Ultracalls have been created which the hypervisor uses to manage the Partition Table entries for NVMs. This allows the Ultravisor to monitor requests to change entries in this table.

The Ultravisor isolates SVMs from each other, from NVMs, and from the hypervisor by keeping the SVMs' partition-scoped page tables and all but their shared data in secure memory and managing the tables so that the SVM memory footprints do not overlap. SVMs may request a normal memory allocation to share data with the hypervisor.

Ultravisor Calls, Hypervisor Calls and Asynchronous Interrupts. All interrupts not directly presented to the SVM that occur in secure state are by default delivered to the Ultravisor. Figure 2 illustrates that in normal state, hypervisor calls go to the hypervisor and asynchronous interrupts are delivered to either the hypervisor or the NVM. In secure state, all hypervisor calls are delivered to the Ultravisor and asynchronous interrupts are delivered to either the SVM or to the Ultravisor. Ultracalls always go the Ultravisor no matter which state the system is running in.

To eliminate side channels caused by leaking state during interrupts, for each hypercall, the Ultravisor saves all of the state of the SVM. It clears all state that is not required for the hypercall, in some cases replacing it with dummy state. The hypercall is reflected to the hypervisor. The return from the hypervisor will come back to the Ultravisor which will return the results to the SVM and restore all other state.

Maintaining Confidentiality. The Ultravisor protects the confidentiality of the SVM when the hypervisor is paging the SVM or dumping the SVM. When data from secure memory

[9]The Partition Table is the only exception to the rule that only processes running in secure state can access secure memory; for purposes of address translation, all processes can access the Partition Table.

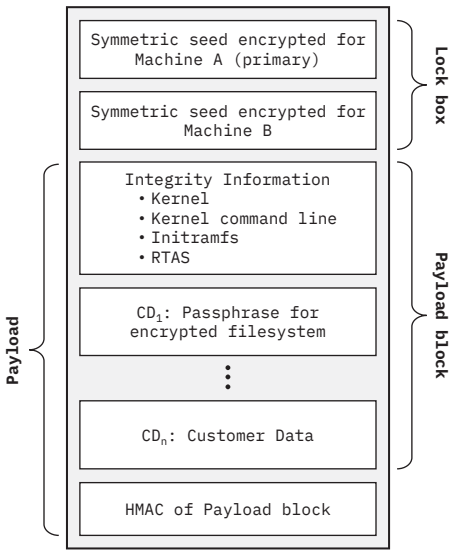

Figure 3. Layout of the ESM operand

are made available to software that is not running in secure state (e.g., the hypervisor), the Ultravisor is responsible for synchronizing the access to the data. Prior to making the data available, the SVM page table entries (PTEs) used to access the data in secure memory are marked invalid and the corresponding TLB entries invalidated by the Ultravisor. The data are then encrypted by the Ultravisor and copied to ordinary memory. Finally, the PTEs that will be used to access the data in ordinary memory are marked valid.

3.2 Verifying the Integrity of Firmware and SVM

For an SVM to be securely started, the target platform must be verified, the SVM must be authorized to run on the platform, and the SVM integrity must be verified. Verifying the platform means determining that it is trusted by the creator of the SVM. The integrity of the SVM must be verified because all SVMs start execution as an NVM. An SVM has integrity if it has not been modified by any unauthorized party and all of the initial parameters are what was specified by the creator. Note that the SVM disk is protected by the Linux encrypted file system selected by the creator. If the platform is verified, the SVM is authorized to run, and the integrity of the SVM is verified, then the SVM can be securely run on the target platform.

All of the required verification is completed as part of the enter secure mode (`ESM`) ultracall (request to transition into an SVM). The first step of the `ESM` ultracall is to copy into secure memory all of the memory associated with the NVM requesting the transition, so that the state cannot be modified after verification prior to execution. If the platform does not verify or the SVM is not authorized to run on the selected target, the Ultravisor will not have access to the symmetric seed required to verify the integrity of the SVM.

Platform Verification. Platform verification requires using the TPM and involves two components, access to the symmetric seed protecting the ESM operand (Figure 3) of the ESM ultracall and verification that the firmware is in the correct state. For the Ultravisor to access the symmetric seed, it must be wrapped in a public key that is associated with a private key in the TPM of the target system. The wrapped symmetric seed is called a *lockbox*.

If there is no lockbox for this system, then the SVM is not authorized to run on the system and the ESM ultracall fails. The TPM is used to extract the symmetric seed from the lockbox. If the TPM will not extract the symmetric seed for the Ultravisor, then there is a problem with the configuration of the hardware or firmware and the ESM ultracall fails. Verifying that the firmware is in the correct state means that the firmware is trusted, the hardware is booting with secure boot enabled, and PEF is enabled on the platform. The state of the hardware is verified through the TPM platform configuration register (PCR) 6 value. This description[10] assumes that the security model allows one symmetric seed per SVM and that this seed is valid on all target systems.

Local Attestation. Local attestation uses the information contained in the ESM operand (illustrated in Figure 3) to verify the integrity of the SVM. If the Ultravisor has access to the symmetric seed, then it will generate an HMAC key and a symmetric key. The HMAC key is used to verify the integrity of the ESM operand payload block. If the integrity check passes, the Ultravisor will use the symmetric key to extract the integrity information from the payload block. This information is used to verify the integrity of the parts of the SVM that could have been tampered with, specifically, the kernel, kernel command line, `initramfs`, and the RTAS area. If these checks pass, the NVM will resume execution as an SVM. The failure of any of these checks causes the request to transition into an SVM to fail, which causes the the NVM to be removed from secure memory and terminated.

Utilizing the TPM. The TPM API supports a secure tunnel, permitting an application to communicate with a TPM through an untrusted path. The tunnel supports parameter encryption and an integrity MAC for both commands and responses, plus anti-replay protection. Applications typically interface to the TPM through a TSS [26], which handles the encryption, MAC and serialization. The Ultravisor exploits the TPM API to establish a secure tunnel through the hypervisor. The Ultravisor uses the TPM to acquire the symmetric seed for the ESM operand associated with the ESM ultracall. The ultravisor reflects a newly added hypervisor call to KVM when it needs to utilize the TPM.

Generating a Lockbox. The SVM owner must authenticate the TPM storage key because the public part will be used to wrap the symmetric seed. Each target machine must be enrolled once. This involves extracting the TPM platform certificate and the storage key and authenticating that the system is from a trusted OpenPOWER vendor.

The *hardware key hash* is a hash of the public keys used to sign the firmware. The owner gets the hardware key hash from the target machine. They validate the hardware key hash against a manifest from a trusted vendor[11].

The tooling that creates an SVM must be supplied with an NVM, the storage key public key for the target system, a policy, and the trusted PCR 6 value(s). PCR 6 represents secure boot validation code, an accepted hardware key hash, secure boot enabled, and PEF enabled. The tooling randomly generates a symmetric seed and a password. Next, it utilizes a virtual or real TPM to run a `TPM duplicate`, which wraps the symmetric seed with the public key and the supplied policy. To avoid a security exposure, this policy should require the PCR 6 value to match[12] and only allow duplication with the password. The password is discarded after duplication. The duplicate returns the lockbox. After creation, the lockbox is inserted into the ESM operand.

It is important to note two features of this design. First, although the tooling that creates an SVM creates a lockbox as part of creating the SVM, a lockbox can also be inserted (or overwritten) into an SVM just prior to execution. The second point is that the protected symmetric seed cannot be duplicated because the required password is always discarded. The creator of the SVM specified which value(s) of PCR 6 (illustrated in figure 4) are acceptable. If any of the values associated with generating PCR 6 changes, a previously authorized SVM will not run[13]. As previously illustrated, SVMs can have more than one lockbox, which authorizes them for more than one machine.

Verifying Hardware State. Our design relies on secure and trusted boot of the firmware stack [17, 45–47, 72, 73]. Only firmware signed by the owner as represented by the hardware key hash will be loaded into the hardware. Secure boot verifies that the firmware on the target system is valid. Trusted boot records information so that the rest of the hardware state can be validated. Platform users can get a verification of the exact firmware loaded into the system by using the remote attestation supported by the TPM.

