Enclave Computing on RISC-V: A Brighter Future for Security?

Ghada Dessouky
Technical University of Darmstadt
Darmstadt, Germany
ghada.dessouky@trust.tu-darmstadt.de

Ahmad-Reza Sadeghi
Technical University of Darmstadt
Darmstadt, Germany
ahmad.sadeghi@trust.tu-darmstadt.de

Emmanuel Stapf
Technical University of Darmstadt
Darmstadt, Germany
emmanuel.stapf@trust.tu-darmstadt.de

***Abstract*—The large attack surface of commodity operating systems motivated academia and industry to develop novel security architectures which provide strong protection for sensitive applications in so-called *enclaves* that only require trust in the hardware and minimal software components. However, the enclave architectures proposed by industry often lack important features, such as secure I/O, and assume threat models which leave out important classes of attacks, such as microarchitectural attacks. Thus, recent works in academia have proposed a new line of enclave architectures with distinct features and more comprehensive threat models, many of which were developed on the open RISC-V architecture. In this paper, we present a brief overview of the RISC-V based enclave architectures proposed in academia, discuss their features, limitations and open challenges which we tackle in our current research.**

***Index Terms*—RISC-V, Trusted Execution Environment (TEE), Enclave, Side-Channel Security**

I. Introduction

For decades, the ongoing discovery of exploitable memory corruption bugs in modern software has fueled a persistent arms race between software attacks and defenses. This gave rise to a spectrum of defenses, such as control-flow integrity (CFI), data-flow integrity (DFI), code-pointer integrity (CPI), and fine-grained address space layout randomization (ASLR). More critically, large code bases of commodity operating systems (OS), have also been found vulnerable in recent years and thus, unsuitable to serve as an underlying Trusted Computing Base (TCB) for protecting sensitive services. This motivated increasing efforts and solutions to integrate hardware-assisted security primitives tightly into the System-on-Chip (SoC) to protect sensitive services, e.g., instruction extensions for hardware-assisted CFI [7], in-process memory isolation [11], or capability systems [27].

One most prominent and promising approach is that of security architectures providing a Trusted Execution Environment (TEE). TEE architectures commonly deploy hardware and software security primitives to enable isolated containers, usually called *enclaves*. Enclaves are used to isolate the execution of sensitive services from all other software, including the OS, and thus, protect them even against strong software adversaries which can compromise the OS. Only a small software/microcode TCB, which configures the underlying hardware security primitives of the system and manages these enclaves, is inherently trusted.

Enclave-based security architectures have been proposed for a variety of computing platforms, ranging from resource-constrained embedded systems, such as Sancus [20], TyTAN [2], TrustLite [16], and TIMBER-V [25], to high-performance computing systems, e.g., industry solutions like Intel SGX [13], AMD SEV [14], ARM TrustZone [1], or academic solutions such as Sanctum [6], Sanctuary [3], or Keystone [17], to name some. While the industry solutions successfully enable a new level of protection against a more privileged software adversary, they often lack important features, such as secure I/O or protection mechanisms against sophisticated software attacks, e.g. cache side-channels attacks, which are typically not included in their threat models.

The advent of open-source hardware and the open architecture RISC-V initiated a new line of research and an opportunity to explore, scrutinize and design enclave architectures across the full stack, both hardware and software. This brought rise to a number of academic solutions such as Sanctum [6], Keystone [17] and TIMBER-V [25], which attempt to address the shortcomings of existing industry solutions, and steer the future for upcoming industry solutions.

In this paper, we provide a brief overview of recently proposed RISC-V based enclave architectures in academia, while highlighting their features, advantages and limitations. We conclude with our envisioned proposal of next-generation enclave architectures on which we are actively working, while shedding light on open challenges for future research.

II. Enclave-based Security Architectures on RISC-V

We introduce next the most well-known RISC-V enclave architectures in academia, Sanctum [6], Keystone [17] and TIMBER-V [25], and discuss their features and limitations, whereas a summarized comparison is shown in Section II-B.

A. Sanctum

In 2016, Costan et al. [6] proposed the Sanctum security architecture whose high-level design is shown in Figure 1.

Adversary Model. The authors assume a strong software adversary who compromises the operating system (OS) executed in the supervisor level. Moreover, the adversary is assumed to use system peripherals to perform Direct Memory Access (DMA) attacks [18] and to conduct cache side-channel attacks

from software [4], [21], [22]. However, all physical attacks, such as fault-injection attacks [15] or cold-boot attacks [12], are considered out of scope.

