INVITED: In Hardware We Trust

Gains and Pains of Hardware-assisted Security

Lejla Batina
lejla@cs.ru.nl
Radboud University, The Netherlands

Patrick Jauernig
patrick.jauernig@trust.tu-darmstadt.de
TU Darmstadt, Germany

Nele Mentens
nele.mentens@kuleuven.be
KU Leuven, Belgium

Ahmad-Reza Sadeghi
ahmad.sadeghi@trust.tu-darmstadt.de
TU Darmstadt, Germany

Emmanuel Stapf
emmanuel.stapf@trust.tu-darmstadt.de
TU Darmstadt, Germany

ABSTRACT

Data processing and communication in almost all electronic systems are based on Central Processing Units (CPUs). In order to guarantee confidentiality and integrity of the software running on a CPU, hardware-assisted security architectures are used. However, both the threat model and the non-functional platform requirements, i.e. performance and energy budget, differ when we go from high-end desktop computers and servers to low-end embedded devices that populate the internet of things (IoT). For high-end platforms, a relatively large energy budget is available to protect software against attacks. However, measures to optimize performance give rise to microarchitectural side-channel attacks. IoT devices, in contrast, are constrained in terms of energy consumption and do not incorporate the performance enhancements found in high-end CPUs. Hence, they are less likely to be susceptible to microarchitectural attacks, but give rise to physical attacks, exploiting, e.g., leakage in power consumption or through fault injection. Whereas previous work mostly concentrates on a specific architecture, this paper covers the whole spectrum of computing systems, comparing the corresponding hardware architectures, and most relevant threats.

ACM Reference Format:
Lejla Batina, Patrick Jauernig, Nele Mentens, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2019. INVITED: In Hardware We Trust: Gains and Pains of Hardware-assisted Security. In *The 56th Annual Design Automation Conference 2019 (DAC '19), June 2–6, 2019, Las Vegas, NV, USA.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3316781.3323480

1 INTRODUCTION

Software attacks can lead to unauthorized access or physical damage to infected devices and networks. To limit potential attacks to only the code containing the flaw, operating system kernels make use of process isolation. However, flaws in the kernel itself can be used to undermine process isolation. Trusted Execution Environments (TEEs) and hardware-assisted approaches try to guarantee a level of security that mitigations for vulnerable software have failed to provide. The basic assumption for hardware-assisted security mechanisms is that hardware is less likely to have software-exploitable vulnerabilities. Building on those primitives, the complexity of the software parts in mitigations can be reduced. As a consequence, industry is advocating hardware-assisted approaches on all fronts: desktop computers (Intel SGX [16], AMD SEV [20]), smartphones (ARM TrustZone [2]), down to low-energy embedded devices (ARM TrustZone-M [43]). In parallel, academia proposed many security solutions based on those industrial hardware trust anchors. A promising approach is to manage hardware by a small software Trusted Computing Base (TCB) to create TEEs, e.g., protecting sensitive user-space code on mobile devices based on ARM TrustZone [7], or improving Intel SGX on RISC-V [11]. However, other types of attacks, besides software attacks, are exploiting the microarchitectural details of the CPU or the physical leakage from cryptographic implementations in hardware or software. Traditional physical attacks either maliciously inject faults [5] or exploit physical leakage from side channels such as timing [23], power consumption [25] or electromagnetic emanations [14] to infer secret information from cryptographic algorithms. Microarchitectural attacks are side-channel attacks that rely on the CPU's architectural specifics [15, 17, 24, 29, 38].
In this paper, we survey different security architectures based on the gains they inherit from underlying hardware security primitives, and present pains caused by subtle state-of-the-art attacks on these security architectures that exploit microarchitectural effects of modern hardware performance enhancements.

2 ADVERSARY MODEL AND REQUIREMENTS

We rely on the classification presented in [1] and distinguish the following types of adversaries: (1) remote adversary, capable of inserting malicious software; (2) local adversary, capable of remote attacks, and controlling and eavesdropping on the communication; (3) physical adversary, capable of performing (non-)intrusive physical attacks—we distinguish between microarchitectural side-channel analysis (SCA) attacks and classical physical attacks. Depending on the computing platform, the importance of each attack model differs, as illustrated in Figure 1. Remote and local attacks are applicable to all types of computing platforms. Classical physical attacks, discussed in Section 5 are not considered a main threat in servers and desktop computers, while they are prominent on IoT devices that allow potential adversaries in close proximity. Today's microarchitectural attacks, on the other hand, are a consequence of performance enhancements in high-end CPUs and are thus mostly applicable to servers and desktop computers, as explained in Section 4.

