SCAAT: Secure Cache Alternative Address Table for mitigating cache logical side-channel attacks

Ameer Shalabi, Tara Ghasempouri, Peeter Ellervee, Jaan Raik
Department of Computer Systems, Tallinn University of Technology, Tallinn, Estonia
{ameer.shalabi,tara.ghasempouri, peeter.ellervee, jaan.raik}@taltech.ee

***Abstract*—Interest in memory systems' security has increased during the last decade due to their vulnerabilities to be exploited by logical side channels attacks. A promising approach for attack detection at run-time is to monitor the cache memory's behavior. However, designing an environment capable of detecting and mitigating these attacks is very challenging. In current monitoring systems, attack mitigation has been largely neglected. To overcome these shortcomings, in this work, we present a secure cache called SCAAT. SCAAT is equipped with an attack mitigation system to handle attacks by remapping where data is stored in the cache to random locations. In addition, SCAAT uses an attack monitor that identifies suspicious behavior that indicates cache logical side-channel attacks. The effectiveness of SCAAT is analyzed and evaluated for several cache configurations in terms of area overhead and performance.**

I. Introduction

Modern digital systems contain significant information that requires high security. In recent years, a large variety of attacks against ICs and memories have been reported. The vulnerability of caches is caused by their characteristics in resource sharing because cache states affect and are affected by all processes. Therefore, one process can infer the cache usage of another process through cache contention [1].

Today, software attacks can compromise cache security through so-called Logical Side-Channel Attacks (LSCA). Adversaries can use these attacks to understand the system‘s secrets by simply observing its behavior. Several examples of cache attacks, reported in recent years, [2–6] represent a serious threat for the semiconductor industry. Patching defenses in the field may be very costly, degrade the performance, and sometimes even create new issues (e.g. reduced battery lifetime of an IoT device) [7]. Therefore, there is a strong need for attack monitoring and mitigating systems for the cache memories.

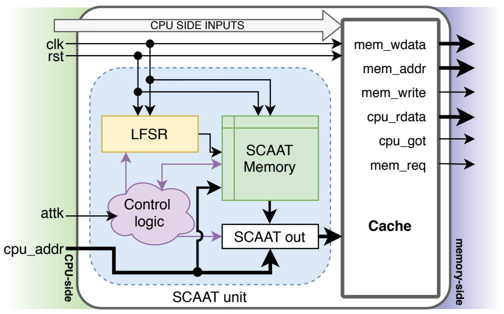


Fig. 1: Block diagram illustrating SCAAT system

Although cache monitoring has been widely studied in the context of high-performance computing, its use for mitigating cache attacks has been almost completely neglected. A limited number of publications has performed offline and online cache monitoring [8–14] for security purposes. However, none of them proposes a solution to ease and handle the attacks after the detection phase.

To this end, some works have proposed mechanisms to obtain secure caches. For instance, [15] proposes a cache-assisted secure execution system called CaSE that can protect against both software attacks and physical memory disclosure attacks on ARM-based devices. In [16], a methodology is proposed to conceal the plain text data from the L2 cache memory. It pursues to make the data unusable in the case an attacker retrieves it, and to avoid the transfer of regular patterns between CPU and cache.

On one hand, approaches in [17–19] use static partitioning where the cache is physically divided into different partitions eliminating cache interference among different applications. However, these approaches cause performance degradation. On the other hand, dynamically-partitioned caches determine the partition size dynamically at run-time. However, dynamic partitioning techniques are still vulnerable to some LSCAs. Examples of dynamically-partitioned caches can be found in [20–24].

Other approaches to gain secure caches are to randomize the memory-to-cache mapping at run-time so that an attacker cannot extract useful information from observing cache contention. However, such cache is slow and power-hungry [1]. I addition, [25] presented a mitigation approach for access-driven side-channel attacks. While the approach in [25] was successful in mitigating access-driven attacks, the mitigation strategy aimed at randomizing all accesses to the cache without directly targeting attacks, which can lead to the cache being exploited using other types of LSCAs.

All in all, in the current approaches, side-channel attacks' monitoring is not coupled with an attack mitigation mechanism and they are mainly limited to alarming the users about the possibility of data leakages.

To overcome the above shortcomings, this work proposes an innovative secure cache called the Secure Cache Alternative Address Table (SCAAT). The SCAAT is a mitigation system that is connected to a monitoring system to detect an attack and mitigate it by remapping where data is stored in the cache to random locations at run time. SCAAT mitigation strategy aims to change the behavior patterns of the cache making it harder to predict and to use these patterns to leak cache behavior-related information.

