On the Cost-E ectiveness of TrustZone Defense on ARM Platform

Naiwei Liu[1]), Meng Yu[2], Wanyu Zang[2], and Ravi Sandhu[1]

[1] Institute for Cyber Security,
University of Texas at San Antonio, San Antonio, TX 78249, USA
naiwei liu@utsa edu
[2] Roosevelt University, Chicago, IL 60605, USA

Abstract In recent years, research e orts have been made to develop safe and secure environments for ARM platform. The ARMv8 architecture brought in security features by design. However, there are still some security problems with ARM. For example, on ARM platform, there are risks that the system is vulnerable to cache-based attacks like side-channel attacks. The success of such attacks highly depends on accurate information about the victim's cache accesses. Cortex-M series, on the other hand, have some design so that the side-channel attack can be prevented, but it also needs a security design to ensure the security of the users' privacy data. In this paper, we focus on TrustZone based approach to defend against cache-based attack on Cortex-A and Cortex-M series chips. Our experimental evaluation and theoretical analysis show the e ectiveness and efficiency of FLUSH operations when entering and leaving TrustZone, which helps in design defense framework based on our research.

Keywords: ARM platform TrustZone IoT security

1 Introduction

In Recent years, many research papers have been focusing on security design on ARM platform. Some of security framework are designed and implemented making use of TrustZone, a secure enclave provided by ARM on both Cortex-A and Cortex-M series. These defense frameworks target to memory protection, process protection and even cache protection. For example, some of the malicious users can utilize the entry/exit of the TrustZone on ARM Cortex-A, launching a cache-based attack, and compromising the message channel between victim threads and the system. As a result, some research papers target to this problem using access control of entry/exit operations, and some papers use isolated cache protection design. The research papers and their implementations can cut down the bandwidth of cache-based attack, with various level of overhead on the whole system.

On the attacker side, many threats are threatening the IoT systems and devices. Some of them focus on systems and some of them are based on ARM

I. You Ed.): WISA 2020, LNCS 12583, pp. 203–214, 2020.
https://doi.org/10.1007/978-3-030-65299-9 16

chips. Cache in these devices becomes the research focus on both single device environment and cloud with multiple devices, or even IoT network connecting smart devices. The attacks can be very e ective on extracting the users' private and secured data, without the permissions and access to the protected enclaves. Side-channel attack among them is a research focus. Malicious hackers can collect performance data, power consumption data or even some 'trash' data to try retrieving useful information. Attackers derive users' information like cryptographic keys, protected or private data by launching attack on the cache, and analyze the information from what they get. Some attackers just try to collect the di erence in access time with di erent memory blocks, and predict what is accessed frequently by the users. The di erence in access time can be collected if the attacker and the victims are sharing data in the cache.

ARM platform, on the other hand, is a di erent environment from traditional x86 structures. It has di erent privilege levels and sets some instructions as privileged operations. For example, cache FLUSH operation on ARM is privileged. On ARMv8-M based on Cortex-M structures, there is a much simpler structure of instructions than other platforms. This is because that ARMv8-M is designed to use in small smart devices. They have limited energy input and are asked to work in a long duration. Some of the devices are powered even by some batteries we can find in grocery stores, so the performance limitation is a thing that must be considered when designing something about security and privacy.

In this paper, we investigate the defense e ectiveness to cache based side-channel attacks on the ARM architecture. We design several tests based on TrustZone on both ARM cortex-A and cortex-M series chips and get the performance data. These can help in design and implementation of defense, while keeping the performance and e ectiveness balanced. Overall, we have following contributions in this paper:

- We investigate the performance overhead of TrustZone related instructions. We analyze the percentage of TrustZone instructions in real life use cases and calculate the overhead brought by these instructions;
- We test FLUSH operation overhead and analyze clock cycles they take on di erent platforms. This helps in the evaluation of cost-e ectiveness on both FLUSH-based attack and defense sides.
- We provide the best/worst case of defense performance based on our experimental results and analysis.

