Exploiting Configurability as a Defense Against Cache Side Channel Attacks

Chenxi Dai and Tosiron Adegbija
Department of Electrical and Computer Engineering
University of Arizona, Tucson, Arizona 85721
Email: {chenxidai,tosiron}@email.arizona.edu

***Abstract*—Caches have significant impact on an embedded system's performance and energy consumption. As a result, much prior research has focused on cache optimizations to minimize energy consumption and improve performance. Caches are also highly susceptible to side channel attacks, wherein an attacker analyzes leaked information from side channels to extract private information. A key challenge of security mechanisms is that they incur overheads, which can potentially impede optimization goals. Since configurability has been widely studied as a viable and effective cache optimization, we explore using configurability as a moving target defense against cache side channel attacks, while minimizing the attendant overheads of designing secure caches. We present experimental results to show that using configurability as a defense mechanism is very promising, and present future research directions towards enabling secure caches through configurability.**

***Index Terms*—Configurable memory, hardware security, cache memories, low-power design, low-power embedded systems, adaptable hardware, side-channel attacks.**

I. Introduction and Motivation

Data encryption is often used to secure sensitive information in modern embedded systems. Typically, cryptography methods encrypt sensitive data so that the data can only be accessed using a secret key. Even though data encryption provides a high amount of security, attackers can still monitor leaked information from side channels, such as execution time, power consumption or heat, to decipher secret keys in order to decrypt sensitive information. Caches are a common target of these side channel attacks because secret keys and private data are usually stored in the cache, especially shared caches (e.g., last level caches, LLCs). An attacker can monitor the cache accesses performed by a critical process (e.g., cryptography), in order to obtain leaking information, which is then analyzed to extract private information. These attacks are especially effective since attackers do not require physical access to observe the side channels; the attacks can be launched remotely using non-privileged operations [8], [10].

Much prior work has studied cache side channel attacks, and defense mechanisms for thwarting these attacks [11], [18], [19]. These defense mechanisms typically introduce overheads, such as energy consumption or execution time. In several cases, the overheads are overlooked as long as security is achieved [14], [18], [19]. Additionally, several cache security approaches require that the caches be redesigned to support the defense mechanisms [18]. In resource constrained embedded systems, these overheads may be prohibitive.

978-1-5090-6762-6/17/$31.00©2017 IEEE

Side channel attacks typically require that a victim's processes be monitored through the side channel for a given period of time. During this period, the attacker must analyze the information obtained through the side channels, such as the mapping of memory accesses to cache sets, in order to deduce useful information about the victim process. The attacker's ability to successfully gather this information relies on a stable cache configuration and a pattern of memory accesses [10]. Given this observation, we propose that cache side channel attacks can be effectively thwarted by dynamically, and frequently, changing the runtime cache configurations. Changing the cache configurations alters the memory access patterns and introduces noise into the side channel information (e.g., energy/power profile); this noise prevents the system stability required for the attacker to successfully gather usable side channel information for accurate analysis.

In this paper, we explore a novel and low-overhead technique for securing the cache's side channels, without the concomitant overheads. We exploit *cache configurability* as a moving target defense (MTD) [22] against side channel attacks. Configurability allows the cache's configurations (cache size, line size, associativity) to be dynamically specialized/tuned to executing applications' resource requirements in order to minimize dynamic energy [9], [20]. Therefore, using configurability as a security mechanism also offers the advantage of achieving other optimization goals (e.g., cache access energy) if the configurations are carefully selected.

To this end, we present techniques for selecting appropriate cache configurations from a cache design space, and dynamically changing the configurations at runtime. We employ configurability as a defense against a recently proposed high-resolution LLC side channel attack [10], and show that our approach effectively prevents the attacker from gathering usable information for carrying out successful attacks. We also evaluate our proposed approach using the *Cache Side-channel Vulnerability (CSV)* metric [21] to quantify the amount of noise introduced by the proposed work. Furthermore, we show, through experimental results, that apart from defending against attacks, the proposed approach also achieves substantial gains in cache access energy and time, without incurring significant overheads. We show that configurability offers much promise as a security mechanism, and present future research directions

for enabling configurability as a defense mechanism against cache side channel attacks.

