of a RISC-V processor system and run-time reconfigurable caches. Simulations are run using Mentor Graphics® ModelSim.

4.2 Reconfigurable Cache Architecture

The aim of this work is to analyze the effectiveness of run-time cache reconfigurations against cache side-channel attacks, rather than one particular cache architecture. Although we use a specific RTL implementation of a reconfigurable cache in this work, any other reconfigurable cache architecture should provide similar security in terms of impeding the success of cache side-channel attacks. A highly reconfigurable cache architecture capable of changing its configuration at a fine granularity at run-time is used in this work. The cache architecture does not employ partial reconfiguration techniques, which are specific to Field Programmable Gate Arrays (FPGA). It is built around small independent memory blocks, which can be used as tag or data storage depending on the current configuration and achieves run-time reconfigurability by changing the logical organization of the memory blocks.

The reconfigurable cache architecture is based on the caches available in the BRISC-V platform [1]. It provides the ability to change the associativity, cache line width, and the number of cache sets at run-time. Since the cache line width can change at run-time, modifications are made to the shared bus between L1 and L2 caches, and the bus interface in each cache. These modifications are necessary to properly handle fetching and writing back cache lines and to maintain cache coherence in a multi-core setup.

Another challenge faced when using a run-time reconfigurable cache is maintaining data consistency across reconfigurations. The cache architecture used in this work performs a complete cache cleanup before a reconfiguration. All the dirty cache lines are written back to the lower cache levels. After a reconfiguration, the cache is empty, and this results in low hit rates temporarily. However, this is not a feature required for the proposed defense mechanism. Even if a cache maintains all of its contents across reconfigurations, the run-time changes in the cache structure will reduce the amount of information gathered by an attacker.

The cache structure is further modified to provide a higher degree of flexibility in terms of how the reconfiguration sequence is triggered. A cache reconfiguration can be triggered by writing to a set of memory-mapped registers. This capability also enables programmable software-based methods to be used in reconfiguring the cache. In other words, attack detection algorithms and associated cache reconfiguration parameters can be implemented as software routines or hardware logic. The modified cache structure can interface with either modality (software or hardware) of detection and reconfiguration since it has an internal module that can act as a pass-through or active component. Due to the modular design, new detection schemes can be easily implemented by replacing the 'detector' module. If a software-based detection method is used, the internal trigger signals can be grounded. Figure 2 shows the architecture of the reconfigurable level 1 cache.

The reconfigurable cache architecture used in this work has some key modifications. The main changes related to run-time reconfiguration are concentrated in the 'cache_memory' module, which instantiates the dual-port SRAM blocks that store the cache contents and meta-data. Although, we mainly analyze the effectiveness of cache reconfigurations against side-channel attacks on the level 1 cache in this work, the same defense can be easily adapted to other cache levels.

Table 1: Overheads for the reconfigurable cache architecture compared to the non-reconfigurable one.

# Memory blocks	Unique configurations	Area	Fmax
20	26	+162%	-23%
40	45	+407%	-38%
80	71	+858%	-55%

For completeness of this work, we provide area overhead and reduction in operating frequency for three design points of the reconfigurable cache architecture used in this research compared to the baseline cache design in Table 1. The cache architecture allows an architect to perform a tradeoff between resource usage and the level of reconfigurability by changing the number of memory blocks and the size of a memory block, while keeping the total memory size constant. Table 1 also provides the number of unique configurations for different levels of resource usage. However, it should be noted that the defense proposed in this work is not specific to a particular cache architecture. Any other cache architecture with run-time reconfiguration capability can also be used in the same manner. Different levels of flexibility in the cache architecture may result in different levels of security. Identifying the number of different configurations required to provide sufficient security requires further analysis.

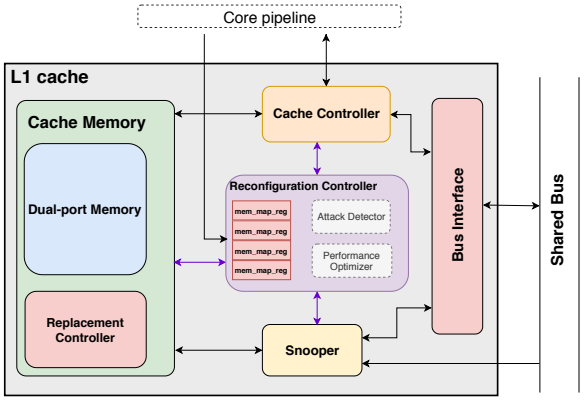

Figure 2: Reconfigurable cache architecture.