The structure of this paper is as follows: in Related Work section, we introduce previous research and recent research on this topic, analyzing their strong contribution and weaknesses; in Overview section, we introduce our environments of development, structure of design and security assumptions; in Implementation section, we provide some details about our design and experiment settings; in Evaluation section, we provide experimental results and discussion; and in Conclusion section we have our conclusions on the research topic.

2 Related Work

2.1 Cache Based Attack

In a cloud computing system or a computer with multiple processes and threads, the Last Level Cache (LLC) is shared among multiple processor cores, making it vulnerable to LLC based side-channel attacks. Unlike L1 cache, LLC is much slower than L1 cache, leading to more difficult set up for side channels. There are di erent ways to launch side-channel attacks, e.g., FLUSH+RELOAD [6,17], PRIME+PROBE [6,7,11], and bus-locking [16].

For example, the FLUSH+RELOAD involves three steps. The attacker first flushes one or more of the desired cache contents using processor-specific instructions (e.g. clflush on x86 processors). Second, the attacker waits for sufficient time for the victim to use (or not to use) the flushed cache area. Finally, the attacker reloads previously flushed cache lines, measuring the reload time for each one of them to infer if it was touched by the victim. FLUSH+RELOAD strategy has been proven very e ectively in many side channel attacks on x86 architecture. For example, Gulmezoglu et al. [6] recovered the AES key of OpenSSL within 15 s. Yarom and Falkner [17] recover a RSA encryption key across VMware VMs using FLUSH+RELOAD attack, and Irazoqui et al. [8] recovered AES keys using similar attack and exploiting the vulnerabilities in cache. For PRIME+PROBE attack, Work [11] recover AES keys in a cross-VM Xen 4.1 using PRIME+PROBE attack. Liu et al. [10] presented a PRIME+PROBE type side-channel attack model against the LLC, which is tested to be practical and threatens the system.

2.2 Hardware Based Defense

Bernstein [2] suggested to add L1-table-lookup instruction to load an entire table in L1 cache, and also load a selected table entry in a constant number of CPU cycles. Page [12] investigated a partitioned cache architecture. Wang and Lee [13–15] proposed new security-aware cache designs to thwart the LLC side channel attack with low overhead. In [15], the Partition-Locked cache (PLcache) was able to lock a sensitive cache partition into cache, and Random Permutation cache (RPcache) randomized the mapping from memory locations to cache sets. In [10], a novel random fill cache architecture that replaces demand fetch with random cache fill within a configurable neighborhood window was proposed. While the hardware solutions provide strong isolations between the victim and the attacker, they require special hardware features that are not immediately available form commodity processors.

2.3 Software Based Defense

Some researchers proposed to modify applications to better protect secrets from side-channel attacks. Brickell et al. [3] proposed three individual mitigation strategies: compact S-box table, frequently randomized tables, and pre-loading

of relevant cache-lines. It compressed and randomized tables for AES. However, it requires manually rewriting the AES implementation and is specific to AES. Cleemput et al. [4] applied the mitigating code transformations to eliminate or minimize key-dependent execution time variations. Crane et al. [5] proposed a software diversity technique to transform each program unique. The approach o ers probabilistic protection against both online and o -line side-channel attacks. In their work, using function or basic-block level dynamic control-flow diversity along with static cache noise results in a performance slowdown of 1.76x–2.02x compared to the baseline AES encryption when using 10%–50% cache noise insertion. Dynamic cache noise at 10%–50% has significantly impact on performance (2.39–2.87x slowdown). However, above software solutions are typically application specific or incur substantial performance overhead.

2.4 Recent Research on ARM TrustZone

In recent years, some papers have discussions and new research findings on ARM platform, especially focusing on TrustZone protection. Zhang et al. [18] proposed an Android protection framework using TrustZone on ARM, protecting VoIP phone calls. It enclaves privacy data so the phone calls cannot be intercepted easily by malicious eavesdropping. Amacher et al. [1] have evaluate the performance of ARM TrustZone using TEEs and di erent benchmarks, but the security concern is out of that paper's scope. Keystone defense framework proposed by Dayeol Lee and others [9] is a good example of defense framework based on TrustZone. It enclaves protected operations and disables sharing in TLBs and memory blocks so there's no side-channel attack based on the vulnerability here. However, the timing side-channel attack is out of that paper's scope. In our discussion, there are still risks of side-channels when exiting from TrustZone, so we need also investigate the vulnerability at the gate of security enclave.