DAC '19, June 2–6, 2019, Las Vegas, NV, USA
© 2019 Copyright held by the owner/author(s). Publication rights licensed to ACM.
ACM ISBN 978-1-4503-6725-7/19/06…$15.00
https://doi.org/10.1145/3316781.3323480

Non-functional requirements such as performance and energy consumption determine which security architectures the computing platforms are capable of integrating, as explained in Section 3.

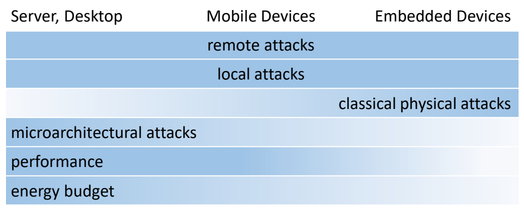

Figure 1: Adversary models and non-functional requirements (the darker the color, the higher the importance).

3 HARDWARE-ASSISTED SECURITY ARCHITECTURES

We categorize existing hardware-assisted security architectures depending on the performance capabilities of the devices on which they are implemented.

3.1 Stationary High-Performance Devices

We present the most popular industrial and academic hardware-assisted security architectures, namely, Intel SGX for x86, and Sanctum for the open RISC-V architecture.

Intel SGX. SGX [10, 16] provides TEEs, also called enclaves, on recent Intel processors. An enclave is used to execute sensitive program code in user space, isolated from a potentially malicious OS or hypervisor. Each enclave is bundled with a regular non-sensitive host application which invokes the enclave. During the enclave setup, the integrity of the enclave code is attested locally or remotely using authenticated measurements of the code. The enclave memory management, exception handling and I/O is performed by the untrusted OS. However, hardware features protect the enclave memory from an unauthorized access by the untrusted OS or hypervisor. SGX's TCB only comprises the CPU hardware and its microcode. SGX encrypts all enclave code and data leaving the CPU. This allows SGX to protect enclaves from Direct Memory Access (DMA) attacks and to persistently store the state of an enclave.

Sanctum. Sanctum [11] was developed for the open RISC-V architecture and resembles Intel SGX regarding its high-level concept. However, SGX's microcode TCB is substituted by a privileged software component called the monitor code. The isolation of the enclave memory is enforced by introducing small hardware changes around the page table walker of the CPU's Memory Management Unit (MMU). In contrast to SGX, Sanctum does not encrypt the enclave code and data in the main memory. However, Sanctum provides a basic DMA attack protection by modifying the memory controller. Unlike SGX, Sanctum implements cache partitioning in the shared last-level cache through memory page coloring. This allows Sanctum to assign cache lines exclusively to an enclave.

3.2 Mobile High-Performance Devices

ARM TrustZone is a widely deployed industrial TEE on mobile devices which provides a single enclave. Sanctuary is an academic TEE that leverages TrustZone to overcome this shortcoming.

ARM TrustZone. ARM TrustZone [2] is used to implement security architectures on ARM-based devices. TrustZone separates the system into two worlds, normal world and secure world. The secure world represents the single enclave (or TEE) of the system. All sensitive apps run in the secure world but are only separated by a software-based isolation. TrustZone can, in contrast to SGX and Sanctum, establish secure channels between peripherals and sensitive apps. The world switch is performed by a privileged component called the monitor code which also verifies all secure world code during boot using digital signatures. The monitor code represents the software TCB of the system, together with all code in the secure world. The separation between both worlds is enforced in hardware by a set of security enhancements of the CPU and other SoC components. TrustZone does not provide cache partitioning but DMA access control by temporarily assigning memory regions exclusively to SoC components (e.g. the CPU or GPU). Since TrustZone only provides a single enclave, the device vendor has to establish a costly trust relationship to all sensitive app developers.

