To appear in the 51st International Symposium on Computer Architecture (ISCA-24)

The Maya Cache: A Storage-efficient and Secure Fully-associative Last-level Cache

Anubhav Bhatla*, Navneet*, and Biswabandan Panda
Indian Institute of Technology Bombay

***Abstract*—The last-level cache is vulnerable to cross-core conflict-based attacks as the cache is shared among multiple cores. A fully associative last-level cache with a random replacement policy can mitigate these attacks. However, it is impractical to design a large last-level cache that is fully associative. One of the recent works, named Mirage, provides an illusion of a fully associative cache with a decoupled tag and data store and a random replacement policy. Mirage is a secure design that mitigates conflict-based attacks with a marginal performance overhead. However, it incurs a storage overhead of 20%, static power overhead of 18.16%, and area overhead of 6.86% compared to a non-secure baseline cache of 16 MB for an 8-core system. One of the primary contributors to the additional storage requirements is the usage of extra invalid tag entries that are used in a skewed way without changing the number of data store entries. These invalid tag entries provide a strong security guarantee. We observe that more than 80% of last-level cache's data store entries are dead on arrival, providing negligible utility in terms of performance improvement as they do not get reused in their lifetimes. Also, in general, the data store entries occupy $\approx$ eight times more storage than tag store entries. Based on these observations, we propose Maya, a storage efficient and yet secure last-level randomized cache that compensates for the additional storage of tag store entries by using fewer data store entries. Maya increases the tag store entries for security and reuse detection and uses fewer data store entries that only store the reused data. Our proposal provides a strong security guarantee, which is one set-associative eviction in 10^{32} line fills at the last-level cache. This is equivalent to a line installed once in 10^{16} years to mount an eviction attack. Maya provides this security guarantee with a 12MB data store that occupies 28.11% less area and 5.46% less static power when compared to a non-secure baseline of 16MB cache.**

I. Introduction

Modern processors use multiple levels of caches to hide the long latency main memory accesses. Typically, the level-1 and level-2 (L1 and L2) caches are private to the core, and the last-level cache (LLC) is shared across all the cores. Cross-core eviction-based side-channel attacks [24] [18] can cause controlled contention at the LLC sets and later can observe the effect of contention by measuring the latency differences between an LLC hit and an LLC miss. Randomized LLC designs are a promising approach to mitigate contention-based cache attacks. Randomized LLCs like CEASER, CEASER-S, Scatter-Cache [29], [30], [40] randomize the address to cache set mapping. However, these designs are prone to probabilistic cache contention attacks and are not secure [7], [14], [27]. The recently proposed SassCache [15] incurs a significant performance slowdown of more than 4% when evaluated on all the memory-intensive SPEC CPU2017 [2] and GAP [4] homogeneous mixes on an 8-core system.

Mirage [32] is a randomized LLC that provides the illusion of a fully associative cache, and it uses a random replacement policy so that an attacker cannot gather any information about a cache line address. Mirage is motivated by the V-way cache [28] and retains the practical set-associative lookups by decoupling placement and replacement from tag store to data store. Mirage uses a set-associative tag-store that over-provisions invalid tags in sets and with load balancing that guarantees new addresses are always filled into invalid tags without causing any conflicts. Cache fills result in global evictions, where a replacement candidate is selected randomly from the entire cache. Mirage guarantees global replacement of cache lines for the lifetime of a computer system; eliminating conflict-based attacks. Note that Mirage does not mitigate cache occupancy attacks, and mitigation of cache occupancy attacks is not in the scope of this paper as a fully associative cache is also prone to occupancy attacks [36].

The Problem. Mirage provides the illusion of a fully associative LLC, incurs a marginal performance overhead, and hence provides a sweet spot in terms of security and performance. However, it incurs an additional storage of 20% at the LLC with a static power overhead of 18.16%, which is a costly tradeoff. For example, for an 8-core system with 16MB baseline LLC, the combined storage of tag store and data store is 16.91 MB, whereas Mirage has a storage requirement of 20.31 MB. This additional storage requirement leads to an increase in static power consumption, from 622mW with the baseline 16MB LLC to 735mW with the Mirage cache. The requirement increases significantly for a large LLC. For a 32-core system with 32 2MB LLC slices, the baseline LLC requires a storage of 67.63MB of tag store plus data store, whereas Mirage requires a storage of 81.25MB LLC space, an additional 13.62MB LLC,

*Both authors contributed equally.

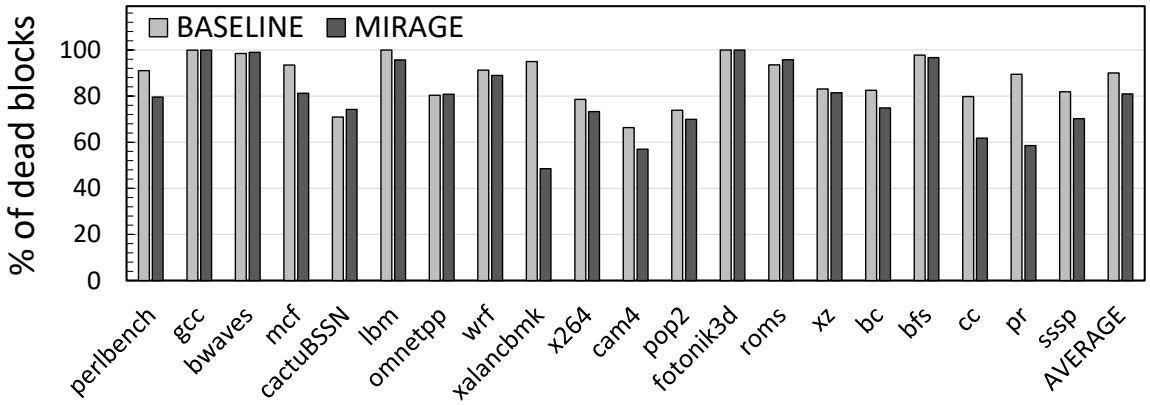

Fig. 1: Percentage of dead blocks inserted into the LLC for 15 memory intensive SPEC CPU2017 [2] benchmarks and five GAP [4] benchmarks on a single core system with a 2MB baseline and Mirage LLCs. The dead block percentage for a given benchmark is the average of dead blocks across all the sim-points [2], [4] of a benchmark. For each sim-point of a benchmark, we warm up for 200M instructions and then simulate 200M instructions.

which is extremely high.

