




# Detecting and Mitigating Conflict-Based Cache Side-Channel Attacks by Monitoring Ping-Pong Accesses Patterns

Hao Ma<sup>1,2</sup>, Zhidong Wang<sup>1,2</sup>, Da Xie<sup>1,2</sup>, Jinchi Han<sup>1,2</sup>, and Wei Song<sup>1,2</sup>(✉) 

<sup>1</sup> State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, CAS, Beijing, China

[songwei@iie.ac.cn](mailto:songwei@iie.ac.cn)

<sup>2</sup> School of Cyberspace Security, University of Chinese Academy of Sciences, Beijing, China

**Abstract.** Conflict-based side-channel attacks allow attackers to monitor victims' access patterns by asserting malicious cache conflicts. While cache randomization has emerged as a potential defense, existing solutions face critical limitations. CEASER-S and DT4+EV10 fail to fully prevent existing eviction set searching algorithms. Other approaches such as MIRAGE and Chameleon suffer from intolerable area and power overheads. To alleviate these limitations, we introduce a randomized cache with attack detection triggered on-demand remapping. A detector is designed to identify active attacks by their distinct ping-pong access patterns and trigger remaps to thwart attacks when they are detected. Our approach achieves sufficient protection against conflict-based side-channel attacks while incurs negligible runtime performance impact with moderate area and power overhead.

**Keywords:** micro architecture · conflict-based cache side-channel attacks · cache randomization · attack detection

## 1 Introduction

To reduce memory access latency, modern computers introduce multi-level cache structures within the system-on-chip architecture between cores and memory. As a critical performance component, the last-level cache (LLC) is shared among all cores to maximize resource utilization. When a sensitive application runs simultaneously with a malicious one on different cores, attackers may utilize cache side-channel attacks to leak sensitive information through the LLC [1, 2]. The cache structure of current LLC unintentionally allows attackers to evict a victim's data by accessing an eviction set – *a group of congruent memory addresses mapping to the same cache set with the victim's data*. This enables attackers to manipulate the cache state and infer sensitive security information from the victim program. Existing studies have demonstrated that such conflict-based attacks have been used to recover encryption keys [3], user privacy data in

the cloud [4, 5], break sandbox defenses [6], inject faults into DRAM [7], and even steal information in what is considered secure trusted execution environments [8].

Cache partitioning was one of the early defenses proposed to defend against conflict-based attacks [5, 9, 10]. By separating private information from ordinary data [11], cache partitioning makes it impossible for attackers to cause conflicts and evict crucial data [7]. However, cache partitioning relies on a trusted operating system to differentiate between private and ordinary data [12]. Furthermore, when privacy data cannot be easily separated from ordinary data using cache partitioning, the approach becomes ineffective.

Cache randomization [13–19] has emerged as a promising defense mechanism. By randomizing the locations of cache blocks [20, 21], it prevents attackers from predicting the address-to-set mapping. Some advanced defense schemes have combined cache randomization with skewing for enhanced protection. For instance, CEASER-S [16] employs a skewed cache structure with periodic remapping to mitigate eviction set searching algorithms such as *Group Elimination* (GE) [21, 22] and *Prime Prune Probe* (PPP) [1–3] but fails to thwart *Conflict Testing* (CT) [16] and *Conflict Testing-Fast* (CT-Fast) [22]. Chameleon Cache [23] strengthens defense by combining a random skewed cache with a Victim Cache (VC). When an eviction occurs in the LLC, the evicted cache block is first moved to the VC, then it evicts an unrelated cache block in VC, thereby obfuscating conflicts and separating contentions. However, each cache miss incurs the relocation of three cache blocks, which both consumes extra power and energy. MIRAGE [24] uses over-provided metadata and separates the meta and data to eliminate set-associativity conflicts, but the authors admit that MIRAGE inevitably results in a 22% area overhead, which substantially reduces LLC resource utilization. Inspired by ZCache [25], DT4+EV10 [26, 27] analyzes the distribution of evictions over LLC cache sets under attack and proposes a lightweight attack detection and on-demand remapping scheme using the traditional set-associative LLC. However, it cannot defend against the latest searching algorithms like *Conflict Testing with Probe+Prune* (CTPP) [28].

Current randomized cache structures exhibit two key limitations: incomplete security and high overheads. In this paper, we propose a new detector to dynamically trigger remaps whenever eviction set searching algorithms are found in action. It is shown that a traditional set-associative LLC can be made secure enough to thwart all existing eviction set searching algorithms. Compared to existing randomized cache designs, our proposal shows advantages in security, cache hit rate, area, and power consumption. Our contributions are as follows:

1. We identify distinct ping-pong access patterns in eviction set searching algorithms: CT and CT-Fast access a single ping-pong address at an elevated frequency, while CTPP, PPP and GE access a massive amount of ping-pong addresses.
2. We propose a new detector to accurately identify the execution of these fast searching algorithms using the ping-pong access patterns.

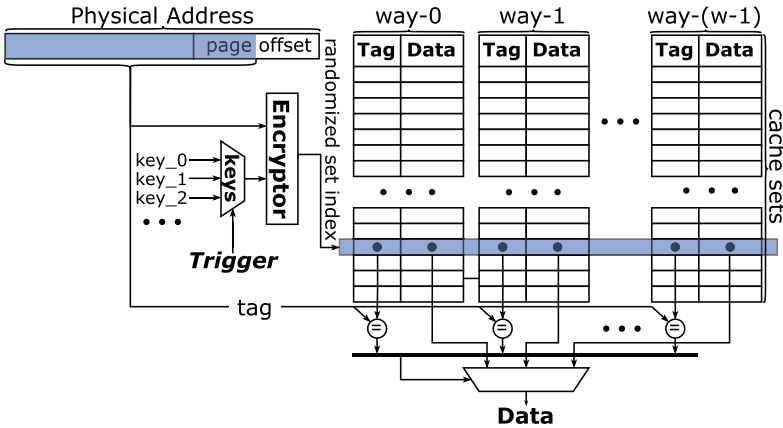
3. We optimize the detector design with a dedicated exclusive tag cache. As a result, the detector introduces only 0.16% runtime, 3.61% area and 3.81% power overhead.

This paper is organized as follows: Sect. 2 covers background knowledge; Sect. 3 analyzes the threat model and ping-pong access patterns; Sect. 4 presents the detector implementation; Sect. 5 evaluates security, performance, and overheads; and Sect. 6 concludes.

## 2 Background

This section introduces the necessary background for understanding the paper, including randomized caches and eviction set searching algorithms.