Secure boot assures that the firmware that is loaded in the system is signed. The validation of the hardware key hash confirms that the signer of the firmware is trusted. The trusted part of the boot extends key values into PCR 6 so that the state of the system can be verified. Figure 4 shows which values are extended into PCR 6. The secure boot verification code is the code used to verify the signatures on loaded firmware, the hardware key hash represents a trusted

[10] Other models are possible.

[11] A list of valid hardware key hashes signed by a trusted vendor or obtained from a trusted vendor's site.

[12] One of the supplied values.

[13] In our security model there is a way for the SVM to be reauthorized.

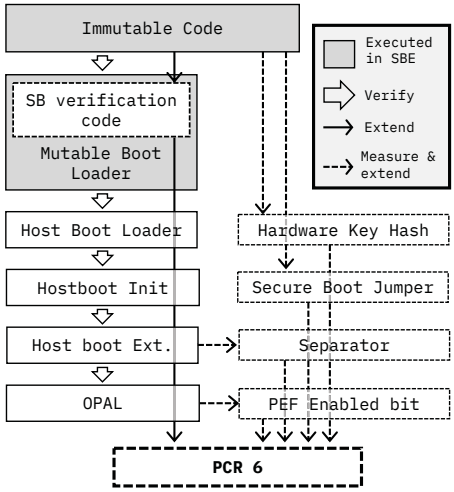

Figure 4. Values extended into PCR 6 for validation of the hardware configuration

firmware supplier, the secure boot jumper confirms whether secure boot is enabled, and PEF Enabled confirms whether PEF is enabled on this machine. The policy associated with the wrapped symmetric seed requires that the PCR 6 value match or the TPM will not use the private key for an `unseal` operation. If the PCR 6 value matches, this operation will return the symmetric seed to the Ultravisor.

3.3 Creating, Running, and Destroying SVMs

All TEEs are legitimately concerned about attacks against their secure computations. Each time the TEE is started, the remote attestation approach exploited by Intel and AMD requires that secrets are not placed into the TEE until after the host machine and the computation have been attested. Attesting the host requires verification with a server at the manufacturer to validate that the measurements from the target environment can be trusted. Attesting the computation requires a verification with the user that the measurements of the computation are correct. However, we are concerned that this approach may not scale to the size of existing cloud infrastructures without potentially introducing bottlenecks or delays. To avoid this issue, infrastructure providers will likely cache results from manufacturers, possibly creating another point of attack.

We agree that verification with the manufacturer and verification of the integrity of the TEE are essential steps. However, our approach differs from the Intel and AMD approaches. Local attestation incorporates the required measurements into the TEE when it is created. It utilizes a secure local verification prior to allowing the TEE to execute. Secrets can be securely incorporated into the `ESM` operand or into the SVM disk because of symmetric encryption. We believe that these steps enable a simpler life cycle management for cloud infrastructures.

An SVM is created by processing an NVM with offline tooling. The conversion adds a parameter to the boot command line indicating the NVM is to become an SVM but does not change the format of the VM's disk. The kernel has been changed to do an `ESM` ultracall if the transition is requested. Conversion also creates the `ESM` operand necessary for the transition from NVM to SVM to work properly. The `ESM` operand is placed in the boot file system or in the `zimage` as appropriate. The tooling requires that the virtual disk associated with the SVM is encrypted, but does not specify which form of disk encryption to use. The passphrase for the encrypted disk must be supplied to the tooling.

Figure 3 illustrates the structure of the `ESM` operand, which contains lockboxes and payload data. Each lockbox is the symmetric seed for this `ESM` operand encrypted with a public key of an authorized machine. The payload area is a group of encrypted blocks followed by HMAC data. The HMAC only covers the payload blocks. The symmetric seed is used to generate a symmetric key and an HMAC key. The symmetric key is used to encrypt each of the payload blocks and the HMAC key is used for integrity checking of the payload blocks. The first payload block contains integrity information for the SVM, hashes of the kernel, kernel command line, initramfs, and RTAS calculated by the tooling. As illustrated, the second payload block contains the pass phrase to the encrypted file system. These two areas are followed by zero or more blocks as specified by the creator of the SVM. Secrets can be placed in payload blocks by the creator of the SVM.

Our analysis indicated that we could allow the first code executed in the VM, `prom_init`, to run without compromising the ability of the Ultravisor to verify the integrity of the VM. The `ESM` ultracall occurs at the end of `prom_init`. Ultracalls are processed directly by the Ultravisor. All SVMs start execution as a normal VM in normal memory. The integrity of the SVM is protected, but only the root file system is encrypted.

Executing SVMs. When the Ultravisor receives the `ESM` ultracall from an NVM, all of the memory associated with the NVM is made secure. In POWER9, this is done by copying the `kernel`, `RTAS`, `initramfs`, and `ESM Operand` into secure memory. After the copy, if the hardware is in the proper state and authorized, the Ultravisor will be able to verify the integrity of the NVM as described in Section 3.2. If the verification fails or the Ultravisor does not have access to the symmetric seed, the attempt to transition to secure state fails. The Ultravisor removes the NVM from secure memory and returns to the hypervisor requesting termination of the NVM. If the verification passes, the NVM will resume execution as an SVM. If the NVM becomes an SVM, a tool has been added to the `initramfs` that will be used by the booting system to get the passphrase of the encrypted disk from the `ESM` operand. Once the SVM has booted, applications within the

SVM can use the ultracall interface to get the information out of payload blocks beyond the passphrase, if present.

Eliminating Side Channels. To eliminate possible side channels between the SVM guest kernel and the hypervisor, we reintroduced some bounce buffering. Significant performance work has been done in the Linux kernel and `virtio` subsystem to eliminate buffer copying, especially between guest kernels and the hypervisor. However, when PEF is enabled, the hypervisor cannot address secure memory. Retaining this design in PEF requires allowing the SVM to compute in buffers that are in normal memory. This potentially enables side channels and/or other attacks by allowing the hypervisor to observe and possibly modify SVM buffers that are actively being used. PEF eliminates this threat by leveraging the bounce buffering support in the guest Linux kernel[14] to pass buffers between the SVM and the hypervisor. Obviously, this increases the overhead for SVMs. The guest kernel in the SVM allocates the `SWIOTLB` pool of shared pages at boot time in normal memory. Whenever the content of a buffer in the SVM needs to be communicated to the hypervisor, the guest kernel allocates a buffer from the `SWIOTLB` pool, and copies the content of the source buffer to the allocated buffer. The allocated buffer is passed to the hypervisor. When the function completes, the contents of the allocated buffer are copied back into the SVM and the buffer is released.

3.4 Integrating SVMs Into The Cloud

As we interacted with cloud providers, end users, and system designers, we encountered three possibly conflicting requirements. Cloud providers must be able to maintain their infrastructure without impacting authorized TEEs. Maintenance includes the ability to migrate a TEE to a different machine, add and remove hardware, or patch hardware especially firmware updates, all transparently. End users want varying degrees of confidentiality. Some are willing to give the secrets controlling their TEEs to their cloud provider and others cannot (for various reasons including regulatory requirements). Users also want the integrity of the target system and their TEE verified prior to execution. Verification includes the TCB firmware and the TEE. The attributes of PEF that help balance these requirement are our utilization of secure and trusted boot, the ability of the user to specify the PCR value (policy) that will be verified by the TPM, and the fact that a lockbox can be inserted into `ESM` operand just prior to execution.

Transparency of maintenance is provided to the cloud provider because they specify the public key associated with the target machine as part of lockbox generation. Confidentiality is addressed through the deployment model. Our attestation model verifies the TCB firmware and the TEE. The user is also given control of the policy, represented by the PCR6 value, that will allow the system to unlock the symmetric seed. Note that we address field upgradeability without disruption of operations by proposing the cloud customer trust the processor/firmware vendor (the vendor's firmware signing public keys) instead of a specific firmware load[15]. Whereas, if the trust model is based on a specific measurement of firmware, then firmware update may break existing TEEs that are not currently running[16] until the new measurement is integrated into the attestation system.

