GIR-Cache: Mitigating Conflict-Based Cache Side-Channel Attacks via Global Indirect Replacement

Hao Ma[1,2], Zhidong Wang[1,2], Da Xie[1,2], Ciyan Ouyang[1,2], and Wei Song[1,2(✉)]

[1] State Key Laboratory of Cyberspace Security Defense, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, China
songwei@iie.ac.cn

[2] 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. MIRAGE suffers from intolerable area and power overheads. Chameleon's relocation mechanism faces the problem of excessive power/energy consumption. To alleviate these limitations, we employ a dual-mapping randomized cache with global indirect replacement (GIR-Cache). A randomized direct-mapped look up table is designed to eliminate dual-index checking overhead by maintaining the active mapping state of each LLC address. Our approach effectively mitigates 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 · global indirect replacement

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

S. K. Cha and J. Park (Eds.): ISC 2025, LNCS 16186, pp. 196–215, 2026.
https://doi.org/10.1007/978-3-032-08124-7_10

from the victim program. Existing studies have demonstrated that such conflict-based attacks have been used to recover encryption keys [3], exfiltrate sensitive user data from cloud environments [4,5], break sandbox defenses [6], and even steal information in what is considered secure trusted execution environments [7].

Cache partitioning was one of the early defenses proposed to defend against conflict-based attacks [5,8,9]. By separating private information from ordinary data [10], cache partitioning makes it impossible for attackers to cause conflicts and evict crucial data. However, cache partitioning relies on a trusted operating system to differentiate between private and ordinary data [11]. Furthermore, when privacy data cannot be easily separated from ordinary data using cache partitioning, the approach becomes ineffective.

Cache randomization [12–18] has emerged as a promising defense mechanism. By randomizing the locations of cache blocks [19,20], it prevents attackers from predicting the address-to-set mapping. As a result, attackers cannot reliably determine congruent addresses beforehand and must instead dynamically discover them during execution, making eviction-based attacks significantly harder to orchestrate. Some advanced defense schemes have combined cache randomization with skewing for enhanced protection. For instance, CEASER-S [15] employs a skewed cache structure with periodic remapping to mitigate eviction set searching algorithms such as *Group Elimination* (GE) [20,21,25] and *Prime Prune Probe* (PPP) [1–3] but fails to thwart *Conflict Testing* (CT) [15] and *Conflict Testing-Fast* (CT-Fast) [21]. Chameleon Cache [22] 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, cache blocks in the VC require periodic reinsertion into the LLC, resulting in three block relocations per cache miss. These operations not only consume extra power but also reduce the available bandwidth. MIRAGE [23] proposes to eliminate attacker-controlled associativity evictions through decoupled metadata storage and multi-step cuckoo relocation. The design incurs a documented 22% storage overhead for metadata structures, with additional system-level impacts including: reduced memory density from metadata partitioning, increased logic complexity for relocation management, and introduced non-negligible runtime performance overheads. Inspired by ZCache [24], DT4+EV10 [25,26] 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) [27].

Current randomized cache structures exhibit two key limitations: insufficient security and high overheads. In this paper, we introduce GIR-Cache to show that traditional set-associative caches can be made secure to thwart all existing eviction set searching algorithms. Compared to existing randomized cache designs, our proposal shows advantages in terms of security, cache hit rate, area, and power consumption. Our contributions are as follows:

1. **Attack-resistant design** using global indirect replacement with dual-mapping, mitigating conflict-based attacks with only 0.53% runtime, 2.47% area, and 3.35% power overhead.
2. **Latency-optimized access** via an index predictor (a randomized direct-mapped lookup table), reducing metadata accesses by 47.3% and dynamic power consumption by 4.5% compared to random selection.
3. **Lightweight implementation** requiring only LLC hit and eviction logic modifications, with no additional metadata storage or ISA changes.

The structure of this paper is organized as follows: Sect. 2 provides necessary background. Section 3 presents the threat model and analyzes why existing eviction set searching algorithms fail under our GIR mechanism. Section 4 details the implementation of GIR-Cache and its lookup table (GIR-LUT). Section 5 evaluates the defense's security and hardware overheads (cache miss rate, relocation frequency, power, and area). Finally, Sect. 6 concludes the paper.

2 Background

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

2.1 Randomized Caches

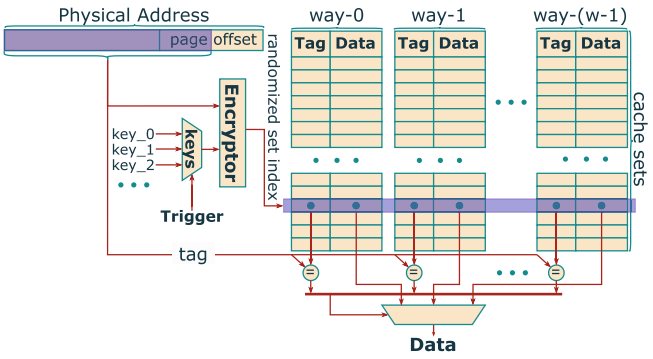

Fig. 1. A randomized set-associative LLC structure.

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 [26]. 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 and 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.

Recent defense mechanisms [15,16,22,23] have adopted skewed cache architectures that enhance security by preventing attackers from precisely predicting cache set indices. This approach invalidates traditional attacks by drastically increasing the required eviction set size, rendering obtained sets ineffective. In skewed caches, addresses are either **fully congruent** (mapping to identical sets across all partitions) or **partially congruent** (inconsistent across partitions). By splitting cache ways into independent skew-partitions – each with a unique mapping key – fully congruent addresses become harder to collect, forcing attackers to rely on partially congruent ones.