II. Background and Related Work

A. Cache Side Channel Attacks

In general, caches (specifically the LLC) are especially susceptible to two key types of side channel attacks: *timing analysis* [17] and *power analysis* [13] attacks. In timing attacks, an attacker observes and analyzes the time it takes to respond to various cache accesses (or a process' overall execution time) in order to extract secrets stored in the victim's cache. Similarly, power attacks monitor the power profile of the cache to identify the occurrences of cache accesses and misses, which can be analyzed (e.g., through differential power analysis [6]) to infer private information. These attacks are easy to implement, do not require special instruments, and are especially effective on embedded systems [18].

Several hardware solutions have been proposed to secure the cache side channel. Wang et al. [18] proposed a partition-based cache design (PLcache) and the random permutation cache (RPcache) for thwarting side channel attacks. The PLcache preloads critical cache lines into the cache and locks those lines, such that an attacker is unable to evict those lines or exploit timing variations from caching those lines. The RPcache randomizes the cache-to-memory mapping by storing the mapping in a permutation table and swapping entries to randomize. Similarly, Domnitser et al. [8] proposed the non-monopolizable cache—another partitioning scheme—which dynamically reserves cache lines for active threads and prevents other co-executing threads from evicting the reserved lines.

Our proposed approach is motivated by the fact that cache side channel attacks (specifically, timing and power analysis attacks) rely on the attacker's ability to consistently monitor the cache's access patterns, and timing and power behaviors through the cache side channels [19]. Thus, introducing noise into the cache access patterns, and timing and power profiles can successfully thwart these attacks. Dynamically tuning the cache configuration at runtime introduces noise by continuously changing the cache access patterns, thereby undermining the attacker's ability to obtain usable side channel profiles.

B. Configurable Caches

Configurable caches have been widely studied as a viable optimization for minimizing the cache's energy consumption [20]. Configurability allows the cache's configurations to be dynamically tuned to executing applications' (or application phases') runtime resource requirements, thereby reducing the energy overheads from over-provisioned caches.

For our work, we use a highly configurable 2 MB L2 cache with 32 KB banks, and a 16 byte physical line size. The cache provides dynamic configuration of the total cache size, associativity, and line size using small bit-width configuration registers, similar to the configurable cache proposed in [20]. The cache has physical ways implemented in individual cache banks that can be shutdown to configure the cache size

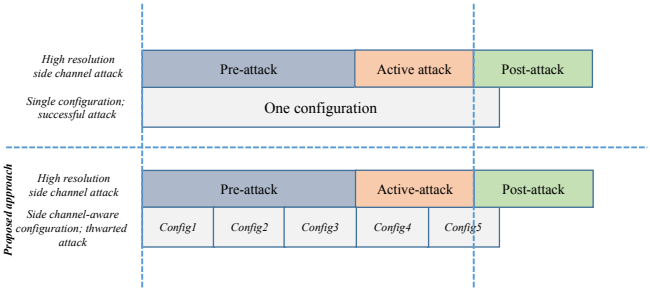

Fig. 1: Illustration of configurability as a defense

(*way shutdown*) or concatenated to configure the associativity (*way concatenation*). Given a base physical line size, multiple physical lines can be fetched to logically configure larger line sizes (*multi-line fetch*). Using this configurable cache, the overall cache energy consumption can be significantly reduced compared to a non-configurable cache [9].

Details of the configurable parameters are presented in Section IV. Augmenting the cache to support this configurability imposes minimal overheads on the cache's hardware and critical path, making it a viable option for resource-constrained embedded systems [20].

III. Configurability as a Defense

Configurable caches provide a new opportunity for thwarting side channel attacks. The proposed work has the key advantage of using a well-studied and effective low-overhead cache optimization technique as a defense mechanism. The goal is to dynamically change cache configurations in order to introduce noise and obfuscate the LLC's timing and power profile, without introducing cache access overheads with respect to the base configuration.