3 Overview

3.1 Background

As multi-core processors become pervasive and the number of on-die cores increases, a key design issue facing processor architects is the security layers and policies for the on-die LLC. With LLC techniques, a CPU might only need to get around 5% data from main memory, which can improve the efficiency of CPU largely. In our implementations, we are using Intel i7-4790 processor, with 8 Mb SmartCache. On ARMv8 Cortex-A platform, we are using Juno r1 Development Platform which has one A57 and one A53 processors on the board. A57 has a 2M LLC on the processor. On Cortex-M platform, we are using ARM Cortex-M4 series chips, the development platform has 3 pipeline stages and no built-in cache.

With the increasing complexity of computing systems, as well as multiple level of memory access, some registers are designed to store some specific hardware events. These registers are usually called hardware performance counters.

We have many tools getting information from those performance counters, thus getting the performance information.

In our implementation, we use perf to collect the execution information of the programs. However, we cannot use perf for collecting timing information of memory access, since it cannot be accurate enough. On this paper we use inline assemblies and consult some related registers to measure time associated information with our side-channels.

3.2 Design on ARM Cortex A

According to our evaluation on current on-the-market systems and applications, we find out that more and more Trusted Execution Environment (TEE) technologies are being used on the implementations of secure system. Besides, most of the implementations are utilizing ARM TrustZone to protect the memory access and critical data. As we are interested in the performance overhead of defending using FLUSH operations on exiting TrustZone, the experiments should start from the measurements of using TrustZone, like the time cost and performance overhead.

Our experiments on ARM Cortex-A are in three di erent steps. For the first step, we test the cost of entering and exiting from TrustZone. After we get the exact data (clock cycles) related to TrustZone, the next step is to measure how much it takes up for the TEEs to call TrustZone related instructions or operations. On the third step, we try to clean the cache every time the system exiting from TrustZone, and see the performance overhead by these FLUSH operations added to the system. As the cache gets FLUSHed every time after the using of TrustZone, the risk of being side-channel attacked can be theoretically cut down to non-exist.

3.3 Overview on ARM Cortex M

Unlike ARM Cortex-A series chips, M-series chips have di erent structure, and with other limitations. Most IoT devices are based on Cortex-A platform, but still a rising trend that more products are using Cortex-M platform. As a result, it is still valuable to investigate the defense against malicious attackers with TrustZone. In this paper, we have similar tests on ARMv8-M platform, measuring the performance of TrustZone, as well as FLUSH operation overhead. Our experiments on Cortex-M are using ARM Versatile V2M-MPS2 Motherboard with ARM Cortex-M4 cores. It o ers 8 Mb of single cycle SRAM, and 16 Mb of PSRAM. It supports the application of di erent ARM Cortex-M classes, from Cortex-M0, to M3, M4, and M7. Besides these support, the development board supports simulation of ARMv8-M.

As mentioned above, on Cortex-M4 series chips, there is no built-in cache. However, the memory structure on M4 is di erent from other structures like x86 and Cortex-A. On that platform, memory blocks are allocated in fixed order,

taking their assigned responsibilities. It is quite di erent from dynamic allocation, and is to the consideration of power consumption and performance overhead. Among these memory blocks, some are acting as 'cache-in-memory', so we can still see them working like cache and operate some instructions to read the working status of it.

The experiments are in two di erent steps. First, we measure the time cost entering and exiting from TrustZone. Next, we implement a program with TrustZone entry/exit instructions, as well as protected running steps. We then test it with controlling of the frequency of entry/exit instructions. We measure the FLUSH operation overhead according to di erent frequencies, and discuss the defense using FLUSH when exiting from TrustZone.

