SCAM: Secure Shared Cache Partitioning Scheme to Enhance Throughput of CMPs

Varun Venkitaraman[1][a], Rishab Ravi[1][b], Tejeshwar Bhagatsing Thorawade[1][c], Nirmal Kumar Boran[2][d] and Virendra Singh[1][e]

[1]*Indian Institute of Technology Bombay, Mumbai, India*

[2]*National Institute of Technology Calicut, Kozhikode, India*

Keywords: Cache Security, Hardware Security, Cache Partitioning, Secure and High-Performance Cache Management Strategy, Cache-Based Side-Channel Attacks.

Abstract: Utility-based dynamic cache partitioning scheme (UCP) improves performance in chip multiprocessors (CMPs) by dynamically way-partitioning the shared Last-Level Cache (LLC) based on each core's utility. At the end of every phase, UCP allocates more ways to the core with higher utility. However, the process of transferring ownership of a cache way from low utility core to high utility core on a cache miss (when there is partition decision change) creates side channels, making shared LLCs vulnerable to data leaks. PASS-P addresses these vulnerabilities by invalidating cache lines before transferring ownership from one core to another after partition change. While it provides security, our analysis shows that PASS-P does not always choose the best cache line for transfer, leaving room for improving performance. To improve performance of the system without compromising on security, we propose **SCAM**, a **s**ecure shared **c**ache p**a**rtitioning sch**em**e. SCAM optimizes the process of selection of transfer candidates, improving performance over PASS-P while maintaining security. SCAM achieves up to 4% performance improvement over PASS-P and reduces LLC misses per kilo instructions (MPKI) by up to 5%. SCAM offers an efficient solution for secure dynamic way-partitioning in shared caches of multi-core systems. It provides better performance without compromising security, making it an effective strategy for protecting against side-channel attacks while ensuring optimal cache utilization.

1 INTRODUCTION

The increasing prevalence of computing systems across diverse applications necessitates robust security measures, particularly for systems handling sensitive data. Extensive research has explored vulnerabilities across the system stack, with a specific focus on hardware security. Prior investigations have revealed numerous hardware flaws, notably side-channel attacks (Percival, 2005; Wang and Lee, 2007; Kong et al., 2008; Ashokkumar et al., 2016; Tromer et al., 2010; Boran et al., 2021; Boran et al., 2022), which exploit unintended inter-program interactions. For instance, timing variations in Advanced Encryption Standard (AES) execution can be leveraged to extract private encryption keys (Bernstein, 2005). These attacks utilize various channels, including execution time, memory access latencies, power consumption, and electromagnetic radiation, to infer sensitive information.

This work focuses on cache-based side-channel attacks. Techniques like Flush+Reload (Yarom and Falkner, 2014) and Prime+Probe (Liu et al., 2015) exploit cache access timing analysis to extract sensitive data. Flush+Reload involves flushing and subsequently reloading specific cache lines to detect timing differences. Prime+Probe populates the cache with attacker-controlled data and probes for timing variations resulting from cache hits or misses. These attacks infer victim program memory locations based on cache hit/miss latency differences.

Prior research has explored mitigating cache-based side-channel attacks, primarily through cache partitioning and randomization (Wang and Lee, 2007). This work focuses on cache partitioning. Static partitioning (Page, 2005), a basic technique, divides cache sets among processes, preventing inter-

[a] https://orcid.org/0000-0002-9871-0638
[b] https://orcid.org/0009-0008-2364-1499
[c] https://orcid.org/0009-0004-3608-0281
[d] https://orcid.org/0000-0003-3942-7899
[e] https://orcid.org/0000-0002-7035-7844

Venkitaraman, V., Ravi, R., Thorawade, T. B., Boran, N. K. and Singh, V.
SCAM: Secure Shared Cache Partitioning Scheme to Enhance Throughput of CMPs.
DOI: 10.5220/0013518700003979
In *Proceedings of the 22nd International Conference on Security and Cryptography (SECRYPT 2025)*, pages 144-155
ISBN: 978-989-758-760-3; ISSN: 2184-7711

process cache sharing and thus mitigating side-channel attacks. However, this approach incurs significant performance overhead due to underutilized cache lines (Wang et al., 2016) and lacks adaptability to dynamic program cache space requirements. Dynamic cache partitioning (DCP) methods (Wang et al., 2016; Qureshi and Patt, 2006; Domnitser et al., 2012; Sanchez and Kozyrakis, 2012; Xie and Loh, 2009), such as Utility-based Cache Partitioning (UCP) (Qureshi and Patt, 2006), dynamically allocate cache resources to optimize utilization. Specifically, Utility-Based Cache Partitioning (UCP) is a dynamic partitioning mechanism that allocates cache resources based on application utility. It estimates the marginal utility of additional cache ways for each workload and adjusts allocations to maximize system throughput.

Although UCP optimizes cache utilization, the flexibility of dynamically changing the partition introduces a critical side-channel vulnerability, enabling data leaks (Boran et al., 2022). During partition changes, cache way ownership gets transferred from a core with lower utility to one with higher utility. This transfer creates an opportunity to obtain sensitive data by mounting attacks such as FLUSH+RELOAD and PRIME+PROBE, as detailed in Section 3. Preventing these exploits without significantly degrading performance is a major challenge. PASS-P identifies and mitigates this security flaw in UCP (Boran et al., 2022). To address this vulnerability, PASS-P invalidates cache lines before transferring ownership. This ensures that sensitive data is not exposed during partition changes. However, such forced invalidation of a cache way can introduce performance degradation for benign application mixes. To address this, PASS-P selects clean cache blocks for transfer. This choice minimizes off-chip writebacks, reducing unnecessary memory traffic. As a result, PASS-P reduces the performance loss without compromising on security.

While PASS-P selects clean cache blocks as transfer candidates, it overlooks the presence of dead blocks and blocks that do not trigger back invalidations. Dead blocks are those that are not accessed after being inserted into the cache, remaining idle until eviction. On the other hand, blocks that trigger back invalidations generate coherence requests when evicted, resulting in increased network traffic. By prioritizing dead blocks, along with blocks that do not cause back invalidations and clean blocks, system performance can be enhanced without compromising security. However, PASS-P's suboptimal candidate selection process undermines cache efficiency. It often fails to prioritize transfer candidates that minimize system's performance overhead, such as dead blocks and non-back-invalidation-inducing blocks, leading to inefficient cache utilization and increased performance penalties. This mismanagement significantly harms overall system performance.