A. Attack Model, Assumptions, and Defense Mechanism

Figure 1 illustrates the attack model considered in this work, and how the proposed approach thwarts the attack. To represent state-of-the-art cache side channel attacks, we use an active attack model recently proposed in [10], which is based on the Prime+probe [11] attack model. The attack comprises of three stages: *pre-attack*, *active attack*, and *post-attack*. In the pre-attack stage, which relies on a stable cache configuration, the attacker first reverse engineers the index mapping of the cache, finds conflict sets for all cache lines, and then finds the page with the lookup table that the victim uses. In the active attack stage, which also depends on a stable configuration, the

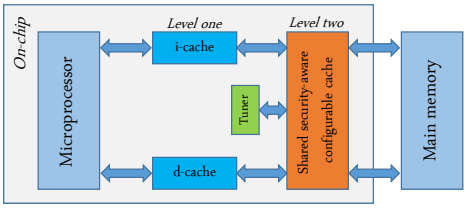

Fig. 2: Major components of the target system

attacker uses information from the pre-attack stage to prime some cache sets with data, and then probe the sets to measure the runtime, in order to determine cache hits or misses. Finally, in the post-attack stage, the attacker uses heuristics to clean up the collected data, and performs analysis to reconstruct the secret key. We direct the reader to [10] for details on this attack model.

Figure 2 depicts the major components of our target system: on-chip microprocessors with separate level one (L1) instruction and data caches with static configurations, a hardware cache tuner, and a shared configurable L2 cache (the LLC). The tuner is a low-overhead hardware structure [4] that orchestrates the tuning process for thwarting side channel attacks. The tuner also contains the *configuration table*, which stores *configuration subsets* for each application/application domain (Section III-B). We assume that the executing applications (or application domains) are known a priori, as it is common to use benchmarks to evaluate the cache design space; however, for future work, we intend to extend our approach to a more generalized system with unknown applications. Similar to the attack model, we also assume that the attacker and victim do not share a core. Allowing simultaneous multi-threading of untrusted processes exacerbates security issues, as it enables the spy to interfere synchronously with the victim [15] and exposes other side channels [3].

Unlike conventional systems where a single configuration is used throughout the attack, our defense approach changes the cache configurations during the attack, in order to introduce noise to the side channel information and obfuscate the cache access patterns and timing/energy profile. Our proposed defense targets the LLC since the LLC is shared between all cores of the system, enabling active attacks [10]. Given a last level cache with a known configuration design space, the proposed approach determines a subset of configurations that the LLC will be randomly reconfigured to at runtime. The reconfiguration will occur at intervals that are less than the minimum amount of time required by the attacker to complete the analysis of the side channel information, i.e., the pre-attack and active attack stages in our attack model (Figure 1).

By changing the cache configuration more often than the minimum required time to complete cache behavior analysis, the attacker is unable to determine collision groups and the critical cache accesses. Furthermore, even if the attacker manages to guess the correct cache configuration, the MTD model causes changes in collision groups and critical cache sets, rendering the attacker's collision groups outdated.

Our proposed approach also mitigates passive timing attacks and power analysis [18], which rely on the difference in timing of cache hits or misses, and the power difference in the bits of an encryption key being 0 or 1. With enough observed traces, the differences become statistically significant and narrows the attacker's search space for the key. Our proposed approach changes the hit and miss behavior of the cache at irregular intervals, and adds noise to the observed traces, thus making these attacks difficult.

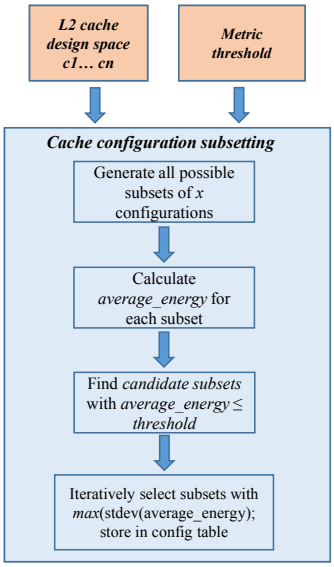

Fig. 3: Overview of design time cache configuration subsetting for determining runtime configuration groups

B. Determining the configuration subsets

We empirically found that only a small subset of configurations is required to introduce sufficient noise to the observed traces. Thus, we developed a design-time technique for determining cache configuration subsets that do not degrade the access energy/time as compared to the base configuration.

Figure 3 illustrates the design-time technique, called *cache configuration subsetting*, for determining the cache configurations for runtime use. Given the L2 cache design space and the target metric (e.g., energy consumption), we first generate all x possible combinations/subsets of configurations $c1...cn$, where n is the total number of configurations in the cache design space. x is the number of configurations in each subset, and represents the configuration variability that will be achieved during runtime. For our experiments, we empirically determined that $x = 5$ provided sufficient variability among the different configurations; larger values of x did not provide much additional variation among the different configurations.