The main contributions of this paper are as follows:

- Combine a monitoring system for cache memories to detect cache LSCAs with a mitigation system to handle the attacks.

978-1-7281-9535-3/20/$31.00 ©2020 IEEE
DOI 10.1109/DSD51259.2020.00043

- Introducing a Secure Cache Alternative Address Table memory called SCAAT that remaps cache addresses to mitigate cache side-channel attacks.
- Analyzing the change of behavior of three benchmarks when executed with SCAAT to observe the false positives generated by the monitoring system and the effectiveness of SCAAT mitigation strategy.
- Analyzing the performance and area of SCAAT compared to a baseline cache.

The rest of this paper is organized as follows. Section II provides the required preliminary information. Section III presents the proposed methodology. Section IV shows the experimental results. Finally, Section V concludes this paper.

II. Preliminaries

For the purpose of testing the SCAAT, the open-source IP core cache [26] was used. The reasoning behind this choice is the parameterizable and generic set-associative nature of the cache. Relevant parameters, their calculations, and constant values used to configure the cache for this work are listed in Table I. A more detailed documentation of the cache design and operations is found in [27].

Furthermore, SCAAT uses the monitor proposed in [28]. When an attack is detected by the monitor, an attack signal $attk$ is sent to the SCAAT unit for attack mitigation. Once the attack is mitigated, the SCAAT unit output is sent to the cache as a new address.

TABLE I: Cache parameters and their definitions.

Parameter	Definition
CACHE_LINES	Number of cache lines.
ASSOCIATIVITY	Number of ways per cache set.
CACHE_SETS	Number of cache sets = $CACHE\_LINES/ASSOCIATIVITY$
CPU_ADDR_BITS	Bit-width of the address sent to the cache from CPU side.
MEM_ADDR_BITS	Bit-width of the memory address sent to the main memory from the cache.
CPU_DATA_BITS	Bit-width of cache to processor (CPU) data interface.
MEM_DATA_BITS	Bit-width of cache to the main memory data interface.
INDEX_BITS	Bit-width of index bits from the CPU address identifying the cache set where tag with TAG_BITS is being accessed. = $log_2(CACHE\_SETS)$
TAG_BITS	Bit-width of tag bits from the CPU address identifying the cache block being accessed. = $CPU\_ADDR\_BITS - INDEX\_BITS$
OFFSET_BITS	Bit-width of offset bits from the CPU address identifying which word within a cache line is being accessed. = $MEM\_DATA\_BITS/CPU\_DATA\_BITS$

III. Proposed Methodology

In this section, we propose a methodology to develop the SCAAT system: a secure cache coupled with a monitoring system to detect the attacks and a mitigating system to ease the attacks when they occur. First, a description of the functionality of the SCAAT system is provided. Second, we describe the SCAAT mitigation strategy for the detected attacks. Third, the SCAAT impact on the behavior of the cache is discussed.

A. Attack mitigation functionality

To mitigate the attacks detected by the monitoring system, the proposed SCAAT is divided into two main subsystems. The first subsystem is a direct-mapped cache or k-way associative cache. The second, a SCAAT unit, which employs a specialized control logic, a specialized memory, and an LFSR. Fig. 1 shows the structure of the SCAAT when connected to a generic direct-mapped or k-way associative cache. The SCAAT unit requires four inputs: clk and rst signals for synchronizing the SCAAT with the cache, the $attk$ signal received from the monitor to signal when an attack has occurred, and the cpu_addr (cache address) being accessed by the CPU. The SCAAT unit has one output: $SCAAT\_out$ that is directly connected to the cache address port.