Our goal is to propose an LLC design that can provide the illusion of a fully associative cache and, hence, the security guarantee without significant storage, power, and performance overhead. Note that one can argue for reducing the overall capacity of the Mirage cache and getting storage overhead similar to the baseline. According to our simulations, this approach leads to an average performance slowdown of around 5%. This slowdown is high as the micro-architecture community has been pushing the LLC performance for the last two decades to achieve performance closer to Belady's policy [35].

Our observations. Mirage increases the tag entries and provides extra tags in the form of invalid tags to provide security. However, it does not change the data store entries, leading to additional storage requirements. Figure 1 shows the fraction of data store entries that are *dead* (not reused after they are installed into the LLC) with the baseline cache and the Mirage for SPEC CPU2017 [2] and GAP [4] benchmarks. On average, more than 80% of the data entries are dead, occupying the LLC data store. This insight is not new and is well-established in the micro-architecture community, leading to research in dead block predictors [21].

Our approach. We propose Maya, a secure and storage-efficient randomized LLC that provides the illusion of a fully associative LLC. Maya uses a smaller data store compared to the baseline, with additional tag entries for security and tracking reuse to avoid any performance overhead due to the reduced data store size. In general, data store entries occupy eight times more storage than tag store entries. So, we can get the maximum benefit in terms of storage neutrality (compared to a non-secure LLC) if we optimize for the data store entries. Second, as most of the data is dead on arrival, these data-store entries are expendable. So, if we can use the extra tags to manage the remainder of the data store intelligently, it is a win-win tradeoff in terms of security, performance, and storage.

The core idea of Maya is motivated by the Reuse Cache [5], where we install a cache line in the data store only after it gets a reuse. To detect the reuse behavior, it uses additional tag-only entries (on top of invalid tag entries) that monitor the reuse behavior, and then on reuse, a data entry is allocated. Maya uses a decoupled tag and data store design. In general, we observe that the Maya cache provides a similar security guarantee as Mirage because the ratio of valid tags to invalid tags at the tag store is similar to Mirage's ratio of valid to invalid tags. For shared memory attacks, the LLC fills are isolated by their respective security domain ID. Overall, we make the following contributions:

(i) We propose Maya, a secure, fully associative, and randomized LLC, which is storage-efficient. The crux of our proposal is a decoupled tag and data store with additional tag entries but fewer data entries (Section III).

(ii) We argue about the security guarantee of Maya in terms of the number of LLC line installs required to mount an eviction-based attack. We prove that such an attacker must perform more than 10^{32} LLC line installs (around 10^{16} years, assuming one LLC fill takes an optimistic one ns.) to get one set-associative eviction, which is much larger than the system lifetime (Section IV).

(iii) Maya provides a strong security guarantee without additional storage (storage savings of 2%). Maya saves the LLC area by 28.11%, and leakage power by 5.46% (Section V).

II. Background

A. Threat model

We assume the following capabilities in our attacker:

(i) She is aware of LLC indexing (through reverse engineering) on a non-secure baseline LLC.

(ii) She can access the LLC by sending memory accesses to her data, but she cannot access any cache line that is not part of her own address space. However, she can accurately differentiate between an LLC hit and an LLC miss.

(iii) She is capable of mounting all the possible LLC

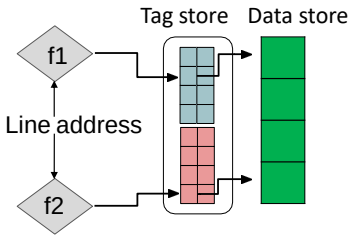

Fig. 2: Mirage cache design with skewed associative tag store.

contention attacks that exploit a timing channel, such as an eviction-based attack and flush-based attacks. She is not capable of mounting cache occupancy-based attacks. For eviction and occupancy-based attacks, she is not restricted by time to form an eviction set and then attack the victim.
(iv) She can flush a cache block from the entire cache hierarchy as long as it is her own data. This way she can expedite the process of creating an eviction set.
(v) The attacker and the victim are running on two different cores on a multi-core system, sharing LLC. Private caches are spatially and temporally partitioned among processes and flushed on context switches, providing isolation. Without loss of generality, our threat model excludes attacks on other micro-architecture units like on-chip interconnect, cache coherence directories, and other port contention-based attacks.

B. LLC Contention Attacks

Eviction-based cache attacks. In eviction-based attacks, an attacker fills its data into an LLC set that conflicts with the victim's data. Later, in the Probe step, the attacker re-accesses its data, and if it observes longer access latency, then it means that the victim has evicted some of the attacker's lines [11], [24], [25].
Shared memory-based attacks. In a shared memory-based attack (like Flush+Reload [42]), an attacker shares its address space with the victim (e.g., shared libraries). The attacker flushes cache lines shared by both the attacker and the victim and observes the victim's access to the same cache lines by observing memory access latency.
Occupancy-based attacks. An LLC occupancy-based attacker observes the LLC space occupied by the victim application. Recent attacks on website fingerprinting [36] exploit the dynamic LLC usage between the attacker and the victim. Note that an occupancy-based attack is possible even with a fully associative cache.
Flush-based eviction attack. A recent work [14] shows that an attacker can mount an eviction-based attack by flushing her private data while creating an eviction set. This method is faster than conventional eviction attacks like Prime+Probe. Note that this is different from shared memory-based flush attacks, where the attacker flushes shared LLC lines.