4.3 Results

In this experiment, we use the ability of the attack program to correctly identify the cache sets accessed by the victim program as the efficacy/performance metric for the attacker. The motive of the attacker is to correctly classify a higher percentage of sets as accessed or not accessed by the victim. We report the average and standard deviation for the accuracy of the attack program across 100 trials. The plots also show the sensitivity and specificity of the attack program.

Figure 3 shows the attack success under different cache configurations. The attacker is fully aware of the cache configuration and there are no run-time reconfigurations performed in these test scenarios. The baseline configuration considered is a 4-way set associative cache with 64-byte wide cache lines and 128 cache sets. The rest of the cache configurations are derived by changing one

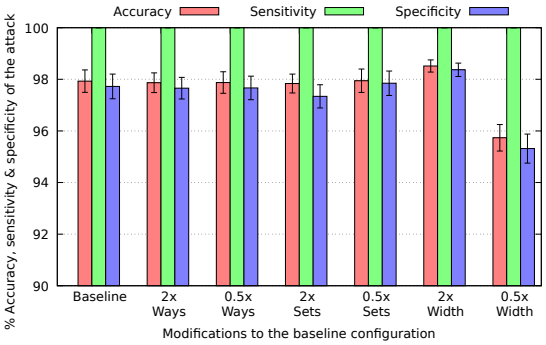

Figure 3: Baseline attack efficacy/performance.

of the parameters from the baseline configuration. The changes include doubling or halving the cache associativity, line width, or the number of cache sets. We observe that the effectiveness of the attack does not vary significantly based on the cache configuration as long as the attacker is fully aware of the exact cache configuration. Accuracy of the attack remains above 95% for all configurations. The attacker can identify all of the cache sets accessed by the victim. Hence the 100% sensitivity. However, the attacker program also incorrectly identifies a few cache sets that were never accessed by the victim program.

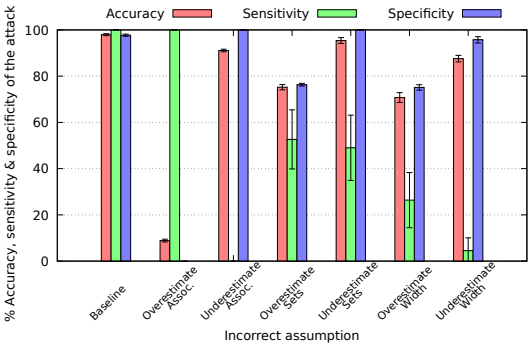

Figure 4: Attack efficacy with incorrect assumptions.

Next, we analyze the effectiveness of the attack when the attacker does not know the correct value of one cache parameter amongst associativity, line width, and number of cache sets. Figure 4 depicts the accuracy of the attack program when the attacker makes an incorrect assumption about one of the cache parameters. We observe that the accuracy and sensitivity of the attack decrease significantly when even one cache parameter is not known.

When the attacker overestimates the associativity of the cache, they construct eviction sets larger than the associativity of the cache. This leads to the attacker evicting part of its known addresses from the cache. Then, the attacker incorrectly identifies the sets that contained the addresses evicted due to their own memory accesses. This increases false positives and reduces the accuracy of the attack. By underestimating the cache associativity, the attacker fails to prime any of the cache sets and ends up identifying none of the sets accessed by the victim, hence zero sensitivity. Similarly, over- and underestimating cache sets result in either the attacker evicting their own addresses from the cache or failing to completely prime the cache. Both result in reducing the accuracy of the attack. Making an incorrect assumption regarding either the cache line width or set count results in an incorrect understanding of address mapping to cache lines. This makes the formulation of eviction sets incorrect and reduces accuracy.