For each configuration subset, we then calculate the average energy consumption achieved by the cache configurations within each subset for a given application or application domain. The average energy consumption is then compared to a predefined energy *threshold* to determine the *candidate subsets* that have $average\_energy \leq threshold$ (the candidate subsets contain *candidate configurations*). The threshold represents the allowed increase in energy consumption as compared to an optimization goal. For example, the threshold could be the base configuration's energy consumption; that is, the average energy must be less than or equal to the base configuration's energy consumption.

Given the subsets with $average\_energy \leq threshold$, we populate the *configuration table* entry for each application with the subset that has the largest standard deviation among the

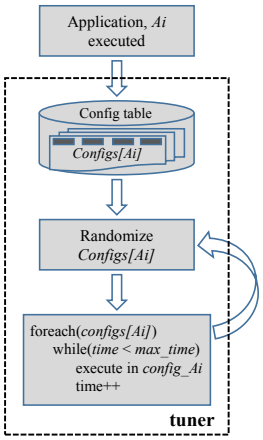

Fig. 4: Runtime cache reconfiguration process

configurations' energy values. Selecting the subsets with large standard deviations ensures that the maximum noise possible is introduced, while minimizing the overheads of ensuring security.

C. Runtime cache reconfiguration

Figure 4 depicts the runtime process for changing the cache configurations to thwart side channel attacks, as orchestrated by the tuner. When an application Ai is executed, the tuner searches for the application's configuration subset, $Configs[Ai]$ in the configuration table, which contains all applications' configuration subsets. The tuner then randomizes the configurations in $Configs[Ai]$, and executes Ai in each of the configurations for a duration *max_time*. Max_time represents a duration of time that is less than the minimum amount of time that the attacker requires to collect information about the cache characteristics (Figure 1). For example, the attack presented in [10] takes at least 155 seconds to complete. Thus, for this attack, using any arbitrary $max\_time < 155sec$ will suffice. When all the configurations have been used, the tuner re-randomizes $Configs[Ai]$, and continues the iterations until Ai is completely executed.

IV. Experiments

A. Evaluation Methodology

We implemented the proposed approach using GEM5 [5] simulations to model a quad-core embedded system microprocessor with base cache configurations similar to the ARM Cortex A15 microprocessor [1]. We used an ARM architecture to illustrate the ability of our approach to minimize overheads in resource-constrained systems; however, the proposed approach applies to other architectures as well.

TABLE I: Cache configurations.

Configuration	Parameters
Static L1 i/d caches	32 KB, 4-way, 64 byte line size 8 KB banks
Configurable L2 cache	128 KB $\rightarrow$ 2 MB, 4- $\rightarrow$ 16-way, 16 $\rightarrow$ 128 byte line size, 32 KB banks

Table I depicts the configurations modeled in our experiments. We modeled a system featuring static private L1 instruction and data caches with 32 KB size, 4-way set associativity, and 64 byte line sizes. The configurable L2 cache featured a base configuration of 2 MB size, 16-way set associativity, 128 byte line size, and 32 KB banks. The L2 cache configuration design space ranged from 128 KB to 2 MB size, 4- to 16-way set associativity, and 16 to 128 byte lines. We used CACTI [12] to determine the cache access energy and latencies for the different configurations shown in Table I.

To evaluate our approach with a variety of applications, we used nine arbitrary benchmarks from the SPEC CPU2006 benchmark suite [2], while ensuring that different kinds of applications (memory and compute intensive) were represented. We assumed that all the applications were critical and required the security mechanism. We cross-compiled the benchmarks for the ARM instruction set architecture, and executed the reference input sets for the first 300 million instructions.

We developed a program to perform the cache configuration subsetting as described shown in Figure 3 (Section III-B). We used the optimal cache configuration (determined through exhaustive search) as the baseline for the threshold. We defined the threshold as $threshold = optimal + optimal * \delta$, where $optimal$ is the access energy achieved by the optimal configuration, and δ is a fraction that represents the increase in energy consumption from the optimal. We also ensured that the $threshold$ value was always less than the base configuration's energy consumption; thus, our approach never degraded the energy configuration as compared to the base configuration. We experimented with several values of δ for each application, and determined the lowest values of δ that yielded candidate subsets. Thus, δ ranged from 0.2 to 0.7 for the different applications. Since an application may have multiple candidate subsets (for example, *bzip2* had 81 candidate subsets for $\delta = 0.4$), our program calculates each candidate subset's standard deviation, and selects the subset with the highest standard deviation for runtime use in the configurable L2 cache.