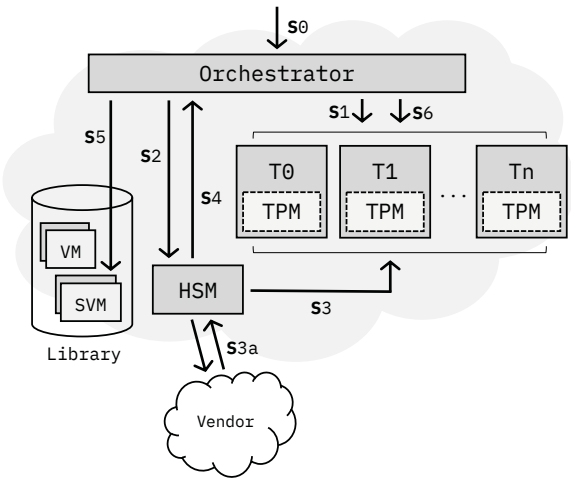

Figure 5. Integrating PEF into cloud environments: (S0) Request SVM execution, (S1) Orchestrator selects target machine, (S2) Orchestrator asks HSM to validate target, (S3) HSM request information (platform cert, storage key structure, HW key hash) from target system, (S3a) HSM validates machine with the vendor, (S4) HSM sends lockbox to the Orchestrator, (S5) Orchestrator retrieves selected SVM and inserts lockbox, (S6) Orchestrator dispatches SVM on target system.

Deployment Model. Hardware Security Modules (HSMs) exist in cloud infrastructures today and are controlled [17] by users but provided by the cloud provider. These HSMs hold keys that are critical to the cloud user and never released to the cloud provider. They perform operations with those keys on behalf of the controlling user. We propose utilizing an HSM to manage the keys associated with TEEs [18]. Our proposal adds an additional operation, generating a lockbox, and some supporting code, to the capabilities the HSM already has.

[14] This is also leveraged by AMD's SEV.

[15] The approach is flexible enough that this could be switched to a specific firmware measurement.

[16] Some systems allow dynamic upgrade of firmware.

[17] Control is represented by having the master secrets of the HSM and does not enable the customer to modify the firmware of the HSM.

[18] If a user is willing to trust the cloud provider with the keys to their TEEs, an existing key management system can hold the keys and a utility can be written to run the necessary protocol to generate the new lockbox.

We propose that this approach reasonably balances the potentially conflicting requirements, but does not require an explicit measurement of the firmware. It requires that the firmware be signed by a trusted vendor. Note that though we introduce local attestation, the Ultravisor can also support remote attestation with the introduction of a new ultracall. This description of deployment assumes that the NVM image that will transition to an SVM is already created and resides in the cloud infrastructure, the associated symmetric seed is already in the HSM, and that given the identity of the NVM, the HSM will select the correct symmetric seed.

Figure 5 illustrates the overall process for running an SVM. The cloud user requests that a running instance of a previously-created NVM (future SVM) be created. The cloud infrastructure selects a target machine and extracts the platform certificate and storage key from the target. The infrastructure forwards the extracted data to the HSM and requests enrollment of the target machine for the NVM. If the enrollment is successful, the HSM will return a lockbox for the target machine to the cloud infrastructure. The infrastructure retrieves the image from its library, inserts the lockbox into the `ESM` operand of the NVM using tooling, and provisions the image that received the lockbox to the target machine. Since the newly inserted lockbox is for the target machine, if there are no other issues, the NVM will successfully transition into an SVM. This model assumes that the customer will have one seed per SVM and that this seed will be valid on all target systems. Other models are possible.

Enrollment Protocol. The HSM, or equivalent function, has to run the enrollment protocol. The enrollment protocol requires that the cloud infrastructure pass the machine indicator (such as IP address), platform certificate, SVM indicator and storage key of the target machine to the HSM. The HSM will validate the platform certificate and check that the storage key properties are correct. If the validation and property checks done by the HSM are successful, then the HSM will generate a random challenge and `make credential`. It will then send an `activate credential` to the TPM on the target machine. The target machine will return a challenge response. If the challenge response is valid, the HSM will generate a lockbox for the target machine and return it to the cloud infrastructure.

The enrollment protocol only has to be run once for each target. Therefore, if the HSM maintains a database of enrolled machines, it can check to see whether the current target machine is enrolled. If it is already enrolled, the HSM can skip enrollment and directly generate the lockbox. Also, each hardware key hash only has to be verified once. The HSM can maintain a database of validated hardware key hashes. Before it runs the protocol to the OpenPOWER vendor, it can check to see if the hash from the target machine has already been validated. Obviously, if a key hash becomes invalid, it has to be purged from the HSM. When a machine is deprovisioned from the cloud infrastructure, the HSM must be informed that the machine is no longer a valid target so it can clean its internal databases. Finally, the primary seed in the TPM must also be cleared when the machine is deprovisioned to assure that that machine cannot be used to extract secrets from previously authorized SVMs.

4 Evaluation

In this section, we evaluate how PEF impacts the CPU and memory performance of VMs. We also analyze the design's impact on I/O operations. In addition, we look at how switching to secure mode during the booting of a VM affects it overall boot time. Finally, we discuss the scale of pertinent software additions and changes needed to implement PEF.

4.1 Experimental Setup

The evaluation is performed on an IBM POWER9 machine with two sockets, twenty physical cores per socket, and four hardware threads per core. The machine has a NUMA memory architecture with 128GB of RAM running a patched Fedora 32 to enable SVMs. (All patches have since been upstreamed.) The machine is equipped with 1Gb networks. When PEF is enabled, 64GB of RAM is reserved for secure memory. The NVMs and SVMs used for evaluation have eight CPUs and 22GB of RAM running Fedora 32. To ensure stable measurements, the VM's VCPUs are pinned to separate physical cores belonging to the first socket. The memory size was chosen to ensure only RAM local to the cores is assigned. Both NVM and SVM use LUKS dm-crypt Linux full disk encryption with passphrase-less boot. All VMs use virtio drivers for both network and block devices with IOMMU enabled and XIVE off. QCOW2 images are used for hard disks which are stored locally on the machine.

The SPEC CPU2017 benchmark is used to evaluate impacts on CPU and memory performance. Specifically, the *intspeed* and *intrate* workloads are used, configured with eight copies and eight threads, running each application three times. The *netperf* [60] benchmark is used to evaluate network performance and is configured to generate a synchronous TCP request/response traffic pattern using 99% confidence level and 5% confidence width for the measured throughput mean value. Traffic flows from the target VM to a separate physical machine with identical hardware configuration as the machine under evaluation. Each experiment is repeated nine times. Block device performance evaluation is done using the *fio* benchmark [8]. The workload consists of synchronously and sequentially reading and writing from/to blocks of a local 10GB file in the VM. We evaluate both the impact of direct and buffered I/O on block accesses. Each workload run is five minutes in duration and is repeated three times.

To evaluate the effects of PEF on a more representative real-world workload, we use the Apache Benchmark (ab) [3]

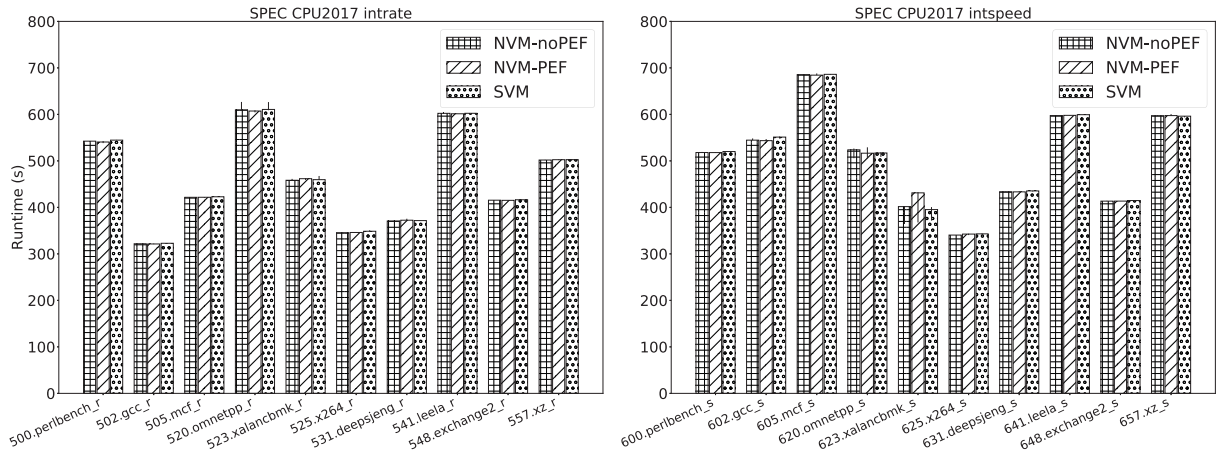

Figure 6. SPEC CPU2017 benchmark results. The vertical bars indicate the min and max values of the runs.