The next step is analyzing the effectiveness of the attack when there are run-time cache reconfigurations. The attack program is designed assuming the baseline cache configuration. We analyze the impact on the accuracy of the attack with cache reconfigurations taking place during different phases of the 'prime+probe' attack. Reconfigurations involving a single cache parameter and multiple cache parameters are analyzed separately. The cases where only one cache parameter changes do not keep the total cache size constant. We allocate sufficient resources at the beginning to accommodate doubling cache associativity, line width, or set count at run-time. In test scenarios where the effective cache size is halved, part of the memory block is not used after the reconfiguration. For the test scenarios where multiple parameters may be reconfigured, the total cache size is kept constant across reconfigurations.

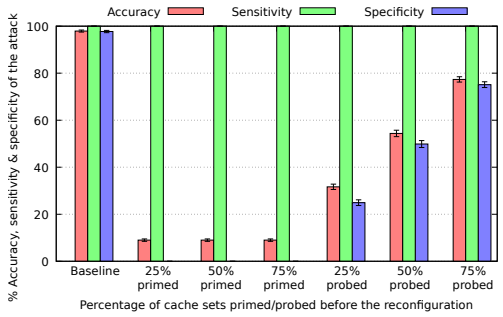

Figure 5: Attack efficacy when associativity is halved.

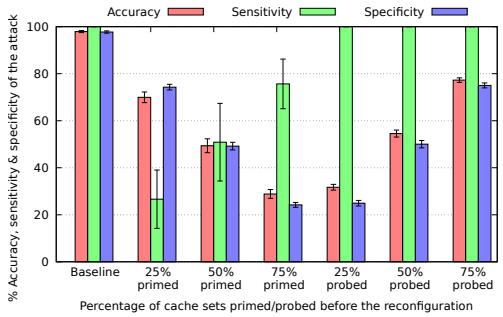

Figure 6: Attack efficacy when associativity is doubled.

Figures 5 and 6 show the impact of doubling and halving the associativity at run-time on the accuracy of the attack. Accuracy increases as the reconfiguration is delayed and the attacker is allowed to complete probing more cache sets. When the reconfiguration is triggered after 75% of the probe phase, attack accuracy is almost 80%. The impact of reconfigurations during the prime phase depends on the exact change made to the cache organization. Reducing the cache associativity results in very low accuracy for the attack. Doubling the associativity shows a lesser impact on the accuracy of the attack. However, the sensitivity of the attack is significantly reduced. Sensitivity increases as more cache sets are primed before the reconfiguration.

Figures 7, 8, 9, and 10 show the results for reconfiguring the number of cache sets and line width during different phases of the attack. We observe similar trends to those seen when reconfiguring cache associativity. Reconfigurations during the prime phase of

the attack decrease the accuracy of the attack by 71% on average. The average reduction in accuracy is 44.4% for the reconfigurations during the probe phase.

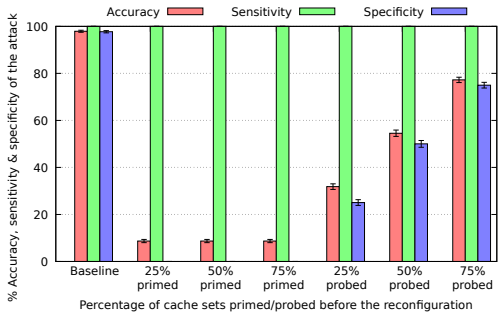

Figure 7: Attack efficacy when cache sets are halved.

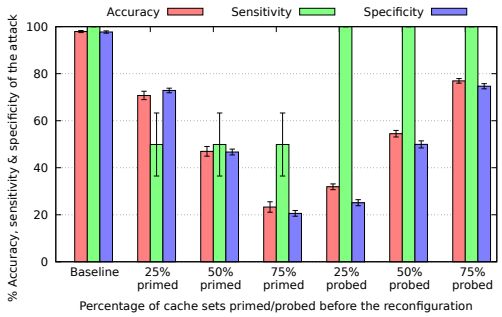

Figure 8: Attack efficacy when sets are doubled.

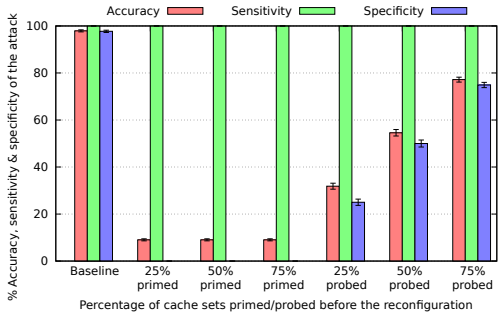

Figure 9: Attack efficacy when line width is halved.