Additionally, a clean cache block is not necessarily a dead block, nor does it always avoid triggering back invalidations. However, by selecting any clean block as a transfer candidate without verifying its dead status or back invalidation properties, we risk evicting blocks that could still be useful. This could also lead to the eviction of blocks in private caches, which may cause unnecessary back invalidations and increase network-on-chip (NoC) traffic. Such actions result in inefficient cache utilization and higher cache miss rates, thereby degrading system's performance. These inefficiencies are more evident when partition changes occur frequently, as suboptimal eviction decisions accumulate, further degrading performance. Without an optimal transfer candidate selection mechanism that considers re-reference likelihood and coherence impact, PASS-P fails to strike an optimal balance between security and performance.

To mitigate these challenges, we introduce SCAM (Secure Shared Cache Partitioning Scheme). SCAM is a secure cache management framework designed to enhance transfer candidate selection while maintaining security. It employs a hierarchical heuristic that prioritizes dead blocks and blocks that do not induce back invalidations, alongside clean cache blocks. This approach improves cache utilization and enhances system performance without compromising security. A comprehensive discussion of SCAM is provided in Section 4.

Cache partitioning at the LLC level can result in the underutilization of available L2 cache space, particularly in certain cache configurations. For example, when only one cache way per set is allocated to a low-utility core in the LLC, the total LLC space assigned to that core may be smaller than the available L2 cache space for the same core, leading to inefficient use of L2 cache. This is because we need to maintain inclusive nature of cache hierarchy. To avoid this issue of L2 cache space under-utilization, SCAM efficiently determines the lower bound for the number of cache ways per set assigned to each core in the LLC. This ensures optimal utilization of private L2 caches and minimizes on-chip storage wastage. Our detailed evaluations show that SCAM outperforms PASS-P, achieving up to a 4% improvement in system performance and a 5.5% reduction in LLC miss-per-kilo-instructions (MPKI). By enhancing system performance while maintaining security, SCAM offers an effective solution for secure last-level cache (LLC) management in multi-core systems, overcoming the limitations of existing methods and advancing secure

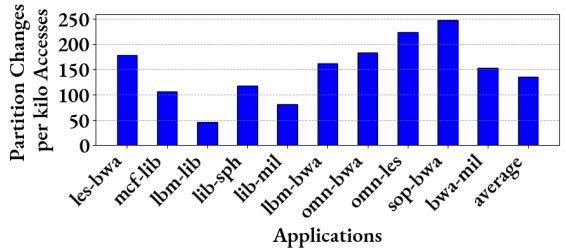

Figure 1: Number of Instances Partition Changes.

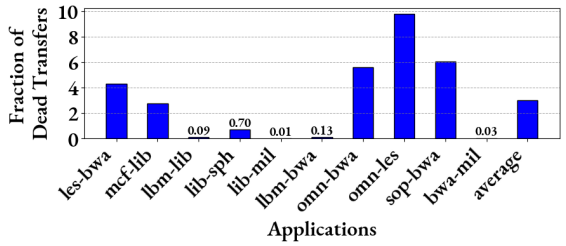

Figure 2: Fraction of PASS-P transfer candidate that is not dead.

cache management practices.

The subsequent sections of this paper are organized as follows: Section 2 delineates the limitations of the PASS-P system and articulates potential avenues for performance enhancement. Section 3 outlines the assumptions underlying an attack on UCP and details the mechanisms by which FLUSH+RELOAD and PRIME+PROBE attacks can be executed against it. Section 4 introduces SCAM, the proposed approach. Section 5 presents a comparative performance analysis of SCAM and PASS-P. Section 7 provides a review of pertinent prior research. Finally, Section 8 presents the concluding remarks.

2 MOTIVATION

In this section, we examine the limitations of PASS-P (Boran et al., 2022) and identify the research gaps that the proposed **SCAM** framework addresses. While PASS-P successfully provides security akin to static partitioning, it exhibits several shortcomings in its transfer candidate selection process, as detailed in the previous section. These limitations negatively affect the overall performance of the system. We begin by analyzing these shortcomings of PASS-P, followed by a discussion of the techniques incorporated into SCAM to mitigate these weaknesses and enhance system performance.

Before addressing the issue of suboptimal transfer candidate selection, we first analyze the frequency of LLC cache partition changes to emphasize the need for an optimal transfer candidate selection scheme.

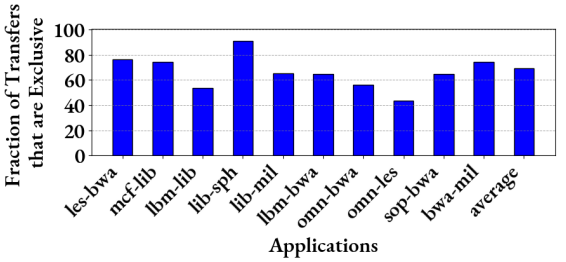

Figure 3: Fraction of PASS-P transfer candidate that does not generate Back Invalidations.

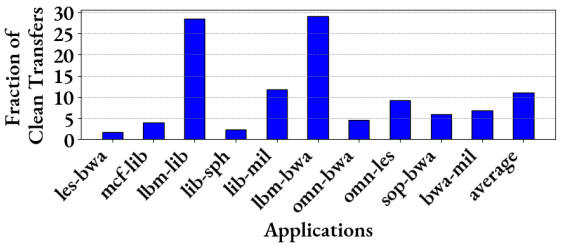

Figure 4: Fraction of PASS-P transfer candidate that is dirty.

We conduct an experiment by modeling PASS-P in the Sniper simulator (Carlson et al., 2014) and evaluate it using the application mixes listed in Table 2. The system configuration used for the evaluation is shown in Table 1. Our evaluation methodology follows the approach discussed in Section 5. Figure 1 presents the results of this experiment, with the x-axis representing the different application mixes and the y-axis showing the number of LLC cache partition changes per kilo LLC accesses. From Figure 1, we observe that the frequency of cache partition changes in PASS-P is considerably high. This highlights the critical need for an optimal transfer candidate selection scheme to reduce performance degradation.

Now to further examine the severity of issues with PASS-P, we conducted an experiment to analyze the types of cache blocks PASS-P transferred during all these partition changes at LLC. We classify the transferred cache blocks into three classes:

1. **Dead Blocks:** Dead blocks are those that are not accessed again after being inserted into the cache until eviction.
2. **Blocks Causing Back Invalidations:** Blocks causing back invalidations are those that, when evicted, generate coherence requests to private caches, increasing network traffic. These cache blocks reside in both private and shared caches.
3. **Clean Blocks:** Clean blocks are unmodified and do not require writebacks when evicted.