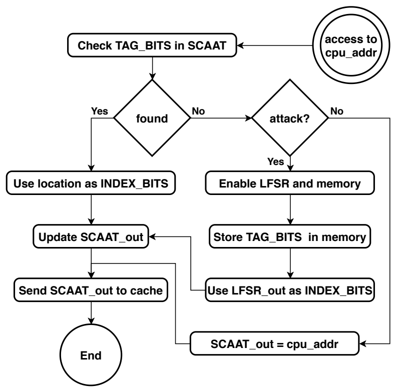


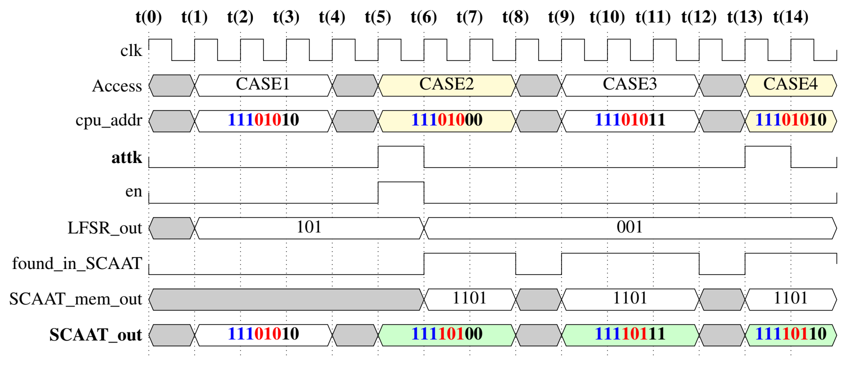
Fig. 2: SCAAT unit control logic transition diagram.

Inputs to the SCAAT unit are used to control and operate two components connected to the control logic:

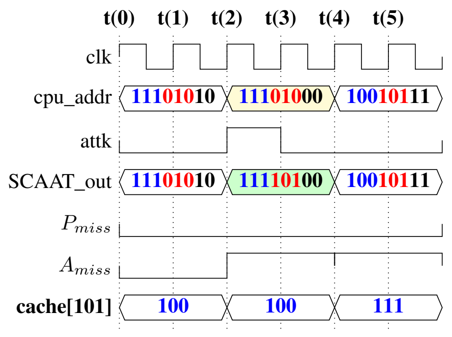
- The LFSR: used to generate pseudo-random bit-vectors of length equal to the $INDEX\_BITS$ parameter.
- The SCAAT memory: a memory module with a single write port and a single read port. It has a data width equal to TAG_BITS and depth equal to $INDEX\_BITS$. The SCAAT memory stores the TAG_BITS of cpu_addr at location $LFSR\_out$. If the input data matches any of the contents of SCAAT memory, the location of the matching data is sent as $SCAAT\_mem\_out$. An additional bit, added as the most significant bit, indicates if the data is found (represented by a logical '1') or not found (represented by a logical '0') in SCAAT memory. This bit is called $found\_in\_SCAAT$ and is used by the control logic.

Fig. 2 illustrates the control logic transition diagram of the SCAAT unit. The cpu_addr must go through the SCAAT unit regardless of the occurrence of an attack. This guarantees that all communication between the CPU and the cache is being checked by the SCAAT unit. If no attack is detected and SCAAT memory does not contain TAG_BITS of cpu_addr, the SCAAT unit is not activated and both the LFSR and the SCAAT memory are not enabled. In this case, cpu_addr is propagated through the SCAAT unit unchanged.

As shown in Fig. 2, both the LFSR and the SCAAT memory are only enabled if an attack is detected by the monitor AND the tag was not found in the SCAAT memory. When an attack occurs on tag T (i.e. TAG_BITS of cpu_addr), control logic checks if tag T has been found in SCAAT memory, if it is, its location in memory is used as $INDEX\_BITS$ of $SCAAT\_out$. If tag T is not in the SCAAT memory, the LFSR and the SCAAT memory are enabled. $LFSR\_out$ is used as SCAAT memory location and Tag T is used as memory input data. At the end of the current clock cycle, the LFSR generates a new pseudo-random number to be used in case of future attacks.



(a)



(b)

Fig. 3: Timing waveform for (a) SCAAT unit when a block is accessed four times and (b) an example scenario demonstrating change in cache access pattern.

B. Attack mitigation strategy

The timing waveform in Fig. 3a illustrates signals and data transition and propagation through the SCAAT unit. The scenario represented in Fig. 3a shows the four possible access cases to the SCAAT. Cache is accessed using 3-bit TAG_BITS (highlighted in blue), 3-bit $OFFSET\_BITS$ (highlighted in red), and 2-bit $INDEX\_BITS$ (highlighted in black). A block referenced by the tag 111 is accessed four times. A single clock cycle is 1 unit of time u donated by the function $t(u)$. Accesses under attack are highlighted in yellow. The generated output from SCAAT unit activation is highlighted in green. This example assumes that a write access takes 3 clock cycles (accesses $CASE1$, $CASE2$, and $CASE3$), while a read access takes 2 clock cycles (access $CASE4$).