B. Results

1) Security Analysis: To enable detailed analysis, and for brevity, we only show the security analysis for *mcf* and *bzip2* to represent memory- and compute-intensive workloads, respectively. Since side channel attacks rely on an attacker's ability to recover traces of the cache's behavior [14], we first evaluated the security of the proposed approach by gathering cache traces and observing the changes in cache access patterns when using the candidate subsets as compared to the base configuration. Figure 5 depicts a snapshot of the cache access pattern for the proposed approach (*configSecurity* in the figure), using different candidate subsets, as compared to the base configuration. For illustration, we only show a snapshot of 100 cache accesses, wherein the configuration is changed after every 20 accesses (the vertical dashed lines represent the configuration transition points).

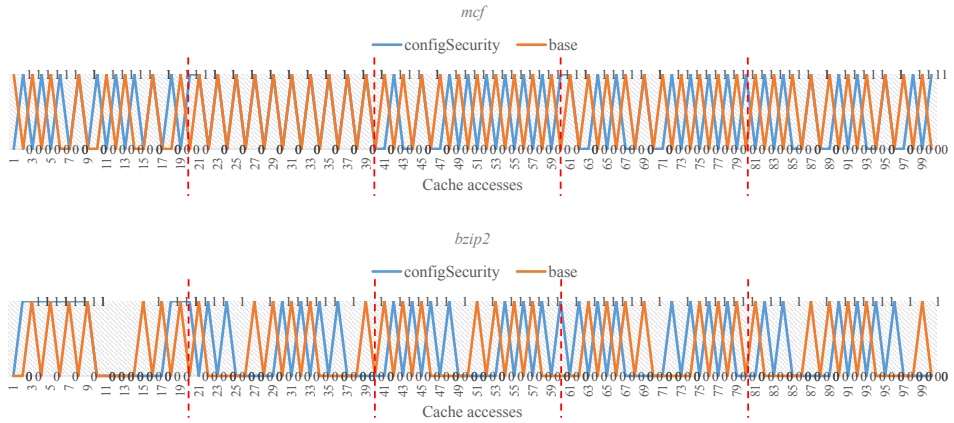

Fig. 5: Snapshot of cache access patterns for different configurations in the candidate subsets as compared to the base configuration. Vertical dashed lines represent points of configuration change; 1 denotes a cache miss; 0 denotes a cache hit

As shown in Figure 5, unlike the base configuration, which yields a relatively stable pattern that can be exploited by an attacker, our approach obfuscates the cache access patterns due to the different cache configurations. Thus, using our approach, the attacker is unable to infer the access behavior of critical cache lines (our experiments assume all cache lines are critical).

To further evaluate and quantify the security achieved by the proposed approach as compared to a base cache, we used the Cache Side-channel Vulnerability (CSV) metric. Previous work [21] has shown that the CSV is an improvement over the Side-Channel Vulnerability Factor (SVF) for measuring side channel security (we direct the reader to [21] for details on the CSV). To calculate the Pearson's correlation coefficient [21], which provides a measure of the CSV, we collected 5000 accesses in the execution trace from the base cache and the trace of the configurable cache. The Pearson's correlation, in our experiments, quantifies how much noise configurability introduces into the cache trace, where 0 means no correlation between the traces (i.e., most difficult to attack) and 1 is completely correlated (easiest to attack).

Let $v_1,\ldots,v_n$ be the configurable cache's execution trace (victim's actual trace) and $a_1,\ldots,a_n$ be the attacker's execution trace. For any access i, v_i and a_i are 0 if the access hits in the cache and 1 if the access misses. We computed the Pearson's correlation coefficient R between the two traces as follows:

$$R = \frac{\sum_{i=1}^{n}(v_i - \overline{v})(a_i - \overline{a})}{\sigma_v \times \sigma_a}$$

where $\overline{v}, \overline{a}$ are the average of victim and attacker traces and σ_a, σ_v are the standard deviations.