C. Recent Advances in Randomized LLCs

CEASER [29]. CEASER encrypts a physical address based on a *key* to get the encrypted address on an LLC access. Even in the encrypted address space, LLC conflicts are possible, and an eviction-based attacker can mount an attack. To mitigate eviction-based attacks, CEASER remaps cache lines with a different *key* after a fixed interval known as the *remapping period*. During a remap period, CEASER remaps with a new key that remaps cache lines into a new LLC set. As per an S&P 2021 paper [38], remapping based on LLC evictions instead of LLC accesses is recommended; an LLC line should be remapped every 2.5 LLC evictions as the authors show that for a 1024 set 16-way LLC slice, the minimum number of LLC evictions needed for finding an eviction is 40.8K [38].
CEASER-S [30] and Scatter-Cache [40]. CEASER-S and Scatter-Cache go one step ahead of CEASER and propose randomization with a skewed associative LLC to mitigate an agile eviction-based LLC attacker that can attack CEASER with slow remapping rates. As per [38], an LLC line should be remapped after 14 and 39 LLC evictions for CEASER-S and Scatter-Cache, respectively.
Mirage [32]. Mirage proposes a fully associative LLC that uses multi-index randomization with a global eviction policy. It provides a proxy for a fully associative LLC with the help of a decoupled tag store and data store. It maintains a set-associative tag lookup and global random eviction for data store using pointer-based indirection. Figure 2 shows Mirage cache, which uses additional invalid tags in a skewed associative tag-store design where cache lines are installed without set conflict. It also uses a load-aware skew-selection policy that guarantees the availability of sets with invalid tags. Mirage is the secure randomized LLC with a security guarantee of one line install in 10^{17} years for mounting an eviction attack. However, it incurs 20% storage overhead. Mirage also proposes Mirage-Lite, a storage-efficient version (17% storage overhead) of Mirage with cuckoo reallocation.

III. The Maya Cache

The Maya cache design has four key components:
(i) It uses a skewed-associative decoupled tag store with additional *invalid* tag entries for security. It also uses extra tag entries for *reuse* detection.
(ii) The tag store of Maya uses a new *priority* bit for each tag entry. Maya stores two kinds of tag entries: priority-0 and priority-1. Priority-0 entries have no associated data entry in the data store until they get a future tag hit and are promoted to priority-1 entries. Priority-1 entries are the entries in the tag store that have corresponding data-store entries.
(iii) For the tag store, Maya uses an insertion policy that keeps tag priorities in mind and is also equipped with *load-awareness* similar to Mirage [32]. Maya uses global random tag eviction in the tag store for only priority-0 entries. This ensures a fixed number of invalid tags are available in the tag store to avoid set-associative-evictions (SAEs).
(iv) Motivated by the Reuse Cache [5], Maya uses a smaller data store that stores entries, which will be *reused*. Maya uses a global random data eviction policy that evicts a data entry randomly *downgrading* its corresponding priority-1 tag entry to priority-0.

A. Tag and data store design

Maya uses a skewed associative and decoupled cache design where each tag entry stores a forward pointer (FPTR), which allows it to point to an arbitrary data entry. A security domain ID (SDID) is also stored for each tag entry to help distinguish between copies brought in by different domains. This helps in mitigating shared memory attacks like Flush+Reload [42].

Skewed-associative design. The tag store is split into two skews [34], each with its independent hash function used to determine the set mapping for a cache line. Each incoming cache line now maps to a set in each of the two skews and can appropriately choose between the two sets.

Priority bit. We introduce an additional bit for each tag store entry, called the *priority* bit. This bit indicates if a valid tag entry has a corresponding valid data entry in the data store. If the priority bit for a valid tag entry is '0', it indicates no valid data entry exists corresponding to this tag. In this case, the forward pointer is invalid. On the contrary, if the priority bit is '1', a valid data entry exists in the data store, along with valid forward and reverse pointers linking these tag store and data store entries.

Extra tag store ways. In the Maya cache design, we provide extra invalid tag ways, similar to Mirage. We also provision additional ways, termed as *reuse ways*, to keep track of valid entries with priority-bit set to zero. These entries do not have a corresponding valid entry in the data store and, thus, an invalid forward pointer. Once such an entry gets a hit in the tag store, its priority is set to '1', and a valid data entry is assigned, along with appropriate forward and reverse pointers. The number of priority-1 entries in the tag store is the same as the total number of data store entries. The cache also holds a fixed number of priority-0 entries to ensure that the data store is only used to store "useful" entries. Additional invalid tags are reserved such that on every line install, there is at least one invalid tag available, and therefore, no SAE occurs. This helps provide security against eviction-based attacks. Note that the sets in the tag store are not statically partitioned for storing priority-0, priority-1, and invalid tag entries. Rather, the total number of entries of each type is kept constant in the tag store once the cache is operating at its maximum capacity.

Decoupled data store. As the tag store is decoupled from the data store, and the data store has fewer entries than the tag store, Maya needs to store the reverse pointer (RPTR), which points to the corresponding tag entry, for each entry in the data store. Figure 3 shows an overview of the Maya cache design.

Fig. 3: Overview of the Maya cache design. White blocks represent invalid tag entries, yellow represents priority-0 tag entries, and green represents priority-1 tag entries.

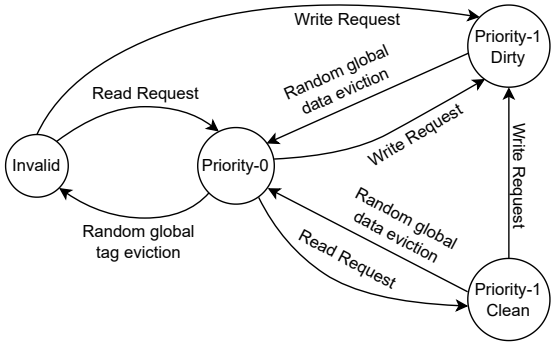

Fig. 4: State transition diagram for tag-store entries in Maya.