A comprehensive empirical analysis reveals significant limitations in PASS-P's algorithm for selecting transfer candidates. Figures 2, 3, and 4 illustrate

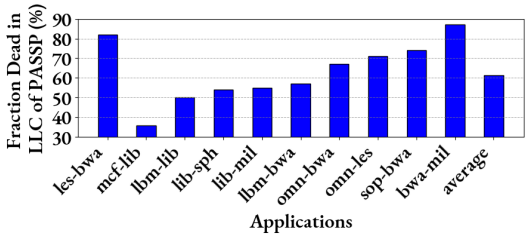

Figure 5: Percentage of Dead Cacheblocks Residing in LLC.

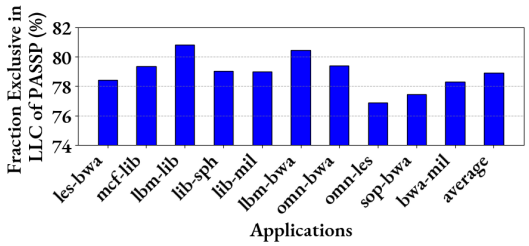

Figure 6: Percentage LLC Cacheblocks residing in Private Caches.

the frequency at which PASS-P evicts suboptimal cache blocks, including non-dead blocks, blocks that trigger back invalidations, and dirty blocks. The simulation environment and evaluation methodology are consistent with those used in the previous experiment. To generate the plots shown in Figures 2, 3, and 4, we incorporated counters into the simulator to track the number of non-dead blocks, dirty blocks, and blocks that trigger back invalidations that got transferred due to PASS-P. Additionally, the necessary infrastructure for detecting the properties of cache blocks was introduced as described in Section 4.

Figure 2 shows that PASS-P evicts non-dead blocks in approximately 3% of cases. This suboptimal decision increases LLC miss rates, leading to performance degradation as frequent cache block insertions occur at the LLC level. Furthermore, as depicted in Figure 3, PASS-P selects blocks that trigger back invalidations in nearly 30% of instances. This behavior generates unnecessary coherence traffic in the network-on-chip (NoC), further stressing system resources. In addition, Figure 4 reveals that PASS-P evicts dirty blocks in 8% of cases when no clean blocks are available within the allocated cache ways. This results in unnecessary writebacks to main memory, adding additional overhead and further diminishing performance.

The previous analysis indicates that PASS-P tends to select suboptimal transfer candidates. However, it is also necessary to investigate whether optimal transfer candidates exist in a set during a cache miss and partition change. To address this, we performed an experiment using the same setup described earlier to

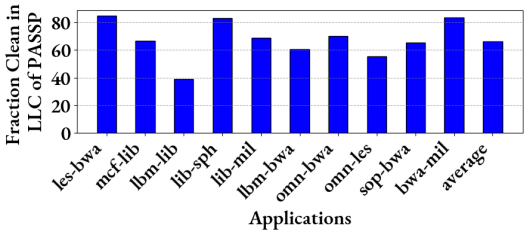

Figure 7: Percentage of Clean Cacheblocks residing in LLC.

analyze the distribution of different cache block properties within the LLC when PASS-P is in operation. Figures 5, 6, and 7 display the distribution of various cache block types—dead blocks, blocks that cause back invalidations, and clean blocks—within the LLC under the PASS-P scheme. Figure 5 reveals a substantial number of dead blocks in the LLC. These dead blocks, which have already completed their useful lifetime, are ideal candidates for eviction. Evicting dead blocks would significantly reduce performance degradation by minimizing unnecessary cache insertions and replacements. Furthermore, Figure 6 illustrates that a considerable fraction of cache blocks trigger back invalidations when evicted. These blocks generate coherence traffic, which increases network congestion and puts additional strain on the system's resources. Such traffic can contribute to system bottlenecks and reduce overall performance.

In addition, Figure 7 shows that a large portion of the LLC is occupied by clean blocks. These clean blocks can be evicted without incurring additional writebacks to main memory, which would otherwise add unnecessary system overhead. Given these observations, it becomes clear that a more effective eviction strategy is required—one that prioritizes dead blocks for eviction while minimizing the impact of blocks that cause back invalidations and unnecessary writebacks. A refined selection approach can not only enhance cache utilization but also improve overall system performance, all while maintaining the security guarantees in dynamically partitioned cache systems. Thus, optimizing the transfer candidate selection process becomes essential to achieving an efficient balance between performance and security.

To address the limitations identified in PASS-P, we introduce SCAM, an advanced cache management strategy designed to optimize transfer candidate selection. SCAM focuses on prioritizing dead blocks, as well as blocks that neither trigger back invalidations nor cause writebacks. This ensures high performance, without compromising on security. By employing a hierarchical transfer candidate selection policy, SCAM effectively reduces LLC misses, alleviates NoC congestion, and minimizes unnecessary

memory overhead. As a result, SCAM significantly boosts overall system performance, outperforming PASS-P in both efficiency and effectiveness.

In summary, SCAM marks a substantial improvement in secure cache management for shared LLCs in multi-core systems. It overcomes the key limitations of PASS-P by refining the transfer candidate selection process and optimizing cache block eviction strategies. SCAM not only enhances system performance but does so without compromising security. This makes it a more efficient and resilient solution for cache management in dynamic partitioning systems. In the following section, we will delve into the details of SCAM's design and implementation.

3 THREAT MODEL

Dynamic cache partitioning schemes are susceptible to Flush+Reload (Yarom and Falkner, 2014) and Prime+Probe (Liu et al., 2015) attacks, particularly when an attacker application can manipulate cache partitioning decisions. For instance, in UCP, an attacker can artificially modify its utility to trigger the reallocation of cache lines to or from itself. Notably, mounting these attacks does not require the attacker process to possess elevated privileges. The illustration of Flush+Reload attack is as follows. We then follow it up with the illustration of Prime+Probe attack.

3.1 Flush+Reload Attack

Flush: The attacker increases its utility to displace all but one cache line from each set, subsequently flushing the retained lines as depicted in step (a) of Fig.8.
Execute: The attacker then decreases its utility to restore the flushed lines, awaiting the victim process's execution, as illustrated in steps (b) and (c).
Reload: The attacker increases its utility again to occupy all but one line in each set and reloads specific addresses of interest, as shown in step (d). The presence of cache hits or misses on these addresses indicates the victim's memory accesses.