$CASE1$ access to the block 111 is performed. No attacks were detected and tag 111 is not found in SCAAT memory, therefore cpu_addr is propagated through the SCAAT unit unchanged.

$CASE2$ is the next access to block with tag 111. This access was found to be under attack by the monitor. Since this is the first ever attack on cpu_addr with tag 111, the LFSR and SCAAT unit must be enabled. The current content of the LFSR register is used as $INDEX\_BITS$ of $SCAAT\_out$. This guarantee that cpu_addr is updated without the need to wait for tag 111 to be stored in SCAAT memory. The LFSR register is updated with the new value and SCAAT memory stores tag 111 at location 101. At the beginning of $t(6)$, SCAAT memory is checked for tag 111, which was stored in the previous clock cycle. At $t(6)$, the MSB of $SCAAT\_mem\_out$ indicates that tag 111 is found in memory at location 101. Once tag 111 is found in memory, the signal $found\_in\_SCAAT$ is set to '1' and location 101 is used as $INDEX\_BITS$ of $SCAAT\_out$.

$CASE3$ access is performed next, but this access was not under attack. Since tag 111 is stored in the SCAAT memory, $found\_in\_SCAAT$ is set to '1' and location 101, where tag 111 is stored, is used as $INDEX\_BITS$ of $SCAAT\_out$.

Finally, $CASE4$ access is performed, but this access, unlike the access made at $CASE3$, was under attack. $INDEX\_BITS$ of $SCAAT\_out$ are changed to reflect the location of tag 111 in SCAAT memory. However, for this access, note that even though an attack was detected by the monitor, the en signal is not set to '1' because the tag was found in SCAAT memory. This prevents generating a new pseudo-random number if an attack repeats on a tag stored in the SCAAT memory.

C. SCAAT impact on cache behavior

Introducing the SCAAT unit to the cache system aims at mitigating LSCAs. This is done by eliminating predictable patterns of cache access. By changing where data is stored in the cache for a given cache access, changes the access and timing patterns of the cache behavior.

Fig. 3b, using the same coloring scheme and address length as in Fig. 3a, shows the timing waveform of three consecutive accesses within a period of 6 clock cycles $t(0)$ to $t(5)$ in a given access pattern. Signal P_{miss} indicates the predicted miss pattern, signal A_{miss} indicates the actual miss pattern, and signal cache[101] shows the tag of the block stored at cache location 101. Assuming no attacks had ever occurred and that the cache locations 010 and 101 are populated with tags 111 and 100 respectively, all three accesses are predicted to be cache hits. However, if an attack occurs on the second access, at $t(2)$, the SCAAT unit is activated. Tag 111 is now mapped to location 101. This, of course, will cause a miss, since location 101 is populated with tag 100. Furthermore, since, after the remapping of tag 111 to location 101 occurred, the third access to tag 100, at $t(4)$, will now also cause a miss since location 101 is populated with tag 111 as a result of the previous access.

The scenario represented in Fig. 3b demonstrates the change in access pattern to the cache by invoking cache misses for accesses that are predicted to be cache hits and vise versa changes the timing pattern of cache responses to access by the CPU. Using the SCAAT unit can reduce the predictability of the cache behavior. As a result, attackers can no longer successfully correlate the information collected during an attack to the actual behavior of the cache.

IV. Experimental Results

The experimental results evaluate the proposed SCAAT in mitigating LSCAs. For the baseline cache architecture, as mentioned in Section II, the open-source IP core cache from [26] is selected. In comparison, the proposed SCAAT will be comprised of the baseline cache with a SCAAT unit fitted to the CPU-side interface of the baseline cache. Attack detection is done using the monitor from [28]. Since the target cache is parameterized, both the baseline cache and SCAAT are evaluated using two types of cache architectures: direct-mapped and 4-way associative caches.

TABLE II: Number of Attack Occurrences (AO), instances of SCAAT activation (SA), and the number of tags stored in SCAAT memory (ST) for each of the benchmarks.

	Direct-Mapped			**4-way Associative**		
	AO	SA	ST	AO	SA	ST
gcc	1556	28552	83	1544	28516	80
gzip	6	268	5	6	268	5
swim	1030	12611	33	1030	12610	33

A. Performance analysis

In order to evaluate and analyze the effect of the proposed SCAAT on the performance of the cache system, three applications from SPEC2000 CPU Benchmark Suite [29] were selected. For evaluating performance, 32-bit CPU data and address busses are connected to the baseline cache and the SCAAT. To achieve this, both a direct-mapped cache and a 4-way associative cache were configured to fit the required cpu_addr length.