### 2.1 Randomized Caches



**Fig. 1.** A randomized set-associative LLC structure.

Randomized last-level caches make it significantly difficult for attackers to search usable eviction sets. As shown in Fig. 1, a randomized cache generates cache set index of an address by encrypting the higher digits of the address after removing the lower cache block offset bits [27]. As the encryption is unknown to the attacker, she must search eviction sets at runtime using fast search algorithms. To limit the time available for searching an eviction set or nullify an eviction set already obtained by an attacker, the cache can update the key used by the encryption, which effectively re-randomize the mapping between addresses and cache set indices. All congruent addresses already obtained by an attacker become useless. However, all cache blocks in the LLC must be relocated according to the new mapping. Single-step relocation directly moves blocks to target sets, evicting random cache blocks. Multi-step relocation recursively handles occupied targets by continuing displacement until finding empty slots or remapped

blocks. While these strategies affect LLC remapping performance, CEASER-S's frequent key changes incur prohibitive overhead – its single-step approach evicts 40–50% of blocks per remapping. In contrast, DT4+EV10 activates remapping only when attacks are detected and employs multi-step relocation to reduce evictions to just 10%.

Modern defense mechanisms employ skewed cache architectures [16, 17, 23, 24] that enhance security by randomizing cache set mappings across multiple skew-partitions. This approach invalidates traditional eviction attacks by: dramatically increasing required eviction set sizes, and rendering obtained sets operationally ineffective. In these designs, addresses are either fully congruent (where the same address maps to identical sets across all skew-partitions) or partially congruent (where the address doesn't consistently map to the same set across all partitions). The use of independent mapping keys per partition makes collecting fully congruent addresses extremely difficult, forcing attackers to rely on partially congruent addresses. The mutual independence of skew-partition mappings forces random eviction selection among missed partitions when accessing partially congruent addresses. This design causes an exponential relationship between the number of skew-partitions and the required partial congruent addresses for target eviction, substantially increasing the attacker's workload.

While skewed caches improve security, excessive skew-partitions hurt performance. Maintaining multiple parallel mappings increases area overhead and complicates access handling and conflict detection, ultimately reducing LLC bandwidth and degrading application performance. Through runtime attack detection coupled with on-demand remapping, we demonstrate that randomized set-associative LLCs simultaneously achieve robust security against existing conflict-based attacks and sustained performance with minimal overhead.

## 2.2 Eviction Set Searching Algorithms

Although cache randomization blocks address mapping prediction but cannot stop runtime eviction set searches that collect congruent addresses. GE [21, 22, 26] initializes with a large random address pool typically exceeding SW addresses (S: cache sets, W: cache ways) that must contain at least W congruent addresses. The method iteratively groups and filters the addresses by dividing them into  $W+1$  groups per round and removing at least one group each time. The attacker tests the remaining groups for further pruning until achieving minimal eviction set, with a time complexity of  $O(SW^2)$  for LLC.

PPP [1–3] begins by accessing a large pool of random addresses to prime the LLC. During the prime stage, mutually conflicting addresses are pruned, and the process continues until all addresses are fully populated in the LLC. The attacker then probes the refined address pool multiple times. Each probe first accesses the target address, then detects evictions through cache misses on re-access target. Those high-latency addresses are proving congruent to the target. The search requires  $O(SW)$  accesses under LRU, or  $O(SW^2)$  under random replacement.

The CT [16] algorithm operates by first accessing a target address, then sequentially testing random addresses to detect congruence. Due to random

replacement, each congruent address has a  $\frac{1}{W}$  probability of evicting the target. The attacker alternates between accessing random and target addresses – when a target access misses, the preceding random address is identified as congruent and added to the eviction set. This process repeats until completion, exhibiting  $O(SW^2)$  time complexity for minimal eviction set contains  $W$  congruent addresses. Notably, the algorithm maintains this complexity ( $O(SW^2)$ ) even for permutation-based replacement policies like LRU.

The CT-Fast [30] algorithm is an optimization of the CT algorithm. It works by finding any random address that is congruent to the target address, which causes the target address to be evicted. After re-accessing the target address, the attacker then accesses all previously found congruent addresses, making target more easily to be evicted.

CTPP [28] combines the advantages of both CT and PPP algorithms, targeting LRU replacement policies. The process begins with a *CT* phase where the attacker quickly builds an initial address pool containing congruent addresses distributed across three types of cache sets: those with more than, exactly, or fewer than  $W$  congruent addresses. This is followed by iterative *Probe* and *Prune* phases – the *Probe* phase eliminates addresses causing cache hits (from undersized sets with fewer than  $W$  congruent addresses), while the *Prune* phase removes addresses resulting in cache misses (from oversized sets with more than  $W$  congruent addresses). Through 3–5 such iterations, the algorithm efficiently converges to a perfect eviction set containing exactly  $W$  addresses.

### 3 Threat Analysis and Defense Methods

All existing eviction set searching algorithms incur large amounts of ping-pong accesses, but the access pattern differs among different algorithms. To be specific, we identify two distinct patterns: a single-address ping-pong pattern asserted by CT and CT-fast algorithms, and a massive-address ping-pong pattern asserted by CTPP, PPP and GE algorithms.

#### 3.1 Threat Model

This paper focuses on preventing attackers from successfully obtain a complete eviction set using existing search algorithms; therefore, we define a successful attack as finding a complete eviction set. Without eviction sets, attackers cannot control target cache sets and all conflict-based side-channel attacks fail.

We assume the attacker can allocate and access arbitrary amount of memory while infer the hit/miss state of a memory access of her own data using high-resolution timers. The victim runs in a separate address space with no shared memory with the attacker; therefore, the attacker cannot directly access data belonging to the victim. The randomized cache structure is publicly available to the attacker but the encryption of the cache set index is hardware controlled and secure (not deciphered).

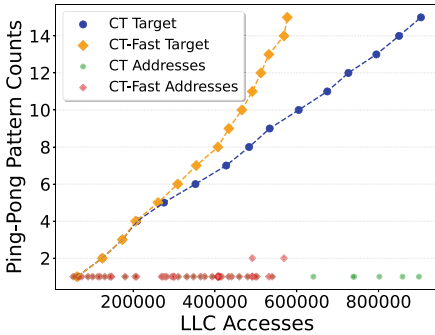
We focus on conflict-based cache side-channel attacks. Other types of cache side-channel attacks, such as reuse-based and occupation attacks [1, 2], are out of the scope of this paper.

### 3.2 Ping-Pong Access and Ping-Pong Distance

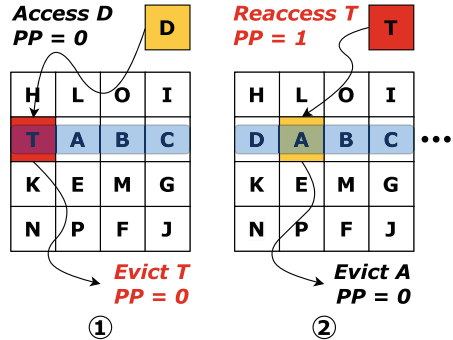
A ping-pong access occurs between the LLC and memory when a cache block is previously evicted from a cache set to make room for another missing block but soon re-fetched from memory due to a re-access [31–33].