Design, Features & Limitations. Sanctum offers isolated execution contexts, called *enclaves*, to protect sensitive services on RISC-V platforms. Every enclave, which runs in the user level, comes bundled with a non-sensitive application which invokes the enclave. The integrity of an enclave is verified prior to its execution using local or remote attestation.

Sanctum relies on the untrusted OS to manage the enclave memory and to provide OS services like interrupt handling or I/O services. However, it has been shown that this may allow adversaries, which compromised the OS, to conduct controlled side-channel attacks on enclaves [19], [24], [28]. The adversary can infer information of the enclave's internal state by observing the enclave page tables [28] or by interrupting the enclave repeatedly [24]. Sanctum mitigates these attacks by storing the enclave page tables in the enclave memory and by making the enclaves interrupt-aware which allows them to detect suspicious interrupt behavior.

Sanctum prevents cache side-channel attacks by two mechanisms: 1) flushing sensitive processor resources on every enclave context switch, namely, the L1 cache and the Translation Lookaside Buffer (TLB), and 2) partitioning the shared L2 cache through memory page coloring which allows to assign cache lines exclusively to enclaves. However, Sanctum's cache partitioning design suffers in practice since all software components of the system need to adhere to the coloring scheme. As a result, the partitioning can only be set at run time if the complete memory layout of the OS is rearranged according to the coloring scheme, which is impractical.

Sanctum's enclaves comprise unprivileged user-level code and thus, cannot support use cases in which a secure connection to a peripheral, e.g., a sensor or GPU, is necessary because this would require privileged driver code. Sanctum provides a basic DMA attack protection which allows to restrict DMA to one specific region of the memory.

Hardware Primitives & TCB. Sanctum enforces the isolation of the enclave code and data by introducing small hardware changes at the Page Table Walker (PTW) which is part of the Memory Management Unit (MMU). The hardware changes guarantee on the one hand that the OS cannot access enclave memory, and on the other hand that an enclave cannot access the OS memory or other enclaves by changing its page tables. The custom PTW prevents a successful address translation of virtual memory addresses which would map to physical memory addresses that the current execution context is not allowed to access. The security critical functionalities of Sanctum are implemented in a software component called the Security Monitor (SM) which represents the Trusted Computing Base (TCB) of the system. The SM runs in the machine level of the RISC-V processor and is verified during a secure boot process. Sanctum's basic DMA protection is implemented by adding two registers to the memory controller.

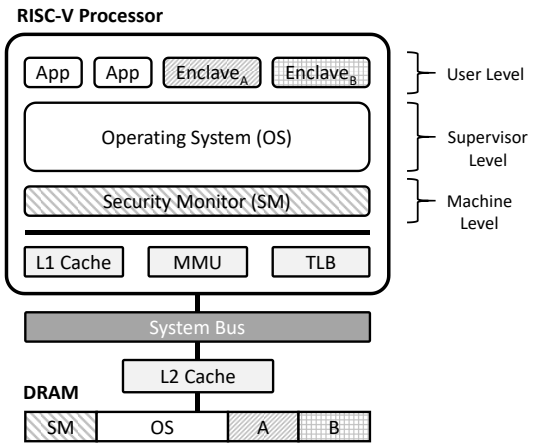

Fig. 1: Design overview of Sanctum which provides user-level enclaves and cache partitioning for the shared L2 cache.

B. Keystone

The high-level design of the Keystone [17] security architecture is shown in Figure 2.

Adversary Model. Keystone was designed to mitigate a strong software adversary able to compromise the OS and perform cache side-channel attacks. Simple physical attacks, in which an adversary is able to read out the content of the DRAM, e.g., by snooping the memory bus, are also considered.

Design, Features & Limitations. Keystone provides enclaves which, in contrast to Sanctum's enclaves, comprise of software running in the user level and supervisor level. In the user level, the enclave app (*EApp*) is executed which contains sensitive application code. In the supervisor level, the enclave runtime provides OS services to the enclave, such as, memory management or interrupt handling. The ability to include critical OS functionalities into the enclave allows Keystone to protect against controlled side-channel attacks which exploit the sharing of sensitive data or code structures, e.g., page tables [28] or interrupt handlers [24].