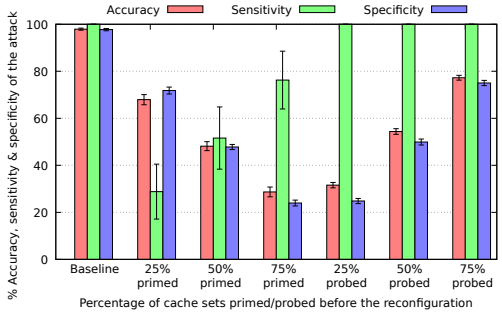

Figure 10: Attack efficacy when width is doubled.

Finally, we analyze the effect of multiple-cache parameters being reconfigured at run-time. This is more realistic because it allows the total cache size to be kept constant, unlike reconfiguring one cache parameter. Figures 11, 12 and 13 show the effect of two cache parameters being reconfigured after 25%, 50% and 75% of the cache sets are primed by the attacker. Similarly, figures 14, 15 and 16 show the effect of reconfiguring two cache parameters at the same time during the probe phase of the attack.

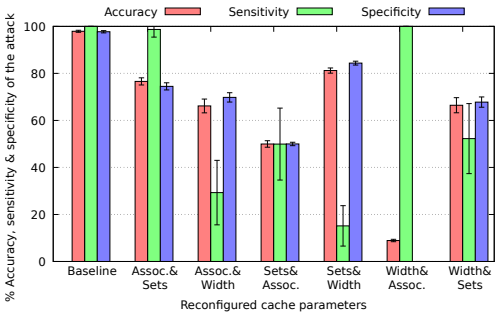

Figure 11: Attack efficacy when two parameters are reconfigured after 25% cache sets are primed.

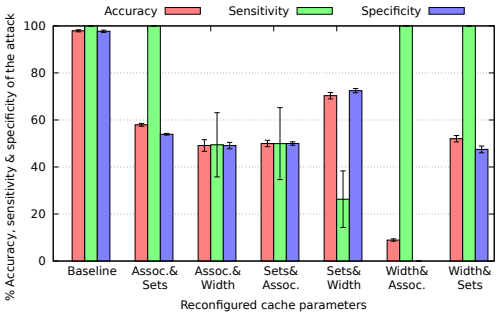

Figure 12: Attack efficacy when two parameters are reconfigured after 50% cache sets are primed.

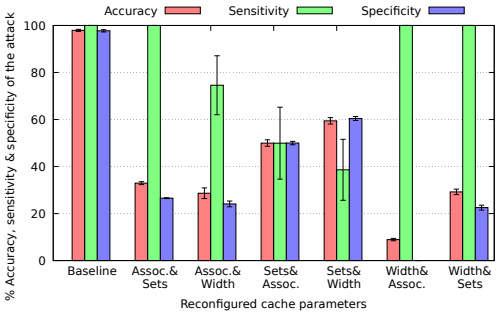

Figure 13: Attack efficacy when two parameters are reconfigured after 75% cache sets are primed.

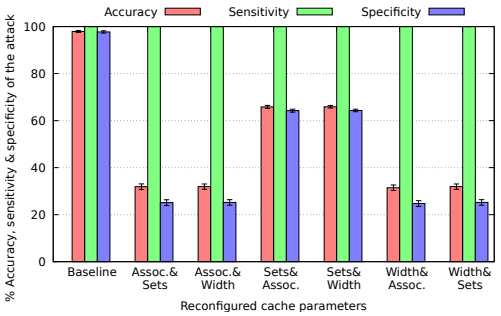

Figure 14: Attack efficacy when two parameters are reconfigured after 25% cache sets are probed.

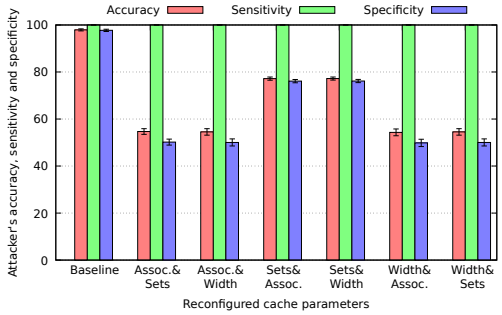

Figure 15: Attack efficacy when two parameters are reconfigured after 50% cache sets are probed.