B. Insertion and eviction policies

Insertion policy. Because of the skewed associative design of the Maya tag store, each new cache line gets mapped to two different sets (one in each skew). The decision of the skew chosen directly affects the distribution of valid tag entries (both priority-0 and priority-1) across the sets. This, in turn, affects the distribution of invalid tag entries available in each set, which is crucial for preventing SAEs and thus maintaining security. Previous works [30], [40] use random skew-selection, which randomly picks one of the skews for the incoming line to be installed in. However, such a policy could lead to an imbalance in the number of available invalid tags, where some sets may end up with no invalid tag and thus become prone to an SAE. Inspired by Mirage, we use a load-aware skew-selection policy, which fills the incoming line into the set with more invalid tags. This promotes balanced use of tags across sets, and an SAE can occur only if both the mapped sets do not have any invalid tag available, which is a rare occurrence. Experimental results show that with a load-aware skew selection and six extra invalid tag entries per skew, an SAE occurs once in 10^{16} years, well beyond the system lifetime.

Note that when a cache line gets filled into the LLC, the corresponding tag is stored in the tag store, with its priority bit set to 0. However, the associated data is not yet stored in the data store, which results in an LLC miss. Subsequently, when a request arrives again for the same tag, the priority is set to 1, the corresponding data is brought into the data store, and the data is available in the LLC for subsequent accesses.

Eviction policy. Maya uses a random global eviction policy, which chooses a random eviction candidate from the entire data store to ensure no information is leaked to an attacker. We term this as *global random data eviction*. When a priority-0 entry gets a hit in the tag store and is upgraded to a priority-1 entry, a random priority-1 entry is chosen globally for eviction from the data store, and its priority bit is reset to '0', thus downgrading it to a priority-0 entry. Maya also introduces global eviction of priority-0 entries from the tag store, called

global random tag eviction. Such an eviction occurs every time a new priority-0 entry is brought into the tag store and a random priority-0 entry is invalidated from the tag store. These two eviction policies ensure that the tag store has a fixed number of invalid tag entries, which is crucial for security.

There can be a case where the tag store has not yet been filled up with the appropriate number of priority-0 entries (until all the reuse ways are filled). In such a case, when a new priority-0 entry gets filled into the LLC, we do not perform global random tag eviction. Similarly, if the data store is not full and a priority-0 entry needs to be upgraded to a priority-1 entry, then we do not perform global random data eviction.

States in the tag store. With Maya, a tag entry can be in one of three possible states: Invalid represents tag entries with their valid bit set to '0'. Priority-0 entries are valid, but their priority bit is set to '0', i.e. they have tag only and no data. Priority-1 entries are valid and with a priority set to '1', i.e. both tag and data are stored in the LLC. Dirty and clean denote if the corresponding data entry has been modified or is up-to-date with the main memory, respectively. Figure 4 shows all possible transitions between these states.

A tag entry starts in the Invalid state when the LLC is initialized. When a demand read comes in for an invalid tag, it transitions to the priority-0 state, and a tag entry is assigned to this tag. If a write request comes for an invalid tag, it is automatically assigned both tag and data entries (priority-1) and marked as dirty. Once a priority-0 entry gets accessed, it is upgraded to a priority-1 entry, and its corresponding data is brought into the cache. It is marked as dirty or clean based on whether it was a write or a read request. When a clean priority-1 entry gets a write request, it is marked as dirty since its data is no longer up-to-date with the main memory. Priority-1 entries can transition to the priority-0 state if selected for *random global data eviction*, where a random priority-1 entry is selected and downgraded to a priority-0 entry. Similarly, a priority-0 entry can go to the invalid state if it gets chosen by *global random tag eviction*.

C. Implementation

Our goal with the Maya cache design is to find the sweet spot between performance, security, and storage overhead. According to the security simulations in Section IV, we require 6 extra invalid tag entries per skew such that no set-associative eviction occurs in the system lifetime. To compensate for the storage overhead of the larger tag store, we reduce the size of the data store to only 6 ways per skew (12 ways in total) instead of the 16 ways per set for the baseline. This, along with the 3 reuse ways per skew, leads to storage savings of around 2% compared to the baseline. If we reduce the data store size further, it will save more storage but also lead to performance loss.

Table I shows the security guarantees with different reuse and invalid tag ways per skew. We observe a reduction in the security guarantee as we increase the number of reuse ways because security is affected by the associativity of the tag store (as shown in Section IV). With six extra invalid ways and one reuse way, we get the storage-efficient Maya that provides the best security guarantee. However, with one reuse way, there is a marginal performance overhead (Figure 5). Therefore, we use Maya with three reuse ways per skew as it offers a sweet spot between performance, security, and storage.

TABLE I: Cache line installs per SAE as the reuse ways vary from 1 to 7 ways with for 5 and 6 invalid ways per skew.

Reuse ways per skew	5 invalid ways per skew	6 invalid ways per skew
1-way	10^{18} (30 yrs)	$2{\cdot}10^{36}$ (10^{19} yrs)
3-ways	10^{16} (180 days)	$4{\cdot}10^{32}$ (10^{16} yrs)
5-ways	$6{\cdot}10^{15}$ (80 days)	$7{\cdot}10^{31}$ (10^{15} yrs)
7-ways	10^{15} (12 days)	$2{\cdot}10^{30}$ (10^{13} yrs)

Figure 5 shows that when we move from one reuse way to three reuse ways, it facilitates reuse prediction. Applications like `fotonik3d` see a normalized performance improvement from 0.97 to 1.04 when we move from one way to three ways. For five and seven reuse ways, there is a slight increase in tag lookup latency, which causes a minor performance drop. Note that the ratio of number of ways for priority-0 to priority-1 entries for one, three, five, and seven reuse ways are $\frac{1}{6}$, $\frac{3}{6}$, $\frac{5}{6}$, and $\frac{7}{6}$, respectively. We do not change the data store entries for the sensitivity study. This is because each data store entry carries almost eight times the number of bits as compared to a tag store entry (Table VIII), which leaves little room for changing the data store size while keeping the storage same with different reuse ways.