While skewed caches enhance security, CEASER-S remains vulnerable to CT/CT-Fast attacks [25], and excessive skewed-partitions can degrade performance. Maintaining multiple concurrent mappings not only incurs area overhead but also complicates both access processing and conflict detection logic. These architectural constraints collectively degrade LLC bandwidth availability and impair runtime performance for applications. GIR-Cache alleviates these limitations by demonstrating that randomized set-associative LLCs can maintain robust security against conflict-based attacks while preserving performance with modest overhead.

2.2 Eviction Set Searching Algorithms

Although cache randomization prevents attackers from deducing address mapping in traditional set-associative caches, it cannot effectively defend against increasingly faster eviction set searching algorithms that gather congruent addresses at runtime. GE [20,21,25] begins with a large random address pool (typically >SW addresses, where **S is cache sets and W is ways**). The pool must contain ≥W congruent addresses. The algorithm iteratively partitions addresses into W+1 groups per round, discarding at least one useless group each iteration until only W addresses remain. The attacker verifies candidate groups until obtaining a minimal eviction set, with LLC time complexity of $O(SW^2)$.

PPP searches eviction sets across different LLC replacement policies [1–3]. The alogrithm begins by priming the LLC with a large random address pool, iteratively pruning conflicting addresses until the LLC is fully populated. The attacker then probes the refined pool: each probe first accesses the target address, then detects evictions through cache misses upon re-access. High-latency addresses indicate congruence with the target. This process requires $O(SW)$ memory accesses under LRU replacement, increasing to $O(SW^2)$ for random replacement.

The CT algorithm first accesses a target address, then sequentially tests random addresses to detect congruence [15]. When a random address conflicts with the target (which occurs with probability $\frac{1}{W}$ under random replacement), alternating accesses between them will reveal congruence through cache misses. Each

random address that causes a target miss is identified as congruent and added to the eviction set. This iterative process continues until completion, requiring $O(SW^2)$ operations to construct a minimal eviction set. The method remains equally effective for permutation-based replacement policies like LRU, maintaining the same time complexity. CT-Fast [29] optimizes the CT algorithm specifically for LRU replacement policies. While its initial phase mirrors the standard CT approach, the process accelerates after finding the first congruent address. With N already collected congruent addresses, the attacker only needs to test W-N candidate addresses in subsequent searches. This is achieved by first accessing the target, then revisiting all N known congruent addresses – making the target more easily to be evicted.

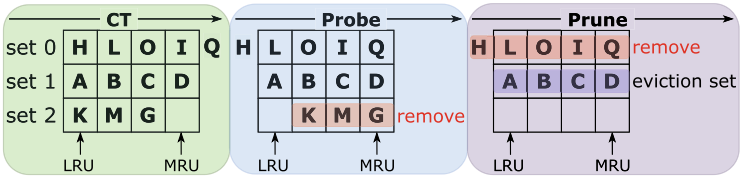

Fig. 2. Simplified model of the CTPP eviction set searching algorithm.

CTPP [27] combines CT and PPP advantages for LRU caches. As shown in Fig. 2, after the CT phase distributes addresses across a set-3, way-4 LRU LLC, three patterns emerge: sets with >W (set 0), =W (set 1), or <W (set 2) congruent addresses. Only sets with exactly W addresses can evict the target – others are eliminated through iterative Probe (removing cache hits from underpopulated sets like set 2) and Prune phases (filtering cache misses from overpopulated sets like set 0). Within 3–5 iterations of these Probe and Prune phases, this converges to a perfect W-sized eviction set (set 1).

3 Threat Analysis and Defense Methods

This section first establishes our threat model, then explains how GIR mechanism works during LLC misses and why conflict-based attacks fail under GIR-Cache's protection mechanism.

3.1 Threat Model

This paper focuses on preventing attackers from successfully obtaining 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 targeting inclusive LLC. Specifically, we only consider SAE-based (Set-Associative Eviction) attacks [23]. 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 Global Indirect Replacement Cache

Prior research [25] demonstrates that in a P skew-partitions set-associative cache, the probability of an address mapping to the same set across all P partitions is $\frac{1}{S^P}$, making it nearly impossible for attackers to obtain perfect eviction sets containing fully congruent addresses. Our defense employs a dual-mapping randomized set-associative architecture to mitigate conflict-based attacks. Each LLC access dynamically selects a mapping via predictor-guided selection for hit determination. During eviction, GIR remaps and relocates target blocks to new sets, forcing the eviction of a random block instead. The relationship between the LLC-missed address and the ultimately evicted address is completely uncorrelated, making any observed conflicts statistically meaningless. As a result, attackers gather only invalid addresses, preventing the searching of a usable eviction set and thwarting further exploitation.

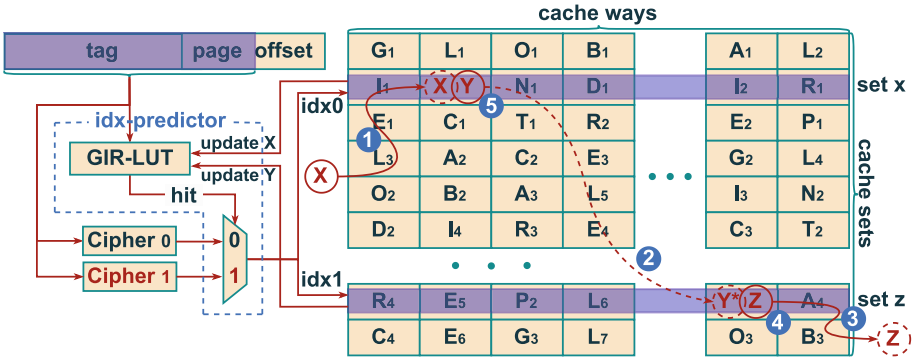

Fig. 3. The situation of gir-cache when handling cache misses.