3.4 Threat Model and Assumptions

In this paper, we assume that the operating system is not compromised so that the attackers are forced to use covert channels or side channels without explicitly violating access control policies enforced by the operating system or other protection mechanisms. We assume that the attacker has sufficient privilege to access the memory access time. This is also needed for the covert channel, and for the performance analysis of the covert channel.

4 Implementations

4.1 Process Structure on Cortex A Platform

As mentioned above, the very first step for our experiment is to calculate the cost of entering and exiting from the TrustZone. On ARM Cortex-A Platform, an instruction smc is used for connecting the secure world and non-secure world. While in normal non-secure world, some code could call privileged smc instruction. Then, secure world monitor will be triggered after validation. After execution of secure code, the return of the execution also calls smc to get back to the normal world. There are many open-source test platform to measure the world switch latency, and in this experiment, we use the well-known QEMU to test. It had been developed since the first patch published in 2011, and been patched by many manufacturers including Samsung, utilizing ARM TrustZone for security design.

The process structure is show at Fig. 1. When there are smc instructions trigger the TrustZone entry/exit, we trap the instructions and start using perf and other time measurement tools to calculate clock cycles they take to finish switching between trust environment and outside memory. We also FLUSH cache every time when we exit from TrustZone and see the di erence in performance overhead by di erent frequency of TrustZone related instructions.

4.2 Process Structure on Cortex M Platform

On ARM-v8 platform, SG/BXNS instructions are used to enter and exit from TrustZone. As there were almost no proper TEEs for ARMv8-M on the market as we were testing, we use a testing program instead. SG (Secure Gate) instruction is called by non-secure world code that wants to trigger TrustZone protection. Unlike Cortex-A structure, on ARMv8-M, the page table is not used, so the memory is fully mapped with di erent regions. When SG instruction is called, the reserved regions for secure world are used to execute the protected part of the code. After the secure execution within TrustZone, the code has an exit called BXNS/BLXNS (Back to Non-Secure) that can lead the execution to other region besides protected ones by TrustZone. We make use of the mechanism of this, and the structure of the testing program is as Fig. 2 shows.

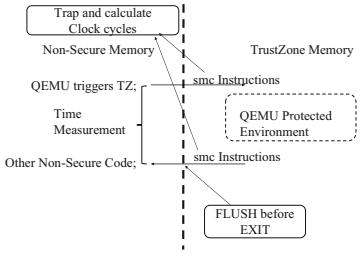

Fig 1 Process structure on Cortex-A

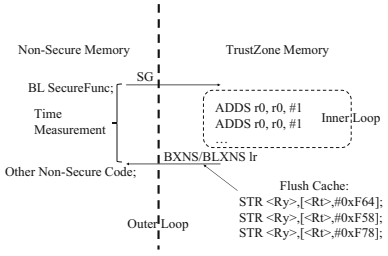

Fig 2 Process structure on Cortex-M

The term 'cache' here on ARMv8-M is part of normal memory being set as 'cacheable'. In other words, it is a region set aside for possible cache using. On Cortex-A series chips or x86 chips, cache flush operations are just some instructions with privileges. However, the case are di erent on ARMv8-M. The allocation of a memory address to a cache address is defined by the designers of the applications. Because of the special structure of ARMv8-M, the cache FLUSH operations are sets of DSB (Data Synchronization Barrier) operations, with address-related instructions.

5 Evaluation

In this section, we introduce our experimental results and discussions, both on ARM Cortex-A and Cortex-M platforms.

5.1 Experimental Results

Cost of Entering and Exiting from TrustZone on Cortex A. QEMU with ARM TrustZone provides us a variety of tests. The tests behave as we

users initiating secure operations from user mode. The test functions validate the TrustZone features of QEMU, and utilizing the features of the functions themselves. We have tests on read/write from non-secure world to secure world and vice versa. The results are shown as Table 1 shows.