Sanctuary. Sanctuary [7] solves the main problem of currently deployed TrustZone-based architectures by providing an arbitrary number of user-space enclaves without introducing new hardware components. In Sanctuary, sensitive apps are not included in the secure world but temporarily isolated on physical cores. The secure world only contains security primitives provided by the device vendor. Therefore, the system TCB only consists of device vendor code. No trust relationships between device vendor and app developers need to be established. The security primitives provide TrustZone functionalities to Sanctuary (e.g DMA access control or secure channels to peripherals). The isolation of the physical cores is enforced by exploiting a feature of ARM's TrustZone-enabled address space controller. In contrast to Sanctum, Sanctuary cannot provide cache partitioning of the shared last-level cache. However, software cache side-channel attacks on sensitive apps are prevented by excluding enclave memory from the shared caches.

3.3 Embedded Devices

Embedded systems have a tight energy budget. Thus, instead of integrating fully-fledged MMUs, these systems use primitive access controllers that limit secure and trusted execution capabilities.

SMART. SMART [12] establishes a dynamic root of trust using hardware-assisted attestation. SMART supports remote attestation but not code isolation. To do so, SMART leverages read-only memory (ROM) containing the attestation code and a secure key that can only be accessed if the program counter is pointing to the ROM region. In detail, attestation is invoked by an untrusted entity that wants to attest code in regular memory. First, the attestation code disables interrupts. Then, the secret key is used to compute an attestation report containing the HMAC of the memory region, input parameters, a nonce and an after-attestation destination address. Then, the report is copied to regular memory, SMART cleans up attestation traces, and jumps to the destination address in the attested code. The attested code can enable interrupts again, execute, and send the attestation report to a verifier. Because of disabled interrupts, SMART is not suitable for real-time applications, and does not consider side-channel attacks or DMA attacks [31] in its threat model. Sancus [33] reduces SMART's TCB to pure hardware.

TrustLite. In contrast to SMART and Sancus, TrustLite [26] provides a fully-fledged TEE for embedded devices. TrustLite leverages an (extended) execution-aware Memory Protection Unit (EA-MPU) and generalizes the concept of a read-only attestation code to freely-configurable regions, called Trustlets, protected in terms of confidentiality and integrity. First, the Secure Loader, stored in ROM, loads the Trustlets into memory and configures the EA-MPU to enforce isolation for each Trustlet's memory. Second, EA-MPU configuration is locked, thus, protection regions are static and a cleanup as in SMART is not needed anymore. Finally, the OS is started. TrustLite provides more flexibility in protecting enclaves than SMART and Sancus. Still, side-channel and DMA attacks are not part of the attacker model. TyTAN [6], an extension of TrustLite for real-time systems, further adds secure boot and secure storage.

4 MICROARCHITECTURAL ATTACKS

In recent years, security research has shown that CPU performance optimizations like caches or transient execution can be exploited by software attackers to infer information from victim processes.

4.1 Software Cache Side-Channel Attacks

In software-based cache side-channel attacks, the attacker measures the duration of memory accesses to infer if a victim process touched certain data. This information can, e.g., be used to attack cryptographic algorithms [34, 35]. Recently, various cache side-channel attacks were presented, e.g, Evict+Time [34], Prime+Probe [34], and Flush+Reload [42]. Attacks are, however, not limited to memory caches: theoretically, any cache structure shared by the attacker and the victim can be exploited, e.g. the TLB [15] or the BTB [28]. Software countermeasures can be implemented in the algorithms that should be protected, e.g. [3, 34], or in privileged software levels that oversee the attacker and victim processes, e.g. [9, 32]. Hardware countermeasures implemented as part of a security architecture either use some sort of cache partitioning [39] or randomize the mapping from memory addresses to cache lines [40].
Software-induced cache side-channel attacks have only been considered in security architectures most recently. SGX and TrustZone do not provide cache side-channel protection on an architectural level for their enclaves [8, 44]. Sanctuary and Sanctum flush core-exclusive caches on the enclave context switches. In contrast to SGX and TrustZone, Sanctum provides partitioning for the shared last-level cache. Sanctuary relies on the TrustZone and can therefore not provide cache partitioning. However, Sanctuary protects from cache side-channel attacks by excluding the Sanctuary memory from the shared caches, while none of the presented security architectures for embedded devices even considers cache side channels.