3.2 Prime+Probe Attack

Prime: The attacker increases its utility to evict all but one cache line from each set and populates these lines with its own data, as shown in step (a) of Fig.9.
Execute: The attacker then decreases its utility to restore the evicted lines, allowing the victim to perform its operations, as illustrated in steps (b) and (c) of Fig.9.

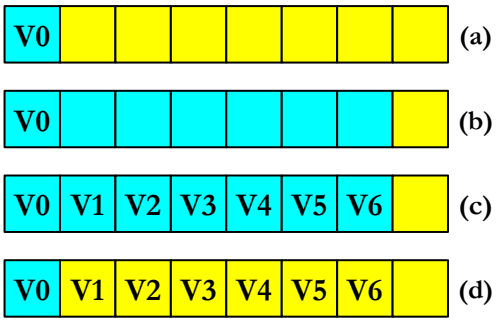

Figure 8: Flush+Reload attack.

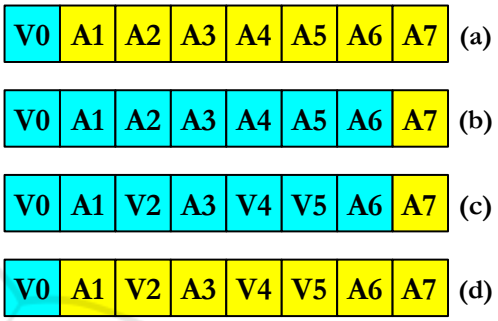

Figure 9: Prime+Probe attack.

Probe: Subsequently, the attacker increases its utility again to evict all but one line from each set and reloads the addresses that were previously primed, as depicted in step (d). The presence of a cache hit or miss on these addresses reveals information about the victim's memory accesses.

In Flush+Reload attacks, both the attacker and victim must share the same code library for the attack to be effective. This sharing enables the attacker to achieve a cache hit during the Reload phase for addresses accessed by the victim during the Execute phase. Despite dynamic cache partitioning protocols generally preventing any single process from occupying all cache lines, thus avoiding starvation, attackers can still extract sensitive information over multiple iterations. Attackers often employ techniques to significantly slow down the victim's execution, such as launching a denial-of-service attack on the Linux Completely Fair Scheduler (CFS), as detailed by Gullasch et al.(Gullasch et al., 2011). SecDCP (Wang et al., 2016) distinguishes processes into confidential and public categories, focusing on safeguarding confidential applications from side-channel attacks. However, SecDCP is still susceptible to Flush+Reload attacks. It only invalidates cache lines transferred from a public to a confidential application if those lines were accessed by the public application. Consequently, lines retrieved by the public attacker during the Reload phase are not invalidated, allowing the at-

tacker to deduce information about the victim's memory accesses. Furthermore, SecDCP does not adjust its partitioning decisions based on the demand of confidential applications, resulting in suboptimal cache partitioning and associated performance degradation.

4 PROPOSAL

UCP is a dynamic cache management technique that optimizes shared cache resources in multi-core systems by adjusting last-level cache (LLC) partitioning based on workload demands. This flexible partitioning prevents cache contention, enhances data locality, and reduces conflict misses. Unlike static schemes, UCP reallocates cache space in real-time, balancing performance and energy efficiency while improving system throughput, especially in multi-application environments. However, UCP's dynamic reallocation introduces a vulnerability. Transferring cache ownership between cores can expose memory access patterns, allowing attackers to exploit techniques like FLUSH+RELOAD or PRIME+PROBE. This side-channel risk arises from cache line transfers, which inadvertently reveal access patterns.

PASS-P mitigates these vulnerabilities by implementing a cache line invalidation strategy. It ensures that cache lines are invalidated before transferring ownership, preventing timing-based attacks during cache access. While PASS-P effectively secures cache partitioning, it still faces performance challenges related to transfer candidate selection, coherence traffic, and cache utilization.

PASS-P predominantly selects clean cache blocks for eviction, but not all clean blocks are optimal. Some might be useful in the future or could trigger unnecessary back invalidations, leading to increased memory traffic. These inefficient evictions negatively impact performance by increasing cache misses and adding load to the system. SCAM improves upon this by prioritizing dead blocks and blocks that do not cause back invalidations or writebacks, enhancing cache efficiency and reducing system overhead. SCAM achieves better performance without sacrificing security. We will now examine the specifics of SCAM, its role in LLC management, and how it facilitates efficient cache management while maintaining security.

4.1 SCAM: Secure Shared Cache Partitioning Scheme

Figure 10 illustrates the process of accessing the last-level cache (LLC) and the role of SCAM in optimizing LLC cache management scheme. When an LLC receives a memory access request, it first determines whether the requested data is present in the cache. In the case of a cache hit, the data is sent to the requesting core, while also being forwarded to higher-level caches to maintain inclusivity. Conversely, if an LLC miss occurs, the request is forwarded to the main memory, and the retrieved data must be inserted into the LLC while also being accessed by the core.

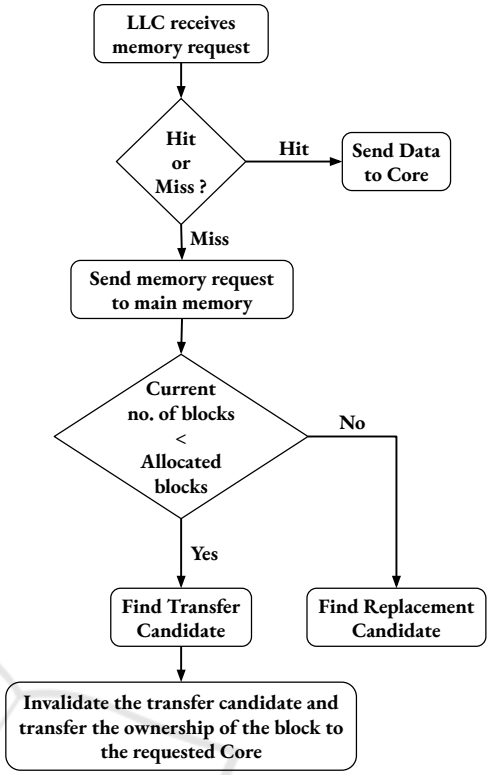

Figure 10: LLC Access Flowchart.