The number of LLC accesses between the eviction and re-access is denoted as the *reuse distance* [34–36], which is an important metric in evaluating the temporal locality of a certain application and the cache efficiency of an LLC design. In this paper, we also find this metric useful in detecting eviction set searching algorithms. However, attackers can maliciously distort this distance by making additional cache accesses. To avoid such distortion, we redefine *reuse distance* as the number of evictions in the cache set before the address is re-fetched from memory, re-termed as *ping-pong distance*, which is found particularly effective and robust for detecting eviction set searching algorithms.

### 3.3 Detection Using Single-Address Ping-Pong Pattern



**Fig. 2.** The high-frequency SA ping-pong access patterns exhibited by the CT and CT-Fast algorithms.



**Fig. 3.** In 4-set, 4-way LRU LLCs, the target address T of CT/CT-Fast exhibits ping-pong (*PP*) counts increasing.

In CT and CT-Fast, attackers access a large amount of random addresses and collect those tested congruent with an attacker chosen target address. These algorithms unavoidably incur frequent ping-pong accesses on the target address. Each attacker accessed random address has a  $\frac{1}{SW}$  probability to successfully evict the target address. Once this occurs, this target address is immediately re-fetched from memory in order to confirm that the random address used to cause the eviction is congruent, and its ping-pong distance is measured as 1. Figure 2 illustrates this phenomenon on a behavioral cache model [37]. The x-axis records LLC accesses, and the y-axis shows the ping-pong accesses count being observed. The number of ping-pong accesses serves as a good trigger because it reflects the count of congruent addresses obtained by the attacker. Once the attacker acquires  $W$

**Algorithm 1.** Single-Address Ping-Pong Detection Mechanism

---

**Input:** Accessed address A  
**Output:** Remap (true or false)

```

1: if LLC_Miss(A) && TC_Hit(A) then
2:    $SAPP \leftarrow TCrdpp(A) + 1$ 
3:    $d \leftarrow SetEvs - TCrdse(A)$ 
4:   if  $d == 1$  &&  $SAPP \geq W - 1$  then
5:     return true
6:   end if
7: end if
8: return false

```

---

congruent addresses, she gains a valid eviction set. Our defense mechanism must invalidate the addresses discovered by the attacker before she can accumulate enough addresses. When the ping-pong access count reaches a certain threshold, the cache can reliably detect an attack and trigger a remap. This allows the system to invalidate as many addresses as possible with minimal remaps. The reason remapping can invalidate the attacker’s previously found addresses is that the LLC’s mapping relationship changes, causing those addresses to no longer map to the same cache set. Additionally, periodic remapping (e.g., CEASER-S) is an inefficient solution because it continues to perform remapping even in the absence of attacks, leading to unnecessary runtime overhead.

As shown in Algorithm 1, we propose a single-address (SA) ping-pong detector dedicated to identify this access pattern. The detector operates at the LLC and during a miss event. When address A is evicted from the LLC, its partial metadata is stored in a specified structure we designed called the *tag cache* (TC). Upon A being subsequently fetched back into the LLC, the detector loads this preserved metadata for updating including: the single-address ping-pong (*SAPP*) count of address A is tracked by the *TCrdpp()* function; the set-evictions count (*old SetEvs*) of the corresponding cache set when A was last evicted, via *TCrdse()* function. The detector then computes the ping-pong distance  $d$  ( $d = \text{current SetEvs} - \text{old SetEvs}$ ). Later using  $d$ , *SAPP*, and LLC ways ( $W$ ) to evaluate whether A qualifies as a risky address and triggers a remap.

For more intuitive demonstration, we depict a simplified 4-set, 4-way LLC using the least recently used (LRU) replacement algorithm in Fig. 3. The cache illustrates the situations ① and ② where target address is captured during CT/CT-Fast algorithms. ① Assuming that the addresses T and A-D are congruent with each other, and T is the target address, A-D are mapped to the same set as the target T. Since these four addresses are loaded into the LLC for the first time, each of them is tagged with a ping-pong (*PP*) count of 0. The following access of D evicts T due to LRU policy, and the *PP* value of T remains 0. The attacker re-accesses T to verify whether it remains in the cache. If T misses, confirming that D is congruent; otherwise, D is not congruent. In this example, T is evicted and re-fetched into the cache, incrementing its *PP* count ( $PP \leftarrow PP + 1$ ). As the attacker interleaves accesses between random

**Algorithm 2.** Massive-Address Ping-Pong Detection Mechanism

---

**Input:** Accessed address A  
**Output:** Remap (true or false)

```

1: if LLC_Miss(A) && TC_Hit(A) then
2:    $d \leftarrow \text{SetEvs} - \text{TCrdse}(A)$ 
3:   if  $d \leq r$  then
4:      $\text{MAPP} \leftarrow \text{MAPP} + 1$ 
5:     if  $\text{MAPP} \geq t$  then
6:       return true
7:     end if
8:   end if
9: end if
10: return false

```

---

addresses and target address T, congruent addresses persistently evict T. Each verification access to T increments its *PP* count. When this count approaches the cache’s associativity threshold, the SA ping-pong detector initiates remapping by modifying the key assignment, thereby invalidating the attacker’s progress.

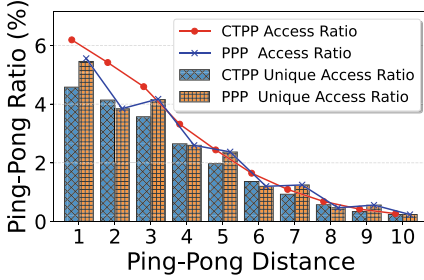
### 3.4 Detection Using Massive-Address Ping-Pong Pattern

CTPP, PPP, and GE are all pruning-based algorithms that eliminate irrelevant addresses from a large candidate set. This requires multiple passes through the candidate set, generating a large number of self-conflicts. Compared to typical program behavior, these self-conflicts essentially produce an order of magnitude more short-distance ping-pong addresses. Effective detection can be accomplished by tracking either the *ping-pong ratio* – computed as *ping-pong accesses divided by total LLC accesses within an observation window* – or the absolute number of ping-pong occurrences during monitoring. When either metric exceeds its predetermined threshold, the detector triggers remapping, thereby effectively countering pruning-based algorithms.

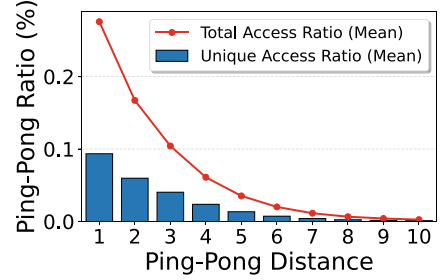
To implement this detection mechanism, as Algorithm 2 shows, we employ a massive-address (MA) ping-pong detector using *detection window* – that spans an average of two accesses per LLC address. When address A misses in the LLC but hits in the TC, the detector loads the metadata from the TC, computes the ping-pong distance  $d$ , and verifies whether  $d$  lies within the detection range of 1 to  $r$ . We set  $r=4$  (configurable,  $r \geq 1$ ) to account for noise-induced distance inflation in pruning-based eviction set searching algorithms detection, providing sufficient margin while tolerating system noise. If this condition is satisfied, the massive-address ping-pong (*MAPP*) count is incremented by 1. When *MAPP*’s value exceeds a specified threshold  $t$  within the same detection window, the detector triggers a remap. At the end of each detection window, we reset to zero both the *MAPP* categorized by  $r$  and the corresponding ping-pong ratios before beginning new data collection in the next window. However, exceptions exist: if the LLC mapping for ping-pong address A differs between its eviction and



subsequent re-fetch, or if the detector’s detection window has changed, then this address will be excluded from the detector’s consideration.