Table II shows the number of attack occurrences (AO), which are occurrences of false positives generated by the monitor and are treated as attacks to be mitigated by SCAAT. The number of SCAAT activation (SA) instances includes both the mitigation of the attack occurrences as well as any accesses to the tags stored (ST) in the SCAAT memory for both cache types. All applications experience a similar number of attack occurrences for both cache types resulting in a similar number of SCAAT activation instances. This shows that the SCAAT was able to mitigate all attacks detected for both cache types.

The results for cache hit rates for the benchmarks are shown in Fig. 4.

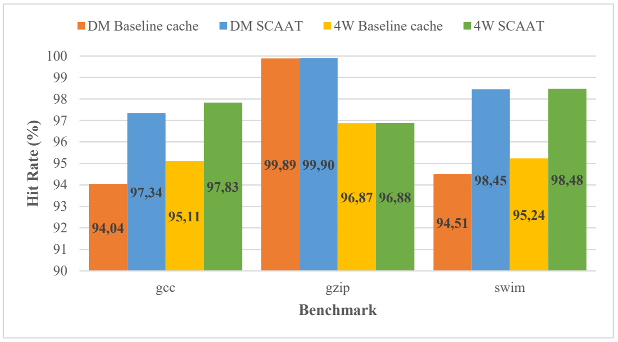


Fig. 4: Hit rate comparison of benchmarks for direct mapped (DM) and 4-way associative (4W) caches.

The results in Fig. 4 show that using the SCAAT does not have a notable effect on the overall performance of the cache. This means that using the SCAAT unit to mitigate attacks comes at no additional cost to the performance of the cache.

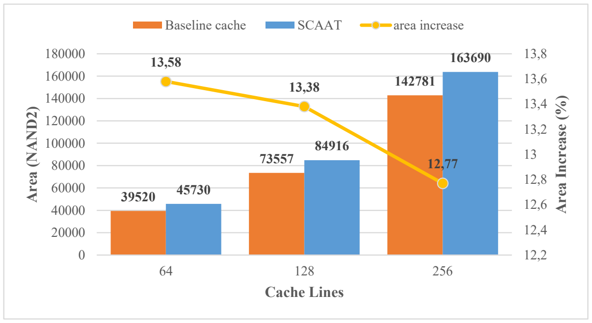
B. Area and critical path delay overheads analysis

In order to evaluate and analyze the area overhead and the critical path delay overhead of the proposed SCAAT in comparison to the baseline cache, several configurations of a 16-bit CPU_ADDR_BITS were synthesized using the Synopsys synthesis suite [30] and AMS 0.18μm CMOS technology library [31]. Table III shows the most notable configuration parameters used for area and critical path delay analysis. In addition to the parameters in Table III, all configurations have a 128-bit MEM_DATA_BITS, allowing up to 4 words per block.

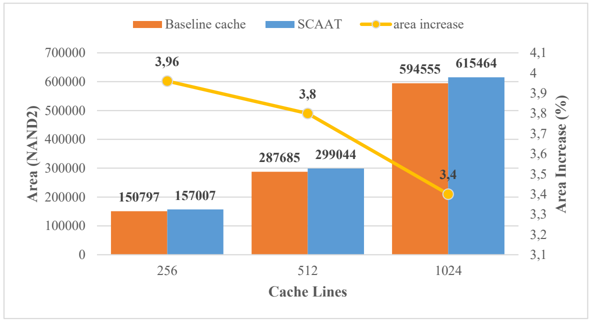
The results for area (2-input NAND (NAND2) gate area equivalent) are shown in Fig. 5. Fig. 5 (a) shows area estimation for direct-mapped baseline cache and direct-mapped SCAAT with 64, 128, and 256 cache lines each. Fig. 5 (b) shows area estimation for 4-way associative baseline cache and 4-way associative SCAAT with 256, 512, and 1024 cache lines each. Additionally, both Fig. 5 (a) and (b) include the percentage of area increase between the baseline cache and SCAAT at each respective number of cache lines. It is important to note that the area reported only includes the synthesized SCAAT unit and cache as shown in Fig 1.