Before inserting new data into the LLC, several conditions must be evaluated to ensure efficient cache utilization. First, it is necessary to verify whether the number of cache ways currently owned by the requesting core is less than its allocated quota, as determined by the dynamic cache partitioning (DCP) scheme's decision engine. If the core has already reached its allocation limit, a cache block from the corresponding set must be evicted based on the replacement strategy, creating space for the new block. However, if the number of cache ways owned by the requesting core is below the allocated threshold, ownership of a cache way must be transferred from another core to the requesting core.

This transfer mechanism is precisely SCAM (Secure Shared Cache Partitioning Scheme). SCAM employs a hierarchical selection process, as outlined in *Algorithm* 1, to determine the optimal cache block

```
Function FindTransferCandidate():
    // First Priority
    for B@LRU to B@(LRU − f × curr_alloc) do
        if B.NotInPrC and B.Dead and B.Clean then
            return B
        end
    end
    // Second Priority
    for B@LRU to B@(LRU − f × curr_alloc) do
        if B.NotInPrC and B.Dead then
            return B
        end
    end
    // Third Priority
    for B@LRU to B@(LRU − f × curr_alloc) do
        if B.NotInPrC and B.Clean then
            return B
        end
    end
    // Fourth Priority
    for B@LRU to B@(LRU − f × curr_alloc) do
        if B.NotInPrC or B.Dead or B.Clean then
            return B
        end
    end
    // Fallback Candidate
    return B@LRU
```

Algorithm 1: Transfer Candidate Selection Algorithm.

for eviction. By following a structured decision-making approach, SCAM enhances system's performance without compromising on cache security. Its step-by-step selection strategy effectively optimizes cache utilization, and enhances overall system efficiency. SCAM follows a step-by-step selection strategy as outlined below:

1. **First Priority:** SCAM gives the highest priority to *dead blocks* that are *clean* and do not cause *back invalidations*. A dead block is a cache block that will not be reused in the future. A clean block has not been modified since it was brought into the cache. Evicting such blocks minimizes performance impact and avoids unnecessary back invalidation requests to higher cache levels.
2. **Second Priority:** If no primary candidate is available, SCAM selects any dead block that avoids back invalidations. This ensures efficient eviction without disrupting other levels of the cache hierarchy.
3. **Third Priority:** If no secondary candidate exists, SCAM searches for any *clean block* that avoids back invalidations. Such blocks are evicted to minimize writebacks and performance penalties.
4. **Fourth Priority:** If the above options are unavailable, SCAM searches in the following order of priority: *any dead block, any clean block, or any block that avoids back invalidations*. The first matching block is selected for eviction.
5. **Fallback Candidate:** If none of the above candidates are found, SCAM evicts the *Least Recently Used (LRU)* cache block as a last resort.

To implement this hierarchical process, SCAM scans half of the cache set, starting from the LRU position. This ensures that the most appropriate block is identified for eviction while limiting unnecessary cache disruptions. If no preferred candidate is found during the scan, SCAM defaults to evicting the LRU block. This fallback mechanism ensures robust and consistent cache management under all scenarios. By following this systematic approach, SCAM reduces performance overhead and improves the efficiency of cache operations while maintaining secure and optimal cache partitioning. SCAM uses additional metadata to track the status of cache blocks:

1. **Deadblock Bit:** This bit indicates whether a cache block has been accessed since it was loaded. A value of 1 means the block is "dead" (unreferenced).
2. **Back-Invalidation Bit:** This bit shows whether evicting the block will cause back invalidations in upper-level caches. A value of 1 means back invalidations will occur.

When a block is inserted into the cache, the deadblock bit is set to 1, and the back-invalidation bit is also set to 1. If the block is accessed, the deadblock bit is reset to 0, indicating the block is active. When the block is evicted from all private caches, the back-invalidation bit is reset to 0. This metadata allows SCAM to make informed decisions about which blocks to evict or transfer.

4.2 Lower Partition Bound

Dynamic cache partitioning schemes, such as SCAM, aim to allocate cache blocks efficiently among all cores in a system. SCAM uses a heuristic that assigns at least one cache block per set in the LLC to each application. However, this allocation method can cause underutilization of private L2 caches. The problem occurs because the LLC's capacity may not be sufficient to fully support the private L2 cache of each core. For example, if a core is allocated only one block per set in the LLC, the total number of usable blocks in the LLC will be smaller than the capacity of the core's private L2 cache. This mismatch results in wasted L2 cache resources, leading to reduced overall system performance.

To overcome this limitation, SCAM enforces a minimum allocation of two cache blocks per set for

every application. By increasing the lower allocation limit, SCAM ensures that the private L2 caches are better utilized. This adjustment minimizes resource wastage and enhances system performance. Through this improved allocation strategy, SCAM achieves a more balanced use of the cache hierarchy, reducing inefficiencies and supporting higher application performance across all cores.

4.3 Decision-Making Engine

SCAM employs a decision-making engine to optimize the partitioning of the shared LLC. This engine draws inspiration from the Utility-based Cache Partitioning (UCP) method (Qureshi and Patt, 2006). The detailed operation of this engine is described below:

4.3.1 Initialization

In a dual-core system with an m-way shared LLC, the cache is evenly divided between the two cores at the start. Each core is allocated $\frac{m}{2}$ ways. Additionally, two Auxiliary Tag Directories (ATDs) are configured to evaluate alternative partitioning schemes:

- **ATD1:** Simulates a configuration where the first core is allocated $(\frac{m}{2}-1)$ ways and the second core receives $(\frac{m}{2}+1)$ ways.
- **ATD2:** Simulates a configuration where the first core is allocated $(\frac{m}{2}+1)$ ways and the second core receives $(\frac{m}{2}-1)$ ways.

4.3.2 Epoch-Based Evaluation:

The cache controller tracks the number of cache misses during each epoch for the actual LLC and the two ATDs. At the end of each epoch, the controller selects the partitioning configuration (from the actual LLC, ATD1, and ATD2) that results in the fewest misses for the next epoch. To ensure adaptability while retaining historical workload behavior, the miss counters for all configurations are reset to half their current value at the beginning of each epoch.

4.3.3 Dynamic Adjustments:

During runtime, the controller validates whether the current set partition aligns with the selected configuration whenever a cache miss occurs. If the set partition does not match, ownership of the cache line is transferred to the other core. These adjustments are only triggered by cache misses, while cache hits leave the existing partitioning unchanged. This approach minimizes unnecessary performance disruptions and preserves the stability of the cache system.