To more intuitively demonstrate GIR-Cache's replacement operation, Fig. 3 shows a miss handling scenario. ① An LLC miss on address X forces random set selection, while its congruent address Y uses the predictor-provided mapping (here, both ultimately mapping to set x via idx0). ② The replacement algorithm evicts Y from set x to create space. ③ Y remaps to set z as Y* using idx1, triggering eviction of a random block Z from set z while updating replacement

metadata. ④ This transfers Y's valid entry from set x to set z, invalidating the original metadata. ⑤ X then occupies the vacated slot in set x. In this scenario, attackers only observe X's miss and Z's eviction – unable to detect Y*'s preservation due to the dual-mapping scheme. Consequently, they collect meaningless random addresses, rendering conflict-based attacks ineffective.

3.3 Thwart Existing Search Algorithms Using GIR

Conflict-based side-channel attacks fundamentally rely on searching eviction sets for target addresses. Figure 4 illustrates the GE algorithm's operation, where an initial pool of N addresses undergoes pruning. The process begins by dividing addresses into W+1 groups, which contain at least W congruent addresses to target T (represented by green boxes B,G,M,S). To eliminate one group, the attacker first accesses T, then sequentially accesses W groups. Upon re-accessing T, a cache miss indicates that the W groups contain at least W congruent addresses, allowing the (W+1)-th group to be discarded. If T results in a cache hit, the algorithm redistributes the N addresses into W+1 new groups without accessing the remaining group. This process repeats until only W (or slightly more) congruent addresses remain.

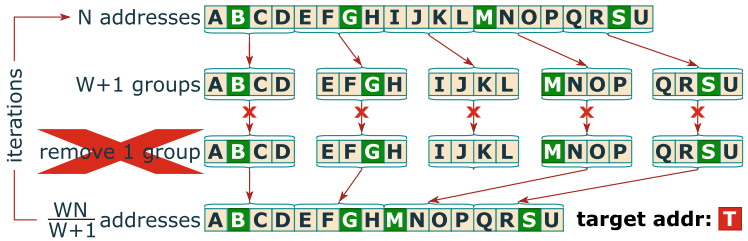

Fig. 4. GE algorithm fails under GIR mechanism (red '×' marks failure steps). (Color figure online)

GIR-Cache thwarts this process through its relocation mechanism. In a conventional set-associative cache, a miss on accessing target T would allow a congruent address to directly evict T from the LLC. However, GIR-Cache breaks this linkage by remapping T to a new set and replacing a block from that set – there is no congruence between the missed address and the ultimately evicted address. As a result, when repeatedly accessing the W groups, GE can no longer reliably evict T, which often remains cache-resident. Since only random addresses have a minimal chance of displacing T from the LLC, GE fails to evict the target using congruent addresses. This prevents GE from eliminating target groups or gathering valid eviction sets, rendering the attack ineffective under GIR-Cache.

Figures 5 and 6 show how GIR's policy disrupts congruent address collection for target T in a 4-set, 4-way LRU cache using CT/CT-Fast and PPP algorithms. Here, ①-③ indicate the execution sequence of the corresponding algorithm,

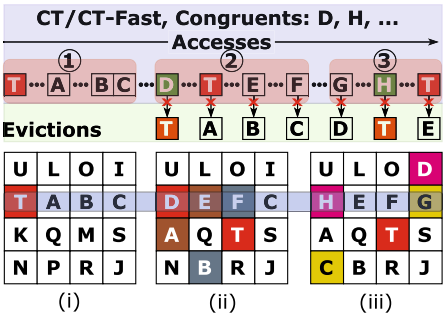

Fig. 5. CT/CT-Fast fails in 4-set, 4-way LRU LLC under GIR (red '×' marks failure steps). (Color figure online)

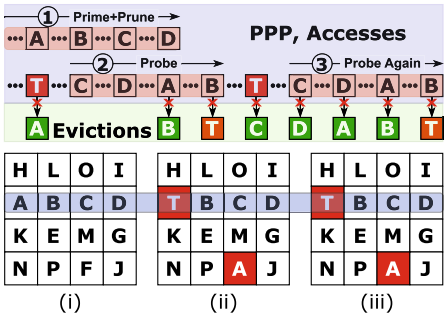

Fig. 6. PPP fails in 4-set, 4-way LRU LLC under GIR (red '×' marks failure steps). (Color figure online)

while (i)-(iii) represent the distribution of cache blocks during the algorithm's execution (Same-colored blocks denote cache-missed and relocated addresses). In CT and CT-Fast (Fig. 5), attackers access a large number of random addresses to collect those congruent with a chosen target address T. The process involves: ① initially accessing T followed by random addresses (A–H...) mapped to T's set; ② Through interleaved accesses to random addresses and target T, observed misses on T identify congruent addresses (e.g., the green blocks D and H); ③ Repeat until four congruent addresses are collected to form the eviction set. However, GIR introduces three critical failure modes: (i) Previously identified congruent addresses (e.g., D/H) lose their ability to evict T; (ii) Eviction attempts trigger T's relocation to new cache sets with random blocks selection (e.g., block M being evicted instead of T when accessing D); (iii) During subsequent testing, the attacker can only observe the evicted random addresses (K, P, N, I), while the genuinely congruent addresses (A–D) remain in the LLC due to relocation.

In the PPP algorithm shown in Fig. 6, ① the Prime+Prune phase begins with the attacker filling the LLC with random addresses (prime set) and subsequently pruning conflicting ones until the remaining addresses such as A–D can coexist in the cache. Timed re-accesses of target T and the prime set identify congruent addresses (LLC misses), though private cache filtering often misaligns LLC and software access orders. ② The Probe phase gathers partially congruent addresses (e.g., the green blocks A and B), and ③ re-probing aligns LLC order to finally collect four addresses (A–D) for T. However, GIR disrupts the PPP algorithm's operation: (i) The pruning phase assumes T's cache set remains unaffected by addresses self-conflicts, but (ii) accessing target T (resulting in a miss) triggers eviction attempts (e.g., selecting address A), which activates GIR's mechanism – relocating A to a new set while randomly ejecting an unrelated block (e.g., F). (iii) Consequently, the collected supposedly-congruent addresses prove arbitrary and invalid, completely corrupting the eviction set searching process.

