SoK: So, You Think You Know All About Secure Randomized Caches?

Anubhav Bhatla*
bhatlaanubhav2001@gmail.com
Indian Institute of Technology Bombay

Hari Rohit Bhavsar*
haribhavsar@cse.iitb.ac.in
Indian Institute of Technology Bombay

Sayandeep Saha
sayandeepsaha@cse.iitb.ac.in
Indian Institute of Technology Bombay

Biswabandan Panda
biswa@cse.iitb.ac.in
Indian Institute of Technology Bombay

Abstract

Over the past decade, numerous side-channel attacks on shared resources, such as the Last-Level Cache (LLC), have exposed security risks in the form of flush-based, conflict-based, and occupancy-based attacks, driving the development of secure cache designs. To defend against conflict-based attacks, which is one of the most effective classes of side-channel attacks, many modern designs randomize LLC set indexing to hinder eviction set construction. Various randomized cache designs have been proposed recently, offering distinct security guarantees. While these designs incorporate several microarchitectural modifications (we call them *security knobs*) over the conventional set-associative cache to ensure security, the individual impact of these microarchitectural modifications has never been evaluated. This leaves a gap in the understanding of randomized LLCs – the design space has not been explored completely and systematically.

In this SoK, we identify and systematically analyze the design knobs employed in state-of-the-art secure randomized cache designs that mitigate conflict-based attacks. Using conventional set-associative caches as our baseline, we study five key knobs: skewing, extra invalid tags, high associativity, replacement policy, and remapping. We also evaluate their impact on occupancy-based attacks. Our findings show that no single knob provides a comprehensive security guarantee. Instead, only specific combinations of knobs yield effective protection, while others offer little to no security benefit.

1 Introduction

Last-level Cache (LLC) hides off-chip memory access latency and improves system performance. Typically, L1 and L2 caches are dedicated to each core, while the shared LLC serves all cores. However, this shared LLC can facilitate side-channel attacks that may reveal sensitive data, such as cryptographic keys [6, 11, 24], user data in the cloud [17], and architecture of neural networks [44]. Three types of attacks are possible in set-associative LLCs: flush-based [46], conflict-based [24] and occupancy-based [32]. In general, these attacks exploit the timing difference between an LLC hit (fast) and a miss (slow). Flush-based attacks are shared memory attacks where the attacker and the victim *share* cache lines, and the attacker flushes the cache line from the LLC and checks for LLC hit and miss. A future reload hit means the victim has filled the cache line into the LLC. In contrast, conflict-based [24] and occupancy-based [32] attacks do not expect shared memory. Conflict-based attackers exploit LLC address-to-set mapping, causing set conflicts where victim lines evict attacker lines, known as a set-associative eviction (SAE). Attackers use SAEs to build an *eviction set*, a collection of addresses that map to the same LLC set. Another class of attacks, called occupancy attacks, involves occupying cache lines to estimate the proportion accessed by a victim process [32]. Occupancy attacks do not need eviction sets.

In recent years, several defense mechanisms have been proposed to counter these attacks, among which LLC randomization has emerged as a promising candidate for flush and conflict-based attacks, and recently for occupancy-based attacks [19]. There have been several proposals for a randomized LLC design. Some of the initial proposals such as RPCache [42], CEASER [28], CEASER-S [29], and ScatterCache [43] were quickly compromised by newer and faster attacks [26, 29, 34], and eventually improved designs were proposed. Recent secure randomized LLC designs, such as Mirage [30], introduce multiple microarchitectural modifications to a conventional set-associative cache to provide security. However, the security guarantee of each of these designs is usually evaluated as a whole, without clarifying the role of each microarchitectural modification. Secondly, many of the designs propose similar modifications or identical security implications. Therefore, it is crucial to identify the set of security "knobs" or modifications used by these designs. We identify, evaluate, and provide a taxonomy of five knobs, both in isolation and in combination. The knobs of interest are: (i) skewing, (ii) extra invalid tags, (iii) high

*These authors contributed equally to this work

associativity, (iv) replacement policy, and (v) remapping. We introduce high associativity as a knob in this study and argue its effectiveness in enhancing security. Even though cache associativity has previously been evaluated [18] in the context of security, no prior work has evaluated high associativity as a security knob. We quantify security using the two metrics – i) *eviction rate*, defined as the probability that an *eviction set* evicts the target address; and, ii) number of evictions to generate an eviction set with real-world eviction set finding algorithms. We iteratively analyze the impact of each knob on the attacker's achievable eviction rate, noting the conditions under which each knob is effective and explaining why it works. For the most promising knob combinations, we use the second metric to make the evaluation stronger.

Through our evaluation, we answer questions such as (i) *why a security knob works and to what extent?* and (ii) *what is a minimal set of knobs to secure a conventional set-associative cache?* This modular systematization enhances the understanding of randomized caches, helping designers and security analysts select optimal designs. Several works have evaluated randomized caches from different perspectives. **CaSA** [15] developed a framework to quantitatively analyze the security of randomized caches against covert channel attacks. **Song et al.** [34] analyzed remapping periods for certain cache designs exploiting design flaws. **CacheFX** [18] proposed a generic framework for assessing cache resilience to side-channel attacks. Our focus differs: we study the extra microarchitectural features added for security, both individually and collectively.

Finally, occupancy-based attacks pose a significant threat to cache security [13], and even complex designs like Mirage [30] cannot fully mitigate them. Partitioning [5, 14] is an effective defense but incurs a substantial performance penalty. For randomized caches, only designs that use partitioning or soft partitioning-based solutions [19, 21] can defend against occupancy-based attacks. Moreover, the security of these designs depends entirely on their partitioning properties. This raises the question: *Can the knobs proposed by secure randomized cache designs mitigate occupancy-based attacks?*

1.1 Key Insights and Contributions

The key insights and contributions of this paper are related to randomized cache designs that primarily mitigate conflict-based attacks. We also provide insights related to a recent randomized cache that mitigates occupancy-based attacks.

Systematic evaluation of security knobs (Section 3). We iteratively analyze the impact of each knob on the attacker's achievable eviction rate, noting the conditions needed for each knob to be effective and explaining why it works. Combinations of different security knobs are also explored, which gives rise to some interesting designs not explored in prior literature. One important insight is that certain knobs are dependent on each other to provide security gains.

Decoupling has no effect on security (Section 3.3.1). We argue that decoupling the tag and data store does not affect the security of a cache design. Designs like Mirage [30] and Maya [7] decouple the tag and data store to accommodate extra invalid tags in the tag store. The security of these designs comes from the additional invalid tags, not from the decoupling itself, which primarily introduces overhead in terms of indirection pointers and design complexity.

High associativity for security (Section 3.4). We identify high associativity as a powerful knob for mitigating conflict-based attacks. Our findings show that increasing cache associativity significantly improves security with minimal changes to conventional set-associative designs, making it a promising direction for secure randomized caches. While associativity up to 16 ways has been explored for security [18], we advocate for extending beyond 16 ways, up to 128 ways.

Trade-off with occupancy-based attacks (Section 4). We examine how various security features impact occupancy-based attacks. First, we find that randomness in skew selection and eviction policy provides some protection, but it is not foolproof. While deterministic eviction policies like LRU are more effective against conflict-based attacks, they are less effective for occupancy-based attacks. Second, in full occupancy-based attacks, the eviction policy—global or local—has minimal impact on security. We evaluate one recent randomized cache design against occupancy-based attack, which provides soft partitioning (e.g., SassCache [19]). Hard partitioning is an ideal mitigation for occupancy-based attacks (e.g., static way-based partitioning). Our results show that SassCache makes such attacks significantly harder than other randomized caches. We also analyze low-occupancy attacks (attacks that use buffers smaller than cache capacity), by extending prior evaluations [13] to high-associativity cache designs, showing that they offer similar resistance to such attacks as SassCache and ScatterCache.

2 Background

2.1 Threat Model

We assume the following capabilities in our attacker, which is a standard threat model [7, 29, 30, 34, 43] for LLC attacks.

- The attacker and victim run on separate cores in a multi-core system, sharing the LLC.
- The attacker has reverse-engineered the mapping from virtual to physical addresses.
- The attacker can access the LLC by sending memory requests to her own data but cannot access cache lines outside her address space. She can accurately probe the LLC hit/miss status for her own data accesses.
- She can flush her cache blocks from the cache hierarchy.
- In the case of randomized caches, she knows the encryption algorithm, but the block cipher key for set mapping is concealed.

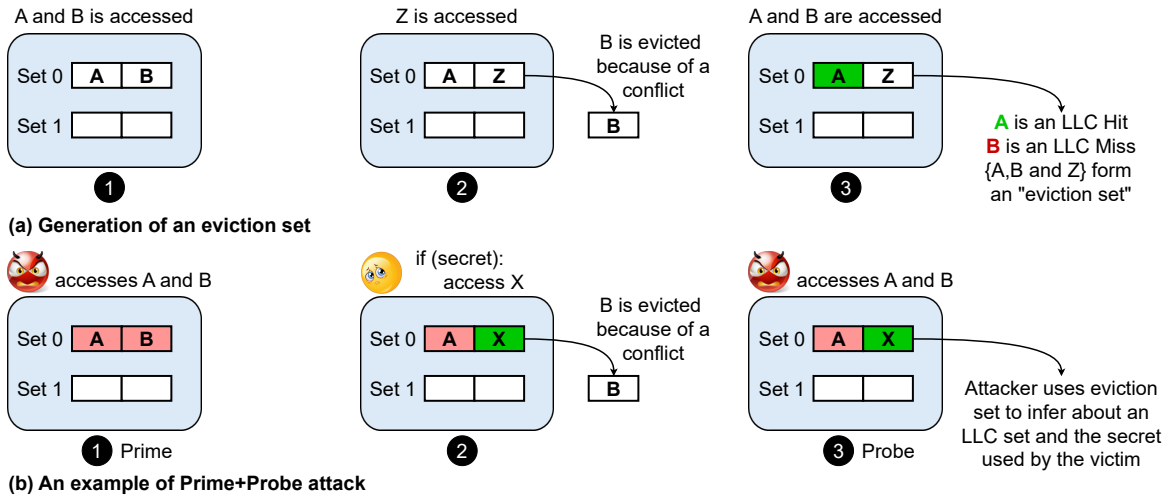

Figure 1: An overview of the steps involved in (a) generating an eviction set, and then using it to mount the (b) Prime+Probe [24] conflict-based attack which leaks information about the victim's secret-sensitive accesses.

2.2 LLC Side-Channel Attacks

In general, three kinds of LLC side channel attacks are prevalent that primarily exploit the latency (timing) difference between an LLC hit and a miss.

Conflict-based (or eviction-based) attacks. These attacks depend on the attacker observing the eviction of its cache lines due to addresses accessed by a victim, causing set conflicts. As a first step, the attacker identifies an eviction set (defined as a collection of addresses with a high degree of contention within a subset of cache lines). For example, such a subset of cache lines is constituted by an entire LLC set for conventional set-associative caches. Figure 1(a) illustrates the process of generating such an eviction set for a set-associative cache. Once the eviction set is established, the attacker can launch a conflict-based side-channel attack, such as Prime+Probe [24]. Figure 1(b) depicts a Prime+Probe attack, where the spy first (❶) *primes* (fills) an LLC set shared with the victim, and then (❷) the victim executes and potentially evicts the spy's cache lines, and finally (❸) the spy *probes* (checks) which lines were evicted to infer the victim's memory access behavior.

Occupancy-based attacks. These attacks rely on the attacker observing changes in its LLC working set due to the victim's cache usage. For example, in website fingerprinting [32], the attacker fills a significant portion of the LLC with its own cache lines and then lets the victim execute. During the *probe* step, the attacker can infer the victim's LLC occupancy based on evictions of its own lines. Unlike set-conflict-based attacks, no eviction set is required for these attacks, but they lead to more coarse-grained information leakage.

Shared memory-based (or flush-based) attacks. These attacks depend on the attacker observing hits on addresses shared with the victim. In the Flush+Reload [46] attack, the attacker *flushes* a shared line from the cache and waits for the victim to execute. In case the victim accessed the shared line, the attacker can infer this during the *reload* step. These shared lines usually consist of read-only shared memory from shared libraries or memory shared through de-duplication by the OS [45]. Such attacks are limited by the requirement of sharing of cache lines between distrusting processes.

2.3 Scope

Our analysis focuses on randomized cache designs that mitigate conflict and occupancy-based side-channel attacks on the LLC. We do not address flush-based attacks, as these attacks are inherently mitigated by randomized cache designs that include a security domain ID (SDID) per cache line, ensuring no cross-SDID flushing of cache lines.

2.4 Eviction Set Creation Algorithms

For the rest of this paper, we abbreviate the number of sets in the cache using S, the number of ways using W, and the size of the cache using N. An eviction set is a collection of addresses built to detect the access of a particular target (victim) address or to evict the target (victim) from the cache. Such eviction sets are constructed using some heuristics as discussed below.