4.2 Transient Execution Attacks

Transient instructions are used to implement CPU performance optimization techniques like out-of-order execution or branch prediction. However, recent attacks showed that there are several flaws in the design of transient execution which an attacker can exploit to alter the microarchitectural state normally invisible to the architecture. In this section, we give an overview over these attacks.
Spectre. Spectre and its variants [22, 24, 27] use mistraining of branch (or return) prediction to transiently execute code even when it is guarded by an access-restricting branch while bypassing all software defenses like bounds checking or CFI. In addition, on modern CPUs, branch prediction buffers are indexed using virtual addresses of the branch instructions [21], allowing mistraining not only from the same address space, but also from different processes.
Meltdown. Meltdown and its variants [22, 29, 36] focus on memory accesses instead of code execution. By exploiting the time window between the cause of an exception and its actual raise at retirement of the causing instructions, Meltdown leaks results of transient instructions that get successively executed during this window. At instruction retirement, the microarchitectural state gets cleaned up, yet, (cache) side channels can be used to extract these results.
Foreshadow. Foreshadow [38] is a transient-execution attack on Intel SGX based on Meltdown's insights. SGX is immune to a plain Meltdown attack as enclave memory usually does not raise memory access exceptions. However, as the OS is in control of all page tables, an attacker can set the `present` or `reserved` bit to force the enclave to raise a page fault (dubbed L1 Terminal Fault, *L1TF* [17], by Intel) [41]. Under those conditions, address translation is aborted, only cache values tagged with the corresponding physical address can be extracted this way. However, arbitrary encrypted enclave pages can be externally forced to be decrypted to the L1 cache using SGX's *secure page swapping*, enabling the attacker to proceed analogously to a normal Meltdown attack.
Consequences. Most of the known transient execution attacks have been mitigated. Yet, trust has been shattered irreparably, e.g., Foreshadow was used to extract attestation keys of Intel SGX. In parallel, new transient execution attacks are emerging [4, 18], even on embedded devices [13]. For most of the hardware-assisted security mechanisms presented in this paper, an extensive evaluation of transient execution attacks has not been presented yet.

5 CLASSICAL PHYSICAL ATTACKS

Physical attacks can be categorized into non-intrusive (passive) and intrusive (active) attacks. In the former case, an adversary, e.g., monitors power consumption while a device performs secret key operations [25]. Other sources of side-channel information, such as electromagnetic (EM) emanations from a chip [14] and timings for different operations performed [23] have also shown to be effective [30]. In contrast, intrusive attacks induce faults in the system that lead to secret information being leaked in the system's output [5]. Physical attacks are performed on both software and hardware implementations of cryptographic algorithms.
Typical countermeasures against passive SCA attacks can be divided into two categories: hiding and masking. Hiding countermeasures aim at breaking the link between the processed data and the side-channel leakage. Masking countermeasures break the link between the actual data and the processed data such that side-channel leakage does not infer the secret information. For an overview, see [30].
While SCA is a passive technique that relies on merely observing leakages, fault attacks are actively provoked and require direct access to the device. Fault attacks aim at exposing sensitive information triggered by irregular conditions that lead to faulty computations. One way to accomplish this is by "glitching" the device, i.e., forcing changes in the values of relevant physical parameters outside the specified intervals. Glitches can be induced through the clock signal, the power supply, EM pulses or optical signals. When

the glitch leads to exploitable faulty computations, fault analysis can be performed. The state of the art in fault attacks and countermeasures can be found in [19]. A recent attack worth mentioning, is demonstrated on ARM TrustZone: CLKSCREW forces a processor to operate beyond its Dynamic Voltage and Frequency Scaling (DVFS) limits in order to leak cryptographic keys [37].

6 CONCLUSION

Learning from the mistakes of speculative execution, computing architectures should be built with security in mind—not only performance. Similar, as the current trend of shrinking the software TCB by adding hardware security extensions is continuing, careful analysis of these components and their interplay is needed more than ever. Building on this underlying TCB, security solutions like cryptographic primitives or TEEs face multi-faceted challenges in the future. Existing TEEs have unsolved issues and are still not widely-adopted, also because of licensing contracts needed to deploy TEE apps in industry solutions. In addition, mobile devices and IoT devices need to face the challenge of protecting against attackers that are in the vicinity of the devices and thus capable of performing physical attacks through side channels or fault injection. In general, it is important to select the optimal security architecture given the energy and performance budget of the application.