The CTPP algorithm fails under GIR-Cache protection. During the CT phase, attackers may successfully locate an initial address pool containing W

supposed-congruent addresses for target T. However, the dual-mapping mechanism ensures most of these addresses are actually random (non-congruent addresses). Subsequent probe and prune phases rapidly eliminate these random addresses. Meanwhile, truly congruent addresses cannot evict target T – they instead relocate T to a new set while randomly ejecting another address, forcing cache hits upon T's re-access. Consequently, attackers only collect random addresses that are useless for mounting effective attacks.

In summary, the GIR mechanism ensures security through target address relocation. When relocated addresses move to new sets and replace a random address, attackers can only observe the missed address and the ultimately evicted address – which share no congruence. This makes it impossible for attackers to use collected random addresses to evict the target, thereby effectively preventing eviction set searching. GIR-Cache defeats all major eviction set searching algorithms (GE/CT/CT-Fast/PPP/CTPP), experimental results (present in Sect. 5) confirming robust security without compromising performance.

4 Hardware Implementation Details

This section presents the hardware design details, covering the pseudo-code for our new hit/eviction handling methods, the GIR-Cache overview structure, and the GIR-LUT design.

4.1 Handling Hits and Evictions in GIR-Cache

Algorithm 1. Optimized GIR-Cache Hit Function

```
    Input: addr
    Output: hit (true/false)
    GirLut[addr]: 1-bit mapping predictor (0/1)
 1: function GIR_HIT(addr)
 2:     i ← GirLut[addr], s0 ← idxer0(addr), s1 ← idxer1(addr)
 3:     (first_s, second_s) ← (i ? (s1, s0) : (s0, s1))
 4:     if hit(addr, first_s) then
 5:         return true
 6:     else if hit(addr, second_s) then
 7:         GirLut[addr] ← (1 ⊕ i)
 8:         return true
 9:     end if
10:     return false
11: end function
```

To implement the GIR strategy, we modified the LLC hit mechanism as described in Algorithm 1. When accessing an address (*addr*) in the LLC, the LLC queries

the GIR-LUT to retrieve the stored index value i while simultaneously computing both potential mapping results for the address – obtaining sets *s0* and *s1* from the two mapping functions (*idxer0()* and *idxer1()*).

This index value determines the primary mapping to use: indexer0 when i is 0, or indexer1 when i is 1. The LLC then performs the first hit check using the indicated mapping function. A successful hit confirms the GIR-LUT's prediction was correct, and the corresponding cache block is returned. When the initial check fails due to prediction error, the system performs a secondary hit verification using the alternative mapping function that was not selected by GIR-LUT. In case of a second hit, the LLC updates the GIR-LUT with the index value $(1 \oplus i)$, correcting the prediction for future accesses. If both checks fail, the address is confirmed as not present in the LLC, and the miss handling proceeds through the standard coherence protocol without modifying the GIR-LUT.

To further support the GIR policy, we redesigned the eviction procedure as specified in Algorithm 2. The algorithm employs an S[W] array structure for cache block organization. The input parameter *addr* specifies the relocation target address, with s[w] representing its associated cache block. The LLC calculates two candidate sets (*s0* and *s1*) for *addr* using both indexers. By evaluating the current set s against *s0* and *s1*, the system determines the remapping strategy. For illustration, we assume *addr* initially maps to *s0* through *idxer0*, then demonstrate its remapping to destination set *ds* via *idxer1* (where $i = 1$).

Algorithm 2. Simplified GIR-Cache Evict Function

```
   Input: addr, s, w
   Cache structure: S[W] is an array representing the cache sets and ways
1: function GIR_EVICT(addr, s, w)
2:     s0 ← idxer0(addr), s1 ← idxer1(addr)
3:     i ← (s==s0), ds ← (i ? s1 : s0)
4:     dw ← replace(ds)
5:     if ds[dw].meta.valid then
6:         evict(ds[dw])
7:     end if
8:     ds[dw] ← s[w], GirLut[addr] ← i
9: end function
```

A special case occurs when both indexers map to the same set (*s0*==*s1*), the LLC bypasses set *ds* relocation and directly evicts the replacement-selected block s[w]. While s[w]'s address is congruent with the missed address, Sect. 3.2's probability analysis ($\frac{1}{S^F}$) demonstrates this collision occurs with only $\frac{1}{S^2}$ probability, rendering it statistically implausible for attackers to accumulate W such congruent addresses. The operation finalizes by depositing the missed block's address, metadata and data in s[w]'s vacated space while simultaneously updating the GIR-LUT with the new mapping index i to preserve predictor accuracy for future accesses.

4.2 GIR-Cache Hardware Overview

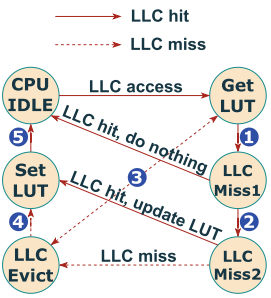

Fig. 7. GIR-LUT operating state machine.

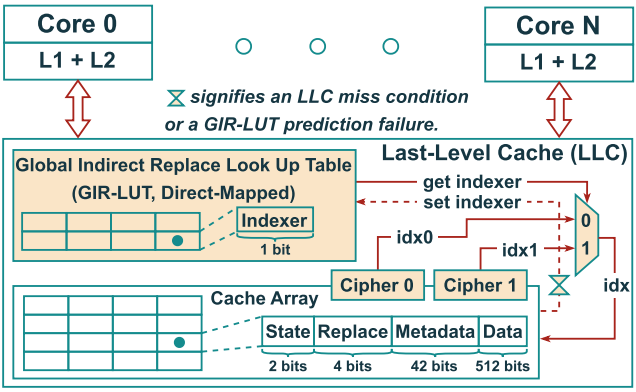

Fig. 8. Overall structure of the GIR-Cache.