We found that for *mcf* and *bzip2*, R was 0.0440 and 0.4141, respectively. Even though *bzip2* had a much larger R than *mcf*, both values were small enough to ensure that the actual side channel trace was obfuscated from the attacker's view. We attribute the significant difference in R to the disparity in *mcf's* and *bzip2*'s memory access characteristics—*mcf* features a much higher runtime variability in memory accesses.

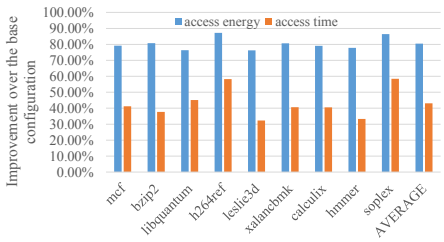

Fig. 6: L2 cache access energy and time achieved by candidate configurations as compared to the base configuration

We also analyzed the changes in energy profile achieved by the different candidate cache configurations (graphs omitted for brevity). We observed that there was significant variation in the energy profiles between different configurations; this variation undermines an attacker's ability to gather stable information from the cache side channel.

2) Access Energy and Time: As expected, employing configurability also benefited the L2 cache's access energy and time as compared to the base configuration. All the candidate subsets significantly improved over the base configuration. We also observed, however, that our approach traded off achieving the optimal configuration for security.

Figure 6 depicts the access energy and time improvement achieved by our proposed approach as compared to the base configuration for all nine benchmarks. The graph represents the average access energy and time when using all five candidate configurations for each benchmark. The configurations were randomly used throughout execution, while the base configuration remained stable in the base system.

On average over all the benchmarks, our approach improved the L2 cache access energy and time by 80% and 43% respectively. Such high improvements were possible due to the a priori knowledge of the applications, which allowed careful configuration subset selection (Section III-B). In addition, this improvement is localized to the L2 cache, and does not

necessarily represent such a magnitude of improvement for the whole memory hierarchy or the whole system. However, these results show the promise of the proposed approach to prevent side channel attacks without introducing cache access energy and time overheads.

3) Hardware and Runtime Overheads: The major sources of overhead imposed by the proposed approach are the hardware overheads resulting from the cache tuner (Section III-A), which stores the candidate configurations and orchestrates the cache configuration process (Figure 4), and the time it takes to reconfigure the cache, which can be measured in terms of the *tuning stall cycles* [16]. We have estimated, using synthesizable VHDL and Synopsys Design Compiler [7] simulations, that the proposed work introduces minimal overheads. The hardware tuner comprises less than 1% area and power overheads, as compared to the ARM Cortex A15 processor. On average, each configuration change accrued 258 tuning stall cycles, which translates to 0.136 μs.

V. Future Research Directions

The work presented herein involves some caveats and assumptions that present opportunities for future research. First, we assumed that the executing applications are known a priori. This assumption limits the applicability of the proposed approach to general purpose embedded systems that execute a wide variety of applications, many of which may not be known at design time. Thus, one key future research direction is extending the proposed approach to systems with unknown applications. Second, further exploration is necessary to consider tradeoffs in changing the number of configurations, x, within each candidate subset. In this work, we empirically determined that $x = 5$ was sufficient, however, different kinds of workloads may require a different number of configurations. Additionally, it may be necessary to have a variable x to fully capture a system's runtime security needs.

Another key variable that necessitates further exploration is how often the cache must be reconfigured in order to thwart side channel attacks. While the proposed approach suffices for a system with a known attack model, we intend to study how configurability can be applied to systems where the attack model is not known beforehand, or systems in which the attack model changes during runtime. We intend to further explore the configuration variability within each candidate subset, and explore how much variability is required to ensure security. Finally, we plan to explore the interplay between L1 and L2 cache configurability for achieving both energy optimization and security, without introducing significant overheads.

VI. Conclusions

In this paper, we explored using configurability as a defense against side channel attacks on the last level cache. Configurability provides a moving target defense, wherein cache configurations are dynamically changed at runtime to obfuscate the cache's behavior. Thus, an attacker is prevented from obtaining sufficient information with which to analyze the cache's characteristics. We presented analysis to show that security can be achieved using configurability, while preventing the concomitant overheads of traditional hardware security techniques (e.g., energy and time overheads). Since we assumed a system where executing applications are known at design time, we presented future research directions for enabling the proposed approach in a system where the runtime applications are not known a priori.