Each tag entry holds 40 tag bits for a 46-bit line address. Three coherence bits for the MOESI coherence protocol and one priority bit are also stored for each tag entry. To map to an arbitrary data entry, an 18-bit FPTR is used. The SDID helps keep track of the domain responsible for bringing in a cache line to allow duplication of shared cache lines. This ensures that the LLC fills of one domain do not affect the fills of another. Maya uses an 8-bit SDID for supporting up to 256 domains. In total, we use a total of 70 bits for each tag entry.

The tag store in Maya is split up into two skews, each with 16K sets (same as a non-secure baseline). Each set consists of six base ways per skew (total 12 ways, corresponding to the number of priority-1 entries), three reuse ways per skew (corresponding to the number of priority-0 entries), and an additional six invalid ways per skew to help maintain system security. With this, we get 192K (16K $\times$ 6 ways) priority-1 entries, 96K (16K $\times$ 3 ways) priority-0 entries, and 192K (16K $\times$ 6 ways) invalid tag entries in the tag store, resulting in 480K total tag store entries. This, multiplied by the total tag bits (70), leads to a tag store of size 4.1MB.

The data store has 192K entries, each storing 512 bits of data (64B cache lines). A data store entry can map to any arbitrary tag store entry, requiring a 19-bit RPTR. A total of 531 bits are stored for each data store entry, resulting in a total data store size of 12.44MB.

For the randomizing function, we use a 12-round PRINCE cipher [6], which is a 64-bit block cipher using 128-bit keys. It

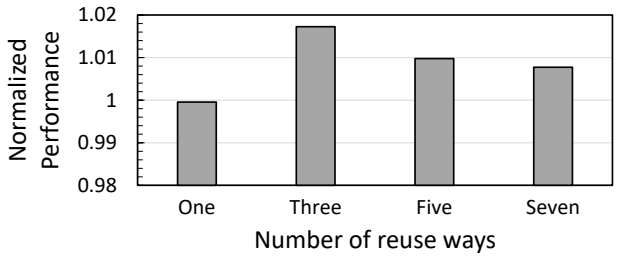

Fig. 5: Effect of the number of reuse ways per skew on the performance of Maya cache, normalized to the non-secure baseline LLC for SPEC CPU2017 homogeneous mixes running on an 8-core system with a warmup of 200M instructions/core and a simulation of 200M instructions/core.

is optimized for latency and has been used in previous works such as [32], [40]. This adds latency of 3 cycles for every LLC lookup. We also assume one additional cycle for tag and data lookup because of the tag-to-data indirection. In total, the Maya cache design requires 4 additional cycles per lookup.

IV. Security Analysis of Maya

Recent advances in eviction-based attacks show that only a few SAEs are required to construct an eviction set and break security. Mirage argued that if an LLC design ensures that no SAEs occur in the system's lifetime, it potentially mitigates future attacks that could break the security of an LLC even with a single SAE. To guarantee security even against such strong attacks, we show that even a single SAE is highly unlikely to occur in the system's lifetime with Maya, and thus, it guarantees security. We consider the worst-case scenario, where every LLC access is a miss as it increases the chances of getting SAEs. An LLC miss can be classified into three categories: demand tag miss, demand or writeback tag hit with priority-0 entry, and writeback tag miss. All these cases cause a change in the tag store state, either by changing the number of entries in a set or changing the composition of a set. Note that a tag hit to priority-1 entry does not lead to any fills in the tag store or data store. so we skipped it for the worst-case scenario. Our security evaluation accommodates all these categories of LLC miss.

A. Bucket-and-Balls Model

To estimate the probability of an SAE for the Maya cache, we use a variation of the bucket-and-balls model as used in [32]. The buckets represent cache sets, the balls denote tag entries, and a ball throw represents a fill. With Maya tag store, we can have two types of balls: priority-0 and priority-1. Priority-0 balls represent tag entries with the priority bit set to '0' (only tag and no data), whereas priority-1 balls represent tag entries with the priority bit set to '1' (both tag and data). The buckets are initialized with a fixed number of priority-0 and priority-1 entries to model the Maya tag store design. This ensures that we model the best-case scenario for the attacker. Table II provides the parameters used for the bucket-and-balls model for a 12MB Maya cache. For the experiment, each

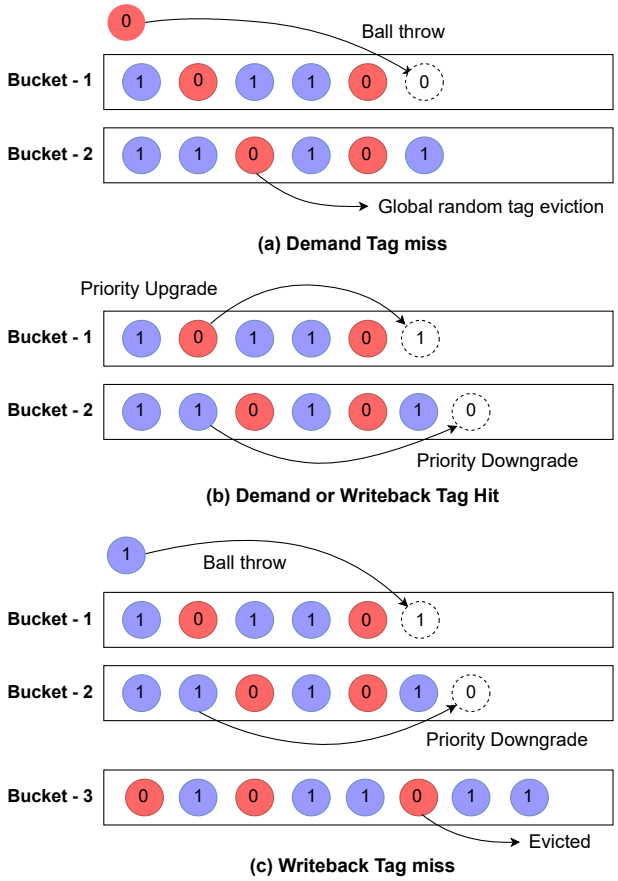

Fig. 6: Bucket-and-Balls model for the three types of LLC accesses: (a) demand tag miss, (b) demand or writeback tag hit, and (c) writeback tag miss

iteration consists of three types of LLC accesses that affect the distribution of balls in the tag store.