Figure 8 presents the hardware implementation of GIR-Cache. Our design enhances a conventional randomized set-associative LLC with a dual-mapping mechanism for address translation. Each memory address undergoes two independent cryptographic transformations through dedicated cipher units before LLC access, with the encryption keys remaining inaccessible to potential attackers. To maintain high-speed LLC access while avoiding the overhead of dual-mapping checks, we implement a direct-mapped GIR-LUT (1-bit *indexer-prediction* per entry: 0 = index0, 1 = index1) that tracks address mappings without LLC modification.

4.3 GIR-LUT Design and Implementation

Figure 7 shows the GIR-LUT state machine workflow during LLC accesses. When the LLC receives a memory request, it queries the GIR-LUT for the active 1-bit *indexer* (using for index-prediction), entering the *GetLUT* state. ① If the prediction yields a cache hit, the state transitions to *CPU IDLE* without GIR-LUT modification. ② On a miss, the system enters *LLCMiss2*, using the alternative index; a hit here triggers a transition to *SetLUT* and inverts the corresponding GIR-LUT bit. For an LLC miss, the state advances to *LLCEvict*, where the controller computes both indexer sets, randomly selects one, and performs replacement. ③ Relocated addresses' GIR-LUT values are verified: *indexer = 0* maps to index1's set, *indexer = 1* to index0's. ④–⑤ The GIR-LUT is updated with the missed addresses' verified indexer and corrected indexers for relocated addresses, ensuring future mapping accuracy and maintaining lookup efficiency.

Building upon PSA-Cache's demonstration that predictive access methods maintain processor efficiency without introducing substantial latency [30], our experimental results further validate this approach. As illustrated in Appendix A Fig. 9, our evaluation running 10G instructions of SPEC-CPU 2017 on Spike [28] shows an arithmetic mean prediction accuracy of 95.8% across all test cases when using GIR-LUT. Without GIR-LUT – when employing a random indexer selection strategy for each access – the same benchmark yields only 52.1% accuracy. This represents an 83.9% improvement in prediction accuracy through GIR-LUT implementation. The enhanced prediction further reduces 43.7% metadata accesses, resulting in approximately 4.5% lower dynamic power consumption compared to the non-GIR-LUT approach.

5 Security and Performance Evaluation

In this section, we evaluate the security of existing defense architectures by executing eviction set searching algorithms and comparing their performance with GIR-Cache. We measure four key metrics: cache miss rate, relocation frequency, power consumption, and area overhead. Using SPEC-CPU 2017 benchmarks (simulating 10G instructions per test case), we present results in Appendix A 10, 11, and 12. Each fig includes an *Average* bar group (rightmost) representing the average across all 23 test cases. Power and area measurements are derived from CACTI-6.5 [32] in a 32 nm technology.

5.1 Experimental Platform

To systematically evaluate our proposal, we first establish the experimental framework based on the processor configuration detailed in Appendix A Table 4. Our evaluation platform emulates Intel Coffee Lake 9th Generation cache architecture through a behavioral cache model [31].

5.2 Security Evaluation and Comparison of Different Caches

Table 1(a) demonstrates that in a 16 MB GIR-Cache (16384-set, 16-way), all existing eviction set searching algorithms fail to collect congruent addresses. Data shows the average number of supposed-congruent addresses identified during distinct attack phases, aggregated from 100 executions per algorithm in the Spike simulation environment. The PPP algorithm initializes its address pool with 1.05 times the LLC's addressable cache blocks, designed to contain at least 16 (LLC ways) congruent addresses. After the warm-up and priming phases, GIR's dual-mapping mechanism enables the attacker to collect approximately 67 presumed-congruent addresses. However, most of these addresses are actually random, resulting from post-relocation collisions between target addresses and random addresses. The pruning phase eliminates 67% of these conflicting addresses, leaving only 22 supposed-congruent addresses cached. During probing, if the target is evicted by a genuine congruent address, it undergoes relocation and subsequently evicts an arbitrary random address. After relocation, these 22 addresses

are unlikely to collide with the post-relocation target address, ultimately keeping the target in the LLC and producing zero true congruent addresses in the probing phase. Similarly, pruning-based algorithms like GE fail to effectively filter address groups. CTPP finds 37 supposed-congruent addresses during CT phase due to dual-mapping, but subsequent probe-prune cycles reduce this to zero. CT and CT-Fast cannot directly evict target addresses – attackers can only collect irrelevant random addresses. These results conclusively demonstrate that all tested algorithms are neutralized by GIR-Cache's protection mechanisms.

However, if attackers forgo searching eviction sets against the LLC and instead attempt to target the GIR-LUT, they would still face significant challenges: First, the GIR-LUT employs hash-based address encryption to achieve randomized mapping, making brute-force attacks far more difficult than searching for congruent addresses in the LLC. Second, even if attackers somehow obtain a congruent address of GIR-LUT and successfully evict the predicted value (causing prediction failure), the LLC would require an additional metadata access. This not only increases dynamic power consumption but also extends the observable timing window from $T_{\text{gir-lut}} + T_{\text{llc-hit}}$ to $T_{\text{gir-lut}} + 2T_{\text{llc-hit}}$ – yet, since $T_{\text{llc-miss}} \gg T_{\text{llc-hit}} \gg T_{\text{gir-lut}}$, the added $T_{\text{llc-hit}}$ delay remains negligible in practice. Crucially, experimental results (Sect. 4.2) demonstrate a 95.8% prediction accuracy for the GIR-LUT, meaning the attacker-introduced delay would likely be masked by the 4.2% baseline prediction failure noise, rendering GIR-LUT congruent address-based attacks ineffective.

Table 1. Security evaluation and comparison of defense schemes normalized to set-associative LLC.

(a) Supposed-congruent addresses collected under GIR.