To create an eviction set, most existing eviction set search algorithms [22, 24, 29, 34] start from a large collection of candidate addresses (called *candidate set*). Then, they filter out addresses from this set that are *congruent* to this target address, and therefore, can evict the target address. Two addresses are said to be congruent if they map to the same subset of cache lines, and can evict each other with non-zero probability. In case this eviction probability is one, we call such addresses *fully congruent*. Otherwise, we say they are

partially congruent. The **Single Holdout** [24] method tests one address at a time from the candidate set. It evaluates this address by testing whether the other addresses in the candidate set still evict the target. If not, this address is included in the eviction set, otherwise, it is excluded. This method requires $O(S^2 \cdot W^2)$ or $O(N^2)$ cache accesses, which is too slow to conduct any practical attacks. The **Group Elimination** [29] [41] method reduces the complexity to $O(N \cdot W)$ or $O(S \cdot W^2)$ accesses. They utilize the simple observation that if the candidate set is divided into $W+1$ groups, then one group certainly does not contain any congruent address and, therefore, can be removed. **Conflict Testing** [34], in contrast, builds the eviction set by testing each address (instead of starting from a candidate set) to check whether or not it is congruent with the target address. The time complexity of this algorithm is $O(S \cdot W^2)$.

In randomized caches, finding addresses that are always congruent to the target is highly unlikely, as the each address have multiple cache lines to map depending on the randomness. Therefore, attackers target partial congruence–addresses that are congruent with the target only with a certain probability. Algorithms like **Prime, Prune and Test** [34] (or Prime, Prune and Probe [26]) add a pruning step to remove addresses from the candidate set which are self-evicting, i.e., addresses which can evict each other with certain probability. The goal is to form a candidate set that remains in the cache, so accessing the target address and then re-accessing the candidate set reveals exactly those elements congruent to the target address. In terms of complexity, these algorithms are similar to conflict testing. Another optimization, **Prune+Plumtree** [22], salvages the pruned addresses to generate other eviction sets, ultimately reaching a complexity of $O(S \cdot W^2 \cdot \log(S))$ for caches with random replacement and $O(S \cdot W \cdot \log(S))$ with LRU replacement. However, the authors mention that "it does not apply to more advanced cache architectures such as skewed randomized caches".

2.5 Randomized LLC Designs

In a traditional set-associative cache, the address-to-set mapping is computed (deterministically) by a subset of the line address bits. Conflict-based attacks exploit this easy-to-discover determinism in address mapping to create efficient eviction sets. To counter this, randomized cache designs introduce *unpredictability* in the mapping, which makes the mapping hard to discover in the first place. However, this is not the only security-enhancing modification. In this section, we survey different generations of randomized caches.

The first generation of secure randomized caches aimed to randomize cache interference between an attacker and a victim process to prevent useful information leakage. One approach was **RPCache** [42], which used a permutation table to randomize the address-to-set mapping. This was followed by **CEASER** [28], which employed a block cipher to randomize the address-to-set-mapping. However, since set conflicts could still occur with block-cipher-based set indexing, CEASER introduced *dynamic remapping* to update the address-to-set mapping (by re-keying the block cipher) before an attacker could create an eviction set, making the mapping *updatable*. **PhantomCache** [38] uses localized randomization within a limited number of cache sets, effectively increasing cache associativity to make the discovery of eviction sets harder. Another secure randomized cache, **H^{2}Cache** [48], used a table-based randomization for the L1 cache and block-cipher-based randomization for the LLC.

The first generation of secure randomized caches was compromised by newer and faster eviction set discovery algorithms [29, 41]. In response, **CEASER-S** [29] and **ScatterCache** [43] introduced a skewed associative design on top of randomized address-to-set mapping, providing a probabilistic defense. In this design, the cache ways are partitioned into multiple skews, each with a unique set mapping, and the incoming addresses are randomly assigned to one of the skews. This obfuscates cache accesses and makes eviction set construction more difficult. These caches represent the second generation of secure randomized caches. A recent design, called **ClepsydraCache** [39], uses a similar randomized design to ScatterCache but uses time-based evictions instead of remapping of the set mapping used in ScatterCache.

In the third generation, **Mirage** [30] builds on the V-Way cache design [27], incorporating load-aware insertion and global random eviction. This approach significantly reduces the probability of set-associative evictions (SAEs), making it highly unlikely for attackers to construct eviction sets. **Maya** [7] is a variant of Mirage that optimizes space efficiency with a reuse-based insertion policy. Interestingly, a relatively older design, called **NewCache** [23], uses a register-based dynamic mapping along with randomization to eventually achieve a fully associative mapping. The abstraction of a fully associative cache was also achieved by the **Chameleon Cache** [40], a modification of CEASER-S featuring a fully associative victim cache with a random replacement policy to decouple set conflicts from evictions. **RECAST** [47] is a set-associative cache that uses core-private caches to protect against cross-core attackers. Each cache line is tagged with a secret bit, which determines the line-to-set mapping, thereby creating the abstraction of a fully associative cache. **Song et al.** [36] modified CEASER by introducing a mechanism that detects ongoing attacks and triggers re-keying.

The fourth generation of secure randomized caches was designed to defend against both occupancy-based and conflict-based attacks. *Cache partitioning* is the most straightforward approach as it provides LLC space isolation. **INTERFACE** [21] is a cache design which combines indirect partitioning and a randomized fully associative cache (like Mirage) to address both attack classes. A more interesting recent design is **SassCache** [19], which only relies on randomization of the address-to-set mapping to achieve security against occu-

Table 1: A comparison of secure randomized cache designs. A register stores the set permutation per word line in table-based indexing, unlike block cipher-based indexing, where the same hardware derives the set for all addresses. Reported performance and storage overheads are based on claims from the respective papers.

Design	Indexing Policy	Remapping?	Skews?	Other Security Knobs	Mitigates Conflict based Attacks?	Mitigates Occupancy based Attacks?	Mitigates Flush based Attacks?	Performance Overhead	Storage Overhead
RPCache [42]	Table-based	✘	✘	N/A	✘	✘	✘	0.15%	1%
NewCache [23]	Table-based	✔	✘	N/A	✔	✘	✔	< 1%	10%
CEASER [28]	Block Cipher-based	✔	✘	N/A	✘	✘	✘	1.1%	0%
CEASER-S [29]	Block Cipher-based	✔	✔	N/A	✘	✘	✔	0.7%	0%
ScatterCache [43]	Block Cipher-based	✔	✔	N/A	✘	✘	✔	2%	5%
PhantomCache [38]	Block Cipher-based	✘	✘	Localized Randomization	✘	✘	✘	1.2%	0.5%
Mirage [30]	Block Cipher-based	✘	✔	Decoupling, Load Aware, Global Eviction	✔	✘	✔	1.7%	20%
H^2Cache [48]	Block Cipher-based	✘	✔	N/A	✘	✘	✘	10.7%	0%
Chameleon [40]	Block Cipher-based	✔	✔	Fully Associative Victim Cache	✔	✘	✘	< 1%	< 0.1%
SassCache [19]	Block Cipher-based	✘	✔	Soft Partitioning	✔	✔	✔	N/A[1]	N/A[2]
ClepsydraCache [39]	Block Cipher-based	✘	✘	Time-based Evictions	✔	✘	✔	1.38%	< 8%
Song et al. [36]	Block Cipher-based	✔	✘	Attack Detection, Multi-step Reallocation	✘	✘	✘	0.89%	1.9%
RECAST [47]	Block Cipher-based	✘	✘	Address-Sensitive Secret Generation	✔	✘	✘	2.29%	1.1%
Maya [7]	Block Cipher-based	✘	✔	Decoupling, Load Aware, Global Eviction	✔	✘	✔	-0.2%	-2%
INTERFACE [21]	Block Cipher-based	✘	✔	Decoupling, Load Aware, Global Eviction, Partitioning	✔	✔	✔	3.4%	14%

pancy attacks. In particular, SassCache enhances the address-to-set mapping of a skewed associative randomized cache like ScatterCache and enables LLC space isolation. As evident from the above survey, each generation of randomized caches introduces new modifications to address existing and potential future attacks. However, a notable gap in the current literature is the lack of an ablation study on the individual capabilities of these security modifications. Such a study would not only enhance our understanding of existing designs but could also inspire new, optimized designs that are better suited to varying resource constraints.

3 Systematization of Randomized Caches

What are security knobs? We define microarchitectural design modifications that are specifically implemented to enhance security in secure randomized cache designs as *security knobs*. Ideally, the block cipher would be the first security knob to study, as it is a critical component in nearly every randomized cache design, providing the unpredictability essential for security. It is advisable to use a block cipher that is robust to *shortcut cryptanalytic attacks* [8, 26], and provides a uniform distribution of the encrypted set indices. Latency of the state-of-the-art ciphers [9, 10] can be an issue here. Additionally, the correct implementation of the block cipher is also crucial to the security properties of a secure randomized cache, as highlighted in [31]. However, since the block cipher cannot be considered optional, we exclude it from further analysis. Instead, we focus on other microarchitectural modifications that serve as security knobs. Identifying security knobs is not always straightforward, as there are often dependencies between different knobs. Some of these dependencies are explicit, while others only become apparent through empirical analysis. Additionally, some knobs are merely attributes or components of another knob and are referred to as *sub-knobs*. A taxonomy of the knobs and sub-knobs employed in the randomized cache designs is presented in Figure 2.

3.1 Evaluation Strategy and Simulation Setup

3.1.1 Metrics

Evaluating the full taxonomy of security knobs is challenging, starting with the selection of appropriate evaluation metrics. Prior work, notably [18], advocates using multiple metrics, including: (i) Relative Eviction Entropy (REE), which quantifies information leakage from attacker-victim conflicts; (ii) the difficulty of constructing eviction sets using state-of-the-art algorithms; and (iii) the effectiveness of attacks on cryptographic targets. Our work also adopts a multi-metric approach. We first use a well-established metric—eviction rate—to identify

[1] The authors have used cache hit rate to quantify the performance of their design instead of misses per kilo instructions. Thus, the impact of this design on performance remains unknown.

[2] No storage overhead analysis has been performed by the authors

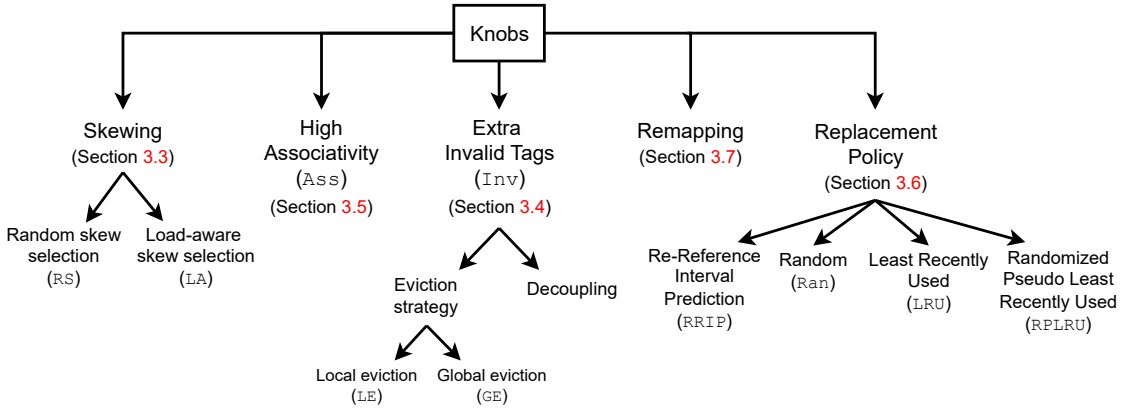

Figure 2: An overview of the knobs and sub-knobs identified in modern secure randomized cache designs, along with their abbreviations used throughout the paper.

cache designs that are promising from a security standpoint, then apply additional metrics for deeper analysis.

Metric-I: Eviction rate. The *eviction rate* is defined as the fraction of times a target address is evicted by an eviction set of a given size, measured over n iterations. In each iteration: (i) a target address x is randomly selected; (ii) its corresponding eviction set is identified; (iii) the target is accessed; (iv) the eviction set is accessed; and (v) it is checked whether the target has been evicted. Prior works [34, 39, 43] also refer to this metric as eviction probability in the context of randomized caches. For strong security, a design should exhibit a low eviction rate even with large eviction sets. This *eviction rate experiment* follows the methodology described in [34]. The rationale behind choosing the eviction rate is as follows:

- The experiment for measuring this metric closely resembles what happens in a real conflict-based attack. For example, in the Prime+Probe attack [24], the eviction rate represents the probability of detecting a secret-dependent victim access.
- Eviction rate is measured for a given eviction set size. If generating an eviction set of that size is already difficult using existing algorithms, it provides a clear, intuitive explanation for a design's security. Notably, the difficulty of eviction set generation is a recognized metric in several works [18], including ours.