Keystone's enclaves are not integrated into the OS and thus, not scheduled like unmodified processes. Whenever an enclave is set up, the state of the OS needs to be stored (and later restored) and the enclave runtime booted which introduces an additional performance overhead. Moreover, providing a runtime for each enclave increases the enclave development effort and leads to duplicate code on the system since many runtimes will provide the same basic functionalities.

In Keystone, device drivers could potentially be included into the enclave runtime to connect to peripherals and perform secure I/O. However, no two-way binding between an enclave and a peripheral is proposed which would enable DMA-capable peripherals to securely communicate with the enclave over enclave memory. As a result, Keystone cannot protect enclaves from DMA attacks [18].

Hardware Primitives & TCB. Keystone enforces the enclave isolation using the Physical Memory Protection (PMP) module specified in the RISC-V ISA manual [10]. The PMP allows to define memory access permissions for the user and supervisor

level (together) and for the machine level. Keystone utilizes the PMP to assign one continuous memory region to each active enclave and to the software in the machine level. All other memory regions are automatically assigned to the OS. The machine level code, which is called Security Monitor (SM), configures the PMP and represents the software TCB of the system. The SM is verified during a secure boot process.

Keystone provides shared L2 cache partitioning by implementing way-based partitioning which assigns complete cache ways to processor cores that execute enclaves. This can lead to cache under-utilization since cache lines that are unused by an enclave cannot be allocated by any other software component.

Keystone can provide protection from simple hardware attacks, e.g., bus snooping, when the executed enclave (EApp + enclave runtime) and the SM fit completely into an on-chip scratchpad memory.

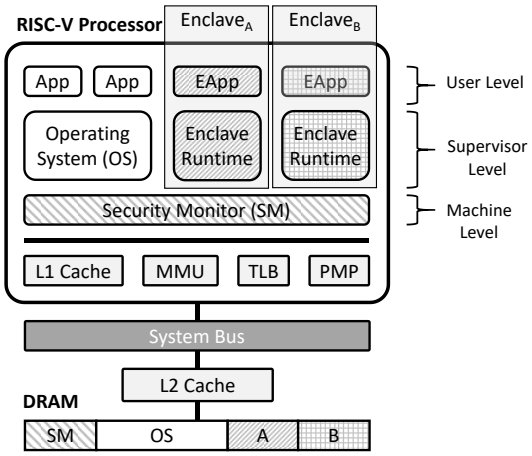

Fig. 2: Design overview of Keystone which provides enclaves (user-level enclave app (EApp) + supervisor-level runtime)

C. TIMBER-V

The high-level design of TIMBER-V [25], which targets embedded systems, is shown in Figure 3.

Adversary Model. A strong software adversary is assumed that compromises the OS, with physical attacks out of scope.

Design, Features & Limitations. The core idea of TIMBER-V is to split applications into a *normal* and a *trusted* domain, whereas memory tagging is used to assign memory words to either one of the domains. All sensitive code and data of an applications is included in the trusted domain. In comparison to Sanctum and Keystone, TIMBER-V's trusted domains allow to create more fine-grained enclaves or *sub-process* enclaves. TIMBER-V also defines a trusted domain in the OS which is called *TagRoot*. The TagRoot enforces the separation between the sub-process enclaves in user level by configuring TIMBER-V's hardware primitives. Moreover, the TagRoot is used by the OS to setup the enclaves and provides some services to the enclaves, e.g., shared memory for communicating with the normal domain, sealing or attestation.

TIMBER-V provides fine-grained enclaves for embedded systems. However, TIMBER-V (as presented) does not provide secure communication channels from enclaves to peripherals, e.g., sensors. Theoretically, drivers and other services can be included in the TagRoot and thus, offered to the enclaves. However, then, TIMBER-V adopts the high-level security model of ARM's TrustZone-A [1] in which a trusted OS provides services to a set of trusted applications. As repeatedly shown, summarized in [5], this security model is flawed since it substantially increases the attack surface of the system.

TIMBER-V does not provide any protection mechanisms against cache side-channel attacks and is vulnerable to interrupt-based controlled side-channel attacks [24] since the enclave interrupt handling is performed by the OS.