Algorithms	S.Con. addrs
Prime,Prune,Probe	67, 22, 0
CT/CT-Fast	0
CTPP(CT Stage)	37
CTPP(Probe - 1)	20
CTPP(Prune - 1)	20
CTPP(Probe - 2)	0
CTPP(Prune - 2)	0

(b) Security comparison normalized to baseline (non-secure set-associative cache).

Structures	GE	CT	CT-Fast	PPP	CTPP
Baseline	100%	100%	100%	92%	95%
CEASER-S	0%	**100%**	**100%**	0%	0%
DT4+EV10	0%	0%	0%	0%	**27%**
MIRAGE	0%	0%	0%	0%	0%
MIRAGE-Lite	0%	0%	0%	0%	0%
Chameleon	0%	0%	0%	0%	0%
GIR-Cache	**0%**	**0%**	**0%**	**0%**	**0%**

Table 1(b) presents defense architectures security effectiveness measured by the success rates of five fast eviction set searching algorithms (each executed 100 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 employs a periodic remap strategy with an average of 100

Table 2. Performance comparison of cache defense structures in MPKI, RelocPKI, and Area normalized to set-associative LLC baseline.

Structures	MPKI (%)		RelocPKI	Area	
	Average	Geo-Mean		Size (mm^2)	Normalized (%)
Set-Associative	100.00	100.00	0	33.447111	100.00
CEASER-S	100.42	110.51	0.090512	34.186495	102.21
DT4+EV10	103.10	125.59	0.394710	34.299493	102.55
MIRAGE	107.54	134.96	0	42.318032	126.52
MIRAGE-Lite	107.67	135.20	0	40.548203	121.23
Chameleon	102.24	112.60	8.412051	34.247375	102.39
GIR-Cache	**100.53**	**107.17**	**2.758631**	**34.272218**	**102.47**

accesses per LLC cache block, offers nearly no defense against CT and CT-Fast. 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. MIRAGE and MIRAGE-Lite eliminate address conflicts at the design level and are capable of defending against all conflict-based cache side-channel attacks. Chameleon implements a defense strategy: when evicting cache blocks in skewed caches, it first relocates the target block to the VC, then evicts another block from VC. This method can decrease contend between cache blocks, preventing attackers from identifying addresses congruence with target addresses. Thus, Chameleon successfully defends against all current conflict-based eviction set searching algorithms. GIR-Cache introduces a global indirect replacement during eviction: selected target blocks undergo remapping and relocation to new cache sets, with the LLC subsequently evicting random addresses. This method prevents attackers from collecting congruent addresses for the target address. Extensive empirical validation through hundreds of experiments conclusively demonstrates GIR-Cache's capability to provide 100% protection against existing eviction set searching algorithms.

5.3 Comparison of Cache Miss Rate and Relocation Frequency

As shown in Table 2 and Appendix A Fig. 10, MPKI (misses per kilo instructions) overhead varies significantly across defense mechanisms, with GIR-Cache achieving the lowest geometric mean increase (7.17%) – substantially outperforming MIRAGE (34.96%) and Chameleon (12.60%). Lower MPKI values indicate higher hit rates and thus better runtime performance. Notably, GIR-Cache maintains this efficiency while providing equivalent security guarantees against LLC address conflict analysis, as all three schemes (including non-remapping MIRAGE) effectively prevent attacker inference. Extreme outliers (e.g., 399.73% overhead for MIRAGE on 505.mcf) highlight the importance of geometric mean comparisons, where GIR-Cache exhibits both the smallest Geo-Mean and second-smallest average overhead (0.53%).

Table 3. Power overhead of cache defense structures normalized to set-associative LLC baseline.

Structures	Power (W)				Power (%)		
	Static	Dynamic	Relocation	Total	Static	Dynamic	Total
Set-Associative	6.151365	0.183213	0	6.334578	100.00	100.00	100.00
CEASER-S	6.271933	0.192186	0.001166	6.464119	101.96	104.90	102.04
DT4+EV10	6.299212	0.232351	0.040905	6.531563	102.40	126.82	103.11
MIRAGE	7.551611	0.289576	0	7.841187	122.76	158.05	123.78
MIRAGE-Lite	7.323553	0.268865	0	7.592418	119.06	146.75	119.86
Chameleon	6.285316	0.287658	0.099229	6.572975	102.18	157.01	103.76
GIR-Cache	**6.292547**	**0.254086**	**0.064643**	**6.546633**	**102.30**	**138.68**	**103.35**

As detailed in Appendix A Fig. 11, we quantify runtime overhead in relocation-based defenses through relocations per kilo instructions (RelocPKI), where lower values correlate with higher cache hit rates and reduced penalties. GIR-Cache demonstrates superior efficiency, achieving a RelocPKI less than one-third of Chameleon's (2.76 vs. 8.41 on average) and a correspondingly 4.23× lower MPKI (0.53% vs. 2.24%). While 87% of benchmarks maintain RelocPKI ≤ 5, exceptions like 503.bwaves and 519.lbm – where stride/irregular access patterns degrade LLC utilization – show spikes exceeding RelocPKI = 15. These outliers underscore GIR-Cache's resilience, maintaining a 1.76× lower geometric mean MPKI (7.17% vs. 12.60%) despite edge cases.

5.4 Comparison of Power and Area Consumption

Appendix A Fig. 12 and Tables 2 and 3 reveal significant power and area overhead variations across schemes. Compared to the baseline (6.151W static/0.183W dynamic), MIRAGE's 75% redundant metadata results in the highest overheads: 26.52% area, 22.76% static power (7.552W), and 58.47% dynamic power (0.290W). MIRAGE-Lite reduces this through 50% extra metadata (21.23% area, 19.06% static power, 46.75% dynamic power), albeit with a 1.72% MPKI increase. Chameleon shows moderate power increases (2.18% static/57.01% dynamic), while GIR-Cache maintains optimal efficiency with merely 2.30% static (6.293 W) and 38.68% dynamic (0.254 W) power growth.