References

[1] Arm. http://www.arm.com. Accessed: December 2016.

[2] Spec cpu2006. http://www.spec.org/cpu2006. Accessed: January 2016.

[3] O. Acıiçmez, W. Schindler, and Ç. K. Koç. Cache based remote timing attack on the aes. In *Cryptographers Track at the RSA Conference*, pages 271–286. Springer, 2007.

[4] T. Adegbija, A. Gordon-Ross, and M. Rawlins. Analysis of cache tuner architectural layouts for multicore embedded systems. In *Performance Computing and Communications Conference (IPCCC), 2014 IEEE International*, pages 1–8. IEEE, 2014.

[5] N. Binkert, B. Beckmann, G. Black, S. K. Reinhardt, A. Saidi, A. Basu, J. Hestness, D. R. Hower, T. Krishna, S. Sardashti, et al. The gem5 simulator. *Computer Architecture News*, 40(2):1, 2012.

[6] M. Bucci, R. Luzzi, M. Guglielmo, and A. Trifiletti. A countermeasure against differential power analysis based on random delay insertion. In *2005 IEEE International Symposium on Circuits and Systems*, pages 3547–3550. IEEE, 2005.

[7] D. Compiler. Synopsys inc, 2000.

[8] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev. Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4):35, 2012.

[9] A. Gordon-Ross and F. Vahid. A self-tuning configurable cache. In *Proceedings of the 44th annual Design Automation Conference*, pages 234–237. ACM, 2007.

[10] M. Kayaalp, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel. A high-resolution side-channel attack on last-level cache. In *Proceedings of the 53rd Annual Design Automation Conference*, page 72. ACM, 2016.

[11] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee. Last-level cache side-channel attacks are practical. In *IEEE Symposium on Security and Privacy*, pages 605–622, 2015.

[12] N. Muralimanohar, R. Balasubramonian, and N. P. Jouppi. Cacti 6.0: A tool to model large caches. *HP Laboratories*, pages 22–31, 2009.

[13] M. Neagu, L. Miclea, and S. Manich. Defeating simple power analysis attacks in cache memories. In *Design of Circuits and Integrated Systems (DCIS), 2015 Conference on*, pages 1–6. IEEE, 2015.

[14] D. Page. Partitioned cache architecture as a side-channel defence mechanism. *IACR Cryptology ePrint Archive*, 2005:280, 2005.

[15] C. Percival. Cache missing for fun and profit, 2005.

[16] M. Rawlins and A. Gordon-Ross. An application classification guided cache tuning heuristic for multi-core architectures. In *Design Automation Conference (ASP-DAC), 2012 17th Asia and South Pacific*, pages 23–28. IEEE, 2012.

[17] V. Saraswat, D. Feldman, D. F. Kune, and S. Das. Remote cache-timing attacks against aes. In *Proceedings of the First Workshop on Cryptography and Security in Computing Systems*, pages 45–48. ACM, 2014.

[18] Z. Wang and R. B. Lee. New cache designs for thwarting software cache-based side channel attacks. In *ACM SIGARCH Computer Architecture News*, volume 35, pages 494–505. ACM, 2007.

[19] Y. Yarom and K. Falkner. Flush+ reload: a high resolution, low noise, l3 cache side-channel attack. In *23rd USENIX Security Symposium (USENIX Security 14)*, pages 719–732, 2014.

[20] C. Zhang, F. Vahid, and W. Najjar. A highly configurable cache architecture for embedded systems. In *Computer Architecture, 2003. Proceedings. 30th Annual International Symposium on*, pages 136–146. IEEE, 2003.

[21] T. Zhang, F. Liu, S. Chen, and R. B. Lee. Side channel vulnerability metrics: the promise and the pitfalls. In *Proceedings of the 2nd International Workshop on Hardware and Architectural Support for Security and Privacy*, page 2. ACM, 2013.

[22] R. Zhuang, S. A. DeLoach, and X. Ou. Towards a theory of moving target defense. In *Proceedings of the First ACM Workshop on Moving Target Defense*, pages 31–40. ACM, 2014.