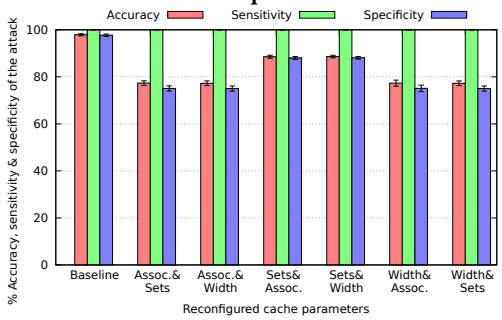

Figure 16: Attack efficacy when two parameters are reconfigured after 75% cache sets are probed.

Similar to reconfiguring one cache parameter, in the cases where the cache is reconfigured during the probe phase of the attack process, the accuracy of the attack improves as more sets can be probed before the reconfiguration. While reconfiguring different parameters results in different levels of impact on the attack performance, the overall trend of increasing accuracy with more sets probed holds. This is intuitive because the attacker is allowed to completely perform the attack on an increasing number of sets in these scenarios. When the reconfiguration takes place after 25% of the cache sets are probed, the accuracy of the attack decreased by 55%. Reconfiguring after 50% and 75% probing reduces accuracy by 36.6% and 17%, respectively.

When the cache is reconfigured during the prime phase of the attack process, the attacker fails to properly prime the cache. This increases the number of false negatives and, therefore, decreases the sensitivity of detecting sets accessed by the victim. Then the probing is done with false assumptions about the cache configuration, which compounds the errors. When the reconfiguration is done after 25%, 50% and 75% of the sets are primed, on average the accuracy of the attack decreases by 40.5%, 51%, and 64%, respectively.

4.4 Evaluation

The test scenarios with no run-time reconfigurations or other programs running on the system yield the highest success for the attacker. This was expected because there is no noise polluting the data collected by the attacker. The simplicity of the attack scenario and the cache hierarchy used in these simulations also result in high success rates for the attack. There are no virtual memory or complex cache line mapping schemes the attacker has to work around. All the caches use physical addresses to map cache lines. The processor is running bare-metal code. Adding other benign processes to the system reduces the success of the attack. This is due to caches now being shared by all the other processes. The attacker cannot distinguish between the accesses from the victim and the other processes not targeted by the attack. This observation is true for both the multi-threaded and multi-core test scenarios we have analyzed.

Run-time reconfigurations drastically reduce the attack's success. Even a change in one cache parameter renders the information gathered by the attacker more or less unusable. This is because the attack is designed assuming a certain cache configuration. Once the cache configuration changes, some of the assumptions made in designing the attack become invalid. Therefore, the information collected by the attacker becomes unreliable. A single reconfiguration was shown to be sufficient to reduce the accuracy of the attack. However, these experiments were conducted using a simple implementation of a cache side-channel attack that does not attempt to identify a cache reconfiguration and the new cache configuration after a reconfiguration. If a sophisticated implementation of the attack was used, multiple reconfigurations may be necessary. A cache with a higher number of possible configurations can counter the attacker's attempts to identify the new cache configuration and then modify the attack to fit the new configuration. When there is a large space of potential configurations, identifying the current cache configuration takes longer and that reduces the time available to mount the actual attack.

Another observation made was the effect of how long the attack continued before a cache reconfiguration takes place. The time taken by the detection mechanism to detect a potential attack and trigger a cache reconfiguration is directly related to the success of the attack. Reconfigurations earlier in probe phase result in a larger reduction in accuracy of the attack. Reconfigurations during prime phase on average result in a higher reduction in accuracy than reconfigurations during the probe phase. This demonstrates the importance of early attack detection.