Demand tag miss. On a demand tag miss, two buckets are randomly chosen, one from each skew, and the ball is installed into the bucket with fewer balls as a priority-0 ball. This models the load-aware skew selection. In the case that both buckets have the same number of balls, one of the two buckets is randomly chosen. If both buckets are full, a bucket spill is caused, and a priority-0 ball needs to be removed from one of the two buckets. This represents a set-associative eviction, which is unfavorable for security. Also, after this ball has been inserted into a bucket, a random priority-0 ball must be randomly evicted from any bucket in either skew. This enables the *global random tag eviction* policy. Figure 6(a) shows the different events that occur on a demand tag miss. Bucket-1 is a randomly chosen bucket where the new priority-0 ball will be inserted, and bucket-2 is another random bucket from which a priority-0 ball will be removed.

Demand or writeback tag hit. To model a demand or writeback tag hit to a priority-0 entry, we choose a random priority-0 ball and upgrade it to a priority-1 ball, modeling a tag hit. Simultaneously, we choose a random priority-1 ball and

TABLE II: Parameters used for the Bucket-and-Balls Model.

Bucket-and-Balls Model	Maya Cache Design
Total priority-0 balls - 96K	Total priority-0 entries - 96K
Total priority-1 balls - 192K	Total priority-1 entries - 192K
Skews - 2	Skews - 2
Buckets/skew - 16K	Sets/skew - 16K
Average priority-0 balls/bucket - 3	Average priority-0 entries/set - 3
Average priority-1 balls/bucket - 6	Average priority-1 entries/set - 6
Bucket capacity - 9 to 15	Ways per skew - 9 to 15

downgrade it to a priority-0 ball (*global random data eviction*). Figure 6(b) summarizes the events on a demand/writeback tag hit with a priority-0 entry. Bucket-1 is a randomly chosen bucket where a priority-0 ball will be upgraded to a priority-1 ball, and bucket-2 is another random bucket from which a priority-1 ball will be downgraded to a priority-0 ball.

Writeback tag miss. For a writeback tag miss in the LLC, we perform priority-1 ball throws using load-aware skew selection. We then downgrade a random priority-1 ball to a priority-0 ball, representing *global random data eviction*. Since the total number of priority-0 balls has increased beyond the steady-state value, we also perform *global random tag eviction*. Figure 6(c) shows the events occurring for this type of LLC access. Bucket-1 is a randomly chosen bucket where the new priority-1 ball will be inserted, and bucket-2 is another random bucket from which a priority-1 ball will be demoted to a priority-0 ball. Furthermore, we randomly choose bucket-3 to evict a priority-0 ball.

B. Empirical Results

Figure 7 shows the expected number of iterations required to get a bucket spill with the given bucket capacity. As we increase bucket capacity from 9 to 15, the frequency of spills drastically reduces. We observe no spills for bucket capacities 14 and 15, and it is impractical to compute the spill frequency for these configurations in a reasonable amount of time (an experiment with one trillion iterations already takes a few days to simulate). We now demonstrate an analytical model to estimate the probability of a spill for 14 and 15 ways/skew.

C. Analytical Model

Using bucket-and-ball simulations for one trillion iterations (three trillion different accesses), we observe no bucket spills for bucket capacities 14 and 15. We propose an analytical approach based on the bucket-and-balls model to estimate these cases' spill frequency. To analytically calculate the probability of a bucket spill, we create a model of our buckets-and-balls system in a spill-free scenario, where the buckets have unlimited capacity. The number of balls in a bucket is modeled as a Birth-Death Markov chain [23], where the birth event corresponds to a ball insertion and the death event to a ball removal. Refer to Table III for the terminology used in the model. A classic result for Birth-Death chains says that the net conversion rate between any two states (here, state refers to the number of balls

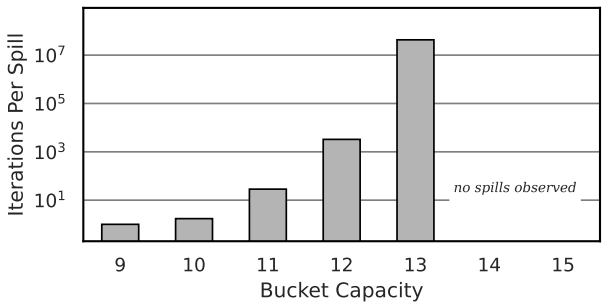

Fig. 7: Number of iterations required to cause a bucket spill. As the bucket capacity increases from 9 to 13, the frequency of bucket spills reduces. We observe no spills in one trillion iterations with bucket capacities of 14 and 15.

in a bucket) becomes zero in the steady state. Using this result, we obtain Equation 1, which equates the transition probability from N to $N+1$ balls to the transition probability from $N+1$ to N balls for a bucket.

$$Pr(N \rightarrow N+1) = Pr(N+1 \rightarrow N) \quad (1)$$

A bucket transitions from N to $N+1$ balls on a ball throw in one of three cases: (i) both buckets randomly chosen from skew-1 and skew-2 have N balls; (ii) the random bucket chosen from skew-1 has N balls and the random bucket from skew-2 has more than N balls; or (iii) the random bucket chosen from skew-2 has N balls and the one from skew-1 has more than N balls. The transition probability from N to $N+1$ balls is given by Equation 2.

$$Pr(N \rightarrow N+1) = Pr(n=N)^2 + 2 \times Pr(n=N) \times Pr(n>N) \quad (2)$$

For a bucket, the transition from $N+1$ balls to N balls can occur only on a global random tag eviction wherein a random priority-0 ball is globally selected for removal from all the balls. The probability of choosing a ball in a bucket with $N+1$ balls is given by the following equation:

$$Pr(N+1 \rightarrow N) = \frac{B_{tot} \times \sum_{r=1}^{r=N+1} \big(r \times Pr(n_0=r, n_1=N+1-r)\big)}{b^0_{tot}}$$

Here, r represents the number of priority-0 balls in a bucket with a total of $N+1$ balls. r varies from 1 to $N+1$ since a bucket with no Priority-0 balls will never be selected for Global random tag eviction.