Generating an eviction set for the eviction rate experiment is a challenging task due to the existence of various eviction set search algorithms. To avoid algorithmic bias, we assume an oracle provides the eviction set, independent of any specific search algorithm. These sets are identified by sampling a large collection of random addresses, determining their set indices using full knowledge of the block cipher key, and selecting addresses partially congruent to the target address. Note that selecting perfect eviction sets doesn't make our evaluation unrealistic; we later assess how real-world eviction set generation algorithms [26, 34] can replicate such sets.

Metric-II: Number of evictions to generate an eviction set. After filtering secure and promising designs based on eviction rate, we evaluate them using state-of-the-art eviction set finding algorithms [26, 34]. The goal is to assess the "difficulty" of constructing an eviction set of a given size, measured by the number of LLC evictions required. This forms the second key metric in our evaluation and directly impacts the critical design knob—the *remapping period*.

We do not explicitly evaluate specific cryptographic targets for conflict-based attacks. While such targets could serve as an additional metric, we argue that the difficulty of evicting a single victim address—as measured by our eviction rate experiments—naturally generalizes to evicting multiple addresses. Prior work [26] demonstrated this by combining eviction sets for multiple distinct addresses to attack AES. Thus, our single-address eviction rate and eviction set construction results reflect the broader difficulty of targeting cryptographic implementations. In contrast, for occupancy-based attacks, we rely primarily on cryptographic benchmarks, as no alternative metrics are yet established in the literature.

Simulation setup: To analyze the impact of various design knobs on security against conflict-based attacks, we extend the open-source behavioral cache simulation model from [35] to incorporate all knobs outlined in Figure 2. Unless otherwise specified, all experiments use a 2MB LLC with 16-way associativity. For eviction set experiment, we set the number of iterations $n = 1000$, based on empirical observations, as most experiments converge reliably with this value.

3.2 Knob 1: Skewing

Traditional cache designs have one set that a particular address can map to; however, under a `Skew-k` cache, an address could go to one of the `k` possible sets, each belonging to one skew. Skew is one of the most important security knobs used in various secure randomized cache designs starting from CEASER-S [29]. We note two possible sub-knobs based on the skew selection (choosing one of `k` possible sets) strategy

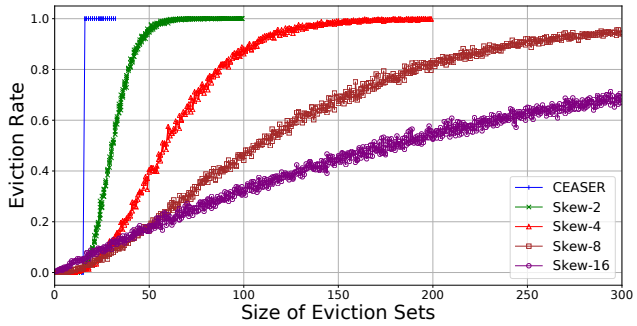

Figure 3: Eviction rate for a varying number of skews (`Skew-k` has `k` skews) with random skew selection and LRU eviction. CEASER does not use any skews and `Skew-2` is a representative for skewed designs such as CEASER-S.

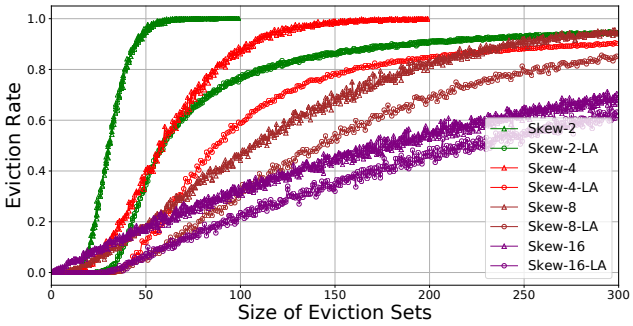

Figure 4: Eviction rate for varying number of skews (`Skew-k` has `k` skews) with random vs load-aware skew selection (`LA`) with LRU eviction.

for an incoming cache line – i) Random skew selection (`RS`); ii) Load-aware skew selection (`LA`).

3.2.1 Random Skew Selection (`RS`)

In random skew selection [29, 43, 48], a new cache line is inserted into a randomly chosen skew. As shown in Figure 3, the eviction rate decreases with the number of skews for a given eviction set size. However, for a fixed number of skews, the eviction rate increases with the eviction set size. The key observation is that there is a non-zero eviction rate even for a small eviction set size, suggesting that even a small eviction set can lead to information leakage.

Explanation. To explain the trend in Figure 3, we refer to [26], which provides the expression for the *eviction probability* (p_e) of an eviction set E of size $|E|$. For a random replacement policy, $p_e = 1-(1-\frac{1}{n_w})^{\frac{|E|}{k}}$, where n_w is the total number of ways across skews and k is the number of skews. For LRU, $p_e = 1 - \text{binom}(\frac{|E|}{k}, \frac{n_w}{k} - 1, \frac{1}{k})$, where binom is the cumulative binomial distribution function. In both cases, p_e decreases as k increases, assuming a fixed $|E|$. The eviction rate in our plots reflects this probability, excluding cache warm-up effects. Notably, even low eviction rates can lead to successful attacks, as shown in prior work [26, 34].

3.2.2 Load-aware Skew Selection (`LA`)

The load-aware skew selection policy compares the remaining capacity of the k-sets that an address can map into, and inserts the address into the set having the largest remaining capacity. In case the remaining capacity is the same for all the skews, the tie is broken with a random skew selection. As shown in Figure 4, load-aware provides slightly better security than random skew selection. However, the gain gradually diminishes with an increase in the skew count.

Explanation. The key difference between random and load-aware skew selection is that the latter depends on the cache state. Let us consider an address e mapping to sets $\{s_1, s_2, \cdots s_k\}$ in skews $\{1, 2, \cdots k\}$. Without loss of generality, we consider the probability that e gets mapped to s_1, which depends on the occupancy of the other skews, which in turn reflects the overall cache state. As a result, e can only evict a target address x when the cache is nearly full—eviction occurs only if all skews are close to capacity. Since eviction set creation algorithms ignore cache state and select e based solely on observed evictions during construction, these sets tend to be ineffective unless the cache is at least $\approx$90% full. However, the eviction rate experiments shown in Figure 4 indicate a slight improvement with load-aware skew selection compared to random skew selection. This counter-intuitive trend, given that higher eviction rates are generally expected as caches fill quickly, was investigated and found to be an artifact of the cache's warm-up state and the method used to compute the eviction rate. A detailed explanation of this behavior is provided in Appendix B. In summary, a minor variation in the eviction rate calculation, which accounts for the size of the warm-up state, reconciles this observation with our overall understanding. We recommend using this alternative calculation, even though it requires significantly more simulation effort, only for a subset of designs with promising security characteristics. Overall, we conclude that load-aware skew selection, even when combined with multiple skews, offers no clear security benefit over random skew selection.

Takeaway

Skewing is useful as it reduces the eviction rate of an eviction set of a particular size. The skew selection policy only has a limited impact on security–load-aware provides a security benefit only when the cache is not full.

3.3 Knob 2: Extra Invalid Tags (`Inv`)

Mirage [30] uses extra invalid tags to ensure that an incoming cache line can find space in its mapped set and not cause an

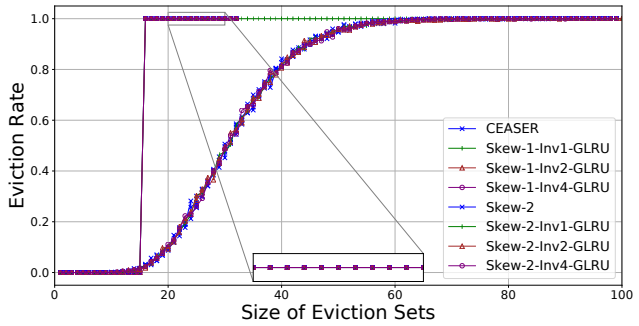

Figure 5: Eviction rate for CEASER and two skews (`Skew-2`) with random skew selection, extra invalid tags (`Inv`), and global LRU eviction (`GLRU`).

SAE. To maintain such invalid tags in the cache, we keep a threshold on the total number of valid entries that can be kept in the cache and trigger the eviction mechanism once this threshold is exceeded. Seeing invalid tags as a knob is motivated by the fact that it involves multiple choices (as sub knobs) to be made regarding cache structure and cache management. The first possible sub-knob is whether to decouple the tag and data store. Another important sub-knob is the eviction policy. Eviction policy is tightly coupled with invalid tags, as the sole reason behind declaring a tag invalid is to control when the evictions are supposed to be made. Therefore, we consider it as a sub-knob for invalid tags.

3.3.1 Decoupled tag store has no security impact

There are two ways in which extra invalid tags can be provisioned in the cache. The first is to use a decoupled tag and data store with extra invalid tag entries in the tag store, which has been used in Mirage. The data store size remains the same as a traditional non-secure cache and the tag store is expanded to store the extra invalid tag ways. An alternate approach to remove the decoupling and instead operate the cache at a lower capacity. In other words, whenever the total valid entries inside the cache exceed a threshold number, we trigger the eviction mechanism so that the cache never fills up. This implies having less number of valid entries in the cache compared to a non-secure cache of the same size.

While these choices indeed qualify as sub-knobs, one observation is that functionality-wise both are the same. We observe this throughout our experiments as the results with and without decoupling are the same. Without loss of generality, we go with no decoupling. Also, we would like to point out that decoupling the tag and data store and adding extra invalid tags leads to a hefty overhead regarding storage, area, and power requirements that Maya [7] takes care of.

3.3.2 Extra invalid tags is not a standalone knob

One crucial observation regarding invalid tags is that they are not useful without skews. A non-skewed cache with invalid tags is nothing but CEASER operating with a lower capacity. This is depicted in Figure 5, where we depict the combination of invalid tags both with and without skews. As expected, skewing has a positive impact on security. Therefore, the rest of our analyses in this subsection will assume a minimum of two skews. However, deciding the proper skew selection policy is complex, as we discuss in the following paragraphs.

3.3.3 Local Eviction (`LE`) vs. Global Eviction (`GE`)

Interplay between skew selection and eviction policy: Given that skews are essential, the next question is which skew-selection policy fits well with extra invalid tags. Our prior observation is that both policies perform almost equally with skews. However, in the context of extra invalid tags, we observe a complex interplay between the eviction policies and the skew-selection policies. We explain this by considering the eviction and skew-selection policies in pairs. The eviction policy in the presence of extra invalid tags can be *local* (`LE`), that is, from the same set as the one in which an insertion occurred; or *global* (`GE`), that is, a valid entry chosen randomly from the entire cache is evicted upon each insertion. Therefore, there are four combinations to evaluate: i) `GE` + `RS`; ii) `GE` + `LA`; iii) `LE` + `RS`; iv) `LE` + `LA`.

Without loss of generality, we begin with the global LRU eviction and random skew selection (`GE+RS`), which is also illustrated in Figure 5. *The observation here is that the combination of skews, invalid tags, global eviction, and random skew selection does not have any advantage over standalone skewing with random skew selection.* Next, we modify the skew selection policy to load-aware skew selection (`GE+LA`). As depicted in Figure 6, invalid tags now become effective, and the eviction rate steadily decreases with an increase in invalid tags. *This establishes invalid tags with global LRU eviction and load-aware skew selection as a useful knob-subknob combination.* The same combination used by Mirage.

Next, we consider local eviction policies. The results with load-aware skew selection (`LE+LA`) are depicted in Figure 7. As observed, *there is no impact of extra invalid tags, and the results are the same as skewing with load-aware.* We note that the results do not change even when local eviction is combined with random skew selection (`LE+RS`). Therefore, we have omitted the plot for this combination.

So far in this evaluation, we have only evaluated the locality of the eviction policies. However, the replacement policy, e.g., LRU or Random, should also be evaluated. We note that one of the most successful design with extra invalid tags, namely Mirage, uses global random replacement. We defer this discussion till Section 3.5. We conclude the current subsection explaining why load-aware skew selection becomes effective in the presence of invalid tags and global eviction.

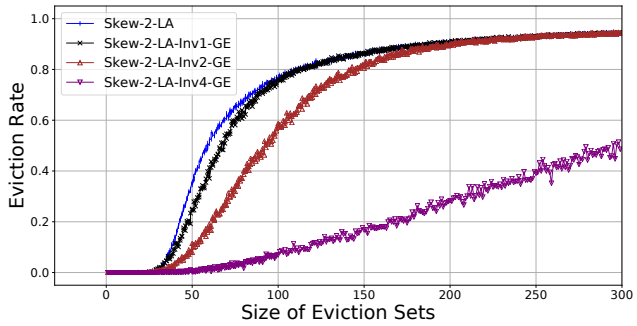

Figure 6: Eviction rate for two skews (`Skew-2`) with load-aware skew selection (`LA`), extra invalid tags (`Inv`), and global LRU eviction (`GLRU`).