As illustrated in Table 2, both Chameleon and GIR-Cache require cache block relocation during LLC miss-induced evictions, exhibiting identical hardware area overheads of 2.39% and 2.47% respectively. Their key architectural difference lies in the relocation mechanism: Chameleon performs at least three distinct cache block relocations per cache miss, while GIR-Cache requires only a single block relocation. This design difference results in Chameleon's RelocPKI being 3.05× higher than GIR-Cache (8.41 vs. 2.76). The power characteristics reveal more nuanced tradeoffs. According to Table 3, while Chameleon's relocation operation consumes 1.54× more power than GIR-Cache (99.229 mW vs. 64.643 mW,

or 53.50% higher), this difference is less than the 3× multiple suggested by their relocation counts. This efficiency stems from Chameleon's use of an 8-block fully associative victim cache (VC) for relocation operations, compared to GIR-Cache's approach of performing candidate set selection across the entire LLC. When considering power consumption per LLC relocation operation but excluding bandwidth effects, the VC-based implementation proves more power-efficient per access. However, GIR-Cache incurs additional dynamic power from maintaining and accessing its randomized direct-mapped GIR-LUT during LLC hits and misses. As detailed in Appendix A Table 4, when the LLC is 16 MB, the GIR-LUT occupies 32 KB, consuming an area of 0.050117 mm^2 (0.146% of the entire GIR-Cache's 34.272218 mm^2). Its static power is 0.010815 W (0.165% of total power) and dynamic power is 0.000858 W (0.013% of total power). In overall power consumption compared to Chameleon, GIR-Cache demonstrates superior efficiency: while maintaining comparable static power (6.293 W vs. 6.285 W, a 0.13% increase), it achieves an 11.81% reduction in dynamic power (0.254 W vs. 0.288 W), ultimately resulting in 0.40% lower total power (6.547 W vs. 6.573 W).

6 Conclusion

This paper investigates the security vulnerabilities in existing randomized cache defenses. Since conflict-based side-channel attacks fundamentally rely on successful execution of eviction set searching algorithms to achieve their malicious objectives, we demonstrate that disrupting these search capabilities can effectively prevent subsequent attacks. Our proposed solution, GIR-Cache, a new randomized cache defense that fundamentally thwarts conflict-based side-channel attacks by disrupting eviction set searching through its global indirect replacement mechanism. The structure employs a dual-mapping set-associative cache with a randomized GIR-LUT that reduces metadata access and dynamic power consumption during LLC hits while ensuring robust protection. During misses, target addresses undergo remapping and relocating to new sets, evicting non-congruent addresses to prevent attackers from detecting cache conflicts and neutralize all major eviction set searching algorithms (e.g., CT, CT-Fast, CTPP, PPP and GE). Remarkably, GIR-Cache delivers robust security against cache attacks with moderate overheads: 0.53% runtime, 2.47% area, and 3.35% power.

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

A Experimental Setup and Details

Figure 9 compares the first-attempt LLC hit success rates between GIR-LUT and random index selection when running SPEC-CPU 2017 benchmarks for 10 billion instructions on GIR-Cache. The rightmost two bars represent averages across all 23 test cases, demonstrating a 95.8% success rate using GIR-LUT

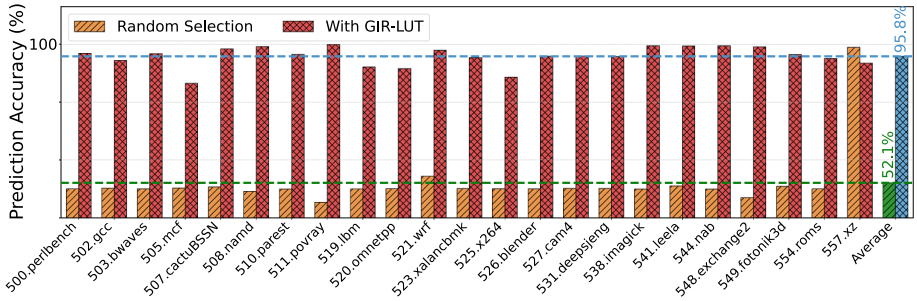

Fig. 9. GIR-LUT prediction accuracy vs. random selected index hit probability in GIR-Cache.

Table 4. Processor and caches configuration

Component	Configuration
Core	8-core, in-order, 3 GHz, IPC = 2
L1 I/D-Cache per Core	32 KB, 64-set, 8-way, LRU, MSI
L2 Cache per Core	256 KB, 1024-set, 4-way, LRU, MSI, Exclusive
LLC (shared across cores)	16 MB, 16384-set, 16-way, LRU, MESI, Inclusive
	32 KB randomized direct-mapped GIR-LUT (1-bit/entry)

versus 52.1% with random selection – a 43.7% reduction in metadata access overhead achieved by GIR-LUT implementation.

Table 4 presents the experimental processor configuration and cache hierarchy specifications, featuring a 16 MB GIR-Cache as the last-level cache with an integrated 32 KB GIR-LUT.

Figures 10, 11, and 12 collectively present experimental results from 10G-instruction SPEC CPU 2017 runs, with each figure's rightmost *Average* bars showing mean values across all test cases.

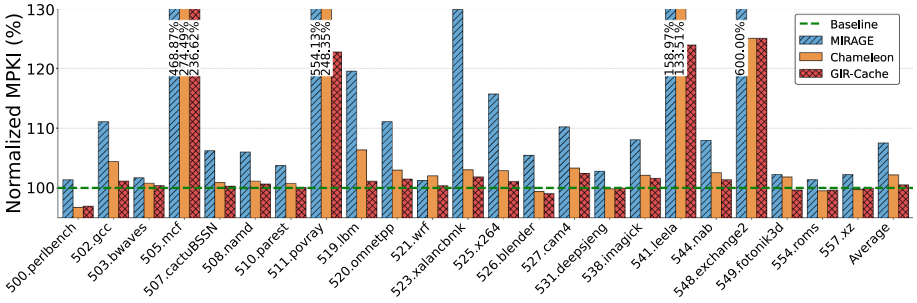

Fig. 10. Misses per K instructions (MPKI) comparison across benchmarks compared to set-associative (baseline).