Using $B_{tot}/b^0_{tot} = (1/3)$ (number of buckets/priority-0 balls) and splitting $Pr(n_0=r, n_1=N+1-r)$ into the conditional probability, $Pr(n_0=r|n=N+1) \times Pr(n=N+1)$, we obtain the following equation:

$$Pr(N+1 \rightarrow N) = \frac{\sum_{r=1}^{r=N+1} \big(r \times Pr(n_0=r|n=N+1) \times Pr(n=N+1)\big)}{3}$$

The expression $\sum_{r=1}^{r=N+1} \big(r \times Pr(n_0=r|n=N+1)$ simplifies to $\mathbb{E}_r[n_0=r|n=N+1]$, which provides the Equation 3.

$$Pr(N+1 \rightarrow N) = \frac{\Big(\mathbb{E}_r[n_0=r|n=N+1] \times Pr(n=N+1)\Big)}{3} \quad (3)$$

TABLE III: Terminology used in the analytical model.

Symbol	Interpretation
$Pr(X \rightarrow Y)$	Probability that a bucket with X balls transitions to Y balls
$Pr(n=N)$	Probability that a bucket contains N balls
$Pr(n_0=X, n_1=Y)$	Probability that a bucket contains X priority-0 balls and Y priority-1 balls
$Pr(n_0=X \mid n=Y)$	Probability that a bucket contains X priority-0 balls and Y total balls
$\mathbb{E}_X[n_0=X \mid n=Y]$	Expected number of priority-0 balls in a bucket with Y total balls
B_{tot}	Total number of Buckets (32K)
b^0_{tot}	Total number of priority-0 balls (96K)

Since priority-0 balls constitute a $(3/9)$ fraction of the total balls in the LLC (refer Table II), $\mathbb{E}_r[n_0 = r|n = N+1] = (3/9)(N+1)$, and Equation 3 simplifies to Equation 4.

$$Pr(N+1 \rightarrow N) = \frac{(N+1) \times Pr(n=N+1)}{9} \quad (4)$$

Using the earlier results from Equation 1, 2, and 4, we get a recursive relation for $Pr(n=N)$ as given in Equations 5, 6.

$$Pr(n=N+1) = \frac{9}{N+1} \times \Big(Pr(n=N)^2 + 2 \times Pr(n=N) \times Pr(n>N) \Big) \quad (5)$$

$$= \frac{9}{N+1} \times \Big(Pr(n=N)^2 + 2 \times Pr(n=N) - 2 \times Pr(n=N) \times Pr(n \leq N) \Big) \quad (6)$$

As we increase n, $Pr(n=N) \rightarrow 0$ and therefore $Pr(n > N) \ll Pr(n=N)$. Using this approximation, Equation 5 can be simplified to Equation 7 for larger values of n (we use this approximate equation once $Pr(n=N)$ becomes smaller than 0.01).

$$Pr(n=N+1) = \frac{9}{N+1} \times Pr(n=N)^2 \quad (7)$$

We simulate the bucket-and-ball model for one trillion iterations and obtain the probability of a bucket with no balls as $Pr_{exp}(n=0) \approx 7.7 \times 10^{-7}$. Using this value in Equation 6, we recursively calculate $Pr_{est}(n=N+1)$ for $N \in [1, 12]$, and then use Equation 7 for $N \in [13, 15]$, when the probabilities become less than 0.01. Figure 8 shows that the estimated values closely match the experimental values. Using the above-described analytical model, we obtain the spill probabilities for 14 and 15 ways.

Frequency of spills. Using the analytical model described above, we calculated a probability estimate for $N = 14, 15$. If we consider a cache design with W ways per skew, the probability of an SAE (or a bucket spill) will be given by $Pr(n=W+1)$. The spill probability follows a double-exponential reduction, as seen in Figure 8. For $W = 13, 14, 15$, an SAE occurs every 10^8, 10^{16}, and 10^{32} line installs, respectively. Thus, the Maya cache design with 15 ways per skew has a frequency of one SAE in $4 \cdot 10^{32}$ line installs or once in around 10^{16} years, effectively providing complete security against eviction-based attacks.

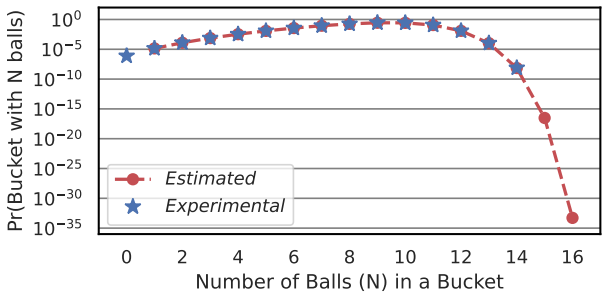

Fig. 8: Probability of a bucket having N balls (Pr(n = N)) - experimental and estimated using the analytical model.

Key management. The key used in Maya is set during the system boot. Although the probability of an SAE is significantly low, in the event of an SAE, the key used in the cipher for set mapping is refreshed followed by a cache flush to ensure security.

Sensitivity to associativity. We now vary the associativity of the Maya tag store, keeping the data store size at 12MB. The base associativity varies from 8 to 36 ways, with the default configuration having 18 total ways (6 base and 3 reuse ways per skew). Table IV shows the rate of SAE for these configurations. We can observe that for the same extra invalid ways per skew, the 8-way configuration is the most secure (one SAE in 10^{23} years), and security reduces as the base associativity increases. However, even for the 36-way configuration, the rate of SAE is once in around 10^{10} years, which is well beyond the system lifetime.

Note that the extensive security analysis provided in this section ensures that Maya is indeed secure. The first property is, of course, the probability of an SAE. Another property is that Maya ensures priority-0 tags themselves cannot be exploited to induce an SAE. Maya surely converges to Mirage, but the seemingly simple yet complex design of Maya does not make it intuitive as to why the security guarantee is the same. The analytical model we ideated is different from Mirage due to which it is important to emphasize the security guarantee rigorously.