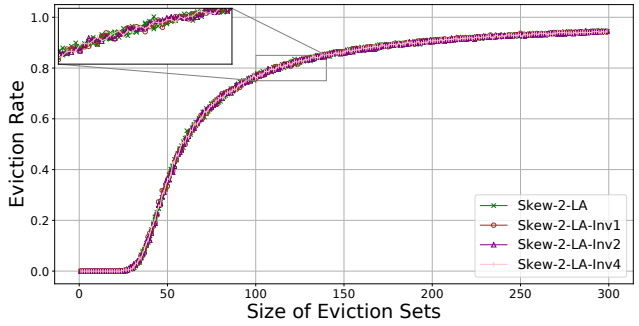

Figure 7: Eviction rate for two skews (`Skew-2`) with load-aware skew selection (`LA`), extra invalid tags (`Inv`), and local LRU eviction.

Why is `LA` important? The positive impact of load-aware skew selection warrants further clarification, as it initially appeared ineffective as a sub-knob of skews (see Section 3.2.2). However, this new observation does not contradict the earlier finding. Load-aware skew selection is effective when the cache is not full, and keeping the cache from becoming full improves both the eviction rate and overall security. This is achieved through the combination of extra invalid tags and global eviction: the cache (including invalid tags) never becomes full, since consuming an invalid tag creates another. However, this mechanism alone is insufficient. An eviction set can still contain enough addresses mapping to a single set in a skew, depleting all its invalid tags and triggering set-associative evictions, compromising security. In such cases, random skew selection becomes ineffective. Load-aware skew selection mitigates this by lowering the likelihood of depleting all invalid tags in a skew, as it tends to distribute insertions across skews more evenly. It effectively requires most sets in other skews to be nearly full, which is unlikely under global eviction. Thus, load-aware skew selection and global eviction work in tandem to keep the probability of any skew becoming full very low, preserving low eviction rates. However, this

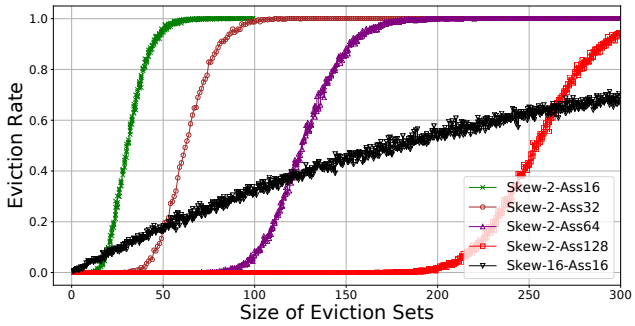

Figure 8: Eviction rate for two skews (`Skew-2`) with random skew selection, local LRU eviction, and associativity (`Ass`).

dynamic is sensitive to the number of extra invalid tags per skew, which influences the global eviction threshold.

Takeaway

Extra invalid tags can enhance resilience against conflict-based attacks. But it has to be paired with skews, global eviction, and load-aware skew selection. No other knob-subknob combinations work. Decoupling the tag and data store has no measurable impact on security.

3.4 Knob 3: High Associativity (`Ass`)

Prior work [18] explored the effect of cache associativity as a security knob. We have also observed this effect in Section 3.2.1, where the eviction probability (p_e) is inversely proportional to the cache associativity. However, [18] limited their evaluation to up to 16 ways. In this section, we extend this analysis to high-associativity designs and demonstrate that they offer significantly stronger security guarantees, making high associativity a powerful design knob against conflict-based attacks attacks.

We begin with the simplest possible configuration of knobs. Once again we note that associativity without skews is nothing but a traditional set-associative cache. As the set of knobs related to extra invalid tags also does not make much sense without skews, starting with skews and associativity seems to be the most obvious choice. Figure 8 shows the trend of eviction rate as we vary the cache associativity. Here we only consider two skews and increasing associativity with random skew selection. We observe that as we increase the cache associativity, the eviction rate decreases for a given eviction set size. Most interestingly, the eviction rate is almost zero up to a certain eviction set size, and this size increases with associativity. As we can observe from Figure 8, `Skew-2-Ass128` performs really well till an eviction set size of ≈ 270.

Explanation. Intuitively, high associativity requires a very large eviction set to evict the target address. This is true even without any skews. Skews, on the other hand, make such evic-

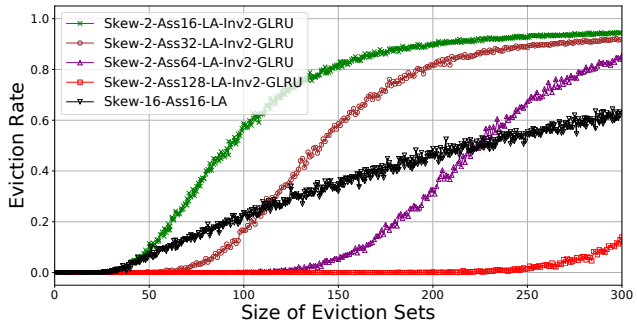

Figure 9: Eviction rate for two skews (`Skew-2`) with load-aware skew selection (`LA`), extra invalid tags (`Inv`), global LRU eviction (`GLRU`), and varying associativity (`Ass`).

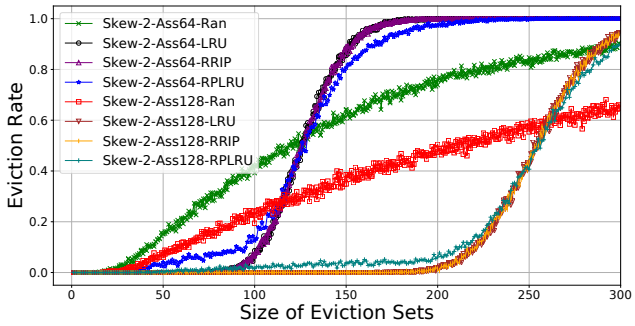

Figure 10: Eviction rate for two skews (`Skew-2`) with random skew selection, varying associativity (`Ass`), and different local replacement policies (`LRU`, `Ran`, `RRIP`, and `RPLRU`).

tion sets probabilistic, thereby decreasing the eviction rate. However, as we can observe from Figure 8, the higher associativity works as a dominating factor, keeping the eviction rate close to zero up to a certain size of the eviction set. The eviction rate increases rapidly after this threshold, but due to skews, the increase in eviction rate is tapered. We have also tested the effect of using load-aware skew selection rather than random skew selection. However, since our setup has neither extra invalid tags nor global eviction, when we consider the cache state to be full, there is no difference in the security provided by the two skew-selection methods.

We also show that skews and high associativity become more effective with load-aware, extra invalid tags, and global eviction (refer Figure 9). The combination of all these knobs can provide a zero eviction probability, even for large eviction sets. However, such a combination of knobs requires various modifications to conventional set-associative caches.

Takeaway

High associativity even with two skews rapidly reduces eviction rate, without requiring any other knobs. Moreover, the eviction rate remains close to zero up to a certain eviction set size.

3.5 Knob 4: Replacement Policy

The replacement policy consists of the cache line insertion policy, state update policy, and cache line eviction policy. So far in this paper (Section 3.3.3), we have considered the LRU replacement policy (both global and local). Prior works [25, 37] have shown the effect of replacement policy on security. In this subsection, we explore five popular policies: *Random replacement* (`Ran`), *least recently used* (`LRU`), *pseudo-least recently used* (`PLRU`), *randomized pseudo-least recently used* (`RPLRU`) [37], and *Re-Reference Interval Prediction* (`RRIP`) [16]. LRU is rarely used in practice due to its implementation complexity; instead, approximations like PLRU are commonly used. Prior work [37] adapted PLRU to skewed randomized caches via RPLRU. Since we focus exclusively on skewed randomized designs in this section, we evaluate only RPLRU and omit PLRU.

We evaluate the high-associativity configurations as well as the successful configurations we have found with invalid tags. Figure 10 shows the effect of the replacement policy on the eviction rate of highly associative skewed caches. An interesting observation is that the random replacement policy significantly reduces the gain obtained due to high associativity, and the trend is almost identical to having 16 skews each with associativity 16. However, we can provide this security with two skews only, making it more practical to implement. Another important point here is that only local replacement makes sense for such designs as these caches are skewed set-associative caches. There is no mechanism (such as invalid tags) to detect if a set is "almost full" and trigger a global eviction. The only way an eviction can be triggered is by detecting if a set is full, and this leaks the same information about the sets as a local eviction. Therefore, we limit our evaluation for this configuration with local eviction policies only. We also observe that RPLRU has a similar behavior to LRU, suggesting that LRU approximations can achieve security comparable to true LRU.

Explanation. We explain why random eviction performs worse than RRIP, LRU, and RPLRU, all of which use reuse information to decide eviction candidates. Consider a set in one of the skews where the target address x resides, containing warm-up entries, the target address, and eviction set entries. To evict the target address, we need n_w eviction set entries. Once these are present, the target can be evicted with certainty. With random skew selection, the eviction rate gradually increases, reaching one after a certain threshold set size. In random eviction, we do not need to completely fill a set, as each entry in the eviction set has a probability of $1/n_w$ of evicting the target. There is an equal probability of self-eviction among eviction set entries, preventing the eviction rate from reaching one. Since each address has a constant probability of evicting

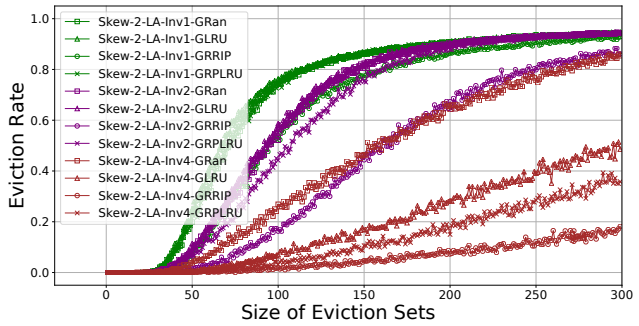

Figure 11: Eviction rate for two skews (`Skew-2`) with load-aware skew selection (`LA`), extra invalid tags (`Inv`), and global replacement policies (`GLRU`, `GRan`, `GRRIP`, and `GRPLRU`).

the target, the exact number of eviction set entries becomes less important. Even with fewer entries, eviction still occurs, leading to a non-zero eviction probability. This makes random eviction less efficient than LRU. Figure 11 shows the effect of using extra invalid tags, load-aware skew selection with global random, global LRU, global RRIP, and global RPLRU eviction. We observe that the global LRU and global RRIP eviction policies perform much better than the global random eviction policy, and this gap widens with an increasing number of extra invalid tags. We also observe that the global RPLRU performs similarly to global LRU while being more practical to implement.

Explanation. The improved security of global LRU over global random eviction stems from a reasoning similar to why local LRU outperforms local random eviction. With global random eviction, every accessed address has a finite probability of evicting the target address. In contrast, under global LRU, the eviction set is accessed immediately after the target, making the target the most recently used cache entry. Therefore, the eviction set can only evict the target address after filling the entire cache, making successful target eviction significantly harder.

Takeaway

Random replacement policies perform worse than LRU and RRIP for conflict-based attacks, as they result in higher eviction rates.

3.6 Knob 5: Remapping

Remapping is a critical knob in many secure randomized cache designs. It is necessary when eviction sets (with a reasonable non-zero eviction rate) can be constructed within a finite time. Remapping changes the address-to-set mapping by updating the block cipher key and make any eviction set constructed for the existing mapping useless. Most of the designs discussed so far do require remapping, except a few leveraging several extra invalid tags (e.g., Mirage), and thereby making eviction rate construction difficult. In this section, we only evaluate configurations requiring remapping (`Skew-2-Ass64` and `Skew-2-Ass128`, and designs with a few invalid tags.).

Using Metric II: The evaluation of remapping differs from the other knobs discussed in Section 3. Instead of assessing its usefulness (which is evident), we evaluate the remapping period. A high remapping period (that is, remapping at a slower rate) for a randomized cache improves performance, but also provides an attacker a larger window to attack. Setting the remapping period is, therefore, a performance security tradeoff. The remapping period cannot be evaluated with the eviction rate metric used so far. Therefore, we resort to our second metric – the *number of evictions to generate an eviction set*. Recall that so far, we have calculated the eviction rate metric based on ideal eviction sets. For the remapping period, however, we need to practically construct the eviction sets using the state-of-the-art eviction set generation algorithms. Given an eviction set size ($|E|$), and a remapping period (R), the expected number of evictions ($\mathcal{E}$) needed to formulate the eviction set for a skewed set-associative cache with S sets, n_w ways, and k skews can be written as $\mathcal{E} = \frac{R \cdot S \cdot n_w}{P'}$ [34]. Here $P' = 1 - \sum_{i=0}^{\frac{|E| \cdot n_w}{k}-1} \binom{R \cdot S \cdot n_w}{i} P^i (1-P)^{R \cdot S \cdot n_w - i}$ and $P = \frac{1}{S \cdot k}$. We use this formula to estimate the remapping period R based on our obtained values of $|E|$ and $\mathcal{E}$ from Metric-II.