Acknowledgements. This work has been supported by the projects HWSec, P3 and S7 within CRC 1119 CROSSING (German Research Foundation, DFG), the Intel ICRI-CARS institute, WearIT4Health (EU/Interreg/EMR) and in part by the Technology Foundation TTW (Project 13499 TYPHOON) from the Dutch government.

REFERENCES

[1] T. Abera, N Asokan, L. Davi, J. Ekberg, T. Nyman, A. Paverd, A. Sadeghi, and G. Tsudik. 2016. C-FLAT: control-flow attestation for embedded systems software. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[2] ARM Limited. 2008. 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.

[3] D. Bernstein, T. Lange, and P. Schwabe. 2012. The Security Impact of a New Cryptographic Library. In *LATINCRYPT (Lecture Notes in Computer Science)*.

[4] A. Bhattacharyya, A. Sandulescu, M. Neugschwandtner, A. Sorniotti, B.k Falsafi, M. Payer, and A. Kurmus. 2019. SMoTherSpectre: exploiting speculative execution through port contention. *arXiv preprint arXiv:1903.01843* (2019).

[5] D. Boneh, R. DeMillo, and R. Lipton. 1997. On the Importance of Checking Cryptographic Protocols for Faults *(Advances in Cryptology - EUROCRYPT '97)*.

[6] F. Brasser, B. El Mahjoub, A. Sadeghi, C. Wachsmann, and P.k Koeberl. 2015. TyTAN: tiny trust anchor for tiny devices. In *52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*.

[7] F. Brasser, D. Gens, P. Jauernig, A. Sadeghi, and E. Stapf. 2019. SANCTUARY: ARMing TrustZone with User-space Enclaves.. In *NDSS*.

[8] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A. Sadeghi. 2017. Software Grand Exposure: SGX Cache Attacks Are Practical. In *11th USENIX Workshop on Offensive Technologies (WOOT 17)*. Vancouver, BC.

[9] M. Chiappetta, E.y Savas, and C. Yilmaz. 2016. Real Time Detection of Cache-based Side-channel Attacks Using Hardware Performance Counters. *Appl. Soft Comput.* (2016).

[10] V. Costan and S. Devadas. 2016. Intel SGX Explained. *IACR Cryptology ePrint Archive* (2016).

[11] V. Costan, I. Lebedev, and S. Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security Symposium*.

[12] K. Eldefrawy, G. Tsudik, A. Francillon, and D. Perito. 2012. SMART: Secure and Minimal Architecture for (Establishing Dynamic) Root of Trust. In *NDSS*.

[13] Mohammad Rahmani Fadiheh, Dominik Stoffel, Clark Barrett, Subhasish Mitra, and Wolfgang Kunz. 2018. Processor Hardware Security Vulnerabilities and their Detection by Unique Program Execution Checking. *arXiv preprint arXiv:1812.04975* (2018).

[14] K. Gandolfi, C. Mourtel, and F. Olivier. 2001. Electromagnetic Analysis: Concrete Results. In *Proceedings of 3rd International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*.

[15] B. Gras, K. Razavi, H. Bos, and C. Giuffrida. 2018. Translation Leak-aside Buffer: Defeating Cache Side-channel Protections with TLB Attacks. In *27th USENIX Security Symposium (USENIX Security 18)*.

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

[17] Intel. 2019. Resources and Response to Side Channel L1 Terminal Fault. https://www.intel.com/content/www/us/en/architecture-and-technology/l1tf.html.

[18] S. Islam, A. Moghimi, I. Bruhns, M. Krebbel, B. Gulmezoglu, T. Eisenbarth, and B. Sunar. 2019. SPOILER: Speculative Load Hazards Boost Rowhammer and Cache Attacks. *arXiv preprint arXiv:1903.00446* (2019).

[19] M. Joye and M. Tunstall. 2012. *Fault analysis in cryptography.*

[20] David Kaplan, Jeremy Powell, and Tom Woller. 2016. AMD memory encryption. *White paper* (2016).

[21] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer. 2018. DAWG: A defense against cache timing attacks in speculative execution processors. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[22] V. Kiriansky and C. Waldspurger. 2018. Speculative buffer overflows: Attacks and defenses. *arXiv preprint arXiv:1807.03757* (2018).