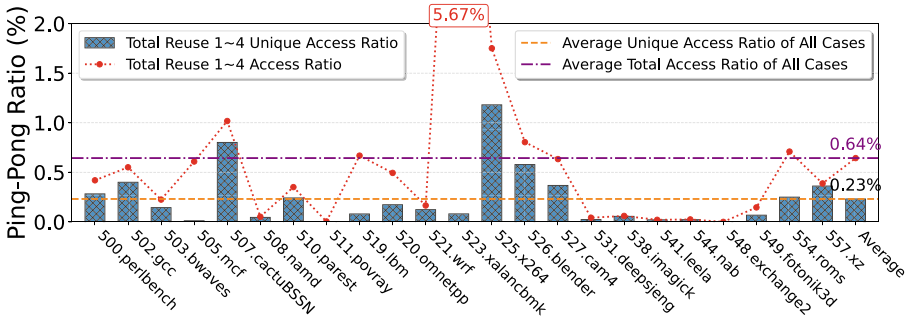


**Fig. 4.** Ping-pong ratio for unique and total accesses in CTPP and PPP algorithms with a ping-pong distance ranging from 1–10.



**Fig. 5.** Average ping-pong ratio for unique and total accesses across ping-pong distances ranging from 1–10 for the SPEC 2017 benchmarks.

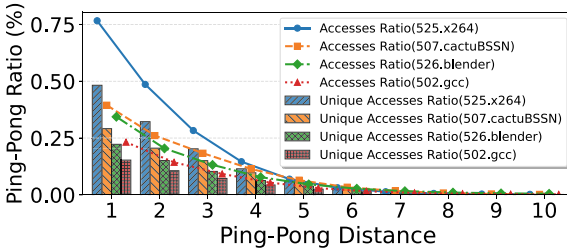
The unique access ratio counts each address only once per detection window, while the total ratio includes all accesses, even duplicates. As shown in Fig. 4, the average ping-pong ratios at distances 1 to 10 are measured across all peak ratios recorded during the detection window in multiple successful executions of the CTPP and PPP algorithms. The results demonstrate that CTPP and PPP achieve significantly higher unique access ratios – approximately 4.6% for CTPP and 5.5% for PPP at distance 1. For comparison, Fig. 5 provides baseline measurements from all SPEC-CPU 2017 benchmark programs [29] across the identical distance range. These benchmark results exhibit sparse address distributions, with an average unique access ratio of merely 0.09% at distance 1.



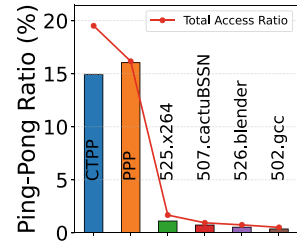
**Fig. 6.** Ping-pong ratio for unique and total accesses across ping-pong distances  $\leq 4$  for the SPEC 2017 benchmarks.

To ensure robustness in real-world environments with system noise from multi-threading and context switching, we implement an expanded detection

window covering ping-pong distances 1 through 4. A more detailed examination of closer-range patterns is shown in Fig. 6, which quantifies ping-pong addresses with distances below 5 and their corresponding access ratios during benchmarks execution. Our experimental data shows a key distinction between total and unique access ratios that impacts detection design. For normal programs (e.g., 519.lbm, 523.xalancbmk, 525.x264), total access counts for ping-pong distances 1–4 far exceed unique counts – 523.xalancbmk shows a  $57\times$  difference (5.67% total vs 0.1% unique ratio). This pattern, reflected in the mean ratios (0.64% total vs 0.23% unique), reveals that total-ratio-based defenses are vulnerable to manipulation through duplicate accesses. Unique access metrics provide more reliable detection as they better capture the characterized patterns of eviction set searching algorithms.



**Fig. 7.** Ping-pong ratios (unique/total) for ping-pong distances 1–10 across the top four test cases.



**Fig. 8.** Distance  $\leq 4$  ping-pong ratios: CTPP/PPP versus top four test cases.

Focusing on the most significant cases, Fig. 7 analyzes the four benchmark programs exhibiting the highest ping-pong ratios from distances 1–10. Figure 8 shows the cumulative ping-pong ratios for distances 1–4, comparing CTPP, PPP, and the four highest-ratio benchmarks. As shown in Fig. 8, 525.x264 exhibits the most pronounced behavior among benchmarks with a 1.2% unique access ratio, this remains substantially lower than CTPP’s 15% and PPP’s 16% ratios. Though not plotted here, experimental data confirms the GE algorithm follows similar patterns, with unique access ratios consistently exceeding normal program behavior. Thus, this method enhances the distinction between regular program execution and eviction set searching algorithms. Our approach delivers three key advantages: enhanced detection accuracy through multi-distance analysis, reduced false positives by focusing on unique accesses, and improved noise immunity via robust threshold design.

## 4 Hardware Implementation Details

Figure 9 illustrates our monitoring architecture that integrates with two components: fine-grained single-address (SA) tracking for detecting access patterns of malicious algorithms (e.g., CT/CT-Fast), and scalable massive-address (MA)

detection for algorithms such as CTPP/PPP/GE. This hybrid design achieves a balance between detection accuracy and hardware efficiency through three key elements: optimized LLC metadata tagging, a dedicated exclusive tag cache structure, and adaptive remapping triggers.

#### 4.1 Ping-Pong Detector

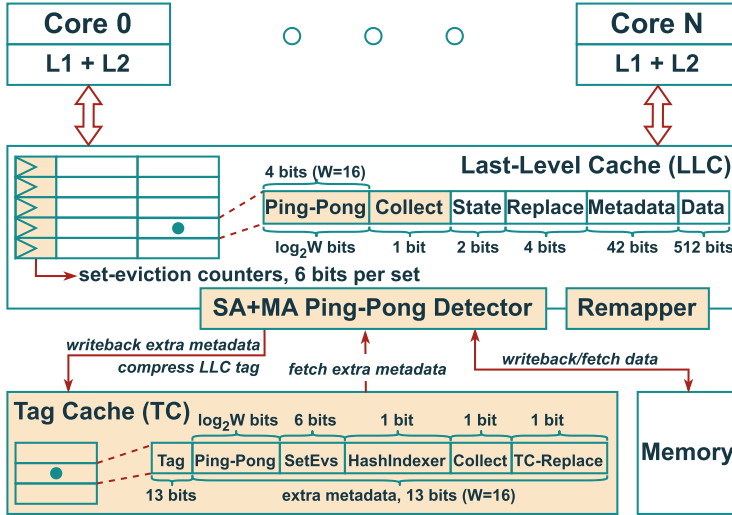


Fig. 9. Overall structure of the ping-pong detector (SA+MA).