Figure 12 shows the number of LLC evictions required to construct eviction sets that achieve a 30% eviction rate for various cache configurations. The 30% threshold is chosen following [34] to enable fair comparison. For `Skew-2-Ass64`, achieving this rate requires an eviction set of 116 entries, which takes an average of 3.7 million LLC evictions using the conflict testing algorithm. `Skew-2-Ass128` requires 241 entries, needing approximately 7.8 million LLC evictions. In contrast, `Skew-16` achieves the same eviction rate with 87 entries and 2.6 million LLC evictions. More precisely, for `Skew-2-Ass128`, $|E| = 241$, and $\mathcal{E} = 7.8 \times 10^6$, which requires a remapping period $R \approx 228$. For `Skew-2-Ass64`, we obtain $R \approx 103$. Notably, for `Skew-16` (with associativity 16) $R \approx 39$, which establishes the efficacy of high-associativity.

Comparing eviction set finding algorithms: In Figure 12, the results presented are with respect to the Conflict Testing [34] eviction set generation algorithm. However, choosing Conflict Testing was not ad-hoc, but based on a comparison between the state-of-the-art eviction set generation algorithms, among which Conflict Testing is found to perform the best. More precisely, we compare Conflict Testing [34] and Prime, Prune and Probe [26] algorithms for eviction set generation, which are widely used for attacking randomized caches. Table 2 shows a comparison of the number of LLC evictions required to generate eviction sets of various sizes for `Skew-2-Ass64` and `Skew-16` using the two algorithms. We can clearly observe that Conflict Testing requires lesser number of LLC evictions. Additionally, Prime, Prune and

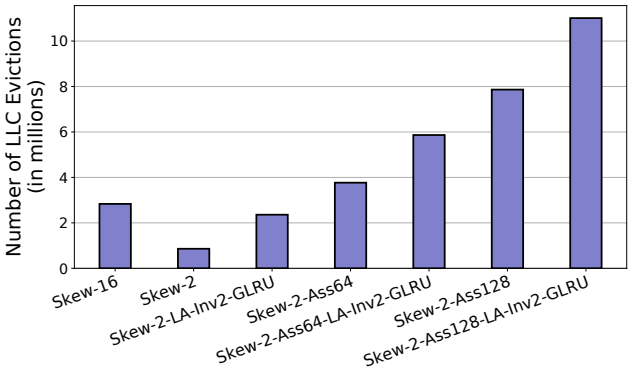

Figure 12: The number of LLC evictions required to create eviction sets that achieve a 30% eviction rate.

Table 2: A comparison of the number of LLC evictions needed to create eviction sets of different sizes using Conflict Testing and Prime, Prune and Probe.

Eviction Set Size	Conflict Testing		Prime, Prune and Probe	
	Skew-2-Ass64	Skew-16	Skew-2-Ass64	Skew-16
10	0.3 million	0.3 million	0.8 million	0.8 million
20	0.7 million	0.6 million	1.6 million	1.5 million
30	1.0 million	1.0 million	2.0 million	2.1 million
40	1.3 million	1.3 million	2.6 million	2.7 million

Probe requires ≈10× the number of LLC accesses as Conflict Testing for the same eviction set size [34]. This makes it infeasible to simulate for large eviction set sizes and therefore we only show results for smaller eviction set sizes. Such results justify our choice of eviction set finding algorithm.

Takeaway

Randomized cache designs with high associativity can have higher remapping periods compared to designs such as CEASER-S and `Skew-16`.

3.7 Sensitivity to Cache Size

We further explore highly associative configurations by varying the cache size from 1MB to 4MB, and also considering a 96MB cache, while observing the eviction set sizes required to reach a 30% eviction rate. Once again we use Metric-II. Figure 13 shows that the eviction rate for `Skew-2-Ass64` and `Skew-2-Ass128` remains unaffected by cache size. This is expected, as the likelihood of evicting a target address depends only on the cache associativity, not on the number of sets. However, as shown in Figure 14, the number of evictions required to construct such eviction sets increases with cache size. This is because current state-of-the-art eviction set search algorithms scale linearly with cache size. Extrapolating to typical LLC sizes, for a 96MB cache, approximately 187 million LLC evictions are required for `Skew-2-Ass64`, and about 381 million for `Skew-2-Ass128`.

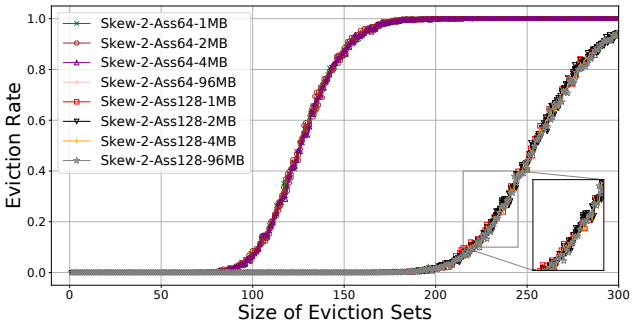

Figure 13: Eviction rate for two skews (`Skew-2`) with random skew selection, high associativity (`Ass64` and `Ass128`), and varying cache size (associativity remains the same).

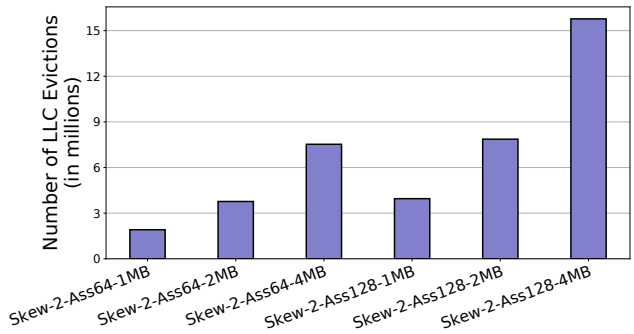

Figure 14: The number of LLC evictions required to create eviction sets achieving 30% eviction rate.

4 The Knobs and Occupancy-based Attacks

Occupancy-based attacks do not leak information at the set level and do not rely on eviction sets. Randomized cache knobs are primarily designed to prevent eviction set creation and they provide no guarantee against occupancy-based attacks. While it is essential for cache designs to defend against conflict-based attacks, they must also avoid unintentionally making occupancy-based attacks easier. This underscores the importance of evaluating cache knobs in the context of occupancy-based threats.

Most defenses against occupancy-based attacks focus on hardware partitioning [5, 14, 20, 21], with SassCache [19] being a notable exception. SassCache employs soft partitioning using block cipher-assisted domain isolation, reducing the likelihood of cross-domain evictions by limiting the bit-width of the cryptographic mapping. To date, there has been limited exploration of cache design knobs specifically targeting

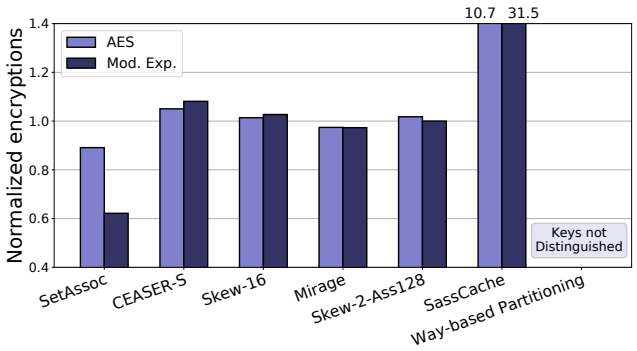

Figure 15: Normalized number of encryptions to distinguish the keys in AES and Modular Exponentiation for various cache designs. Normalization is done with respect to `FA-RR`.

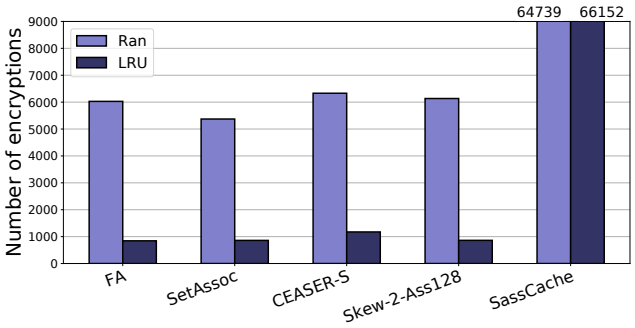

Figure 16: Number of encryptions to distinguish the keys in AES for different replacement policies (`Ran` and `LRU`).

occupancy-based attacks. SassCache's soft partitioning remains the only prominent strategy. We begin by analyzing the effectiveness of knobs originally intended for conflict-based attacks in mitigating occupancy-based threats. We then evaluate soft-partitioned designs like SassCache. For comparison, we also consider an ideal static partitioned design that ensures complete domain isolation. We do not explore other partitioning schemes, as our focus is on evaluating knobs within randomized cache designs.

Evaluated designs. To streamline our analysis, we evaluate a selected set of representative designs from Section 3: a skewed design (CEASER-S), an extra invalid tags-based design (Mirage), a highly skewed design (`Skew-16`), and a highly associative design (`Skew-2-Ass128`). For soft-partitioned randomized designs, we evaluate SassCache with a coverage of 39% ($t=-1$). We also evaluate a static way-based partitioned design as the ideal mitigation against occupancy-based attacks. All designs are compared to a fully associative design with random replacement (`FA-RR`).

Benchmarking strategy. For the occupancy-based attack evaluation, we use AES (OpenSSL implementation with T-tables) and modular exponentiation, measuring the number of encryptions required to distinguish between two secret keys. The attacker begins by filling the cache with a randomly chosen *occupancy set*—a collection of addresses capable of occupying the cache. The attack proceeds by priming the cache with this set, allowing the victim to execute, and then probing the same set to count the number of cache misses. Based on these miss statistics, the attacker attempts to differentiate between encryptions using different keys. This evaluation follows the methodology of CacheFX [18], and we use the same simulator. However, we explore additional design variants, many introduced through our knob-based design exploration. We collect 100,000 encryption traces per key and report the median number of encryptions required to distinguish between the keys.

Observations. Figure 15 shows the number of encryptions required to distinguish keys in AES and Modular Exponentiation, normalized to `FA-RR`. We observe that all the cache designs that use knobs designed for mitigating conflict-based attacks perform quite similarly to `FA-RR`, with the exception of the set-associative design. However, SassCache (soft partitioning) provides a much better security compared to other randomized cache designs that are secure to conflict-based attacks. Moreover, a way-based static partitioned design provides complete protection.

Explanation. Let us first explain why the set-associative cache performs relatively worse than other designs. This is because, in randomized cache designs, the randomness (due to skew selection or eviction policy) enhances security. Specifically, there are always addresses in the occupancy set that map to the same cache set, and the same can happen for victim accesses. These colliding addresses create self-evictions. For highly deterministic schemes, such as a set-associative cache, the self-evictions for both the victim and attacker are also deterministic, as each address can only map to one set in the cache. In contrast, in designs with more randomness, self-eviction noise is random. Since occupancy-based attacks focus on the total number of misses, any deterministic self-eviction for the victim will be an artifact of the victim's access pattern and, therefore, more of a signal than noise for the attack. On the other hand, random self-evictions are imposed by the cache design and do not (entirely) depend on the victim's access pattern. This randomness adds noise and significantly increases the number of encryptions needed for an attack. To better understand the impact of deterministic vs. randomized eviction policies, we further examine the LRU replacement policy (see Figure 16). As shown, the lack of randomness significantly reduces the efficacy of schemes using deterministic policies compared to those with random eviction. The trend is similar for other deterministic replacement policies, such as RRIP, and thus the results for RRIP were omitted.

Next, we explain why all the randomized designs perform almost the same with respect to occupancy-based attacks in Figure 15. We attribute this to the evaluation strategy where the entire cache has been occupied. *Due to full occupancy, the adversary can observe all misses caused by the victim's*

access irrespective of whether the replacement policy is local or global. This is the main reason behind all representative designs having almost the same performance with respect to the occupancy-based attacker.

Finally, we examine SassCache and static way-based partitioning. SassCache guarantees a certain degree of isolation between the cache entries controlled by different security domains, while static partitioning ensures that there is no leakage between the security domains. While both designs provide significantly better security against occupancy-based attacks compared to other randomized cache designs, SassCache still cannot fully block leakage of information between domains, and thus is still vulnerable to occupancy-based attacks compared to a static way-based partitioned design, which cannot be broken by any conflict or occupancy-based attack.

Low-occupancy-based attacks. So far, we have focused on attacks that fully occupy the cache to extract sensitive data such as AES keys. Recent work [13] extends this threat model to *low-occupancy-based attacks*, which can be used for covert channels, process fingerprinting, and key recovery. [13] uses *guessing entropy*, defined as the expected number of guesses an attacker needs to make to correctly guess the secret key, as their metric to measure the threat. The lower the guessing entropy, the easier for the attacker to obtain the secret key. Their findings show that such attacks are feasible on Mirage even with only 10% cache occupancy. Moreover, at low occupancy levels, Mirage performs worse than ScatterCache and CEASER-S for covert channel attacks, indicating that *the locality of replacement impacts security under low occupancy*. This trend is attributed to Mirage's global random eviction policy [13]. We extend their analysis to our high-associativity designs (`Skew-2-Ass64` and `Skew-2-Ass128`) as shown in Figure 17, and find that these designs offer similar resilience to SassCache and ScatterCache, and do not exhibit the same vulnerability as Mirage.