5 RECONFIGURATION AS AN ALTERNATIVE RESPONSE

In this work, we assumed that there is a detection mechanism in place that will detect potential cache side-channel attacks and trigger cache reconfiguration. Future work includes implementing a real-time side-channel attack detection mechanism. Most of the prior work in real-time side-channel attack detection utilizes learning-based methods [5] [2] [27]. While prior work has shown high accuracy in attack detection, false positives are not eliminated by those detection methods. A false positive detection can result in terminating a benign process. The alternative is human intervention when the detection mechanism identifies a potentially malicious process. However, this is also not a practical solution due to the sheer number of processes running on a system. For certain use cases, such as a cloud service provider, most of the processes will be running client code, which the service provider has little to no information on. Terminating a client's benign process can have financial repercussions.

Run-time cache reconfigurations can be used to complement some of the real-time detection methods proposed in previous works. Cache reconfiguration is orthogonal to attack detection, and, therefore, the detection mechanism can be implemented independently. It can be hardware- or software-based. As discussed

earlier, terminating a process is not an acceptable response in most instances of an attack detection system flagging a process as suspicious. Reconfigurable caches provide another viable response to a potential side-channel attack. If the detection mechanism detects potential attacks with varying degrees of confidence, it is important to have a defense that does not include potentially terminating a benign application. In a system with run-time reconfigurable caches, a low confidence detection of a potential attack could trigger a cache reconfiguration. High confidence detections can still terminate the suspicious process. With this scheme in place, even if the detection method has incorrectly flagged a benign process, it will only experience performance degradation, as opposed to termination.

6 CONCLUSION

In this work, we analyzed the viability of adaptive caches as a defense mechanism against cache side-channel attacks. Our observations indicate that the success of a cache side-channel attack is severely impeded by run-time cache reconfigurations. We observed that a single cache reconfiguration can reduce the ability of the attacker to identify the cache sets accessed by the victim drastically. The accuracy of the attack decreased by 57.6% on average when a single cache parameter reconfiguration was performed without maintaining the total cache size. Reconfiguring two parameters while keeping the cache size constant resulted in an average reduction of 44.3% in accuracy. We were also able to demonstrate the importance of early attack detection with our results. Against a 'prime+probe' attack, cache reconfigurations taking place after 75%, 50%, and 25% of the probe phase resulted in decreasing accuracy for the attack. Reconfigurations during the prime phase resulted in even lower success for the attack. We discussed how run-time cache reconfigurations can be used as an alternative response to potential side-channel attacks instead of terminating the suspicious process. This reduces the potential of terminating a benign process due to a false positive from the attack detection mechanism. With a run-time cache reconfiguration, the process will experience performance degradation as opposed to being terminated. The major takeaways of this work are: (i) reconfigurable caches are an effective defense mechanism against cache side-channel attacks; (ii) they provide a strong defense with a negligible performance overhead compared to previously proposed methods; (iii) the speed of detecting a potential attack is paramount to the success of the defense; (iv) using run-time cache reconfigurations to thwart cache side-channel attacks reduces the pressure on attack detection; (v) reconfiguring even a single cache parameter reduces the accuracy of a cache side-channel attack significantly.

REFERENCES

[1] Sahan Bandara, Alan Ehret, Donato Kava, and Michel Kinsy. 2019. BRISC-V: An Open-Source Architecture Design Space Exploration Toolbox. In *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays (FPGA '19)*. ACM, New York, NY, USA, 306–306. https://doi.org/10.1145/3289602.3293991

[2] Mohammad-Mahdi Bazm, Thibaut Sautereau, Marc Lacoste, Mario Sudholt, and Jean-Marc Menaud. 2018. Cache-based side-channel attacks detection through intel cache monitoring technology and hardware performance counters. In *Third International Conference on Fog and Mobile Edge Computing (FMEC)*. IEEE, 7–12.

[3] Daniel J Bernstein. 2005. *Cache-timing attacks on AES*. Available at http://cr.yp.to/papers.html#cachetiming.

[4] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F. Wenisch, Yuval Yarom, and Raoul Strackx. 2018. Foreshadow: Extracting the Keys to the Intel SGX Kingdom with Transient Out-of-Order Execution. In *27th USENIX Security Symposium (USENIX Security 18)*. USENIX Association, Baltimore, MD, 991–1008.

[5] Marco Chiappetta, Erkay Savas, and Cemal Yilmaz. 2016. Real time detection of cache-based side-channel attacks using hardware performance counters. *Applied Soft Computing* 49 (2016), 1162–1174.