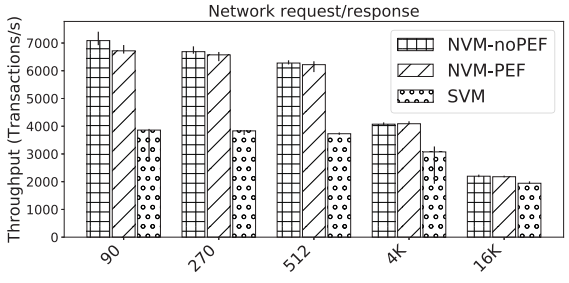

Figure 7. Network performance results. Each transaction is a request and corresponding response. Message sizes: 90, 270, 512, 4K, and 16K byte. The vertical bars indicate the min and max values of the runs.

to stress an Apache HTTP server running inside the respective VMs. The benchmark is used to retrieve static files of varying sizes (200B to 2MB) and with different levels of concurrency. Each run consisting of requests for a specific file is executed for two minutes.

It should be noted that the focus of our evaluation is to analyze the relative performance of SVMs and NVMs in order to highlight any performance impact due to our design and not the absolute performance of the machine under various workloads. In addition, our work has primarily focused on the functionality and security aspects of PEF rather than the performance optimization of SVMs. As part of our evaluation, we point out areas where improvements can be made in future iteration of this technology.

4.2 Performance Impact

To ensure our design does not impact the performance of NVMs, we compare results between an NVM running on the machine with no-PEF enabled and the same VM running on the same machine when PEF is enabled. The impact of our design on CPU and memory performance can be seen in Figure 6. As expected, the results indicate no significant performance impact when PEF is enabled. Similarly, we compare the performance of NVMs to SVMs. As can be seen, there is also no significant performance differences between SVMs and NVMs. This is attributable to the fact that no encryption is used by the SVMs to protect data in memory. Such form of encryption has been found to cause performance degradation when memory access is not sequential [29].

The impact to the network performance is shown in Figure 7. The synchronous request/response traffic pattern is designed to measure any overhead introduced by our PEF design. As shown, the throughput achieved between NVMs running non-PEF and PEF-enabled firmware is virtually the same, indicating no major impact to network performance of NVMs. However, there is a significant throughput degradation between normal and secure VMs of nearly 45% when message sizes are small. This is most likely due to the overhead associated with the bounce buffers that are used in the I/O path of SVMs and the cost of context switching between SVM and the host (see Subsection 3.1). This performance difference is gradually reduced to ~10% as the message size gets larger as can be seen in Figure 7 because the total number of messages, and therefore, context switches, is reduced.

Similar to the network performance trend, secure VMs have an initial ~77% drop in block device performance starting at the 4K block size when compared to NVMs (see Figure 8). As in the network I/O case, as the block size increases, this performance difference is gradually reduced. This can be observed below block size of 32K but since the block performance for NVMs peak above that block size, it is unclear if this relative improvement continues. However, it is clear that SVM's block performance continues to improve as block size becomes larger.

To see how much context switching can be a factor, we evaluate the use of block I/O buffering since this can reduce the total amount of interrupts generated. As can be seen in

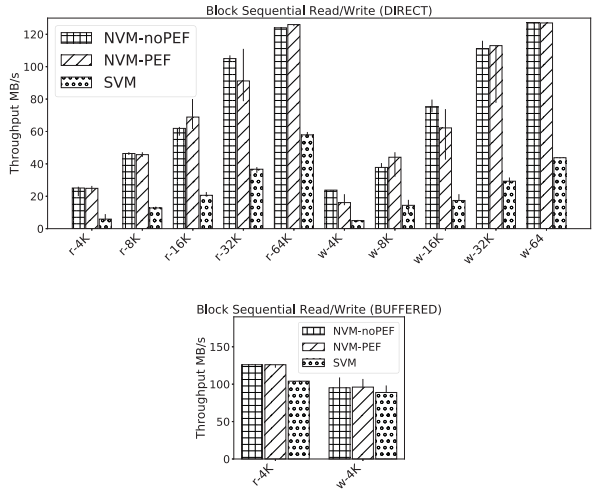

Figure 8. Block performance results. Synchronous and sequential read/write blocks from/to a 10GB file. Block sizes: 4K, 8K, 16K, 32K, and 64K. Top graph uses direct I/O and bottom graph uses buffered I/O. The vertical bars indicate the min and max values of the runs.

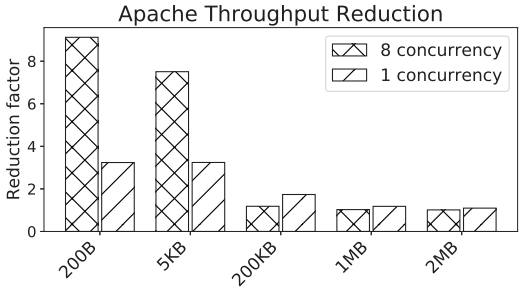

Figure 9. Apache server performance overhead of SVM compared to NVM on PEF when requesting different file sizes and concurrency levels.

the bottom graph of Figure 8, at 4K block size, the performance difference between NVMs and SVMs is less than 18% for reads and 8% for writes. With buffered I/O, larger block sizes do not improve the throughput in our experiments, hence, we omit those results.

We do not know the exact reason why the relative performance reduction between NVMs and SVMs for block differs from that of network I/O since they share the same overhead source. We suspect it may have to do with how the drivers and the respective devices interact, resulting in more VM exits. As we continue to optimize our design, we plan to investigate this issue further.

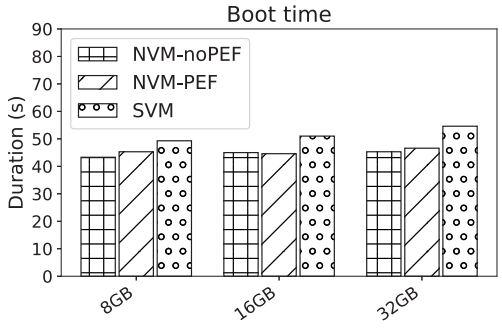

Figure 10. Boot times of NVMs and SVMs with 8GB, 16GB, and 32GB of memory.

To evaluate how much the overhead in PEF affects a real-world application, Figure 9 shows the results of running the *ab* benchmark on an Apache HTTP server hosted by the respective VMs. We do not show the relative performance difference between NVMs running on PEF and non-PEF configurations as they are similar. As can be seen, the smaller requests at higher volume (higher concurrency level) affects the performance of SVMs the greatest. The reduction to performance improves significantly to about 10% when the requested file size is 2MB. This is consistent with our evaluation using the microbenchmarks discussed above.

We also investigated the impact of PEF on the boot time of VMs. This can be particularly important for applications with sensitive VM boot time such as serverless computing or container orchestration platforms.

For our evaluation, boot time is measured from the time the VM is started to when an SSH connection can be established. As can be seen in Figure 10, SVM boot time is slower than their NVM counterparts. There is a roughly 13%-16% increase in boot time and the difference seems to increase as the size of VM memory gets larger. There are two main reasons for the increase in boot time. The first reason is due to a memory copy that is performed early in the boot process when switching from an NVM to an SVM. This initial memory copy has a fixed time given a particular VM image. The second reason is likely due to the extra work associated with creating metadata in the host kernel for reserving secure pages. This amount of work is proportional to the VM's memory size and can explain the boot time increase when memory size gets larger.

4.3 Extent of Software Modifications

We assess the complexity of our design by looking at the amount of code needed to implement the Ultravisor as well as the changes needed to support PEF in Linux. Table 2 shows the amount of code and compiled binary size for each of the major software components in the platform. The Ultravisor has close to 39K LoC and utilizes the TPM Software Stack

Table 2. Size of key software components on the platform.

Component	KLOC	K bytes
Ultravisor	39	897
TSS	37	263
Kernel/KVM	3000	2500

(TSS) (37K LoC) for interfacing with the TPM. It should be noted that the Ultravisor only uses parts of the functions that remain in the TSS so there is further room for reduction. In addition, the Ultravisor code is actively being improved upon so we expect further code trimming to occur. Table 2 also compares the Ultravisor to the size of a minimal Linux KVM kernel that supports running SVMs. While not small in an absolute sense, when compared to production ready hypervisors and OSs like the Xen Hypervisor (~500K LoC [30]) and the Linux KVM (~3M LoC), the Ultravisor is relatively small. Another interesting reference point is the security monitor design from the Keystone[53] project, which is conceptually similar to the Ultravisor but leverages RISC-V hardware capabilities and has roughly 10K LoC.