Takeaway

Deterministic replacement policies offer worse security than random ones against occupancy-based attacks. Local eviction policies provide better security than global ones for low-occupancy-based attacks, while performing similarly for full-occupancy-based attacks.

5 Open Problems

Unified security metric for occupancy-based attacks. In this work, we used eviction rate and the difficulty of eviction set generation as metrics to evaluate cache resilience against conflict-based attacks. For occupancy-based attacks, we relied on cryptographic workloads such as AES and modular exponentiation, due to the lack of established, targeted metrics in the literature. With the growing threat of occupancy-based attacks, there is a pressing need for a unified metric to quantify cache resilience in such scenarios.

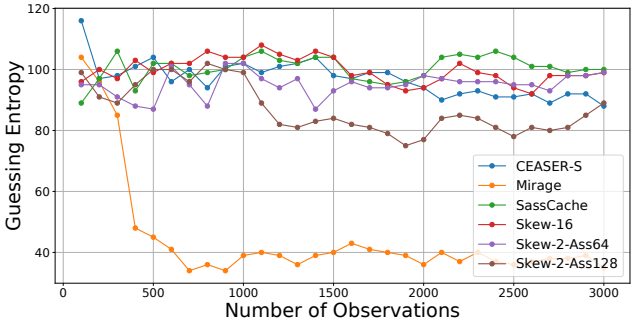

Figure 17: Guessing entropy for AES key recovery across a 50% occupancy rate for varying number of observations.

SassCache security against multiple adversary processes. SassCache provides a thorough security evaluation for scenarios with a single attacker domain. However, multi-domain attacks are also realistic. In such cases, the authors suggest that cloud vendors select the parameter t based on the expected co-location probability, using the formula $C_t = 1-(1-C)^{n_d}$. However, accurately predicting the number of attacker domains in advance is often impractical and may lead to security vulnerabilities if the actual number exceeds the expectation. Although t is reconfigurable in hardware, a comprehensive analysis of the multi-domain attacker threat and the effectiveness of the proposed mitigation remains essential. Finally, more alternatives should be explored for occupancy-based attack mitigation, which are agile and scalable, and yet provide the same security as hard partitioning.

6 Conclusions

The key findings from our knob-based systematization can be summarized as follows: (i) Skewing is the most versatile knob and should be included in all randomized cache designs. (ii) Extra invalid tags result in designs closest to fully associative caches, offering strong security against conflict-based attacks, but require complex knob combinations, increasing design complexity. Moreover, even with this increased design complexity, we do not get any advantage against occupancy-based attacks. Additionally, designs with global eviction, such as Mirage, have been shown to be vulnerable to low-occupancy-based attacks. (iii) High associativity, even with just two skews, provides strong security against conflict-based attacks. However, they do not provide any benefit against occupancy-based attacks. In general, they require relatively less number of knobs to achieve good security. However, remapping is required which invalid tag-based designs can avoid. Overall, while there is no clear winner in this study, from a viewpoint of design complexity versus security tradeoff, high-associativity designs should be considered as viable options.

7 Acknowledgments

We thank the authors of [13, 18, 35] for generously supporting us to use their code for our various experiments. We would also like to thank the anonymous reviewers for their constructive comments that helped in improving this work. This work is supported by the Trust Lab Research Grant 2024.

8 Ethics Considerations

Disclosures. This paper does not introduce any novel or previously unknown attacks. Instead, it focuses exclusively on attacks that have already been disclosed and documented in prior work. All the attacks discussed in this paper are well-established and widely recognized within the community.

Experiments using private systems. All research activities undertaken for this paper were conducted exclusively on systems owned and managed by the authors and their affiliated institution. The experiments were carried out using software-based, open-source simulators to replicate the required conditions and analyze system behaviors. Consequently, no real-world systems were accessed, and no vulnerabilities were discovered, exploited, or tested on live or operational systems.

Terms of service. All experiments used open-source simulators, specifically the simulation models from [35], [18], and [13]. The tools have been cited appropriately to ensure transparency and acknowledge their contribution.

Cache vulnerabilities. In this work, we provide robust metrics for evaluating both existing and novel secure randomized cache designs. We offer fresh insights into previous analyses and develop new experiments to better understand the inner workings of secure randomized caches. Our primary focus is on identifying security knobs that are effective and resilient against adversaries, such as conflict-based and occupancy-based attackers. We successfully identified a minimal cache design that defends against conflict-based attacks. While the risk of occupancy-based attacks is not new and has long been recognized by the community, we emphasize that most existing secure cache designs acknowledge their vulnerability to these attacks, with some proposing mitigations. Ultimately, we stress the importance of developing new cache designs that specifically account for occupancy-based attacks, which represent a significant threat to modern systems.

9 Open Science

Our work offers new insights into the evaluation of existing secure randomized cache designs through both established and novel experiments, leveraging open-source simulators. The eviction rate and eviction set search results were obtained using the open-source behavioral cache simulation model introduced by [35]. We have modified this model and intend to release both the modifications and the accompanying scripts to facilitate the replication of our findings during artifact evaluation. Simulations on occupancy-based attacks were conducted by extending the open-source CacheFX simulator [18]. For the low-occupancy-based attack evaluation, we extend the open-source setup provided [13].

We have publicly released all the tools and scripts used in this work for the security analysis of cache designs, ensuring full reproducibility of our security results. The permanent link for our source code can be found at `https://doi.org/10.5281/zenodo.15529618`. We now provide a brief overview of the various tools and scripts used. A more detailed version of the artifact is available [4], as the "Artifact Appendix".

- **cache-model:** An extended version of the behavioral cache simulation model by [35]. This model is used to generate results for Figures 3–14, 18–19, and Table 2.
- **cachefx:** An extended version of the CacheFX simulator introduced by [18]. This simulator is used to generate results for Figures 15–16.
- **low-occupancy:** An extended simulation model used for low-occupancy-based attack simulations introduced by [13]. This model is used to generate results for Figure 17.
- **`buildAll.sh`:** Script used to build all source files for the cache model, CacheFX, and low-occupancy simulators.
- **`genAllFigs.sh`:** Script used to run all experiments and generate PDF files for Figures 3–19.
- **`genTable.sh`:** Script used to run experiments and generate Table 2.
- **`requirements.txt`:** A list of the Python libraries needed for smooth functioning of all scripts. These can be installed using:
  `pip3 install -r requirements.txt`
- **`README.md`:** Provides a description of the directories and detailed steps to build projects and run experiments to reproduce results from this work.

A Appendix: Evaluating Design Trade-offs

Security. We evaluate the security of the shortlisted designs based on the number of LLC evictions required to construct an eviction set. A higher number of required evictions indicates stronger resistance to conflict-based attacks. We use the Conflict Testing algorithm for eviction set generation, as it outperforms alternatives such as Prime, Prune and Probe. High-associativity designs, such as `Skew-2-Ass128`, and invalid-tag-based designs, like `Skew-2-Ass128-LA-Inv2-GLRU`, offer strong protection by making eviction set construction significantly more difficult and maintaining short remapping periods. Other designs, such as Mirage and SassCache, achieve an extremely low probability of set-associative evictions, rendering eviction set discovery nearly impossible.

Performance. We evaluate a range of secure randomized cache designs using the ChampSim [3] microarchitecture simulator. Our baseline is a non-secure 8-core system with a

Table 3: A comparison of secure randomized cache designs, the number of LLC evictions to create an eviction set with 30% eviction rate, and their performance, storage, power overheads.

Design	LLC Evictions Needed to Create Eviction Set	Knobs Used	Performance Overhead	Logic Overhead	Dynamic Power Overhead	Static Power Overhead
Skew-2 (CEASER-S)	0.5 million	Skews, Remapping	-1.3%	1.9%	2.5%	2%
Skew-16 (ScatterCache)	2.8 million	Large number of skews, Remapping	0.1%	1.7%	0.5%	1.4%
Skew-2-LA-Inv2-GLRU	2.3 million	Skews, Load-aware, Invalid Tags, Global Eviction, Remapping	1.3%	1.9%	5.2%	2%
Mirage	Not Possible	Skews, Load-aware, Invalid Tags, Global Eviction	0.2%	18.6%	-0.2%	19.6%
Skew-2-Ass64	3.8 million	Skews, High Associativity, Remapping	-2.1%	2.3%	1.8%	3.3%
Skew-2-Ass128	7.9 million	Skews, High Associativity, Remapping	-2.2%	2.4%	1.7%	6.4%
Skew-2-Ass64-LA-Inv2-GLRU	6.1 million	Skews, High Associativity, Load-aware, Invalid Tags, Global Eviction, Remapping	-2.2%	2.3%	1.8%	3.3%
Skew-2-Ass128-LA-Inv2-GLRU	11.0 million	Skews, High Associativity, Load-aware, Invalid Tags, Global Eviction, Remapping	-2.5%	2.5%	1.7%	6.4%
SassCache (coverage = 39%)	Not Possible	Skews, Soft Partitioning	2.3%	2.4%	7.4%	1.2%

16MB, 16-way set-associative LLC, employing the LRU replacement policy and 64-byte cache lines. We select 15 homogeneous workloads (42 sim-points) from SPEC CPU2017 [2], focusing on benchmarks with more than one LLC miss per kilo-instruction (MPKI) in a single-core 2MB cache configuration. Each simulation includes a 50-million-instruction warm-up phase, followed by the execution of 1.6 billion instructions across eight cores (200 million per core) within the region of interest. To compare performance across cache designs, we use the weighted speedup [33] metric for 8-core systems, normalizing all results to the non-secure baseline. Most designs incur only marginal performance overheads, with the largest slowdowns observed for `Skew-2-LA-Inv2-GLRU`, due to only 75% valid cache entries, and SassCache, due to soft partitioning between cores.

Logic. We analyze the total storage requirements of each design by calculating the number of bits per tag and data entry. Most designs incur a modest 2–3% storage overhead, primarily due to the addition of secure domain ID (SDID) bits in each tag. A notable exception is Mirage, which has significantly higher overhead due to a 75% larger tag store and pointer-based indirection between tag and data entries. Additionally, several designs require extra logic, such as circuitry for load-aware skew selection and block ciphers for address randomization. We compare this logic overhead to the 5.5 million gate equivalents (GEs) used in the 1MB L2 cache of the BROOM chip [12], scaled to a 2MB cache.

Power. We estimate dynamic and static power overheads using P-CACTI [1], configured in sequential access mode and modeled for 7nm FinFET technology. For LLC static power, most designs exhibit minimal overheads, with high-associativity designs incurring slightly higher overheads (up to 6%) due to more complex lookup circuitry. Mirage incurs a 19.6% static power overhead, attributed to a 75% increase in tag entries and the use of indirection pointers between tag and data arrays. The baseline has a static LLC power of ≈520mW.

For dynamic power, we focus on the LLC hit and miss power. Since each LLC miss goes directly to the DRAM, we also need to account for the DRAM access power. We estimate tag and data read/write energy using P-CACTI, and assume a DRAM read/write energy of approximately 60nJ. As DRAM access is much more expensive than LLC access in terms of energy, designs with lower LLC miss rates achieve better overall dynamic power efficiency compared to standard 16-way associative designs. For example, high-associativity caches, despite their higher dynamic access energy, have smaller dynamic power overheads than their 16-way associative counterparts. The baseline dynamic power (both LLC and DRAM access) is around 1800mW.

B Appendix: Impact of Warm-up States

In Section 3.2.2, we observed some non-intuitive trends in eviction rate results. We argue that these trends stem from the specifics of the eviction rate experiment rather than limitations of the metric itself. Following the method in [34], we gradually increase the size of the warm-up set in iterations by retaining cache entries from one iteration to the next, starting from an empty cache. This approach averages over varying cache states, smoothing out the influence of any particular initial state. It also significantly reduces simulation time by avoiding per-iteration warm-up. *However, this averaging over warm-up states is what leads to the confusing trends noted in Section 3.2.2.*

Warm-up state and load-aware skew selection. To elaborate, we refer to the eviction rate experiments in Figure 4, where we compare random and load-aware skew selection. Surprisingly, load-aware skew selection appears to offer some improvement over random selection, which seems counterin-

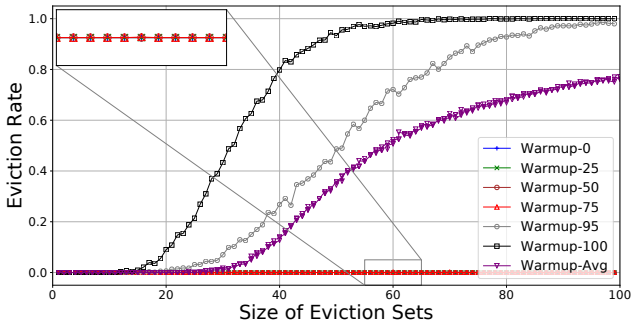

Figure 18: Eviction rate for two skews with load-aware skew selection, LRU eviction and different warm-up states. For example, `Warmup-75` refers to a `Skew-2-LA` cache with a 75%-filled warm-up state. `Warmup-Avg` refers to a `Skew-2-LA` cache with an average warm-up state as per [34].