To support both SA and MA ping-pong detection mechanisms, we extended the LLC with additional metadata. Experiments revealed that recording per-set evictions (*SetEvs*) by a 6-bit counter provides sufficient precision when computing the ping-pong distance. To minimize the increase in cache miss rate caused by remapping, we employ an upper threshold in the detector before the attacker accumulates  $W$  congruent addresses to reduce the frequency of remaps. Thus, we implemented 4-bit ( $\log_2 W, W = 16$ ) *ping-pong* for each cache block. To handle potential distance calculation errors from address remapping during ping-pong intervals, we added a 1-bit *hash indexer*. When an address is evicted from the LLC, its *hash indexer* is written back to the TC. Upon subsequent access to this address, the *hash indexer* is retrieved from the TC and loaded back into the LLC. The detector then verifies whether the current LLC mapping matches the previously stored mapping. If they are consistent, the address is considered valid and proceeds to ping-pong distance calculation. However, if the mappings differ, it indicates that the same address has been remapped to different cache sets before and after remapping, consequently associating with different set eviction

counters. This inconsistency would lead to erroneous ping-pong distance computation, and therefore the address is discarded from further processing.

For MA detection, we track unique ping-pong addresses using a 1-bit *collect* flag per address (initially 0). This flag is set to 1 on first access and cleared during remapping. The unique counter increments only when the flag transitions from 0 to 1; subsequent accesses to already collected addresses (flag=1) are ignored. We also implemented a  $(\log_2 2 * S * W)$ -bit global counter to support detection windows. When fetching back the extra metadata of an address from TC to LLC, only the storage overhead for *ping-pong* and *collect* metadata needs to be maintained in the LLC, while the remaining extra metadata does not require LLC storage (since a complete backup of the total extra metadata is preserved in the TC). These metadata are exclusively used for active computations: distance measurement, index verification, and collection tracking. Once the calculations are completed, the total extra metadata in the LLC will be updated to the current state, and the backup in the TC will be cleared to free space.

## 4.2 Exclusive Tag Cache Design

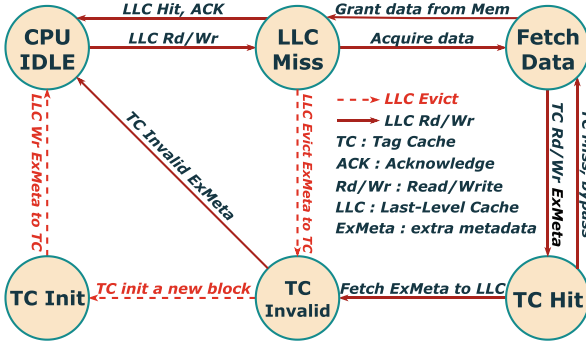
To preserve extra metadata for evicted LLC addresses, we implemented an on-chip exclusive tag cache (TC) instead of using main memory. This design decision is supported by three key considerations: First, memory-based approaches would necessitate architectural changes, conflicting with our goal of lightweight deployment. Second, accessing metadata off-chip would introduce unacceptable bandwidth and power consumption overheads. Third, our TC solution efficiently utilizes chip space by only storing metadata for recently evicted blocks, avoiding the memory waste that would occur with unused address tracking throughout program execution.

The TC maintains minimal overhead, it only stores extra metadata for recently evicted blocks and requires no additional consistency management. For our 4MB LLC implementation (organized as 4096 sets, 16-way), we designed a 26KB tag cache (512 sets, 16-way). Each cache block contains two 13-bit fields: one for extra metadata and another for encrypted address tags.

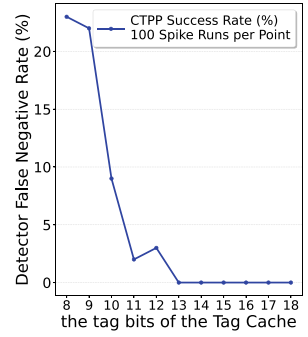
## 4.3 Interactions Between LLC and Tag Cache

Figure 10 illustrates the interaction protocol between the LLC and TC through a two-phase state machine. During the access process (represented by solid arrows), the LLC only queries the TC when cache misses occur to retrieve necessary extra metadata. When the TC contains the requested extra metadata (TC hit), the system transfers this metadata to the LLC while simultaneously invalidating the TC copy to retaining only the latest version in the LLC. This reduces TC hardware overhead. In cases of TC misses, the system bypasses extra metadata handling altogether, fetching data directly from memory.

The eviction process (denoted by dashed arrows) follows a different protocol: evicted LLC blocks always write their extra metadata back to the TC, requiring new slot allocations due to the exclusive nature of their relationship. To prevent



**Fig. 10.** State machine for the interaction between LLC and TC extra metadata.



**Fig. 11.** TC's tag bits vs. CTPP success rate.

attackers from maliciously flushing the TC, we prioritize retaining extra metadata with higher *ping-pong* counts in the TC for detection purposes. As a result, we redesigned the TC's replacement strategy as follows:

1. Small Ping-Pong First: Evict blocks with lower *ping-pong* counts first, preserving blocks with higher *ping-pong* counts to enhance the speed and accuracy of algorithms detection (e.g., CT, CT-Fast).
2. Random: If all blocks in a set have equal *ping-pong* counts, random replacement is used to select the eviction candidate (we introduce 1-bit *TC-Replace* per TC cache block).

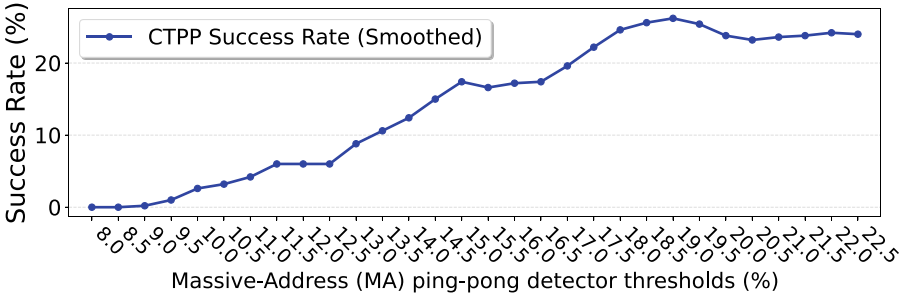
#### 4.4 Tag Compression of the Tag Cache

To further reduce TC storage overhead, our design implements an efficient address compression scheme that transforms 42-bit physical addresses into compact 13-bit hashed values for TC tag storage. This optimization is feasible because the TC tags serve only for hit/miss determination within the TC itself, without requiring back-probing to the LLC. As evidenced in Fig. 11, this compression strategy demonstrates remarkable effectiveness: the system completely suppresses CTPP algorithm (0% success rate) when using  $\geq 13$ -bit tags, while shorter tags ( $\leq 12$ -bit) induce progressive performance degradation due to accelerating hash collision rates. The hashing mechanism thus achieves an optimal trade-off – it reduces storage overhead by 69% (from 42 to 13 bits) while preserving perfect attack detection capability, ensuring both space efficiency and security robustness.