Table 3 illustrates that the total code change needed for the Linux kernel and QEMU is about 1.3K LoC. The first set of changes include code to paravirtualize the host kernel so that it can make use of Ultravisor services such as invoking *ultracalls* to update VMs' page table entries, modify secure registers, and return control to the Ultravisor. In addition, some code was added to expose Ultravisor logs to userspace. Separately, the kernel's Heterogeneous Memory Management (HMM) subsystem must be enhanced to provide the kernel the ability to associate secure memory to SVMs in collaboration with the Ultravisor. The HMM is also updated to coordinate movement of data from secure memory to normal memory and vice-versa.

Changes to the kernel are also necessary to support secure guest enablement. This includes code to perform *ultracalls* to switch to secure mode, share/unshare pages with the Hypervisor, and use page sharing for SWIOTLB buffers, Virtio Queue, Virtual Processor Area (VPA), and External Interrupt Virtualization Engine (XIVE). Furthermore, QEMU has to be updated to enable communication with the physical TPM, creation of a VM with capability to turn secure, and coordination of SVM cleanup.

All of the above changes to the Linux kernel and QEMU have been accepted and merged upstream by the respective projects as of this writing. The Ultravisor was open sourced in December of 2020[34, 35, 37–39][19].

[19]Instructions to set up PEF-enabled software stack: https://github.com/open-power/ultravisor/wiki/How-to-build-and-run-Secure-VM-using-Ultravisor-on-a-OpenPOWER-machine

Table 3. Code changes in the Linux kernel to support PEF.

Component	Lines of Code
Host kernel paravirtualization	248
Host kernel HMM	267
Secure guest kernel enablement	461
QEMU	292

5 Limitations

The integrity of a confidential computation must be maintained at all times. When this project started, Linux encrypted file system lacked runtime integrity protections. The absence of integrity protection meant that certain attacks from a malicious Hypervisor could be successful against an SVM. Subsequently, Google upstreamed [51, 52] a version of FS-verity [70] that provides runtime integrity protection for read-only files in a read/write file system. Approaches to provide runtime integrity protection for read/write files are also being pursued. Secure assignment of peripherals to a confidential computation remain an active open issue. PEF is designed to be able to leverage solutions to these issues when available.

As noted in our threat model we excluded physically probing the memory of the system. Our research was done using POWER9 which does not have hardware encryption of memory. Consequently, on POWER9 it is theoretically possible to probe memory during the hardware boot and observe the password passed from OPAL to the Ultravisor. POWER9 servers are primarily used in enterprise IT or cloud infrastructure environments that are all restricted access locations. In these environments, physical attack is a reduced concern. However, POWER10 has announced hardware memory encryption, Transparent Memory Encryption (TME) [32, 33]. TME removes this exposure by protecting the confidentiality of the memory from physical probing. It is not used for isolation or confidentiality between components running in the system. TME does not affect the size of the Ultravisor.

Migration of VMs is a well understood technology and a key capability for cloud providers. AMD has announced support for migration of encrypted virtual machines. The initial implementation of the Ultravisor does not support migration, though there are no hardware constraints. Migration utilizes a protected key known to the source and target systems, requires authorizing the SVM on the target, and metadata so that the migrated SVM resumes at the correct target. Enabling migration also enables suspend/resume of an SVM in place.

Dynamic allocation of secure memory (DASM) is the ability to change any page into or out of secure state and is not supported by POWER9. There was insufficient time to design hardware support in the product cycle where the initial hardware changes were implemented. The absence of

DASM increases the size of the ultravisor because it must do memory management for SVMs instead of monitoring the memory management done by the Hypervisor. Though supported by POWER9, PEF does not support transactional memory.

Currently, starting an SVM requires that all of the memory must be available. Prior research [12], IBM Z Secure Execution [15], and AMD SEV-SNP [2] all support DASM with memory over-commit. Memory over-commit could have been implemented with the partitioned memory design, but it would have required a significant effort that would be discarded when DASM is available.

Previous research [12] demonstrated hardware and software approaches to sharing memory between TEEs. Because the Ultravisor manages and controls the page tables for SVMs, this can be implemented. The Ultravisor does the encryption and decryption of secure memory when required. Consequently, the availability of hardware-based memory encryption may further reduce the size of the Ultravisor.

DASM, memory over-commit, and memory sharing between SVMs are interrelated. The hardware support for DASM will affect how memory over-commit and memory sharing are implemented. Although these two features can be implemented on the current architecture, it was decided that it was better to wait until the hardware support for other features is resolved.

6 Related Work

There is a long history of hardware changes to increase the security of computer systems [21, 48, 56, 59, 61]. These changes are often a combination of hardware and software methods [10, 11, 50, 67]. Focusing on TEEs specifically, since the initial development of TEEs in the embedded systems space starting with ARM TrustZone [5, 63, 66], major processor vendors including Intel, AMD and IBM have developed and commercialized processors with TEE support. Broadly speaking, there are two classes of TEEs: process-based [13, 14, 22, 55, 57, 58, 68] and VM-based [2, 15]. A significant amount of effort from both industry and academia has been focused on improving TEE implementations as well as extending TEEs beyond the CPU.

In this section, we outline relevant TEE designs from existing works and start by discussing more recent TEEs that are directly related to PEF. Specifically, Table 4 compares key attributes and features of PEF to Intel SGX, Keystone using RISC-V, Intel TDX, and AMD SEV. PEF has many similarities with these TEEs but PEF is different[20] in that the secure entity (e.g., VM) is always encrypted, even before deployed on the platform, thus allowing for a model of inserting user secrets before attestation. In addition, PEF is currently distinct in its support for the sharing of secure memory between SVMs which can have performance benefits for certain classes of applications [1]. Furthermore, PEF supports a local attestation model that offers a different cost trade-off consideration especially in cloud deployments. Finally, PEF is currently the only commercially available TEE that is fully open sourced.

[20] Z Secure Execution for Linux One also has encrypted TEEs.

	SGX	Keystone	TDX	SEV	PEF
Attributes					
No architectural changes	✗	✓	✗	✗	✗
Physical Attacks protection	✓	✓	✓	✓	1
Features					
Open Source Implementation	✗	✓	✗	✗	✓
Multi-threading/core Support	✓	2	✓	✓	✓
Support Shared Memory	✓	✓	✓	✓	✓
Sharing Secure Memory	✗	✗	✗	✗	✓
VM TEE	✗	3	✓	✓	✓
Dynamic secure memory assignment	✗	✓	✓	✓	4
Application transparrency	5	✓	✓	✓	✓
Attestation Model Local/Remote	R	R	R	R	L

Table 4. Comparison of related work supporting confidential computing. (1) POWER10 introduces physical memory protection. (2) Keystone has multi-threading support, but does not yet support multi-core. The impact of multi-core is unclear. (3) While not yet demonstrated, Keystone can support virtualizion if the RISC-V processor has hypervisor state. (4) Additional hardware is required for PEF to support dynamic assignment of secure memory. (5) Scone[20] extends SGX to make applications transparent within a container.

Intel's SGX [58] SGX protects portions of an application, called *enclaves* that are explicitly entered and exited. This reduces the transparency, since the developer has to slice the application and protect the interface. To ease application development, Haven [9] shows how SGX can protect an unmodified binary by creating a unikernel inside the enclave, but with the risk of an increased TCB and reaching the memory limit of protected memory [22]. Scone [6, 20] extends the usability of SGX while reducing the TCB by allowing an application inside a Docker container to run in an enclave. Graphene-SGX [18] adds multiprocess support to SGX enclaves. Sanctum [23] improves on the SGX model by incorporating an open and smaller TCB while providing greater enclave isolation using a page-coloring-based cache partitioning scheme to prevent certain types of software-based side-channel attacks.