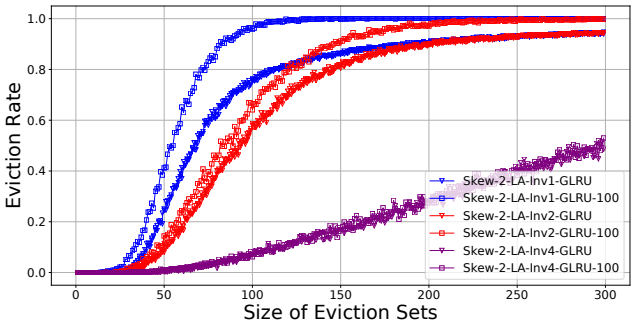

Figure 19: Eviction rate for two skews (`Skew-2`) with load-aware skew selection (`LA`), extra invalid tags (`Inv`), and global LRU eviction (`GLRU`) with different warm-up states. For example, `Skew-2-LA-Inv2-GLRU-100` refers to a `Skew-2-LA-Inv2-GLRU` cache with a 100%-filled (excluding the invalid entries) warm-up state.

tuitive. As discussed in Section 3.2.2, *any eviction set should be ineffective in the presence of load-aware skew selection, unless the cache is nearly full. However, since the cache being full is a common case, the overall improvement for non-full states should be negligible.* However, the eviction rate experiment averages results across all warm-up sizes, giving equal weight to states with low cache occupancy. This skews results and artificially lowers the eviction rate, creating a misleading impression. *While the trends in Figure 4 may seem promising from a design perspective, they do not reflect the true security behavior.* To validate this, we modified the experiment to compute eviction rate separately for each warm-up level. As shown in Figure 18, eviction sets become effective only at high warm-up percentages. At that point, *the benefit of load-aware skew selection vanishes*, since with a full cache, tie-breaking effectively makes it behave like random skew selection. This supports our interpretation. While averaging over cache states provides useful design-space insights and faster simulation (as done in [34] and throughout this paper) it does not always capture edge-case behaviors. Therefore, we use the more detailed, per-state eviction rate analysis selectively for promising designs.

Load-aware skew selection with invalid ways. Building on the previous discussion, we performed the enhanced eviction rate analysis for all promising designs but highlight only the most notable findings here. One such case involves combining load-aware skew selection with invalid ways. Figure 6 in Section 3.3.3 shows the standard eviction rate results for this combination. For comparison, Figure 19 in this section presents the enhanced eviction rate, calculated per warm-up state size. Specifically, Figure 19 captures the effect of starting from a full cache (excluding extra invalid tags) on configurations using load-aware skew selection, extra invalid tags, and global eviction. We observe a significant discrepancy between the standard and enhanced eviction rates when the number of invalid tags is small. This gap arises because the original experiment does not emphasize the more probable full-cache scenarios. In a fully occupied cache, eviction set addresses need only displace invalid tags in the target set, which is easier when there are fewer such tags. As the number of extra invalid tags increases, this warm-up state effect diminishes. Notably, Mirage—the most effective design leveraging invalid tags—employs a sufficiently large number of them. However, while Mirage focuses on blocking set-associative evictions, our analysis uses a unified eviction rate-based metric, providing a broader perspective.

As a final takeaway, we argue that the issue arises not from the eviction rate metric itself, but from how it is computed to speed up evaluations. Both evaluation speed and accuracy are important. While a nearly full cache is generally common, it is not guaranteed. For example, if a target system is underutilized for a while, the attacker can choose this as an attack window. It can then craft a warm-up scenario to their advantage by exploiting such an underutilized system. Therefore, we recommend not overlooking warm-up effects, but performing enhanced eviction rate evaluation only for a selected subset of designs.

References

[1] Pcacti tool, Online. Available: https://sportlab.usc.edu/downloads/.

[2] SPEC CPU 2017 traces for champsim. https://hpca23.cse.tamu.edu/champsim-traces/speccpu/index.html, February 2019.

[3] ChampSim simulator. http://github.com/ChampSim/ChampSim, May 2020.

[4] Artifact Appendix: SoK: So, You Think You Know All About Secure Randomized Caches? https://github.com/AnubhavBhatla/systematization-of-secure-randomized-caches/blob/main/Artifact-Appendix.pdf, 2025.

[5] Kerem Arikan, Abraham Farrell, Williams Zhang Cen, Jack McMahon, Barry Williams, Yu David Liu, Nael B. Abu-Ghazaleh, and Dmitry Ponomarev. Tee-shirt: Scalable leakage-free cache hierarchies for tees. In *31st Annual Network and Distributed System Security Symposium, NDSS 2024, San Diego, California, USA, February 26 - March 1, 2024*, pages 1–18, 2024.

[6] Daniel J. Bernstein. Cache-timing attacks on AES. 2005.

[7] Anubhav Bhatla, Navneet, and Biswabandan Panda. The maya cache: A storage-efficient and secure fully-associative last-level cache. In *2024 ACM/IEEE 51st Annual International Symposium on Computer Architecture (ISCA)*, pages 32–44, 2024.

[8] Rahul Bodduna, Vinod Ganesan, Patanjali SLPSK, Kamakoti Veezhinathan, and Chester Rebeiro. Brutus: Refuting the security claims of the cache timing randomization countermeasure proposed in ceaser. *IEEE Computer Architecture Letters*, 19(1):9–12, 2020.

[9] A. Bogdanov, L. R. Knudsen, G. Leander, C. Paar, A. Poschmann, M. J. B. Robshaw, Y. Seurin, and C. Vikkelsoe. Present: An ultra-lightweight block cipher. In *Cryptographic Hardware and Embedded Systems - CHES 2007*, pages 450–466, 2007.

[10] Julia Borghoff, Anne Canteaut, Tim Güneysu, Elif Bilge Kavun, Miroslav Knezevic, Lars R. Knudsen, Gregor Leander, Ventzislav Nikov, Christof Paar, Christian Rechberger, Peter Rombouts, Søren S. Thomsen, and Tolga Yalçin. PRINCE - A low-latency block cipher for pervasive computing applications (full version). *IACR Cryptol. ePrint Arch.*, page 529, 2012.

[11] Billy Brumley and Risto Hakala. Cache-timing template attacks. In *Advances in Cryptology - ASIACRYPT 2009*, pages 667–684, 12 2009.

[12] Christopher Celio, Pi-Feng Chiu, Krste Asanović, Borivoje Nikolić, and David Patterson. Broom: An open-source out-of-order processor with resilient low-voltage operation in 28-nm cmos. *IEEE Micro*, 39(2):52–60, 2019.

[13] Anirban Chakraborty, Nimish Mishra, Sayandeep Saha, Sarani Bhattacharya, and Debdeep Mukhopadhyay. Systematic evaluation of randomized cache designs against cache occupancy. *To appear In 35th USENIX Security Symposium (USENIX Security 25)*, 2025.

[14] Ghada Dessouky, Emmanuel Stapf, Pouya Mahmoody, Alexander Gruler, and Ahmad-Reza Sadeghi. Chunked-cache: On-demand and scalable cache isolation for security architectures. In *29th Annual Network and Distributed System Security Symposium, NDSS 2022, San Diego, California, USA, April 24-28*, pages 1–18, 2022.

[15] Bourgeat et al. Casa: End-to-end quantitative security analysis of randomly mapped caches. In *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, pages 1110–1123, 2020.

[16] Jaleel et al. High performance cache replacement using re-reference interval prediction (rrip). In *Proceedings of the 37th Annual International Symposium on Computer Architecture*, ISCA '10, page 60–71, 2010.

[17] Ristenpart et al. 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*, CCS '09, page 199–212, 2009.

[18] Daniel Genkin, William Kosasih, Fangfei Liu, Anna Trikalinou, Thomas Unterluggauer, and Yuval Yarom. Cachefx: A framework for evaluating cache security. In *Proceedings of the 2023 ACM Asia Conference on Computer and Communications Security*, ASIA CCS '23, page 163–176, 2023.

[19] Lukas Giner, Stefan Steinegger, Antoon Purnal, Maria Eichlseder, Thomas Unterluggauer, Stefan Mangard, and Daniel Gruss. Scatter and split securely: Defeating cache contention and occupancy attacks. In *2023 IEEE Symposium on Security and Privacy (SP)*, pages 2273–2287, 2023.

[20] Nadja Ramhöj Holtryd, Madhavan Manivannan, and Per Stenström. Scale: Secure and scalable cache partitioning. In *2023 IEEE International Symposium on Hardware Oriented Security and Trust (HOST)*, pages 68–79, 2023.

[21] Yonas Kelemework and Alaa R. Alameldeen. INTERFACE: An Indirect, Partitioned, Random, Fully-Associative Cache to Avoid Shared Last-Level Cache Attacks. In *2024 International Symposium on Secure and Private Execution Environment Design (SEED)*, pages 37–49, 2024.

[22] Tom Kessous and Niv Gilboa. Prune+plumtree - finding eviction sets at scale. In *2024 IEEE Symposium on Security and Privacy (SP)*, pages 3754–3772, 2024.

[23] Fangfei Liu, Hao Wu, Kenneth Mai, and Ruby B. Lee. Newcache: Secure cache architecture thwarting cache side-channel attacks. *IEEE Micro*, 36(5):8–16, 2016.

[24] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. Last-level cache side-channel attacks are practical. In *2015 IEEE Symposium on Security and Privacy*, pages 605–622, 2015.

[25] Moritz Peters, Nicolas Gaudin, Jan Philipp Thoma, Vianney Lapôtre, Pascal Cotret, Guy Gogniat, and Tim Güneysu. On the effect of replacement policies on the security of randomized cache architectures. In *Proceedings of the 19th ACM Asia Conference on Computer and Communications Security*, ASIA CCS '24, page 483–497, New York, NY, USA, 2024. Association for Computing Machinery.

[26] Antoon Purnal, Lukas Giner, Daniel Gruss, and Ingrid Verbauwhede. Systematic analysis of randomization-based protected cache architectures. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 987–1002, 2021.

[27] M.K. Qureshi, D. Thompson, and Y.N. Patt. The v-way cache: demand-based associativity via global replacement. In *32nd International Symposium on Computer Architecture (ISCA'05)*, pages 544–555, 2005.

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

[29] Moinuddin K. Qureshi. New attacks and defense for encrypted-address cache. In *Proceedings of the 46th International Symposium on Computer Architecture*, ISCA '19, page 360–371, New York, NY, USA, 2019.

[30] Gururaj Saileshwar and Moinuddin Qureshi. MIRAGE: Mitigating Conflict-Based cache attacks with a practical Fully-Associative design. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1379–1396, 2021.

[31] Gururaj Saileshwar and Moinuddin Qureshi. The Mirage of breaking MIRAGE: Analyzing the modeling pitfalls in emerging "attacks" on MIRAGE. *IEEE Computer Architecture Letters*, pages 121–124, 2023.

[32] Anatoly Shusterman, Lachlan Kang, Yarden Haskal, Yosef Meltser, Prateek Mittal, Yossi Oren, and Yuval Yarom. Robust website fingerprinting through the cache occupancy channel. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 639–656, 2019.

[33] Allan Snavely and Dean M. Tullsen. Symbiotic jobscheduling for a simultaneous multithreaded processor. *SIGOPS Oper. Syst. Rev.*, 34(5):234–244, nov 2000.

[34] Wei Song, Boya Li, Zihan Xue, Zhenzhen Li, Wenhao Wang, and Peng Liu. Randomized last-level caches are still vulnerable to cache side-channel attacks! but we can fix it. In *Proceedings - 2021 IEEE Symposium on Security and Privacy, SP 2021*, pages 955–969, 2021.

[35] Wei Song and Peng Liu. 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 (RAID 2019)*, pages 427–442, 2019.

[36] Wei Song, Zihan Xue, Jinchi Han, Zhenzhen Li, and Peng Liu. Randomizing set-associative caches against conflict-based cache side-channel attacks. *IEEE Transactions on Computers*, 73(4):1019–1033, 2024.

[37] Florian Stolz, Jan Philipp Thoma, Pascal Sasdrich, and Tim Güneysu. Risky translations: Securing tlbs against timing side channels. *IACR Trans. Cryptogr. Hardw. Embed. Syst.*, 2023(1):1–31, 2023.

[38] Qinhan Tan, Zhihua Zeng, Kai Bu, and Kui Ren. Phantomcache: Obfuscating cache conflicts with localized randomization. *Proceedings of Network and Distributed System Security Symposium*, pages 1–17, 2020.