Table 1 TrustZone-related instruction cost on Cortex-A

Tests	Direction	Average cost clock cycles)	Time on 800 Mhz
P0_nonsecure_check_register_access	Non-secure to Secure	1950	2.43 us
P0_secure_check_register_access	Secure to Non-secure	2200	2.75 us

Percentage of TrustZone Related Instructions. We write a script based on the above write/read code. In the script, there is a loop called in and runs several times as a workload. We use Ubuntu 16.10 as the normal world OS, with 26 processes running on background, including the workload we use for testing. We count the smc-related instructions that belongs to TrustZone-related operations, and analyze the attributions of them. According to our test, the instructions takes up less than 6% of the total instructions running, with these three di erent categories as shown on Table 2.

Table 2 Di erent categories of TrustZone-related instructions

Type	Percentage
Non-secure to Secure Test R/W	2.87%
Secure to Non-secure Test R/W	2.91%
Others (Access from Background)	0.01%

In normal using conditions, however, the manufacturers are not using TrustZone that often. Thus, the test here can be the upper bound or 'worst case' of the utilization of TrustZone-Related instructions. Normally, the non-secure world does not have to call in the secure world too often.

Performance Overhead by FLUSH Operations. It is already known that ARM TrustZone on Cortex-A series are not going to clean the cache when exiting from the secure world to non-secure world. As a result, there are possibilities for the attackers to make the most of the last level cache and conduct cache-based attacks. For example, the side-channel attack of FLUSH+RELOAD, PRIME+PROBE are both found practical on the environment with TrustZone on ARM Cortex-A, some even with a fiercely high bandwidth. On the other hand, if we can FLUSH the cache every time on the 'exit' to the normal non-secure world, then it can be expected that the bandwidth of the side-channel

attack can be limited to a number that is worthless to the attackers to gather the information possibly leaked by the smc operations.

We still test the performance using our test model. In this test, we are adding cache FLUSH operations on every smc instruction that calling exit from the secure world to non-secure world. On that situation, we measure the performance overhead by comparing the clock cycles of execution. At the same time, we change the percentage of TrustZone-related instructions to see the di erence in the overhead. The results are shown on Fig. 3 and Fig. 4.

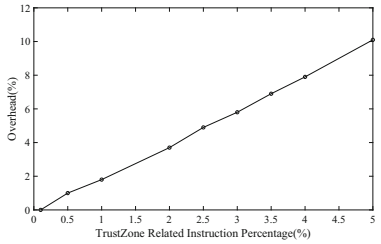

Fig 3 TrustZone related instructions and their overhead

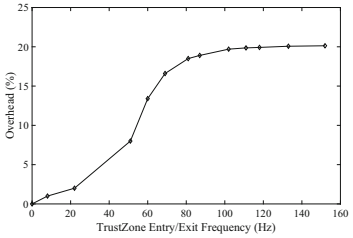

Fig 4 TrustZone entry/exit frequency and FLUSH overhead

Experimental Results on ARMv8 M. According to our experiments, the testing case triggering TrustZone operations SG and BXNS. As every region is fixed in the memory, the costs of entering and exiting from TrustZone are surprisingly much lower than ARM Cortex-A series chips. The results are shown at Table 3.

Table 3 TrustZone-related instructions cost on ARMv8-M

Operation	Direction	Cost on average (clock cycles)
SG	Non-Secure to Secure	3.5
BXNS/BLXNS	Secure to Non-Secure	5.2

We measure the performance of the FLUSH operations using our testing program shown at Fig. 2. We add FLUSH operations before executing BXNS/BLXNS operations to ensure there is nothing left when exiting from TrustZone. We measure the overhead by the FLUSH operations, and we also change the outer loop to have di erent frequencies of TrustZone entries and exits. The results are shown at Fig. 5.

5.2 Discussions

TrustZone Usage Frequency and Flush Overhead. According to our experimental results, on ARMv8 platform, the system is connecting with TrustZone with very low frequency, taking up less than 10% of the instructions at most. Some specific instructions trigger the secure gate of TrustZone. However, when the contexts running in secured memory finish, TrustZone does not clean the cache before exit, leaving some risks here. Based on low frequency and overhead from TrustZone related instructions, we can FLUSH the cache every time when exiting from TrustZone, and still keep a low overhead of less than 20% on Cortex-A chips. This design will let the system manufacturer to put protected or private contexts into TrustZone and with no worries about side-channel attack when exiting from it.