## 5 Security and Performance Evaluation

To systematically evaluate our proposal, we first establish the experimental framework based on the processor configuration detailed in Appendix A Table 3. Our evaluation platform emulates Intel Coffee Lake 9th Generation cache architecture through a behavioral cache model [37].

### 5.1 Security Analysis and Comparison



**Fig. 12.** CTPP success rates under varying MA ping-pong detector thresholds, with each point representing 100 independent Spike experiments [29].

We selected (W-1) as the threshold for the SA detection mechanism. When the ping-pong count of an address reaches this threshold, the SA detector triggers a remap, effectively defending against CT/CT-Fast in nearly all cases. For the MA detection threshold, we conducted thousands of experiments to select a ping-pong ratio that falls below the minimum thresholds of CTPP, PPP, and GE. The results are shown in Fig. 12, the x-axis denotes the thresholds (measured as unique ping-pong ratios) and the y-axis quantifies defense effectiveness through CTPP’s success rate. In this context, lower y-axis values indicate better defensive effectiveness. The curve is derived from raw experimental data using a moving average calculation with a sliding window of 5 for smoothing.

As the thresholds increase, the overall success rate of CTPP shows an upward trend. Each point in the coordinate system represents the result of 100 independent experiments run on a Spike. Some points exhibit a decline, which is due to variations in the address pool size generated during the CT phase of CTPP. The *Probe+Prune* phase is more susceptible to noise interference, as the attacker aims to precisely control the replacement distribution of the set containing the target address through random addresses. Additional noise reduces the robustness of this control, leading to a decrease in the success rate of constructing a perfect eviction set. When the detector’s ping-pong distance range sets to 1–4, the total unique accesses count is approximately 11141, corresponding to an LLC ping-pong ratio of 8.5%, with a success rate of 0% for CTPP (with 15% ratio for PPP). Thus, when the detector’s collected unique accesses exceed the threshold ratio during a detection window, it triggers a remap, preventing attackers from searching a complete eviction set using CTPP, PPP and GE algorithms.

Table 1 presents defense architectures security effectiveness measured by the success rates of five fast eviction set searching algorithms (each executed 100

**Table 1.** Comparison of security and performance across defense schemes.

| Structures            | GE        | CT          | CT-Fast     | PPP       | CTPP       | MPKI         | RPGI        | Power         | Area          |
|-----------------------|-----------|-------------|-------------|-----------|------------|--------------|-------------|---------------|---------------|
| set-associative †     | 100%      | 100%        | 100%        | 92%       | 95%        | 0.00%        | 0           | 0.00%         | 0.00%         |
| CEASER-S              | 0%        | <b>100%</b> | <b>100%</b> | 0%        | 0%         | 0.81%        | 1.5         | 0.39%         | 1.59%         |
| DT4+EV10              | 0%        | 0%          | 0%          | 0%        | <b>27%</b> | 0.30%        | <b>32.8</b> | 2.60%         | 2.54%         |
| MIRAGE                | 0%        | 0%          | 0%          | 0%        | 0%         | 0.81%        | 0           | <b>17.34%</b> | <b>19.22%</b> |
| MIRAGE-Lite           | 0%        | 0%          | 0%          | 0%        | 0%         | 0.83%        | 0           | <b>13.71%</b> | <b>15.13%</b> |
| <b>P-P Detector ‡</b> | <b>0%</b> | <b>0%</b>   | <b>0%</b>   | <b>0%</b> | <b>0%</b>  | <b>0.16%</b> | <b>4.9</b>  | <b>3.81%</b>  | <b>3.63%</b>  |

– All percentage values are normalized to the baseline:

† *Set-associative llc* serves as the baseline (undefended) configuration.

‡ *P-P Detector: single-address+massive-address* ping-pong detector.

times on Spike), where lower success rates indicate stronger defenses. The experimental results indicate that under the traditional set-associative LLC architecture, the success rates of GE, CT and CT-Fast are nearly 100%. However, due to noise during Spike runtime, the success rates of PPP and CTPP drop to approximately 92% and 95%, respectively. Compared to existing defense solutions, CEASER-S, which employs a periodic remap strategy with an average of 100 accesses per LLC cache block, offers nearly no defense against CT and CT-Fast. In contrast, ping-pong detector provides almost 100% defense against both CT and CT-Fast. MIRAGE, which eliminates address conflicts at the design level, is capable of defending against all conflict-based cache side-channel attacks. DT4+EV10, combining the detector with a on-demand remap strategy, achieves nearly 100% defense against CT and CT-Fast but fails to defend against CTPP, which the ping-pong detector can defend against with almost 100% effectiveness. Additionally, ping-pong detector also offers near-complete defense against GE and PPP.

## 5.2 Performance Analysis and Comparison

Evaluation on SPEC CPU 2017 benchmarks (10B instructions/case) demonstrates that the ping-pong detector achieves strong security with small performance overhead. Compared to a traditional set-associative LLC, it reduces the average miss rate (represented by misses per kilo instructions, MPKI) to 0.16% – the lowest among all evaluated defenses (Table 1). This reduction reflects higher cache hit rates, better data retention, and faster execution, as detailed in Appendix A Fig. 13(a). Notably, performance impacts vary by workload: 548.exchange2 benefits from a 16.7% MPKI reduction, whereas 508.namd and 523.xalancbmk exhibit moderate increases of 18.8% and 4.6%, respectively.

For the remap-based defense scheme, we quantify runtime overhead using remaps per giga instructions (RPGI). As shown in Appendix A Fig. 13(b), the SA mechanism dominates remap overhead (>89%) in 96% of cases. Exceptions include 523.xalancbmk (sequential pattern), where MA and DT4+EV10 show distinct overheads (17 vs. 430), and 519.lbm (irregular access), where MA avoids

false positives but SA raises RPGI above 30. Overall, our ping-pong detector reduces RPGI by 85% (4.9 vs. DT4+EV10’s 32.8) and improves MPKI by 47%.

Power analysis shows the randomized LLC dominates power consumption (96.6% of total), while detector components ( $\leq 1.20\%$ ) and remaps (2.2%) introduce minimal overhead — resulting in a 10.93% dynamic, 3.53% static, and 3.81% total average power increase across 23 test cases compared to baseline. As detailed in Appendix A Fig. 13(c), static power remains the primary contributor in both baseline and detector configurations, with the detector increasing static power by 3.5% (1.627W vs. 1.572W), 99.5% of which stems from the LLC. Dynamic power grows modestly with instruction count, adding 6.6 mW (detector: 67.0 mW vs. baseline: 60.4 mW), primarily from the randomized LLC (64.7 mW), while remapping and tag cache contribute 1.51 mW and 0.80 mW, respectively. Compared to defenses like MIRAGE (19.22% area, 17.34% power overhead), our solution achieves robust security with far lower costs: 3.63% area (81% reduction) and 3.81% power (78% reduction), as shown in Table 1. Component-level ratios are further dissected in Appendix A Table 2, where Randomized LLC includes detection logic and Remap quantifies remapping overhead.