[39] Jan Philipp Thoma, Christian Niesler, Dominic Funke, Gregor Leander, Pierre Mayr, Nils Pohl, Lucas Davi, and Tim Güneysu. ClepsydraCache – preventing cache attacks with Time-Based evictions. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1991–2008, Anaheim, CA, August 2023.

[40] Thomas Unterluggauer, Austin Harris, Scott Constable, Fangfei Liu, and Carlos Rozas. 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 (SEED)*, pages 13–24, 2022.

[41] Pepe Vila, Boris Köpf, and José Francisco Morales. Theory and practice of finding eviction sets. *2019 IEEE Symposium on Security and Privacy (SP)*, pages 39–54, 2018.

[42] Zhenghong Wang and Ruby B. Lee. New cache designs for thwarting software cache-based side channel attacks. *SIGARCH Comput. Archit. News*, 35(2):494–505, 2007.

[43] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. ScatterCache: Thwarting cache attacks via cache set randomization. In *28th USENIX Security Symposium (USENIX Security 19)*, pages 675–692, 2019.

[44] Mengjia Yan, Christopher W. Fletcher, and Josep Torrellas. Cache telepathy: Leveraging shared resource attacks to learn DNN architectures. In *29th USENIX Security Symposium (USENIX Security 20)*, pages 2003–2020, 2020.

[45] Fan Yao, Milos Doroslovacki, and Guru Venkataramani. Are coherence protocol states vulnerable to information leakage? In *2018 IEEE International Symposium on High Performance Computer Architecture (HPCA)*, pages 168–179, 2018.

[46] Yuval Yarom and Katrina Falkner. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the 23rd USENIX Security Symposium, 2014*, pages 719–732, 2014.

[47] Xingjian Zhang, Haochen Gong, Rui Chang, and Yajin Zhou. Recast: Mitigating conflict-based cache attacks through fine-grained dynamic mapping. *IEEE Transactions on Information Forensics and Security*, 19:3758–3771, 2024.

[48] Xingjian Zhang, Ziqi Yuan, Rui Chang, and Yajin Zhou. Seeds of SEED: H2Cache: Building a Hybrid Randomized Cache Hierarchy for Mitigating Cache Side-Channel Attacks . In *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, pages 29–36, 2021.

USENIX Security '25 Artifact Appendix: SoK: So, You Think You Know All About Secure Randomized Caches?

Anubhav Bhatla*, Hari Rohit Bhavsar*, Sayandeep Saha, Biswabandan Panda
Indian Institute of Technology Bombay

A Artifact Appendix

A.1 Abstract

The artifact is used to evaluate the security of various cache designs discussed in the paper. We provide the source files and scripts required to reproduce all security results (Figures 3–19 and Table 2). A comprehensive README file is also included to facilitate easy reproduction of the results.

A.2 Description & Requirements

This section lists all the hardware and software dependencies required for the experimental setup to run all security simulations. Given below is a description of the important directories and scripts provided in the artifact:

- **cache-model:** An extended version of the behavioral cache simulation model. This model is used to generate results for Figures 3–14, 18–19, and Table 2.
- **cachefx:** An extended version of the CacheFX simulator used to generate results for Figures 15–16.
- **low-occupancy:** An extended simulation model for our low-occupancy-based attack simulations. This model is used to generate results for Figure 17.
- **`buildAll.sh`:** Script used to build all source files for the cache model, CacheFX, and low-occupancy simulators.
- **`genAllFigs.sh`:** Script used to run all experiments and generate PDF files for Figures 3–19.
- **`genTable.sh`:** Script used to run experiments and generate Table 2.
- **`requirements.txt`:** A list of the Python libraries needed for the smooth functioning of all scripts.
- **`README.md`:** Provides a description of the directories and detailed steps to build projects and run experiments to reproduce results from this work.

*These authors contributed equally to this work

A.2.1 Security, privacy, and ethical concerns

Our study does not introduce any novel or previously unknown attacks. Instead, it focuses exclusively on attacks that have already been disclosed and documented in prior work. All the attacks discussed in our evaluation are well-established and widely recognized within the community. The experiments were carried out using software-based, open-source simulators to replicate the required conditions and analyze system behaviors. No real-world systems were accessed, and no vulnerabilities were discovered, exploited, or tested on live or operational systems. Therefore, there are no ethical concerns in this work.

A.2.2 How to access

Our artifact is available through Zenodo. The artifact can be accessed at https://doi.org/10.5281/zenodo.15529618.

A.2.3 Hardware dependencies

We implemented and evaluated our artifact on a server with the Intel Xeon Gold 6342 CPU, 128GB of RAM, and 4TB of disk storage. To effectively parallelize simulations and ensure timely completion, we recommend utilizing a large number of CPU cores, ideally distributed across multiple servers.

A.2.4 Software dependencies

We test our artifact on a system running Ubuntu 20.04.6. However, any operating system which supports C++ and Python is sufficient to run our experiments. The Python packages required have been listed in the `requirements.txt` file. We also require docker and the C++ boost library for our simulations. The instructions to install this library and the Python libraries can be found in the `README.md` file.

A.2.5 Benchmarks

Our experiments evaluate the eviction rate of cache designs, number of LLC evictions required for eviction-set-generation, and the guessing entropy for key recovery. These experiments are self-sufficient and do not require any benchmarks for evaluation.

A.3 Set-up

A.3.1 Installation

We require that python, pip3, and docker are already installed on the system. For installing required Python libraries, run:

```
$ pip3 install -r requirements.txt
```

We also require the C++ boost library, installed using:

```
$ sudo apt install libboost-all-dev
```

The source files can be compiled using the following:

```
$ bash buildAll.sh build
```

A.3.2 Basic Test

An easy way to check if the simulation set-up is complete is to generate all figures and the table using our existing results. These can be generated using:

```
$ bash genAllFigs.sh 1
$ bash genTable.sh 1
```

If the set-up is complete, 17 PDF files will appear, named `figurex.pdf`, where x ranges from 3 to 19. Table 2 will also be printed on the terminal.
We recommend removing these figures before moving forward, especially if one aims to reproduce all the security results from scratch. Use the following command:

```
$ rm *.pdf
```

A.4 Evaluation workflow

We provide flexibility in how one can reproduce our security results. A single script can be used to run experiments for all security figures (Figures 3–19) at once. We also provide the option to use our existing simulation results to generate the figures without a fresh run:

```
$ bash genAllFigs.sh 1
```

In order to generate all figures using a fresh simulations, run:

```
$ bash genAllFigs.sh 0
```

Similarly, to generate Table-2 using existing results, run:

```
$ bash genTable.sh 1
```

A fresh set of simulations for Table-2 can be run using:

```
$ bash genTable.sh 0
```

All scripts that will be discussed later have the same option of using either our existing results (set option to `1`) or running a fresh set of simulations (set option to `0`). We assume that the user wants a fresh set of simulations, and set this option to `0` by default in Section A.4.2.

A.4.1 Major Claims

(C1): Increasing the number of skews helps reduce the eviction rate. Additionally, load-aware skew selection improves eviction rate. These are validated by **E1** described in Section 3.2, with results in Figures 3 and 4. We later talk about how this depends on the warm-up state in **Ex**.

(C2): Only the combination of skews with load-aware skew selection, invalid tags and global LRU eviction provides a security benefit. Additionally, increasing the invalid tags improves the security of such cache designs. These are validated by **E2** described in Section 3.3, with results in Figures 5, 6 and 7.

(C3): High associativity provides significant security benefits. This is validated by **E3** described in Section 3.4, with results in Figures 8 and 9.

(C4): Random replacement policy performs worse than LRU and RRIP for conflict-based attacks, as it results in higher eviction rates. This is validated by **E4** described in Section 3.5, with results in Figures 10 and 11.

(C5): High associativity designs require significantly more LLC evictions to create eviction sets that achieve a 30% eviction rate. Adding the knobs of load-aware skew selection, invalid tags, and global eviction further improves the security. These are validated by **E5** described in Section 3.6, with results in Figure 12.

(C6): The conflict testing algorithm is much faster than Prime, Prune and Probe in terms of number of LLC evictions needed for same-sized eviction sets. This is validated by **E6** described in Section 3.6, with results in Table 2.

(C7): Eviction rate is independent of the cache size (assuming same associativity). However, the number of LLC evictions required to create eviction sets that achieve a 30% eviction rate increase with cache size. These are validated by **E7** described in Section 3.7, with results in Figures 13 and 14.

(C8): SassCache and way-based partitioned cache designs perform significantly better than other randomized cache designs against occupancy-based attacks. Also, a random replacement policy is better suited against occupancy-based attacks compared to a deterministic replacement policy. These are validated by **E8** described in Section 4, with results in Figures 15 and 16.

(C9): Mirage is vulnerable to low-occupancy-based attacks compared to other randomized cache designs discussed in this work. This is validated by **E9** described in Section 4, with results in Figure 17.

(C10): The security benefit of designs using load-aware skew selection has a strong dependence on the cache warm-up state. This is validated by **E10** described in Appendix B, with results in Figures 18 and 19.

A.4.2 Experiments

(E1): [Impact of skews and skew selection policy] [48 compute-hour]: This experiment analyzes the impact of skews and the skew-selection policy on cache security.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 3
$ python3 get-figure.py 0 4
```

Results: `figure3.pdf` and `figure4.pdf` files are generated.

(E2): [Combinations of skew selection policies, invalid tags, and eviction policy] [72 compute-hour]: This experiment analyzes the security of various knob combinations including skews with random and load-aware skew selection, invalid tags, and local and global eviction.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 5
$ python3 get-figure.py 0 6
$ python3 get-figure.py 0 7
```

Results: `figure5.pdf`, `figure6.pdf` and `figure7.pdf` files are generated.

(E3): [Impact of high associativity] [48 compute-hour]: This experiment analyzes the security impact of high associativity on designs with just two skews, and also invalid-way-based designs.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 8
$ python3 get-figure.py 0 9
```

Results: `figure8.pdf` and `figure9.pdf` files are generated.

(E4): [Impact of replacement policy] [48 compute-hour]: This experiment analyzes the security impact of the replacement policy on designs with just two skews and high associativity, and also invalid-way-based designs.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 10
$ python3 get-figure.py 0 11
```

Results: `figure10.pdf` and `figure11.pdf` files are generated.

(E5): [Number of LLC evictions required for eviction-set-creation] [96 compute-hour]: This experiment evaluates the number of LLC evictions, required to create an eviction set achieving 30% eviction rate, on various designs.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 12
```

Results: `figure12.pdf` file is generated.

(E6): [Comparison of eviction-set-creation policies] [96 compute-hour]: This experiment compares the conflict testing and prime, prune and probe algorithms based on the number of LLC evictions they require to create same-sized eviction sets.
Execution: To run the experiment:

```
$ bash genTable.sh 0
```

Results: Table-2 is printed on the terminal.

(E7): [Sensitivity to cache size] [96 compute-hour]: This experiment analyzes the impact of cache size on the eviction rate as well as on the number of LLC evictions required for eviction-set-creation.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 13
$ python3 get-figure.py 0 14
```

Results: `figure13.pdf` and `figure14.pdf` files are generated.

(E8): [Security against occupancy-based attacks] [96 compute-hour]: This experiment analyzes the security of various cache designs against occupancy-based attacks.
Execution: To run the experiment:

```
$ cd cachefx/
$ python3 get-figure.py 0 15
$ python3 get-figure.py 0 16
```

Results: `figure15.pdf` and `figure16.pdf` files are generated.

(E9): [Security against low-occupancy-based attacks] [30 human-minute + 128 compute-hour]: This experiment analyzes the security of various cache designs against low-occupancy-based attacks.
Execution: To run the experiment:

```
$ cd low-occupancy/
$ sudo docker run -it -v $(pwd)/randomized
_caches:/home/randomized_caches
randomized-caches
$ cd randomized_cache_hello_world/
$ bash setup.sh
$ cd ../ ; bash buildAES.sh
$ bash genNumbers.sh
$ bash getGE.sh
$ exit
$ sudo mv randomized_caches/results results
$ python3 get-figure.py 0
```

Results: `figure17.pdf` file is generated.

(E10): [Impact of warm-up states] [48 compute-hour]: This experiment analyzes the impact of warm-up state on the eviction rate, and how it causes deviation in results from the original eviction-rate experiment.

Execution: To run the experiment:

```
cd cache-model/
python3 get-figure.py 0 18
python3 get-figure.py 0 19
```

Results: `figure18.pdf` and `figure19.pdf` files are generated.

A.5 Notes on Reusability

We highly encourage others to use this work in order to analyze other cache designs and configurations. For `cache-model` simulations, one can change the cache configuration in `cache-model/config/cache.json`. If needed, the source code inside `cache-model/cache` can be modified to implement additional cache designs. New experiments can be implemented by modifying the source code in `cache-model/test`. For `cachefx` simulations, new configurations can be added in `cachefx/configs` and new cache designs can be implemented using the files in `cachefx/Cache`.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at https://secartifacts.github.io/usenixsec2025/.