TrustZone Discussion on Cortex M. Unlike Cortex-A series, ARMv8-M based on Cortex-M structure is designed to have low energy cost and with much simpler system, which is thought to fit for mobile or home devices. At this case, the performance overhead brought by security protection should be controlled in a very low number. According to our experimental results, on Cortex-M structure, the secure gate instructions take much less clock cycles to execute, making it a good choice on the basis of security design. When we add FLUSH operations on exit instructions, we have even lower overhead comparing with Cortex-A chips, having less than 10% overhead at most. It is a practical design for the manufacturer to introduce and not hard to develop. On the other hand, they could put protected data and instructions into the secure enclave of TrustZone.

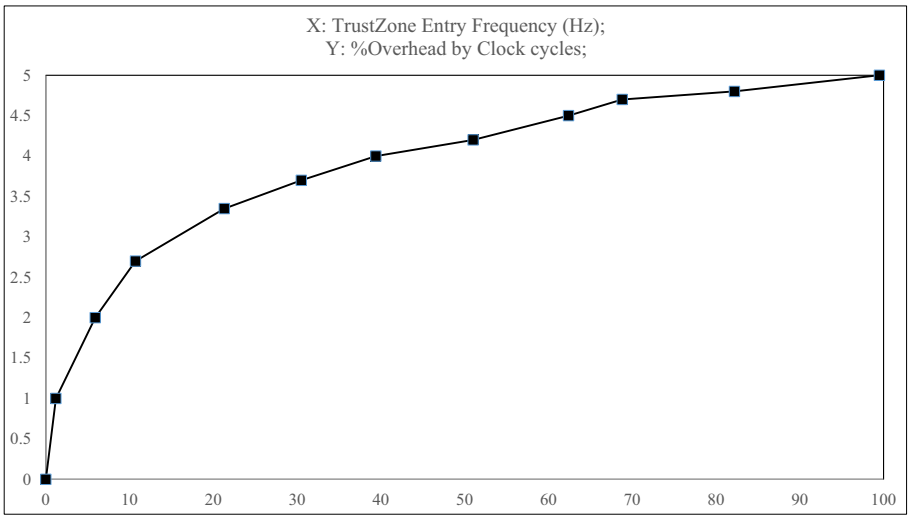

Fig 5 TrustZone entry/exit frequency and FLUSH overhead on ARMv8-M

Cache Based Defense on ARM Platform. Though we have no perfect way to take the place of validating cache and cleaning the TLB entries, we still have some idea for possible solutions, because there are some potential for speeding up and getting better performance. For example, we can move the FLUSH operations out from the privileged level, and try implementing another framework to ensure the security of this type of operations, while maintaining low overhead. In this paper, we quantitatively discuss the security design for dealing with FLUSH operation requests, and there are still some more topics to research on.

6 Conclusion

In this paper, we have some discussion on the e ectiveness and cost of attack and defense based on ARM platform. We start from investigating the cache-based attacks. Then we design and implement some tests on ARM platform, both on ARM Cortex-A and ARMv8-M series chips. It is shown that the side-channel attack and other types of exploitations are practical and serious, causing loss to users' privacy and security. From our experimental results, TrustZone can be utilized to help defending against side-channel and covert channel attacks, but it must have an adaptive ways to manage cache operations. On the other hand, it is practical to implement FLUSH based defense on ARM platform, with reasonable overhead and good e ectiveness.

In the future, we need to develop some defense framework on ARM platform, based on FLUSH operations and secure gate entry/exit instructions. The challenge will be the di erence in structures of ARMv8 platform, and real-life limitations like power consumption, portable needs and other challenges. However, it is promising that ARM platform can provide the users with an environment in balance of performance, privacy, security and good mobility as well.

Acknowledgements This paper and research project are sponsored by NSF CREST Grant HRD-1736209 and NSF Grant 1634441. The grants are for security research on cloud and systems. This research is performed in the Institute for Cyber Security (ICS) lab in University of Texas at San Antonio, and Computer Science Department in Roosevelt University.