TABLE III: Cache configuration for area estimation.

	Cache Type					
Parameter	Direct-Mapped			4-way associative		
Cache Lines	64	128	256	256	512	1024
Associativity	1	1	1	4	4	4
Cache (KB)	1	2	4	4	8	16
SCAAT memory (B)	64	112	192	64	112	192



(a) Area estimation for direct-mapped caches.



(b) Area estimation for 4-way associative caches.

Fig. 5: Area estimations (NAND2 equivalent) by cache type.

Fig. 5 (a) shows that an area increase between 12.77% and 13.58% occurs when the SCAAT is implemented using a direct-mapped cache architecture compared to a baseline cache of the same architecture and size. However, the percentage of area increase has been shown to decrease when the number of cache lines doubles. This can be largely attributed to the compact nature of the SCAAT memory as it only stores the TAG_BITS of cpu_addr.

Fig. 5 (b) shows that, compared to the direct-mapped caches, a much lower area increase of 3.40% to 3.96% occurs when the SCAAT is implemented using a 4-way associative cache architecture compared to a baseline cache of the same architecture and size. In addition to the SCAAT memory's compactness mentioned before, another factor for the reduction of area increase is the associativity of the cache that allows a much larger number of cache lines to be addressed using the same number of $INDEX\_BITS$. Since the size of SCAAT memory is largely determined by the $INDEX\_BITS$ parameter, a smaller SCAAT memory is needed relative to the size of the caches. This contributes to SCAAT memory taking a far less area percentage of the SCAAT system area. Implementing SCAAT with k-way associative cache architecture with high associativity can lead to a further decrease in area overhead.

Fig. 5 shows the advantage of using SCAAT in terms of area. While many of the current approaches require fixed architecture to increase the security of the cache at high area overhead, SCAAT can be constructed using any cache architecture with introducing low area overhead. In addition, the SCAAT does not require any change to the architecture of the cache itself.

As for the critical path delay, a clock period of $10ns$ is assumed for all the configurations. The evaluated path delay from cpu_addr input going through the SCAAT unit to the cache address input. On average, for each bit of cpu_addr, the delay is $4.83ns$ i.e. 0.48 of a clock cycle. This delay is caused by the sequential nature of the SCAAT memory i.e. half a clock cycle is needed to write a TAG_BITS of cpu_addr to the SCAAT memory. This delay, however, has no effect on the operation time of the SCAAT. As mentioned in Section III-A and demonstrated in Fig. 3a between $t(5)$ and $t(6)$, $LFSR\_out$ is used as $INDEX\_BITS$ of $SCAAT\_out$ in the duration of time needed to store TAG_BITS in SCAAT memory. This eliminates any effect of delay caused by the sequential nature of the SCAAT memory.

V. Conclusions

In this work, a Secure Cache Alternative Address Table (SCAAT) is proposed. SCAAT, when connected to a monitoring system, has been shown to mitigate Logical Side-Channel Attacks (LSCA) by remapping address under attack to random locations. This approach shows several advantages over current approaches to cache security. On one hand, current approaches to cache security suffer from performance degradation and area overheads, however SCAAT was shown to have little effect on the cache performance as well as introducing low area overhead between 12.77% and 13.58% for direct-mapped and between 3.40% to 3.96% for k-way associative caches. On the other hand, while current approaches require special modifications or fixed cache architecture to increase security, SCAAT can be constructed using any direct-mapped or k-way associative cache architectures with no special modifications. This makes the SCAAT a feasible and scalable option for increasing the security of the cache system.

References

[1] F. Liu *et al.*, "Newcache: Secure cache architecture thwarting cache side-channel attacks," *IEEE Micro*, vol. 36, pp. 8–16, 2016.

[2] D. J. Bernstein, "Cache-timing attacks on aes," Tech. Rep., 2005.

[3] C. Percival, "Cache missing for fun and profit," in *Proc. of BSDCan 2005*, 2005.

[4] D. A. Osvik *et al.*, "Cache attacks and countermeasures: The case of AES," in *Topics in Cryptology - CT-RSA 2006, The Cryptographers' Track at the RSA Conference 2006, San Jose, CA, USA, February 13-17, 2006, Proceedings*, 2006, pp. 1–20.