Hardware Primitives & TCB. TIMBER-V implements memory tagging by making changes to the Memory Protection Unit (MPU) of the system which enforces the separation between all processes and between the normal and trusted domain of the currently executed process. Furthermore, a hardware tag engine is introduced which checks every memory access using the tags stored in memory. TIMBER-V requires 2 bit tags for every 32 bit of memory (6.25% memory overhead) and introduces custom instructions for checking and manipulation the tags. Additional tag engines are required at every peripheral with Direct Memory Access (DMA) capabilities in order to protect from DMA attacks [18].

The software TCB of the system consists of the TagRoot, which verifies the MPU configuration set by the OS, and all software running in the machine level of the processor.

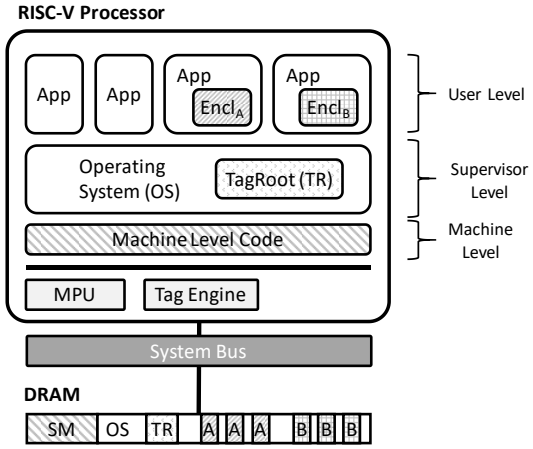

Fig. 3: Design overview of TIMBER-V which provides sub-process enclaves on embedded systems.

III. Open Challenges

In the following, we discuss open challenges of enclave-based security architectures and give an outlook on how these challenges could be tackled in next-generation architectures.

Customizable enclave architectures. The RISC-V security architectures presented in Section II each provide a specific type of enclave which either comprises of user-level code, user-level and supervisor-level code or isolates only parts of a process. In other words, all architectures adopt a rigid *one-size-fits-all* approach to enclaves which require the developer

Name	Enclave Type			Dynamic Cache Side-Channel Resilience	Controlled Side-Channel Resilience	Enclave-to-Peripheral Binding
	User-Level	User/Supervisor-Level	Sub-Process			
Sanctum [6]	●	○	○	◐	●	○
Keystone [17]	○	●	○	●	●	○
TIMBER-V [25]	○	○	●	○	◐	○

to adapt sensitive services to the enclaves' features and requirements, instead of adapting the enclave to the needs of the sensitive service which is limiting and cumbersome in practice.

The rapidly growing diversity of services that process sensitive data, e.g., biometric authentication, digital keys, Machine Learning as a Service (MLaaS), among many others, demand service-specific requirements from the underlying security architecture. One such requirement is to establish secure two-way binding between peripherals and enclaves, which is inevitable in many IoT and AI usage scenarios in which peripherals, such as sensors and GPUs are used. A two-way binding between enclaves and peripherals includes performing access control on the accesses from enclaves to peripherals (over MMIO) and also on the accesses from DMA-capable peripherals to enclave memory.

As mentioned in Section I, commercial enclave architectures do not consider side-channel attacks in their threat models. Sanctum and Keystone propose cache partitioning schemes, yet these suffer from practical limitations or provide coarse-grained allocation of cache resources to enclaves. Thus, another open challenge is to incorporate adequate, practical and customizable microarchitectural defenses [8], [23], [26] against side-channel attacks into enclave architectures.

We envision next-generation enclave architectures, on which we are actively working, which provide customizable enclaves of different types where the privilege level of the enclave can be selected as stipulated by the sensitive service's functionality and security requirements. Exclusive binding of individual enclaves to peripherals (both MMIO and DMA) should also be supported to enable emerging IoT and AI use cases. Moreover, the architecture should provide flexible mechanisms by-design to configure the required side-channel protection level at run time for different enclaves individually and independently. These configurable protection mechanisms need to be tightly integrated throughout the full stack of a computing platform. One approach that we are currently pursuing is to allow configurable and on-demand fine-grained cache resource partitioning that can be flexibly customized for every enclave individually.

Reconfigurable and adaptive platform security. The challenge of providing flexible side-channel defenses is motivated by numerous attacks performed on commercial enclave architectures [4], [29]. However, incorporating these defenses can only protect against known or anticipated threats. Unknown or unanticipated attacks might still require a more radical upgrade of the entire system, at both the hardware and software layers, to make computing platforms more future-proof.