Figure 10 compares cache miss rates per kilo instructions (MPKI) across defense structures, with the y-axis showing percentage overhead relative to a conventional set-associative LLC baseline (green dashed line). Each benchmark group on the x-axis displays three adjacent bars (left to right: MIRAGE, Chameleon, GIR-Cache). The arithmetic mean baseline miss rate for 505.mcf, 511.povray, 541.leela, and 548.exchange2 is 0.003651 MPKI, while the defenses exhibit increases of 399.73%, 241.53%, and 209.01% respectively on these workloads. Geometric means (34.96%, 12.60%, 7.17%) further contextualize these outliers, with GIR-Cache consistently achieving the lowest overhead.

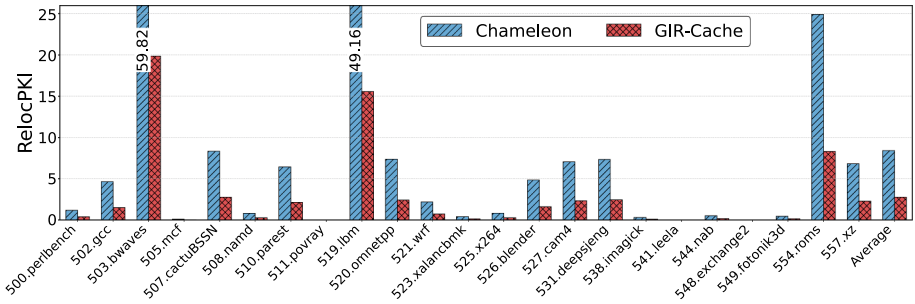

Fig. 11. Relocations per K instructions (RelocPKI) across benchmarks.

Figure 11 compares RelocPKI between Chameleon and GIR-Cache across all benchmarks. The y-axis measures relocation frequency (per kilo instructions), with GIR-Cache's average (2.76) markedly lower than Chameleon's (8.41). Test cases are grouped by workload type, with outliers labeled (e.g., 503.bwaves and 519.lbm exceeding RelocPKI = 15 due to stride/irregular accesses).

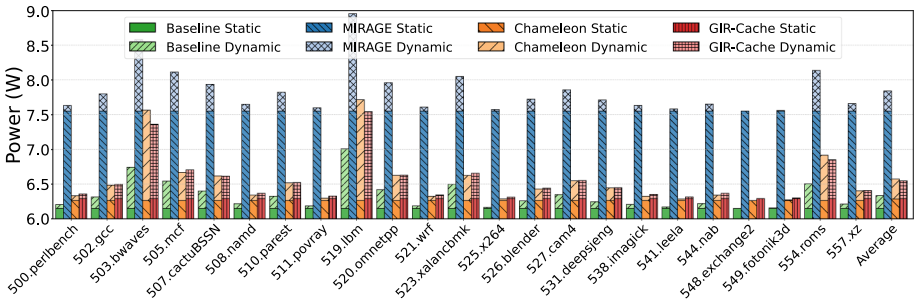

Fig. 12. Power comparison across benchmarks compared to set-associative (baseline).

The power analysis in Fig. 12 employs stacked-bar visualization to distinguish base (static power) and variable (dynamic power) components, while comparatively evaluating four architectures: conventional set-associative cache and three

security-enhanced designs (MIRAGE, Chameleon, and GIR-Cache), with the rightmost column explicitly showing cross-architecture average power values for reference.

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: Pointcheval, D. (ed.) CT-RSA 2006. LNCS, vol. 3860, pp. 1–20. Springer, Heidelberg (2006). https://doi.org/10.1007/11605805_1
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., et al.: 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. Hähnel, M., Cui, W., Peinado, M.: High-Resolution Side Channels for Untrusted Operating Systems. In: 2017 USENIX Annual Technical Conference, pp. 299–312 (2017)
8. Page, D.: Partitioned cache architecture as a side-channel defence mechanism. IACR Cryptology ePrint Archive 2005:280 (2005)
9. 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)
10. 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)
11. 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)
12. Ramkrishnan, K., Zhai, A., McCamant, S., Yew, P.C.: New attacks and defenses for randomized caches. arXiv preprint arXiv:1909.12302 (2019)
13. 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)
14. Liu, F., Lee, R.B.: Random fill cache architecture. In: 2014 47th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 203–215 (2014)
15. Qureshi, M.K.: New attacks and defense for encrypted-address cache. In: 46th Annual International Symposium on Computer Architecture, pp. 360–371 (2019)

16. 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)
17. 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)
18. Tan, Q., Zeng, Z., Bu, K., Ren, K.: PhantomCache: obfuscating cache conflicts with localized randomization. In: NDSS (2020)
19. Oren, Y., Kemerlis, V.P., Sethumadhavan, S., Keromytis, A.D.: The spy in the sandbox: practical cache attacks in Javascript and their implications. In: Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, pp. 1406–1418 (2015)
20. 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)
21. 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)
22. 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)
23. 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)
24. Sanchez, D., Kozyrakis, C.: The ZCache: decoupling ways and associativity. In: 43rd Annual IEEE/ACM International Symposium on Microarchitecture, pp. 187–198. IEEE (2010)
25. 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)
26. 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)
27. 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)
28. UC Berkeley, RISC-V International: Spike RISC-V ISA Simulator Documentation (2023). https://chipyard.readthedocs.io/en/latest/Software/Spike.html
29. Purnal, A., Turan, F., Verbauwhede, 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)
30. Calder, B., Grunwald, D., Emer, J.: Predictive sequential associative cache. In: Proceedings of the Second International Symposium on High-Performance Computer Architecture, pp. 244–253 (1996)
31. 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)
32. 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)