## 6 Conclusion

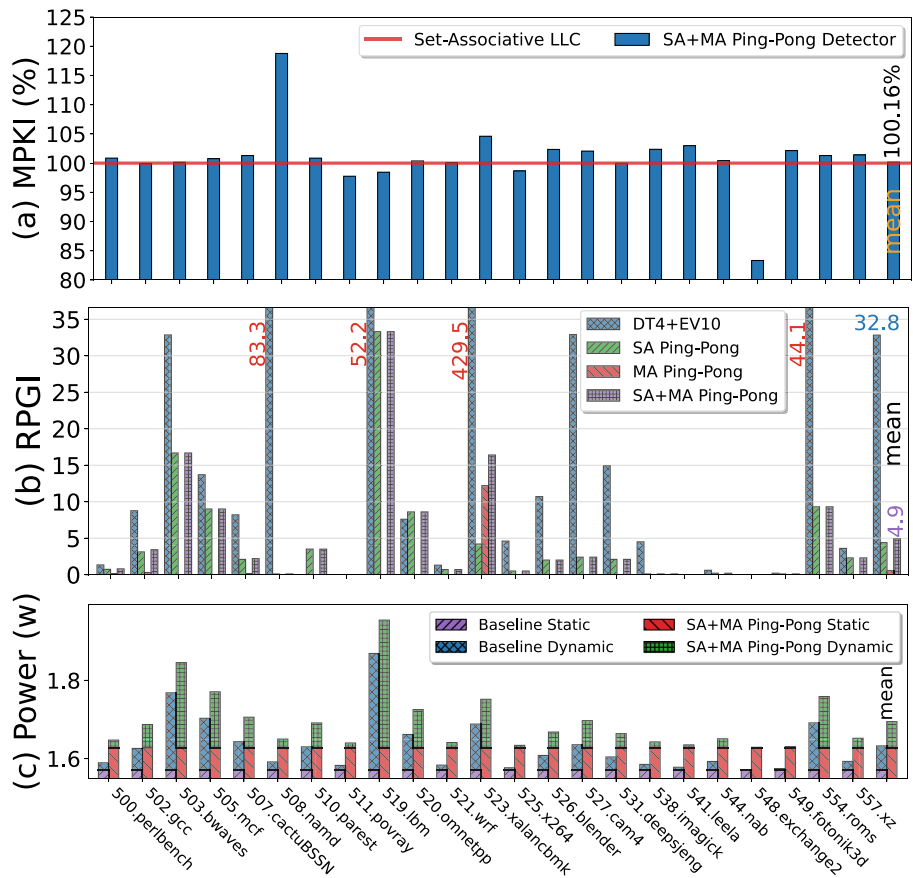
This paper investigates the security vulnerabilities in existing randomized cache defenses. We observe two distinct ping-pong access patterns: single-address (e.g., CT, CT-Fast) and massive-address (e.g., CTPP, PPP, GE). Conflict-based side-channel attacks fundamentally depend on the successful execution of eviction set searching algorithms to achieve their malicious objectives. Therefore, we argue that by obstructing the attacker’s ability to search for eviction sets, subsequent attacks can be effectively thwarted. To defend existing eviction set searching algorithms, we develop a ping-pong detector capable of identifying both patterns. The solution incorporates an optimized tag cache design to minimize runtime impact while providing effective detection. Compared to an undefended set-associative LLC, the ping-pong detector employs two complementary protection schemes: its single-address mechanism effectively counters CT and CT-Fast, while the massive-address mechanism defends against pruning-based algorithms including CTPP, PPP, and GE. Remarkably, the defense structure incurs moderate runtime overhead: only 4.9 remaps are triggered in executing every billion instructions on average, and the LLC cache miss rate increases by merely 0.16%; for area and power overhead, it introduces 3.63% and 3.81% respectively. These measurements suggest our approach can mitigate conflict-based attacks while maintaining reasonable system efficiency.

**Acknowledgements.** This work was partially supported by the National Natural Science Foundation of China under grant No. 62172406.



### A Experimental Setup and Details

Figure 13 presents the MPKI, RPGI, and power consumption of the detector across each SPEC-CPU 2017 benchmark; Table 2 details the power breakdown of the SA+MA ping-pong detector components; Table 3 specifies the experimental platform configuration.



**Fig. 13.** Performance analysis showing (a) misses per K instructions (MPKI) comparison, (b) remaps per G instructions (RPGI) comparison, and (c) power consumption across benchmarks compared to baseline.

**Table 2.** Power breakdown of ping-pong detector (SA+MA) components

| Power Types   | Randomized LLC | Tag Cache | Remap |
|---------------|----------------|-----------|-------|
| dynamic power | 96.6%          | 1.2%      | 2.2%  |
| static power  | 99.5%          | 0.5%      | 0.0%  |
| total power   | 96.55%         | 1.20%     | 2.25% |

**Table 3.** Processor and caches

| Component                 | Configuration  |
|---------------------------|--|
| Core                      | 2-core, in-order, 3GHz, IPC=2                                      |
| L1 I/D-Cache per Core     | 32KB, 64-set, 8-way, LRU, MSI                                      |
| L2 Cache per Core         | 256KB, 1024-set, 4-way, LRU, MSI, Exclusive                        |
| LLC (shared across cores) | 4MB, 4096-set, 16-way, LRU, MESI, Inclusive                        |
| Tag Cache                 | 26KB, 512-set, 16-way, Exclusive<br>Random + Small Ping-Pong First |

## References

1. Liu, F., Yarom, Y., Ge, Q., Heiser, G., Lee, R.B.: Last-level cache side-channel attacks are practical. In: 2015 IEEE Symposium on Security and Privacy, pp. 605–622 (2015)
2. Osvik, D.A., Shamir, A., Tromer, E.: Cache attacks and countermeasures: the case of AES. In: Topics in Cryptology–CT-RSA 2006, pp. 1–20. Springer (2006)
3. Percival, C.: Cache missing for fun and profit. In: BSDCan Ottawa (2005)
4. Ristenpart, T., Tromer, E., Shacham, H., Savage, S.: Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds. In: Proceedings of the 16th ACM Conference on Computer and Communications Security, pp. 199–212 (2009)
5. Liu, F., Ge, Q., Yarom, Y., Mckeen, F., Rozas, C., Heiser, G., Lee, R.B.: Catalyst: Defeating last-level cache side channel attacks in cloud computing. In: 2016 Ieee International Symposium on High Performance Computer Architecture, pp. 406–418 (2016)
6. Irazoqui, G., Eisenbarth, T., Sunar, B.: S\$A: A shared cache attack that works across cores and defies VM Sandboxing – and its application to AES. In: 2015 IEEE Symposium on Security and Privacy, pp. 1–15 (2015)
7. Gruss, D., Maurice, C., Mangard, S.: Rowhammer.js: a remote software-induced fault attack in javascript. In: Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 300–321. Springer (2016)
8. Hähnel, M., Cui, W., Peinado, M.: High-Resolution side channels for untrusted operating systems. In: 2017 USENIX Annual Technical Conference, pp. 299–312 (2017)