[6] Joan Daemen and Vincent Rijmen. 1999. AES Proposal: Rijndael. Available at https://csrc.nist.gov/csrc/media/projects/cryptographic-standards-and-guidelines/documents/aes-development/rijndael-ammended.pdf.

[7] Chenxi Dai and Tosiron Adegbija. 2017. Exploiting configurability as a defense against cache side channel attacks. In *2017 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*. IEEE, 495–500.

[8] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2012. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Transactions on Architecture and Code Optimization (TACO)* 8, 4 (2012), 35.

[9] Dmitry Evtyushkin, Dmitry Ponomarev, and Nael Abu-Ghazaleh. 2016. Jump over ASLR: Attacking branch predictors to bypass ASLR. In *The 49th Annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Press, 40.

[10] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+ Flush: a fast and stealthy cache attack. In *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. Springer, 279–299.

[11] Hossein Hosseinzadeh, Mihailo Isakov, Mostafa Darabi, Ahmad Patooghy, and Michel A. Kinsy. 2017. Janus: An uncertain cache architecture to cope with side channel attacks. *IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS)* (2017), 827–830.

[12] Ralf Hund, Carsten Willems, and Thorsten Holz. 2013. Practical timing side channel attacks against kernel space ASLR. In *2013 IEEE Symposium on Security and Privacy*. IEEE, 191–205.

[13] 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 preprint arXiv:1801.01203* (2018).

[14] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown. *arXiv preprint arXiv:1801.01207* (2018).

[15] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B Lee. 2015. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*. IEEE, 605–622.

[16] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. 2015. Last-Level Cache Side-Channel Attacks are Practical. In *2015 IEEE Symposium on Security and Privacy*. 605–622. https://doi.org/10.1109/SP.2015.43

[17] Ross Mcilroy, Jaroslav Sevcik, Tobias Tebbi, Ben L Titzer, and Toon Verwaest. 2019. Spectre is here to stay: An analysis of side-channels and speculative execution. *arXiv preprint arXiv:1902.05178* (2019).

[18] Yossef Oren, Vasileios P Kemerlis, Simha Sethumadhavan, and Angelos D Keromytis. 2015. The spy in the sandbox: Practical cache attacks in javascript and their implications. In *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1406–1418.

[19] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache attacks and countermeasures: the case of AES. In *Cryptographers' track at the RSA conference*. Springer, 1–20.

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

[21] R Rivest et al. 1983. Cryptographic communications system and method, US 4405829 A. (1983).

[22] Pepe Vila, Boris Köpf, and José Francisco Morales. 2018. Theory and practice of finding eviction sets. *arXiv preprint arXiv:1810.01497* (2018).

[23] Zhenghong Wang and Ruby B Lee. 2007. New cache designs for thwarting software cache-based side channel attacks. In *ACM SIGARCH Computer Architecture News*, Vol. 35. ACM, 494–505.

[24] Zhenghong Wang and Ruby B Lee. 2008. A novel cache architecture with enhanced performance and security. In *Proceedings of the 41st annual IEEE/ACM International Symposium on Microarchitecture*. IEEE Computer Society, 83–93.

[25] Yuval Yarom and Katrina Falkner. 2014. FLUSH+ RELOAD: a high resolution, low noise, L3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*. 719–732.

[26] Yuval Yarom, Daniel Genkin, and Nadia Heninger. 2017. CacheBleed: a timing attack on OpenSSL constant-time RSA. *Journal of Cryptographic Engineering* 7, 2 (2017), 99–112.

[27] S. Yu, X. Gui, and J. Lin. 2013. An approach with two-stage mode to detect cache-based side channel attacks. In *The International Conference on Information Networking 2013 (ICOIN)*. 186–191. https://doi.org/10.1109/ICOIN.2013.6496374

[28] Chuanjun Zhang, Frank Vahid, and Walid Najjar. 2003. A highly configurable cache architecture for embedded systems. In *30th Annual International Symposium on Computer Architecture, 2003. Proceedings.* IEEE, 136–146.

[29] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2012. Cross-VM side channels and their use to extract private keys. In *Proceedings of the 2012 ACM conference on Computer and communications security*. ACM, 305–316.