[5] D. Gruss *et al.*, "Cache template attacks: Automating attacks on inclusive last-level caches," in *Proceedings of the 24th USENIX Conference on Security Symposium*, ser. SEC'15. Berkeley, CA, USA: USENIX Association, 2015, pp. 897–912. [Online]. Available: http://dl.acm.org/citation.cfm?id=2831143.2831200

[6] Y. Yarom *et al.*, "Cachebleed: A timing attack on openssl constant time RSA," in *Cryptographic Hardware and Embedded Systems - CHES 2016 - 18th International Conference, Santa Barbara, CA, USA, August 17-19, 2016, Proceedings*, 2016, pp. 346–367. [Online]. Available: https://doi.org/10.1007/978-3-662-53140-2_17

[7] G. Doychev and B. Köpf, "Rigorous analysis of software countermeasures against cache attacks," *SIGPLAN Not.*, vol. 52, Jun. 2017.

[8] S. Deng *et al.*, "Cache timing side-channel vulnerability checking with computation tree logic," in *HASP '18*, 2018.

[9] S. Deng *et al.*, "Analysis of secure caches using a three-step model for timing-based attacks," *HaSS*, 2019.

[10] Z. He and R. B. Lee, "How secure is your cache against side-channel attacks?" in *MICRO*, 2017.

[11] M. Chiappetta *et al.*, "Real time detection of cache-based side-channel attacks using hardware performance counters," *ASC*, 2016.

[12] S. Briongos *et al.*, "Cacheshield: Protecting legacy processes against cache attacks," *CoRR*, 2017.

[13] M. Mushtaq *et al.*, "Nights-watch: A cache-based side-channel intrusion detector using hardware performance counters," in *HASP*, 2018.

[14] T. Zhang *et al.*, "Cloudradar: A real-time side-channel attack detection system in clouds," 2016.

[15] N. Zhang *et al.*, "Case: Cache-assisted secure execution on arm processors," in *2016 IEEE Symposium on Security and Privacy (SP)*. IEEE, 2016, pp. 72–90.

[16] M. Neagu *et al.*, "Interleaved scrambling technique: A novel low-power security layer for cache memories," in *2014 19th IEEE European Test Symposium (ETS)*. IEEE, 2014, pp. 1–2.

[17] R. B. Lee *et al.*, "Architecture for protecting critical secrets in microprocessors," in *ISCA'05*, June 2005.

[18] D. Zhang *et al.*, "Language-based control and mitigation of timing channels," *SIGPLAN Not.*, vol. 47, Jun. 2012.

[19] D. Zhang *et al.*, "A hardware design language for timing-sensitive information-flow security," *SIGPLAN Not.*, vol. 50, Mar. 2015.

[20] Y. Wang *et al.*, "Secdcp: Secure dynamic cache partitioning for efficient timing channel protection," in *53nd ACM/EDAC/IEEE DAC*, June 2016.

[21] L. Domnitser *et al.*, "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," *ACM TACO*, Jan. 2012.

[22] V. Kiriansky *et al.*, "Dawg: A defense against cache timing attacks in speculative execution processors," in *51st IEEE/ACM MICRO*, 2018.

[23] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," *SIGARCH Comp. Arch. News*, 2007.

[24] M. Yan *et al.*, "Secure hierarchy-aware cache replacement policy (sharp): Defending against cache-based side channel atacks," in *ISCA '17*, 2017.

[25] B. Niazmand *et al.*, "Design and verification of secure cache wrapper against access-driven side-channel attacks," in *2019 22nd Euromicro Conference on Digital System Design (DSD)*, 2019, pp. 672–676.

[26] Chair of VLSI Design, Diagnostics and Architecture. (2016) PoC - Pile of Cores. Technische Universität Dresden. [Online]. Available: https://github.com/VLSI-EDA/PoC

[27] T. B. Preußer *et al.*, "The portable open-source ip core and utility library poc," in *2016 International Conference on ReConFigurable Computing and FPGAs (ReConFig)*, 2016, pp. 1–6.

[28] C. Reinbrecht *et al.*, "Lid-cat: A lightweight detector for cache attacks," in *25th IEEE European Test Symposium*, Tallinn, ESTONIA, 2020.

[29] S. Sair and M. Charney, "Memory behavior of the spec2000 benchmark suite," Technical report, Tech. Rep., 2000.

[30] Synopsys design compiler. [Online]. Available: https://www.synopsys.com/

[31] AMS 0.18um CMOS process - AMS Process Technology. [Online]. Available: https://ams.com/process-technology