RISC-V based TIMBER-V [71] uses memory tagging to isolate code and data in applications, effectively enabling the creation of enclaves in RISC-V-based embedded processors. Due to fine-grained memory tagging, TIMBER-V is able to support both the SGX and TrustZone programming model. Also based on RISC-V processors, Keystone [53] provides a framework for building customized TEEs by leveraging hardware security primitives and a programmable software abstraction layer. Keystone is unique because, as reported, it did not require changes to RISC-V architecture for implementation. Additionally, as a framework it can concurrently support different TEE architectures. It makes use of a privileged reference monitor to control and assign enclaves to

confined memory areas and leverages hardware features in the RISC-V processor to enforce memory isolation. Applications must be developed in conjunction with a run time designed specifically for the Keystone framework.

Intel's TDX is Intel's approach [40–43] to providing a VM based TEE for x86 architecture. Unlike the 64-bit extensions, Intel's architecture is not the same as AMD's. TDX exploits the SGX infrastructure where possible, but it does not have the same limitations. Our summary of its feature in Table 4 is based upon the white paper and architecture documents.

AMD's SEV [2] provides secure VM execution which most closely resembles the functionality of PEF. SEV uses memory encryption for confidentiality which makes sharing memory between secure VMs challenging. In addition, similar to Intel SGX, attestation requires verifying with AMD the authenticity of the platform before a user's key is injected into the VM to decrypt the secret data.

ARM TrustZone [5] partitions the processor into Secure and Normal worlds, where code in the Normal world cannot access resources in the Secure world. This is enforced by hardware separation of the two worlds with a Secure Monitor that runs in the Secure world, responsible for switching between the worlds. Applications must be tailored to make use of the two worlds – keeping sensitive data and code in the Secure world while still being able to communicate with the Normal world. Sanctuary [16] increases the flexibility of ARM's TEE by providing mechanisms allowing enclaves to be run in the Normal world's user-space.

XOM [55] is a set of architecture changes that enforces program confidentiality, execution isolation, and copy/tamper resistance to applications. It does this through the use of program and data encryption, data tagging, and a private on-chip data store. The design introduces new instructions including "secure load" and "secure store", which tell the CPU to perform integrity verification on the values loaded and stored. For this reason, the *application transparency* is limited: developers must tailor a particular application to this architecture. Aegis [68] subsequently fixed an attack in XOM by providing protection against replay attacks using an integrity tree and implemented optimizations to improve performance overhead.

Secret-Protecting Architecture [54] provides a mechanism to run code in a tamper-protected environment. The authors design in [24] a mechanism extending a root of trust to supporting devices. However, this technique does not have the transparency of other techniques. For example, there can be only one Secret-Protected application installed and running at a given point in time. The ability of protected applications to make use of system calls is likewise limited.

Flicker [57] provides a hardware-based protection mechanism that functions on existing, commonly deployed hardware (using mechanisms available in the TPM). As with other solutions, this incurs only minimal performance overhead. It provides a protected execution environment, but without requiring new hardware changes. The trade-off is that software has to be specifically developed to run within this environment. The protected environment is created by locking down the CPU using TPM late-load capabilities, so that the protected software is guaranteed to be the only software running at this point. System calls and multi-threading in particular do not work for this reason. Moreover, hardware interrupts are disabled, so the OS is suspended while the protected software is running. Thus, software targeted for Flicker must be written to spend only short durations inside the protected environment.

Iso-X [25] considers a threat model identical to PEF. However, the granularity of the protected portions is similar to SGX, and requires explicitly placing the secure code in a *compartment*.

Overshadow [19] provides guarantees of integrity and privacy protection similar to AEGIS, but implemented in a virtual machine monitor instead of in hardware. This approach has the advantages of making the implementation transparent to software developers. For example, software shims are added to protected processes to handle system calls seamlessly. This means, however, that there is no protection provided against malicious system software. The *cloaking* approach, where the OS sees an encrypted version, is similar to the mechanism employed in PEF.

7 Conclusions

In this paper, we introduced PEF, an open source virtual machine-based Trusted Execution Environment (TEE) for confidential computing for OpenPOWER (Power ISA). Our contributions include a novel partitioning of the TEE between hardware and firmware, a design that maximizes the use of existing security components, and one that simplifies life cycle management for SVMs by utilizing local attestation.

Our work also accomplishes the following: enabling the conversion of existing VMs into SVMs with new supported tooling (backward compatibility), demonstrates minimal impact on NVMs, when PEF is active and none if it is disabled, introduces a small number of new Ultravisor interfaces, creates a smaller TCB than exists for a normal VM, enables existing applications (with minimal restrictions) to run in an SVM, enables secure sharing between SVMs, and provides protection of each SVM from the rest of the system and the rest of the system from each SVM.

Acknowledgments

The authors would like to acknowledge the significant contributions to Protected Execution Facility by Mike Anderson, Thiago J. Bauermann, Christopher J Engel, Ronald Kalla, Memmet Kayaalp, Jens Leenstra, Dimitrios Pendarakis, Peter A. Sandon, and Sarah Wright. Without their efforts this project would not have been successful.

References

[1] Adil Ahmad, Juhee Kim, Jaebaek Seo, Insik Shin, Pedro Fonseca, and Byoungyoung Lee. 2021. CHANCEL: Efficient Multi-client Isolation Under Adversarial Programs. In *Annual Network and Distributed System Security Symposium (NDSS)*. Internet Society, 11710 Plaza America Drive, Suite 400, Reston, VA 20190.

[2] AMD. 2020. AMD SEV-SNP: Strengthening VM Isolation-with Integrity Protection and More. White paper. https://www.amd.com/system/files/TechDocs/SEV-SNP-strengthening-vm-isolation-with-integrity-protection-and-more.pdf

[3] Apache. 2021. ab - Apache HTTP server benchmarking tool. on-line. https://httpd.apache.org/docs/2.4/programs/ab.html

[4] L. B. Arimilli, B. Blaner, B. C. Drerup, C. F. Marino, D. E. Williams, E. N. Lais, F. A. Campisano, G. L. Guthrie, M. S. Floyd, R. B. Leavens, S. M. Willenborg, R. Kalla, and B. Abali. 2018. IBM POWER9™ processor and system features for computing in the cognitive era. *Journal of Reproduction and Development* 62 (2018), 1.

[5] ARM. 2009. ARM Security Technology Building a Secure System using TrustZone Technology. White paper. https://static.docs.arm.com/genc009492/c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf

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

[7] Will Authur, David Challener, and With Kenneth Goldman. 2015. *A Pratical Guide to TPM 2.0*. Apress, One New York Plaza, Suite 4600 New York, NY 10004-1562. https://www.apress.com/us/book/9781430265832

[8] Jens Axboe. 2020. fio. github.com. https://github.com/axboe/fio There are 187 contributors to this repo.

[9] Andrew Baumann, Marcus Peinado, and Galen Hunt. 2015. Shielding applications from an untrusted cloud with haven. *ACM Transactions on Computer Systems (TOCS)* 33, 3 (2015), 1–26.

[10] David E Bell and Leonard J LaPadula. 1975. *Computer security model: Unified exposition and multics interpretation*. Technical Report ESD-TR-75-306. MITRE Corp., Bedford, MA,.

[11] Kenneth J Biba. 1977. *Integrity considerations for secure computer systems*. Technical Report. MITRE CORP BEDFORD MA.

[12] Richard Boivie, , Ek Ekanadham, Bhushan Jain, Eric Hall, Guerney D H Hunt, Mohit Kapur, Mehmet Kayaalp, Elaine Palmer, Dimitrios Pendarakis, David Safford, and Ray Valdez. 2017. *Hardware Support For Malware Defense and End-To-End Trust*. Technical Report. IBM T. J. Watson Research Center.

[13] Rick Boivie. 2012. *Secureblue++: Cpu support for secure execution*. Technical Report RC 25287. IBM T. J. Watson Research Center.

[14] Rick Boivie and Peter Williams. 2013. *Secureblue++: Cpu support for secure execution*. Technical Report RC 25369. IBM T. J. Watson Research Center.

[15] C. Borntrager, J. D. Bradbury, R. Bundgen, F. Busaba, L. C. Heller, and V. Mihajlovski. 2020. Secure Your Cloud Workloads with IBM Secure Execution for Linux on IBM z15. *IBM Journal of Research and Development* 64, 5/6 (2020), 1–1.

[16] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2019. SANCTUARY: ARMing TrustZone with User-space Enclaves. In *26th Annual Network and Distributed System Security Symposium, (NDSS)*. Internet Society, 11710 Plaza America Drive, Suite 400 Reston, VA 20190, Session 1A: Mobile Security.