Table 1: Experimental Framework.

Simulator	**Sniper Multi-core Cycle Accurate Simulator**
CPU core	Dual core, 2.67 GHz, 4-wide fetch, 128-entry ROB
L1 I/D Cache	32 KB, 4-way, LRU, private, 4 cycles access time
L2 Cache	512 KB, 8-way, LRU, private, 8 cycles access time
Last Level Cache (LLC)	2 MB per core, 16-way, LRU, shared, 30 cycles access time

By leveraging this adaptive partitioning mechanism, SCAM ensures that the shared LLC dynamically responds to workload demands while maintaining low performance overhead. In the next section, we evaluate SCAM's performance against PASS-P. The results demonstrate SCAM's ability to deliver enhanced system performance and robust security in dynamic cache partitioning scenarios.

5 EVALUATION FRAMEWORK

In this section, we provide a detailed explanation of the simulation environment used for evaluating **SCAM**. We also present a comparative analysis of **PASS-P** and SCAM.

To evaluate performance, we used the cycle-accurate **Sniper Simulator** (Carlson et al., 2014). This tool allowed us to analyze two benchmarks running concurrently on separate cores. The simulation configurations are summarized in Table 1. Our study focuses on benchmark pairs selected from the **SPEC CPU2006** suite (Henning, 2006). These benchmarks frequently update the UCP partitioning strategy, making them suitable for testing SCAM's dynamic cache management.

For each application, we used traces representing one billion instructions. These traces were identified using **Simpoints** to reflect typical application behavior. Before starting the analysis, the system was warmed up with 200 million instructions to ensure steady-state performance.

The results of our evaluation are shown in Fig. 11 and Fig. 13. The x-axes in these figures are labeled with the application names and their respective types. For better clarity, Table 2 lists all evaluated applications. These applications were chosen based on their cache behavior, particularly their **Last-Level Cache (LLC)** miss rates. Applications with moderate (50% to 75%) to very high (above 75%) LLC miss rates were selected.

High LLC miss rates are significant because they increase the likelihood of destructive interference in

Table 2: Application list.

Application Name	Abbreviation	LLC Miss Rate (in %)	Application Name	Abbreviation	LLC Miss Rate (in %)
leslie 3d	les	80.2	sphinx3	sph	91.56
GemsFDTD	gem	66.99	wrf	wrf	77.95
mcf	mcf	63.49	bwaves	bwa	96.42
soplex	sop	56.51	milc	mil	83.64
lbm	lbm	98.96	omnetpp	omn	58.54

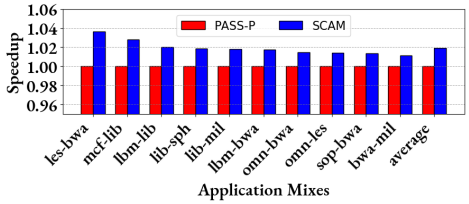

Figure 11: System's Performance Analysis.

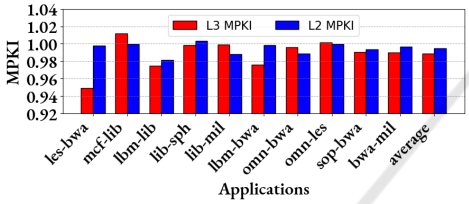

Figure 12: Cache Performance Analysis.

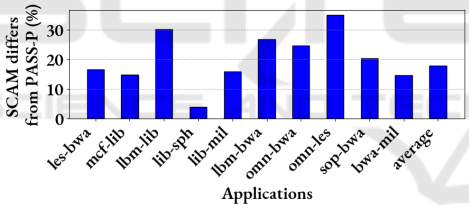

Figure 13: Fraction of times SCAM transfers a cache block different from that of PASS-P.

shared caches. Such interference is critical to evaluate the effectiveness of SCAM in reducing performance bottlenecks. Additionally, the selected applications frequently triggered partition decision changes when run concurrently, providing valuable insights into SCAM's dynamic partitioning capabilities.

The results presented in this section represent the **average performance metrics** across all possible application combinations from the SPEC CPU2006 benchmark suite. This approach ensures a comprehensive analysis of SCAM's behavior across a wide range of workloads.

In the next subsection, we compare SCAM with PASS-P in detail. We highlight the key improvements in performance achieved by SCAM, demonstrating that these enhancements do not compromise the strong security guarantees provided by dynamic cache partitioning.

6 RESULTS & ANALYSIS

In this section, we conduct a detailed performance evaluation of **SCAM** and compare its efficiency with the **PASS-P** scheme across diverse application combinations. The results demonstrate SCAM's ability to outperform PASS-P in both performance and cache management. As shown in Fig.11, SCAM achieves a maximum performance improvement of 4% and an average enhancement of 2% compared to PASS-P. These results indicate that SCAM's optimized transfer candidate selection has a direct and measurable impact on system performance. Further validation is provided by Fig.13, which highlights the differences in cache block transfers between SCAM and PASS-P. On average, SCAM selects different cache blocks for transfer 18% of the time, with a maximum difference of **34%** observed for a specific application mix (*omn-les*). This significant variation underscores the importance of SCAM's careful approach to transfer candidate selection during cache partition adjustments.

Fig.12 illustrates the impact of SCAM on L2 and LLC misses per kilo instructions (MPKIs). On average, SCAM reduces L2 MPKI by 1%, with the most significant reduction reaching **2%** for certain application mix (*lbm-lib*). Also, SCAM reduces LLC MPKI by an average of 1.2% with the reduction being as high as 5% for *les-bwa* application mix. These improvements reflect SCAM's ability to enhance cache efficiency by reducing cache misses and optimizing resource utilization. However, it is important to note that some application mixes exhibit an increase in both L2 and LLC MPKIs. This seemingly counter-intuitive outcome is attributed to shifts in cache access patterns caused by SCAM's optimization strategies. These variations highlight the complex interplay between SCAM's dynamic partitioning approach and the workloads' cache access behaviors.

Table.3 provides a comparative analysis of the properties of cache blocks transferred by PASS-P and SCAM during partition changes. The table presents the average values observed across all application mixes used in our evaluation. To evaluate the effectiveness of PASS-P and SCAM, we profiled the characteristics of all cache blocks that underwent trans-

Table 3: Transfer Candidate Property.