References

1. Amacher, J., Schiavoni, V.: On the performance of ARM TrustZone. In: Pereira, J., Ricci, L. (eds.) DAIS 2019. LNCS, vol. 11534, pp. 133–151. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-22496-7_9
2. Bernstein, D.J.: Cache-timing attacks on AES, Technical report (2005)
3. Brickell, E., Graunke, G., Neve, M., Seifert, J.-P.: Software mitigations to hedge AES against cache-based software side channel vulnerabilities (2006)
4. Cleemput, J.V., Coppens, B., De Sutter, B.: Compiler mitigations for time attacks on modern x86 processors. ACM Trans. Archit. Code Optim. **8**(4), 23:1–23:20 (2012)

5. Crane, S., Homescu, A., Brunthaler, S., Larsen, P., Franz, M.: Thwarting cache side-channel attacks through dynamic software diversity. In: 22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, 8–11 February 2014 (2015)
6. Gülmezoğlu, B., İnci, M.S., Irazoqui, G., Eisenbarth, T., Sunar, B.: A faster and more realistic *Flush Reload* attack on AES. In: Mangard, S., Poschmann, A.Y. (eds.) COSADE 2014. LNCS, vol. 9064, pp. 111–126. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21476-4_8
7. Irazoqui, G., Eisenbarth, T., Sunar, B.: S$A: a shared cache attack that works across cores and defies VM sandboxing - and its application to AES. In: The Proceedings of 2015 IEEE Symposium on Security and Privacy, San Jose, CA, 17–21 May 2015, pp. 591–604. IEEE (2015)
8. Irazoqui, G., Inci, M.S., Eisenbarth, T., Sunar, B.: Wait a minute! A fast, cross-VM attack on AES. In: Stavrou, A., Bos, H., Portokalidis, G. (eds.) RAID 2014. LNCS, vol. 8688, pp. 299–319. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-11379-1_15
9. Lee, D., Kohlbrenner, D., Shinde, S., Song, D., Asanović, K.: Keystone: a framework for architecting tees. arXiv preprint arXiv:1907.10119 (2019)
10. Liu, F., Lee, R.B.: Random fill cache architecture. In: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 203–215, December 2014
11. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: 2015 IEEE Symposium on Security and Privacy, pp. 605–622, May 2015
12. Page, D.: Partitioned cache architecture as a side-channel defence mechanism (2005). Accessed 22 Aug 2005. page@cs.bris.ac.uk 13017
13. Shih, M.-W., Lee, S., Kim, T., Peinado, M.: T-SGX: eradicating controlled-channel attacks against enclave programs. In: Proceedings of the 2017 Annual Network and Distributed System Security Symposium (NDSS), San Diego, CA (2017)
14. Wang, Z., Lee, R.B.: A novel cache architecture with enhanced performance and security. In: 2008 41st IEEE/ACM International Symposium on Microarchitecture, pp. 83–93, November 2008
15. Wang, Z., Lee, R.B.: New cache designs for thwarting software cache-based side channel attacks. In: Proceedings of the 34th Annual International Symposium on Computer Architecture, ISCA 2007, pp. 494–505 (2007)
16. Wu, Z., Xu, Z., Wang, H.: Whispers in the hyper-space: high-bandwidth and reliable covert channel attacks inside the cloud. IEEE/ACM Trans. Netw. **23**(2), 603–614 (2015)
17. Yarom, Y., Falkner, K.: FLUSH+RELOAD: a high resolution, low noise, L3 cache side-channel attack. In: 23rd USENIX Security Symposium, USENIX Security 2014, San Diego, CA, August 2014, pp. 719–732. USENIX Association (2014)
18. Zhang, P., Liu, Z., Ma, C., Zhang, L., Han, D.: KPaM: a key protection framework for mobile devices based on two-party computation. In: 2019 IEEE Symposium on Computers and Communications (ISCC), pp. 1–6. IEEE (2019)