[17] Claudia Carvalho. 2018. Using the TPM NVRAM to Protect Secure Boot Keys in OpenPOWER. (August 2018). https://www.youtube.com/watch?v=8baIs5h-tI4&list=UL8baIs5h-tI4&index=269 Linux Security Summit NA.

[18] Chia che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In *2017 USENIX Annual Technical Conference (USENIX ATC 17)*. USENIX, Santa Clara, CA, 645–658. https://www.usenix.org/conference/atc17/technical-sessions/presentation/tsai

[19] Xiaoxin Chen, Tal Garfinkel, E Christopher Lewis, Pratap Subrahmanyam, Carl A Waldspurger, Dan Boneh, Jeffrey Dwoskin, and Dan RK Ports. 2008. Overshadow: a virtualization-based approach to retrofitting protection in commodity operating systems. *ACM SIGOPS Operating Systems Review* 42, 2 (2008), 2–13.

[20] Scone Confidential Computing. 2021. https://sconedocs.github.io/sconify_image

[21] Fernando J Corbató and Victor A Vyssotsky. 1965. Introduction and overview of the Multics system. In *Proceedings of the November 30–December 1, 1965, fall joint computer conference, part I*. Spartan Books, Washington, DC, 185–196.

[22] Victor Costan and Srinivas Devadas. 2016. Intel SGX Explained. *IACR Cryptol. ePrint Arch.* 2016, 86 (2016), 1–118.

[23] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *25th USENIX Security Symposium (USENIX Security 16)*. USENIX, Austin, TX, 857–874. https://www.usenix.org/conference/usenixsecurity16/technical-sessions/presentation/costan

[24] Jeffrey S Dwoskin and Ruby B Lee. 2007. Hardware-rooted trust for secure key management and transient trust. In *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 1601 Broadway, 10th Floor New York, NY 10019-7434, 389–400.

[25] Dmitry Evtyushkin, Jesse Elwell, Meltem Ozsoy, Dmitry Ponomarev, Nael Abu Ghazaleh, and Ryan Riley. 2014. Iso-x: A flexible architecture for hardware-managed isolated execution. In *2014 47th Annual kilburn1961atlas/ACM International Symposium on Microarchitecture*. IEEE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264 USA, 190–202.

[26] Kenneth G. Goldman. 2015. IBM's TPM 2.0 TSS. https://sourceforge.net/projects/ibmtpm20tss/

[27] Christopher Gonzalez, Eric Fluhr, Daniel Dreps, David Hogenmiller, Rahul Rao, Jose Paredes, Michael Floyd, Michael Sperling, Ryan Kruse, Vinod Ramadurai, et al. 2017. 3.1 POWER9™: A processor family optimized for cognitive computing with 25Gb/s accelerator links and 16Gb/s PCIe Gen4. In *2017 IEEE International Solid-State Circuits Conference (ISSCC)*. IEEE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264 USA, 50–51.

[28] Trusted Computing Group. 2019. *Trusted Platform Module Library Specification, Family 2.0* (level 00, revision 01.59 ed.). Trusted Computing Group, 3855 SW 153rd Drive, Beaverton, Oregon 97003. https://trustedcomputinggroup.org/work-groups/trusted-platform-module/

[29] C. Göttel, R. Pires, I. Rocha, S. Vaucher, P. Felber, M. Pasin, and V. Schiavoni. 2018. Security, Performance and Energy Trade-Offs of Hardware-Assisted Memory Protection Mechanisms. In *2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS)* (Salvador, Brazil). IEEE Computer Society, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264 USA, 133–142.

[30] Synopsys: Black Duck Open Hub. 2020. Xen Project (Hypervisor) Statistics. on-line. https://www.openhub.net/p/xenproject-hypervisor/analyses/latest/languages_summary

[31] IBM. 2019. *POWER9™ Processor Programming Model Bulletin*. Technical Report. IBM. https://wiki.raptorcs.com/w/images/1/15/POWER9_Processor_Programming_Model_Bulletin_090919.pdf

[32] IBM. 2020. IBM Power Systems Announces POWER10 Processor. https://www.ibm.com/blogs/systems/ibm-power-systems-announces-power10-processor/