Property	Quantity for PASS-P (%)	Quantity for SCAM (%)	Property	Quantity for PASS-P (%)	Quantity for SCAM (%)
only dead	2.18	0.81	only not in private cache	0.53	2.65
only clean	0.11	0	dead and clean	11.13	2.06
dead and not in private cache	6.28	20.98	dead, clean and not in private cache	76.39	70.98
clean and not in private cache	3.36	2.51	only LRU and not dead, not clean, and in private cache	0.02	0

fer during the execution phase in which performance statistics were recorded. This profiling enables a direct comparison of the decision-making processes of PASS-P and SCAM. From the data in Table.3, it is evident that SCAM successfully avoids transferring cache blocks that merely satisfy the clean property without meeting additional criteria (refer to the *only clean* property in Table.3). This ensures that SCAM does not prioritize cache blocks solely based on clean property, preventing unnecessary evictions that could impact performance.

Additionally, SCAM refrains from transferring LRU blocks that do not satisfy any of the three key properties required for optimal eviction decisions (refer to *only LRU and not dead, not clean, and in private cache* property in Table.3). By doing so, SCAM prevents inefficient cache block transfers that would otherwise degrade system performance. Furthermore, an analysis of the *only not in private cache* and *dead and not in private cache* categories in Table.3 highlights SCAM's ability to mitigate NoC traffic pressure. By prioritizing blocks that are unlikely to be re-referenced after insertion, SCAM ensures efficient cache utilization while reducing unnecessary communication overhead. These optimizations collectively enhance cache management efficiency, improve system throughput, and reinforce security against side-channel threats.

Overall, the evaluation results emphasize the importance of SCAM's strategic selection of transfer candidates. By prioritizing dead blocks and considering back invalidations, SCAM improves performance without compromising the security of the shared **Last-Level Cache (LLC)**. This dual focus on performance and security establishes SCAM as a robust and effective dynamic cache partitioning strategy.

6.1 Hardware Overhead Compared to PASS-P

SCAM enhances transfer candidate selection with minimal hardware overhead, ensuring efficient cache management without added complexity. Each cache block is augmented with an additional bit per core to track its presence in private caches, enabling more informed eviction decisions and reducing unnecessary data movement.

To further optimize cache utilization, SCAM introduces a re-reference bit per block to detect dead blocks. This bit indicates whether a block has been accessed since insertion, allowing SCAM to prioritize evicting blocks unlikely to be reused. By distinguishing between frequently accessed and obsolete data, SCAM minimizes performance overhead while improving cache efficiency.

Moreover, SCAM leverages the existing dirty bit in cache metadata to identify clean blocks, eliminating the need for additional storage. This ensures that clean blocks can be selected as transfer candidates without incurring unnecessary writebacks, maintaining both performance and security. With these lightweight yet effective modifications, SCAM refines cache management while preserving low hardware complexity. The next section explores related research on shared cache management, highlighting existing solutions and their limitations.

7 RELATED WORK

Mitigation strategies for addressing cache-based side-channel attacks can be broadly divided into two main categories: **cache randomization** and **cache partitioning** (Wang and Lee, 2007).

Cache randomization techniques aim to obscure the mapping between main memory addresses and cache lines, making it difficult for attackers to infer sensitive information. These techniques disrupt predictable access patterns, a key factor exploited in side-channel attacks. For example, the **RPCache** mechanism (Wang and Lee, 2007) uses a permutation table to randomize the address mapping within the cache subsystem. This randomization ensures that cache accesses become unpredictable, significantly reducing the likelihood of successful side-channel attacks. Another approach, **CEASER** (Qureshi, 2018), employs a low-latency block cipher to encrypt cache

addresses. By encrypting the address-to-cache-line mapping, CEASER makes the process opaque and difficult for attackers to reverse-engineer.

In contrast, **cache partitioning** strategies focus on isolating cache resources between different processes. This separation ensures that one process cannot access or infer data from another, thereby minimizing the risk of side-channel leakage. For example, static partitioning techniques (Page, 2005) enforce strict boundaries between processes, providing robust security guarantees. However, this strict separation often leads to performance degradation, as cache lines allocated to one process may remain underutilized, wasting valuable cache resources.

To address the inefficiencies of static partitioning, **dynamic cache partitioning (DCP)** methods were introduced. These techniques, such as **Utility-based Cache Partitioning (UCP)** (Qureshi and Patt, 2006), adaptively allocate cache lines based on their utility. UCP dynamically adjusts partitions to improve cache utilization, boosting performance by optimizing resource allocation. Despite these benefits, dynamic partitioning introduces new vulnerabilities. Shared cache lines can still be exploited by attackers to perform side-channel attacks.

For example, **SecDCP** (Wang et al., 2016) categorizes processes into confidential and public groups, offering targeted protection for sensitive processes. However, this approach remains vulnerable to specific attacks, such as the Flush+Reload technique, as discussed in Section.3. This compromises the security guarantees of SecDCP, leaving critical gaps in its defense mechanisms.

Other methods, such as **COTSknight** (Yao et al., 2019) and **DAWG** (Kiriansky et al., 2018), also enhance cache security but introduce additional complexities. COTSknight uses advanced cache monitoring and allocation mechanisms to detect suspicious behaviors. While effective in some scenarios, it fails to mitigate Flush+Reload attacks entirely and incurs a performance penalty of up to 5% compared to an insecure baseline. DAWG, on the other hand, isolates cache accesses through secure way partitioning. However, this approach leads to performance slowdowns ranging from 0% to 15%, depending on the workload, when compared to an approximate LRU baseline.

Compared to these methods, **PASS-P** (Boran et al., 2022) strikes a better balance between performance and security. PASS-P achieves an average slowdown of just 0.35% and a maximum of 2.2% relative to the insecure UCP baseline. Additionally, UCP itself provides a performance improvement of 10.96% over LRU (Qureshi and Patt, 2006), highlighting PASS-P's competitive performance. However, as we analyze in Section 2, PASS-P has its own limitations in terms of optimal transfer candidate selection during partition changes.

For L1 caches, methods like **NoMo** (Domnitser et al., 2012) offer a tunable tradeoff between performance and security without requiring software or operating system modifications. NoMo makes minor changes to cache replacement policies to mitigate side-channel risks. However, its fully secure configuration aligns with static partitioning, leading to performance degradation of up to 5% and an average slowdown of 1.2% compared to LRU.