[23] P. Kocher. 1996. Timing attacks on implementations of Diffie-Hellman, RSA, DSS, and other systems. In *Annual International Cryptology Conference*.

[24] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2018. Spectre attacks: Exploiting speculative execution. *arXiv:1801.01203* (2018).

[25] P. Kocher, J. Jaffe, and B. Jun. 1999. Differential Power Analysis. In *Advances in Cryptology: Proceedings of CRYPTO'99*.

[26] P. Koeberl, S. Schulz, A. Sadeghi, and V. Varadharajan. 2014. TrustLite: A security architecture for tiny embedded devices. In *Proceedings of the Ninth European Conference on Computer Systems*. ACM.

[27] E. M. Koruyeh, K. Khasawneh, C. Song, and N. Abu-Ghazaleh. 2018. Spectre returns! speculation attacks using the return stack buffer. In *12th USENIX Workshop on Offensive Technologies WOOT 18)*.

[28] S. Lee, M. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado. 2017. Inferring Fine-grained Control Flow Inside SGX Enclaves with Branch Shadowing. In *26th USENIX Security Symposium (USENIX Security 17)*.

[29] M. Lipp, M.l Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, et al. 2018. Meltdown: Reading kernel memory from user space. In *27th USENIX Security Symposium (USENIX Security)*.

[30] S. Mangard, E. Oswald, and T. Popp. 2007. *Power Analysis Attacks: Revealing the Secrets of Smart Cards (Advances in Information Security)*.

[31] A. Markettos, C. Rothwell, B. Gutstein, A. Pearce, P. Neumann, S. Moore, and R. Watson. 2019. Thunderclap: Exploring vulnerabilities in Operating System IOMMU protection via DMA from untrustworthy peripherals. In *NDSS*.

[32] R. Martin, J. Demme, and S. Sethumadhavan. 2012. TimeWarp: Rethinking Timekeeping and Performance Monitoring Mechanisms to Mitigate Side-channel Attacks. In *Proceedings of the 39th Annual International Symposium on Computer Architecture (ISCA '12)*.

[33] J. Noorman, P. Agten, W. Daniels, A. Strackx, R.and Van Herrewege, C. Huygens, B. Preneel, I. Verbauwhede, and F. Piessens. 2013. Sancus: Low-cost trustworthy extensible networked devices with a zero-software trusted computing base. In *22nd USENIX Security symposium*.

[34] D. Osvik, A. Shamir, and E. Tromer. 2006. Cache attacks and countermeasures: the case of AES. In *RSA Conference*.

[35] C. Percival. 2009. Cache missing for fun and profit. (2009).

[36] J. Stecklina and T. Prescher. 2018. Lazyfp: Leaking fpu register state using microarchitectural side-channels. *arXiv preprint arXiv:1806.07480* (2018).

[37] Adrian Tang, Simha Sethumadhavan, and Salvatore Stolfo. 2017. CLKSCREW: exposing the perils of security-oblivious energy management. In *26th USENIX Security Symposium (USENIX Security 17)*.

[38] J. Van Bulck, M. Minkin, O. Weisse, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, T. Wenisch, Y. Yarom, and R. Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security)*.

[39] Z. Wang and R. Lee. 2007. New Cache Designs for Thwarting Software Cache-based Side Channel Attacks. *SIGARCH Comput. Archit. News* (2007).

[40] Z. Wang and R. Lee. 2008. A Novel Cache Architecture with Enhanced Performance and Security. (2008).

[41] O. Weisse, J. Van Bulck, M. Minkin, D. Genkin, B. Kasikci, F. Piessens, M. Silberstein, R. Strackx, T. Wenisch, and Y. Yarom. 2018. *Foreshadow-NG: Breaking the virtual memory abstraction with transient out-of-order execution.* Technical Report.

[42] Y. Yarom and K. Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack.. In *USENIX Security Symposium*.

[43] Joseph Yiu. 2015. ARMv8-M architecture technical overview. *ARM WHITE PAPER* (2015).

[44] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. Hou. 2016. TruSpy: Cache Side-Channel Information Leakage from the Secure World on ARM Devices. Cryptology ePrint Archive, Report 2016/980.