9. Page, D.: Partitioned cache architecture as a side-channel defence mechanism. IACR Cryptology ePrint Archive **2005**, 280 (2005)
10. Kim, T., Peinado, M., Mainar-Ruiz, G.: System-level protection against cache-based side channel attacks in the cloud. In: 21st USENIX Security Symposium, pp. 189–204 (2012)
11. Gruss, D., Maurice, C., Fogh, A., Lipp, M., Mangard, S.: Prefetch side-channel attacks: bypassing SMAP and kernel ASLR. In: Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security, pp. 368–379 (2016)
12. El-Sayed, N., Mukkara, A., Tsai, P.-A., Kasture, H., Ma, X., Sanchez, D.: KPart: a hybrid cache partitioning-sharing technique for commodity multicores. In: 2018 IEEE International Symposium on High Performance Computer Architecture, pp. 104–117 (2018)
13. Ramkrishnan, K., Zhai, A., McCamant, S., Yew, P.C.: New attacks and defenses for randomized caches. arXiv preprint [arXiv:1909.12302](https://arxiv.org/abs/1909.12302) (2019)
14. Wang, Z., Lee, R.B.: A novel cache architecture with enhanced performance and security. In: 2008 41st IEEE/ACM International Symposium on Microarchitecture, pp. 83–93 (2008)
15. Liu, F., Lee, R.B.: Random fill cache architecture. In: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 203–215 (2014)
16. Qureshi, M.K.: New attacks and defense for encrypted-address cache. In: 46th Annual International Symposium on Computer Architecture, pp. 360–371 (2019)
17. Werner, M., Unterluggauer, T., Giner, L., Schwarz, M., Gruss, D., Mangard, S.: ScatterCache: thwarting cache attacks via cache set randomization. In: 28th USENIX Security Symposium, pp. 675–692 (2019)
18. Doblas, M., Kostalampros, I.-V., Moreto Planas, M., Hernández Luz, C.: Enabling hardware randomization across the cache hierarchy in linux-class processors. In: Fourth Workshop on Computer Architecture Research with RISC-V, pp. 1–7 (2020)
19. Tan, Q., Zeng, Z., Bu, K., Ren, K.: PhantomCache: obfuscating cache conflicts with localized randomization. In: NDSS (2020)
20. Oren, Y., Kemerlis, V.P., Sethumadhavan, S., Keromytis, A.D.: The spy in the sandbox: P[ractical] cache attacks in javascript and their implications. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1406–1418 (2015)
21. Vila, P., Köpf, B., Morales, J.F.: Theory and practice of finding eviction sets. In: 2019 IEEE Symposium on Security and Privacy, pp. 39–54 (2019)
22. Song, W., Liu, P.: Dynamically finding minimal eviction sets can be quicker than you think for side-channel attacks against the LLC. In: 22nd International Symposium on Research in Attacks, Intrusions and Defenses, pp. 427–442 (2019)
23. Unterluggauer, T., Harris, A., Constable, S., Liu, F., Rozas, C.: Chameleon cache: approximating fully associative caches with random replacement to prevent contention-based cache attacks. In: 2022 IEEE International Symposium on Secure and Private Execution Environment Design, pp. 13–24 (2022)
24. Saileshwar, G., Qureshi, M.K.: MIRAGE: mitigating conflict-based cache attacks with a practical fully-associative design. In: 30th USENIX Security Symposium, pp. 1379–1396 (2021)
25. Sanchez, D., Kozyrakis, C.: The ZCache: decoupling ways and associativity. In: 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 187–198. IEEE (2010)

26. Song, W., Li, B., Xue, Z., Li, Z., Wang, W., Liu, P.: Randomized last-level caches are still vulnerable to cache side-channel attacks! But we can fix it. In: 42nd IEEE Symposium on Security and Privacy, pp. 955–969 (2021)
27. Song, W., Xue, Z., Han, J., Li, Z., Liu, P.: Randomizing set-associative caches against conflict-based cache side-channel attacks. *IEEE Trans. Comput.* **73**(4), 1019–1033 (2024)
28. Xue, Z., Han, J., Song, W.: CTPP: a fast and stealth algorithm for searching eviction sets on intel processors. In: Proceedings of the 26th International Symposium on Research in Attacks, Intrusions and Defenses, pp. 151–163 (2023)
29. Bucek, J., Lange, K.-D., Kistowski, J.v.: SPEC CPU2017 — Next-generation compute benchmark. In: Companion of the 2018 ACM/SPEC International Conference on Performance Engineering, pp. 41–42 (2018)
30. Purnal, A., Turan, F., Verbaauwhede, I.: Prime+ Scope: Overcoming the observer effect for high-precision cache contention attacks. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security, pp. 2903–2917 (2021)
31. Wang, K., Yuan, F., Hou, R., Ji, Z., Meng, D.: Capturing and obscuring ping-pong patterns to mitigate continuous attacks. In: 2020 Design, Automation & Test in Europe Conference & Exhibition, pp. 1408–1413 (2020)
32. Wang, K., Yuan, F., Hou, R., Lin, J., Ji, Z., Meng, D.: CacheGuard: a security-enhanced directory architecture against continuous attacks. In: Proceedings of the 16th ACM International Conference on Computing Frontiers, pp. 32–41 (2019)
33. Yuan, F., et al.: PiPoMonitor: mitigating cross-core cache attacks using the auto-cuckoo filter. In: 2021 Design, Automation & Test in Europe Conference & Exhibition, pp. 1697–1702 (2021)
34. Demme, J., et al.: On the feasibility of online malware detection with performance counters. In: Proceedings of the 40th Annual International Symposium on Computer Architecture, pp. 559–570 (2013)
35. Chen, A., et al.: Detecting covert timing channels with time-deterministic replay. In: 11th USENIX Symposium on Operating Systems Design and Implementation, pp. 541–554 (2014)
36. Fang, H., Doroslovački, M., Venkataramani, G.: Reuse-trap: re-purposing Cache reuse distance to defend against side channel leakage. In: 2020 57th ACM/IEEE Design Automation Conference, pp. 1–6 (2020)
37. Han, J., Wang, Z., Ma, H., Song, W.: Spike-FlexiCAS: a RISC-V processor simulator supporting flexible cache architecture configuration. *J. Softw.* **36**(9), 1–15 (2025)
38. Muralimanohar, N., Balasubramonian, R., Jouppi, N.: Optimizing NUCA organizations and wiring alternatives for large caches with CACTI 6.0. In: 40th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 3–14 (2007)
39. UC Berkeley, RISC-V International: Spike RISC-V ISA Simulator Documentation (2023). <https://chipyard.readthedocs.io/en/latest/Software/Spike.html>