SCAM, the solution we propose, builds upon the foundational principles of PASS-P to address these challenges. SCAM effectively combines the strong security guarantees of static partitioning with the performance benefits of dynamic partitioning techniques. By refining transfer candidate selection and optimizing partition changes, SCAM provides robust protection against side-channel attacks while minimizing performance overhead. This makes SCAM a highly efficient and secure solution for dynamic cache partitioning, striking a careful balance between security and system performance.

8 CONCLUSION

This research demonstrates that secure cache partitioning can be optimized for performance without compromising security. To achieve this, we introduce SCAM, a novel secure cache management framework that addresses the performance limitations inherent in the transfer candidate selection strategy employed by PASS-P, a state-of-the-art secure dynamic cache partitioning (DCP) protocol. Experimental results indicate that SCAM yields a performance improvement of up to 4% compared to PASS-P. By minimizing back invalidations and prioritizing the removal of dead blocks, SCAM not only enhances cache efficiency but also strengthens security by mitigating potential attack vectors.

Designed with versatility in mind, SCAM seamlessly integrates with shared cache levels in systems utilizing various DCP protocols. This adaptability ensures robust and scalable cache management across a diverse range of application scenarios. Furthermore, SCAM provides a foundation for addressing emerging security threats, such as Meltdown (Lipp et al., 2020) and Spectre (Kocher et al., 2020). Future extensions to mitigate these advanced vulnerabilities will further solidify SCAM's position as a high-performance and secure cache management solution.

In summary, SCAM represents a significant advancement in the field of secure cache management. By effectively balancing performance optimization with robust security measures, it effectively addresses the evolving requirements of contemporary computing environments.

ACKNOWLEDGEMENTS

This work was supported in part by Indo Japanese Joint Lab Grant and AI powered adaptive cyber defence framework sponsored by NSCS(Government of India), Security of futuristic technology sponsored by MEITY(Government of India) and ISRO Respond.

REFERENCES

Ashokkumar, C., Giri, R. P., and Menezes, B. (2016). Highly efficient algorithms for aes key retrieval in cache access attacks. In *2016 IEEE European symposium on security and privacy (EuroS&P)*, pages 261–275. IEEE.

Bernstein, D. J. (2005). Cache-timing attacks on aes.

Boran, N. K., Joshi, P., and Singh, V. (2022). Pass-p: Performance and security sensitive dynamic cache partitioning. In *SECRYPT*, pages 443–450.

Boran, N. K., Pinto, K., and Menezes, B. (2021). On disabling prefetcher to amplify cache side channels. In *2021 25th International Symposium on VLSI Design and Test (VDAT)*, pages 1–6. IEEE.

Carlson, T. E., Heirman, W., Eyerman, S., Hur, I., and Eeckhout, L. (2014). An evaluation of high-level mechanistic core models. *ACM Transactions on Architecture and Code Optimization (TACO)*.

Domnitser, L., Jaleel, A., Loew, J., Abu-Ghazaleh, N., and Ponomarev, D. (2012). Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks. *ACM Transactions on Architecture and Code Optimization (TACO)*, 8(4):1–21.

Gullasch, D., Bangerter, E., and Krenn, S. (2011). Cache games–bringing access-based cache attacks on aes to practice. In *2011 IEEE Symposium on Security and Privacy*, pages 490–505. IEEE.

Henning, J. L. (2006). Spec cpu2006 benchmark descriptions. *ACM SIGARCH Computer Architecture News*, 34(4):1–17.

Kiriansky, V., Lebedev, I., Amarasinghe, S., Devadas, S., and Emer, J. (2018). Dawg: A defense against cache timing attacks in speculative execution processors. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 974–987. IEEE.

Kocher, P., Horn, J., Fogh, A., Genkin, D., Gruss, D., Haas, W., Hamburg, M., Lipp, M., Mangard, S., Prescher, T., et al. (2020). Spectre attacks: Exploiting speculative execution. *Communications of the ACM*, 63(7):93–101.

Kong, J., Aciicmez, O., Seifert, J.-P., and Zhou, H. (2008). Deconstructing new cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 2nd ACM workshop on Computer security architectures*, pages 25–34.

Lipp, M., Schwarz, M., Gruss, D., Prescher, T., Haas, W., Horn, J., Mangard, S., Kocher, P., Genkin, D., Yarom, Y., et al. (2020). Meltdown: Reading kernel memory from user space. *Communications of the ACM*, 63(6):46–56.

Liu, F., Yarom, Y., Ge, Q., Heiser, G., and Lee, R. B. (2015). Last-level cache side-channel attacks are practical. In *2015 IEEE symposium on security and privacy*, pages 605–622. IEEE.

Page, D. (2005). Partitioned cache architecture as a side-channel defence mechanism. *Cryptology ePrint Archive*.

Percival, C. (2005). Cache missing for fun and profit.

Qureshi, M. K. (2018). Ceaser: Mitigating conflict-based cache attacks via encrypted-address and remapping. In *2018 51st Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 775–787. IEEE.

Qureshi, M. K. and Patt, Y. N. (2006). Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches. In *2006 39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*, pages 423–432. IEEE.

Sanchez, D. and Kozyrakis, C. (2012). Scalable and efficient fine-grained cache partitioning with vantage. *IEEE Micro*, 32(3):26–37.

Tromer, E., Osvik, D. A., and Shamir, A. (2010). Efficient cache attacks on aes, and countermeasures. *Journal of Cryptology*, 23:37–71.

Wang, Y., Ferraiuolo, A., Zhang, D., Myers, A. C., and Suh, G. E. (2016). Secdcp: secure dynamic cache partitioning for efficient timing channel protection. In *Proceedings of the 53rd Annual Design Automation Conference*, pages 1–6.

Wang, Z. and Lee, R. B. (2007). New cache designs for thwarting software cache-based side channel attacks. In *Proceedings of the 34th annual international symposium on Computer architecture*, pages 494–505.

Xie, Y. and Loh, G. H. (2009). Pipp: Promotion/insertion pseudo-partitioning of multi-core shared caches. *ACM SIGARCH Computer Architecture News*, 37(3):174–183.

Yao, F., Fang, H., Doroslovački, M., and Venkataramani, G. (2019). Cotsknight: Practical defense against cache timing channel attacks using cache monitoring and partitioning technologies. In *2019 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 121–130. IEEE.

Yarom, Y. and Falkner, K. (2014). {FLUSH+ RELOAD}: A high resolution, low noise, l3 cache {Side-Channel} attack. In *23rd USENIX security symposium (USENIX security 14)*, pages 719–732.