However, hardware, unlike software, cannot be updated or patched once fabricated in silicon. This also means that extensive security analysis is required at the pre-silicon design phase to verify the security of the underlying hardware design. As we have witnessed through our work with Hack@DAC [9], this is, however, very challenging to achieve in practice even with state-of-the-art hardware verification and analysis techniques.

We envision a paradigm shift to a more comprehensive design approach based on a secure open platform that enables flexible updatability of *both* hardware and software primitives at run time, such that it can thwart future attacks as well as adapt to different functionality and security requirements. We propose to augment the underlying hardware with just-in-time reconfigurable hardware elements, thus enabling hardware security updates and secure reconfiguration of selected hardware components/features even after deployment. One approach would be to enable different granularities of this reconfigurability for different features, largely depending on how tightly integrated these features are into the processor pipeline and their impact on the critical path (and thus performance). This ranges from the coarse-grained per-process enabling/disabling of tightly integrated microarchitectural optimizations, such as speculative execution, or selecting 1-of-x cache partitioning mechanisms to enable at run time, all the way to a more fine-grained clean-slate reconfiguration of logic tiles that interface less tightly with key components of the processor. These logic tiles can be reconfigured entirely to implement different processor security extensions, e.g., complete execution tracing or only control-flow tracing or enforcement.

Realizing this platform requires tackling many open challenges, e.g., devising systematic techniques to identify the root causes of vulnerable information leakage and which security-critical components require this hardware reconfigurability. Other challenges are identifying the security-performance trade-offs that need to be incorporated by design, which level of reconfigurability is feasible for different features without degrading performance prohibitively in today's complex SoCs, and how to most efficiently design and integrate them into enclave architectures.

ACKNOWLEDGMENT

We thank our anonymous reviewers for their valuable and constructive feedback. This work was funded by the Intel Collaborative Research Institute for Collaborative Autonomous & Resilient Systems (ICRI-CARS) and by the Deutsche Forschungsgemeinschaft (DFG) – SFB 1119 – 236615297.

REFERENCES

[1] ARM Limited. Security technology: building a secure system using TrustZone technology. http://infocenter.arm.com/help/topic/com.arm.doc.prd29-genc-009492c/PRD29-GENC-009492C_trustzone_security_whitepaper.pdf, 2008.

[2] Ferdinand Brasser, Brahim El Mahjoub, Ahmad-Reza Sadeghi, Christian Wachsmann, and Patrick Koeberl. Tytan: tiny trust anchor for tiny devices. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6. IEEE, 2015.

[3] Ferdinand Brasser, David Gens, Patrick Jauernig, Ahmad-Reza Sadeghi, and Emmanuel Stapf. Sanctuary: Arming trustzone with user-space enclaves. In *NDSS*, 2019.

[4] Ferdinand Brasser, Urs Müller, Alexandra Dmitrienko, Kari Kostiainen, Srdjan Capkun, and Ahmad-Reza Sadeghi. Software grand exposure: Sgx cache attacks are practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*, 2017.

[5] David Cerdeira, Nuno Santos, Pedro Fonseca, and Sandro Pinto. Sok: Understanding the prevailing security vulnerabilities in trustzone-assisted tee systems. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P), San Francisco, CA, USA*, pages 18–20, 2020.

[6] Victor Costan, Ilia A Lebedev, and Srinivas Devadas. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security Symposium*, 2016.

[7] Lucas Davi et al. HAFIX: Hardware-Assisted Flow Integrity eXtension. In *ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2015.

[8] Ghada Dessouky, Tommaso Frassetto, and Ahmad-Reza Sadeghi. HybCache: Hybrid Side-Channel-Resilient Caches for Trusted Execution Environments. 2020.

[9] Ghada Dessouky, David Gens, Patrick Haney, Garrett Persyn, Arun Kanuparthi, Hareesh Khattri, Jason M Fung, Ahmad-Reza Sadeghi, and Jeyavijayan Rajendran. Hardfails: insights into software-exploitable hardware bugs. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 213–230, 2019.

[10] RISC-V Foundation. The risc-v instruction set manual, volume ii: Privileged architecture, documentversion 20190608-priv-msu-ratified". https://riscv.org/specifications/privileged-isa/, 2019.