[33] IBM. 2020. IBM Reveals Next-Generation IBM POWER10 Processor. https://newsroom.ibm.com/2020-08-17-IBM-Reveals-Next-Generation-IBM-POWER10-Processor
[34] IBM. 2020. Instructions to set up the PEF-enabled software stack. https://github.com/open-power/ultravisor/wiki/How-to-build-and-run-Secure-VM-using-Ultravisor-on-a-OpenPOWER-machine
[35] IBM. 2020. PEF/Ultravisor code. https://github.com/open-power/ultravisor
[36] IBM 2020. *Power ISA ™ Version 3.1.* IBM. soft copy distribution: https://wiki.raptorcs.com/w/images/f/f5/PowerISA_public.v3.1.pdf.
[37] IBM. 2020. svm-build tools. https://github.com/open-power/svm-tools
[38] IBM. 2020. Ultravisor enabled pnor. https://github.com/rampai/op-build
[39] IBM. 2020. Ultravisor enabled skiboot. https://github.com/rampai/skiboot
[40] Intel. 2020. *Architectural Specification: Intel® Trust Domain Extensions (Intel® TDX) Module* (document number: 344425-001us ed.). Intel Corporation, 2200 Mission College Blvd.,Santa Clara, CA 95054-1549 USA.
[41] Intel. 2020. *Intel® TDX Virtual Firmware Design Guide* (document number: 344991-001us ed.). Intel Corporation, 2200 Mission College Blvd.,Santa Clara, CA 95054-1549 USA.
[42] Intel. 2020. *Intel® Trust Domain CPU Architectural Extensions* (document nubmber: 343754-001us ed.). Intel Corporation, 2200 Mission College Blvd.,Santa Clara, CA 95054-1549 USA.
[43] Intel. 2020. Intel® Trust Domain Extensions. White paper. https://software.intel.com/content/dam/develop/external/us/en/documents/tdx-whitepaper-v4.pdf
[44] International Business Machines Corporation 2016. *Linux on Power Architecture Platform Reference* (advance ed.). International Business Machines Corporation. https://openpowerfoundation.org/?resource_lib=linux-on-power-architecture-platform-reference
[45] Nayna Jain. 2019. A Deep Dive into OpenPOWER Host Secure Boot. (October 2019). https://www.youtube.com/watch?v=I0NgIRYRhtw OpenPower Summit EU.
[46] Nayna Jain. 2019. OpenPOWER Secureboot Host OS Key Management. (August 2019). https://www.youtube.com/watch?v=yfdbuzvptsg Linux Security Summit.
[47] Nayna Jain and Thiago J Bauerman. 2018. Using Linux as a secure boot loader for OpenPOWER Servers. (October 2018). https://www.youtube.com/watch?v=hwB1bkXQep4 Linux Security Summit Europe.
[48] Tom Kilburn, R Bruce Payne, and David J Howarth. 1961. The ATLAS supervisor. In *Proceedings of the December 12-14, 1961, eastern joint computer conference: computers-key to total systems control*. Association for Computing Machinery, New York, NY, USA, 279–294.
[49] Tom Kolan, Hillel Mendelson, Vitali Sokhin, Kevin Reick, Elena Tsanko, and Greg Wetli. 2020. Post-silicon validation of the IBM POWER9™ processor. In *2020 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264 USA, 999–1002.
[50] Butler W Lampson. 1974. Protection. *ACM SIGOPS Operating Systems Review* 8, 1 (1974), 18–24.
[51] Michael Larabel. 2019. Google's FS-VERITY File Authentication Call For Inclusion in Linux 5.4 Kernel. https://www.phoronix.com/scan.php?page=news_item&px=Linux-5.4-FS-VERITY-PR
[52] Michael Larabel. 2020. FS-VERITY Seeing Performance Enhancements With Linux 5.6 Kernel. https://www.phoronix.com/scan.php?page=news_item&px=Linux-5.4-FS-VERITY-PR
[53] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Krste Asanović, and Dawn Song. 2020. Keystone: An Open Framework for Architecting Trusted Execution Environments. In *Proceedings of the Fifteenth European Conference on Computer Systems* (Heraklion, Greece). Association for Computing Machinery, New York, NY, USA, 16 pages.
[54] Ruby B Lee, Peter CS Kwan, John P McGregor, Jeffrey Dwoskin, and Zhenghong Wang. 2005. Architecture for protecting critical secrets in microprocessors. In *32nd International Symposium on Computer Architecture (ISCA'05)*. IEEE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264 USA, 2–13.
[55] David Lie, Chandramohan Thekkath, Mark Mitchell, Patrick Lincoln, Dan Boneh, John Mitchell, and Mark Horowitz. 2000. Architectural support for copy and tamper resistant software. *Acm Sigplan Notices* 35, 11 (2000), 168–177.
[56] Alastair JW Mayer. 1982. The architecture of the Burroughs B5000: 20 years later and still ahead of the times? *ACM SIGARCH Computer Architecture News* 10, 4 (1982), 3–10.
[57] Jonathan M McCune, Bryan J Parno, Adrian Perrig, Michael K Reiter, and Hiroshi Isozaki. 2008. Flicker: An execution infrastructure for TCB minimization. In *Proceedings of the 3rd ACM SIGOPS/EuroSys European Conference on Computer Systems 2008*. ACM, 1601 Broadway, 10th Floor New York, NY 10019-7434, 315–328.
[58] Frank McKeen, Ilya Alexandrovich, Alex Berenzon, Carlos V Rozas, Hisham Shafi, Vedvyas Shanbhogue, and Uday R Savagaonkar. 2013. Innovative instructions and software model for isolated execution. *Hasp@ isca* 10, 1 (2013).
[59] Shigeru Motobayashi, Takashi Masuda, and Nobumasa Takahashi. 1969. The HITAC5020 time sharing system. In *Proceedings of the 1969 24th national conference*. Association for Computing Machinery, New York, NY, USA, 419–429.
[60] netperf. 2021. Netperf. on-line. https://github.com/HewlettPackard/netperf
[61] Elliott I Organick. 2014. *Computer system organization: the B5700/B6700 series*. Academic Press, 111 Fifth Avenue, New York, NY.
[62] Ramachandra Pai, Michael V. Le, Guerney D. H. Hunt, and Hani Jamjoom. 2021. Documentation: Enabling PEF, Creating SVM, and Running Performance Evaluation. on-line. https://github.com/mvle/eurosys2021_PEF_OpenPOWER/wiki
[63] Mohamed Sabt, Mohammed Achemlal, and Abdelmadjid Bouabdallah. 2015. Trusted execution environment: what it is, and what it is not. In *2015 IEEE Trustcom/BigDataSE/ISPA*, Vol. 1. IEEE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264 USA, 57–64.
[64] Satish Kumar Sadasivam, Brian W Thompto, Ron Kalla, and William J Starke. 2017. IBM POWER9™ processor architecture. *IEEE Micro* 37, 2 (2017), 40–51.
[65] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2020. Malware Guard Extension: abusing Intel SGX to conceal cache attacks. *Cybersecurity* 3, 1 (2020), 2.
[66] Carlton Shepherd, Ghada Arfaoui, Iakovos Gurulian, Robert P Lee, Konstantinos Markantonakis, Raja Naeem Akram, Damien Sauveron, and Emmanuel Conchon. 2016. Secure and trusted execution: Past, present, and future-a critical review in the context of the internet of things and cyber-physical systems. In *2016 IEEE Trustcom/BigDataSE/ISPA*. IEEE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264 USA, 168–177.
[67] Leroy Smith. 1975. *Architectures for secure computing systems*. Technical Report. MITRE CORP BEDFORD MASS.
[68] G Edward Suh, Dwaine Clarke, Blaise Gassend, Marten Van Dijk, and Srinivas Devadas. 2003. AEGIS: architecture for tamper-evident and tamper-resistant processing. In *ACM International Conference on Supercomputing 25th Anniversary Volume*. ACM, 1601 Broadway, 10th Floor New York, NY 10019-7434, 357–368.
[69] Brian Thompto. 2016. POWER9™: Processor for the cognitive era. In *2016 IEEE Hot Chips 28 Symposium (HCS)*. IEEE, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720-1264 USA, 1–19.
[70] Theodore Ts'o. 2018. File System-level Integrity Protection. USENIX Association, Oakland, CA.
[71] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. 2019. TIMBER-V: Tag-Isolated

Memory Bringing Fine-grained Enclaves to RISC-V. In *26th Annual Network and Distributed System Security Symposium, (NDSS)*. Internet Society, 11710 Plaza America Drive, Suite 400 Reston, VA 20190, Session 10.

[72] George Wilson. 2017. Defeating Invisible Enemies: Firmware Based Security in OpenPOWER Systems. (September 2017). https://events.static.linuxfound.org/sites/events/files/slides/op-stboot-lss-2017-v7.0.pdf Linux Security Summit.

[73] George Wilson. 2019. Open Power Secure and Trusted Boot. (December 2019). https://www.youtube.com/watch?v=4GUddIZZ3GA Linux Security Summit.

A Artifact Description

Our artifacts include the code and complete instructions for enabling PEF on standard OpenPOWER or POWER9 machines as well as the steps for creating an SVM and running all the experiments described in our paper. It also includes all of the necessary scripts, configuration files, and non-licensed software needed to replicate our work.

Three versions of the POWER9 chip have been produced: DD 2.1 (initial), DD 2.2 (second), and DD 2.3 (current). Each version replaced the prior version in manufacturing. Our instructions will work on either a DD 2.2 or DD 2.3 processor. The recommended version for PEF is DD 2.3.

The landing page for the documentation can be found here [62]. Embedded in the documentation are instructions tailored for the EuroSys Artifact Evaluation Committee titled *"How to build and run Secure VM using Ultravisor on a OpenPOWER machine for Eurosys."* The instructions for running the performance benchmarks also refer to (S)VMs we create for the Artifact Committee. Because we supplied equipment for the evaluation, the step that required physical access to the equipment had already been done. Note that the two machines as well as VMs referenced in those instructions are no longer available. Consequently, these instructions should be ignored.

The instructions [34] in the wiki are detailed and contain videos at key points. There are three major tasks:

1. Enabling PEF on standard OpenPOWER or POWER9 machine,
   a. Configuring the hardware for easy development
   b. Installing the firmware
   c. Installing and configuring the operating System.
2. Creating an SVM from an NVM, and
3. Running the experiments described in the paper.

A.1 Enabling PEF

If you have the ability to sign firmware that can be loaded into your OpenPOWER or POWER9 machine, then you have the option of not disabling secure boot, but you must run the firmware through the signature process every time you build it. In a normal development/test environment, it is easier to disable secure boot, per the instructions, which requires one-time physical access to the machine. Similarly, re-enabling secure boot requires one time access. Also, for your OpenPOWER machine, you need to confirm that the TPM vendor is Nuvoton. Although the Ultravisor is TPM vendor agnostic, we give you instructions for configuring the Nuvoton TPM (found in POWER9 machines). For the Ultravisor to be able to utilize the TPM in your system, the correct kernel driver must be loaded and the TPM enabled. As far as we know, there are Linux drivers available for all TPM vendors.

If your objective is to build the Ultravisor and test, experiment, or play at the virtual machine level, following the instructions in the Wiki is sufficient. However, if you want to debug or modify the Ultravisor code in any way, it would be very useful to review the information in the doc directory of the github [35]. You should also acquire a copy of the Linux PAPR documentation [44]. This describes all of the hypercalls that the Ultravisor supports.

A.2 Creating an SVM

All SVMs are created from an NVM. If you are starting from scratch, then you have to create a VM before you can create an SVM. Whether you create a VM or use an existing VM, it will be converted into an SVM by following the instructions we supplied. If you choose to use an existing VM, it must be upgraded to a kernel that supports PEF. The required version are listed in the instructions.

A.3 Running Performance Evaluation

We provide all the necessary scripts, configuration files, and references to benchmarking software needed to replicate our experiments. However, due to licensing issues, we do not provide access to SPEC CPU2017.

The target VMs should be configured so that each VCPU is pinned to a separate physical core and have at least 22GB of memory. The memory should also be allocated from the same NUMA node if applicable.

The block benchmark creates large 10GB files and so the VM should have at least 50GB of free disk space. Two machines are needed for the networking benchmark. Only one needs to be an OpenPOWER POWER9. Scripts are provided to open firewall holes in order for the client and server processes to connect.