[11] Tommaso Frassetto, Patrick Jauernig, Christopher Liebchen, and Ahmad-Reza Sadeghi. Imix: In-process memory isolation extension. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 83–97, 2018.

[12] J Alex Halderman, Seth D Schoen, Nadia Heninger, William Clarkson, William Paul, Joseph A Calandrino, Ariel J Feldman, Jacob Appelbaum, and Edward W Felten. Lest we remember: cold-boot attacks on encryption keys. *Communications of the ACM*, 52(5):91–98, 2009.

[13] Intel. Intel Software Guard Extensions Programming Reference. https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf, 2014.

[14] David Kaplan, Jeremy Powell, and Tom Woller. Amd memory encryption. https://developer.amd.com/wordpress/media/2013/12/AMD_Memory_Encryption_Whitepaper_v7-Public.pdf, 2016.

[15] Yoongu Kim, Ross Daly, Jeremie Kim, Chris Fallin, Ji Hye Lee, Donghyuk Lee, Chris Wilkerson, Konrad Lai, and Onur Mutlu. Flipping bits in memory without accessing them: An experimental study of dram disturbance errors. In *41st Annual International Symposium on Computer Architecture*, ISCA, 2014.

[16] Patrick Koeberl, Steffen Schulz, Ahmad-Reza Sadeghi, and Vijay Varadharajan. Trustlite: A security architecture for tiny embedded devices. In *Proceedings of the Ninth European Conference on Computer Systems*, page 10. ACM, 2014.

[17] Dayeol Lee, David Kohlbrenner, Shweta Shinde, Dawn Song, and Krste Asanović. Keystone: A framework for architecting tees. *arXiv preprint arXiv:1907.10119*, 2019.

[18] A Theodore Markettos, Colin Rothwell, Brett F Gutstein, Allison Pearce, Peter G Neumann, Simon W Moore, and Robert NM Watson. Thunderclap: Exploring vulnerabilities in operating system iommu protection via dma from untrustworthy peripherals. In *NDSS*, 2019.

[19] Mathias Morbitzer, Manuel Huber, Julian Horsch, and Sascha Wessel. Severed: Subverting amd's virtual machine encryption. In *Proceedings of the 11th European Workshop on Systems Security*, page 1. ACM, 2018.

[20] Job Noorman, Pieter Agten, Wilfried Daniels, Raoul Strackx, Anthony Van Herrewege, Christophe Huygens, Bart Preneel, Ingrid Verbauwhede, and Frank Piessens. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *22nd USENIX Security symposium*, 2013.

[21] Dag Arne Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of AES. In *RSA Conference*, 2006.

[22] Colin Percival. Cache missing for fun and profit, 2005.

[23] Moinuddin K. Qureshi. Ceaser: Mitigating Conflict-based Cache Attacks via Encrypted-Address and Remapping. 2018.

[24] Jo Van Bulck, Frank Piessens, and Raoul Strackx. Nemesis: Studying microarchitectural timing leaks in rudimentary cpu interrupt logic. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pages 178–195. ACM, 2018.

[25] Samuel Weiser, Mario Werner, Ferdinand Brasser, Maja Malenko, Stefan Mangard, and Ahmad-Reza Sadeghi. Timber-v: Tag-isolated memory bringing fine-grained enclaves to risc-v. In *NDSS*, 2019.

[26] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. ScatterCache: Thwarting Cache Attacks via Cache Set Randomization. 2019.

[27] Jonathan Woodruff, Robert NM Watson, David Chisnall, Simon W Moore, Jonathan Anderson, Brooks Davis, Ben Laurie, Peter G Neumann, Robert Norton, and Michael Roe. The cheri capability model: Revisiting risc in an age of risk. In *2014 ACM/IEEE 41st International Symposium on Computer Architecture (ISCA)*, pages 457–468. IEEE, 2014.

[28] Yuanzhong Xu, Weidong Cui, and Marcus Peinado. Controlled-channel attacks: Deterministic side channels for untrusted operating systems. In *2015 IEEE Symposium on Security and Privacy*, pages 640–656. IEEE, 2015.

[29] Ning Zhang, Kun Sun, Deborah Shands, Wenjing Lou, and Y Thomas Hou. Truspy: Cache side-channel information leakage from the secure world on arm devices. *IACR Cryptology ePrint Archive*, 2016:980, 2016.