D. Need for Domain IDs

Until now, we've explored the Maya cache design, highlighting its ability to safeguard against eviction-based side-channel attacks. In these discussions, we presumed that the victim and the attacker didn't share any cache lines. In situations where they do share cache lines, various attacks like Flush+Reload [42], Flush+Flush [17], Flush+Prefetch [16], and Evict+Reload [18], could potentially leak victim data.

Previous works [10], [22], [40] suggested placing the victim and the attacker in different domains and maintaining duplicate copies of shared lines in the cache for each security domain. This ensures that flush from one domain does not evict

TABLE IV: Cache line installs per SAE as the base associativity of the tag-store varies from 8 ways to 36 ways. 18-ways (6+3): On average, it consists of 6 base and 3 reuse ways/skew.

	Associativity		
Invalid Ways	**8-ways (3+1)**	**18-ways (6+3)**	**36-ways (12+6)**
4 extra ways/skew	10^{10} (7 s)	10^{8} (0.1 s)	10^{7} (9 ms)
5 extra ways/skew	10^{20} (10^{3} yrs)	10^{16} (180 days)	10^{14} (1 day)
6 extra ways/skew	10^{40} (10^{23} yrs)	10^{32} (10^{16} yrs)	10^{28} (10^{11} yrs)

another domain's copy. Scatter-Cache [40] employs a Security-Domain-ID (SDID) in combination with the physical cache-line address to determine the set index through its index derivation function (IDF). This approach allows a shared line to be filled and mapped to different sets for various domains, leading to duplication. However, this method doesn't guarantee duplication because two domain copies might still get mapped to the same set and thus not be duplicated. Mirage stores the SDID (8 bits) of the domain installing the line along with the tag of the line in the tag store. This guarantees the duplication of shared lines. Similarly, Maya includes an 8-bit SDID for each tag entry, accommodating a maximum of 256 domains. This ensures that the LLC fills of one domain do not affect the fills of another domain, and thus, the system is secure against shared-memory attacks. The length of the SDID can be adjusted to support more or fewer domains, depending on the requirement. Maya also mitigates Reload+Refresh [9] attack as it guarantees global evictions with random replacement.

E. Maya and the private caches

In the case of a private L1 or L2 cache, which is mostly shared by a 2-way simultaneous multi-threading (SMT) processor, it is relativity an easy design choice to partition the private L1 or L2 among two threads as done in the prior works [12]. Usage of randomization, global random replacement, and additional reuse ways at L1 can lead to performance degradation as high as more than 10%. Similarly, the cache coherence directory can be partitioned [41].

So overall, the cache hierarchy will have heterogeneous solutions for security (partitioning at the private caches and Maya at the LLC). Note that the interaction between private caches and the Maya cache is seamless as a load request from a core will probe its private but partitioned L1/L2 cache and its directory as per the partitioning technique used and then based on the SDID and the key used by Maya, the request will get filled into the Maya cache. A response from the Maya cache fills the private but partitioned L1/L2 based on the partition assigned to the SDID.

F. The cat and mouse game

Sophisticated attacker. One can argue that the timing difference between accessing a priority-1 entry and a priority-0/invalid tag entry can be exploited to mount a new timing-based side-channel attack. However, in the Maya cache design, the entries owned by the victim and the attacker have different

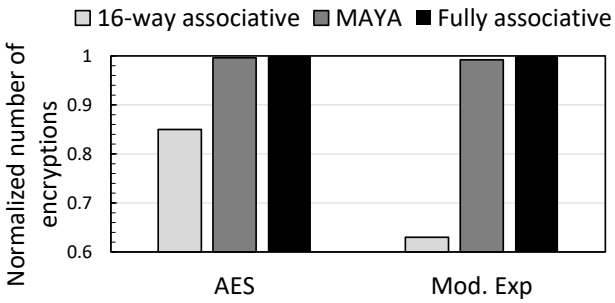

Fig. 9: LLC occupancy attack: number of encryptions required to break AES and modular exponentiation with a 16-way associative cache, Maya cache, and a fully associative cache. The number of encryptions is normalized to a fully associative cache with a random replacement policy.

SDIDs and are filled in isolation. The only way the attacker can exploit the reuse-based fills is by mounting an eviction-based attack, which we have already shown to be impossible since there are no set-associative evictions for 10^{16} years.

A sophisticated attacker can use the addresses under her control to occupy the LLC first, and then use `clflush` instructions to directly evict part of the addresses and start accessing new addresses (similar to a flush-based eviction attacker as discussed in Section II-B). Maya is secure against this sophisticated attack. The strong security guarantee provided by Maya is not just because of randomized mapping and global random eviction but also due to the presence of six extra invalid ways per skew. This leads to a partially filled tag store even at full cache capacity, which makes it impossible for a sophisticated attacker to cause an SAE within the lifetime of a system (10^{16} years). We make a provision in our security model to accommodate the sophisticated attacker. The attacker accesses its private data and then uses `clflush` operation to evict some of the accessed data. We make sure that the attacker does not trigger any global random eviction. The security guarantee for Maya remains intact with these modifications.

Cache occupancy attack and Maya. The Maya cache, by design, does not mitigate cache occupancy attacks and even a fully associative cache is prone to cache occupancy attacks. However, the Maya cache, while mitigating conflict-based attacks, should not make it easier to mount an occupancy-based attack. To evaluate the same, we mount an LLC occupancy-based attack. We attack AES (OpenSSL implementation with T tables) and modular exponentiation and compare the number of encryptions required to break the keys. To make it a strong attack, we simulate AES and modular exponentiation with two different keys, each having different reuse profiles at the LLC so that an attacker can exploit the Maya cache. The goal of the attacker is to distinguish these keys based on the reuse profiles. We run the attack 1,000,000 times and report the median of number of encryptions required for distinguishing the keys.

Normalized to a fully associative cache that uses a random replacement policy, the Maya cache behaves almost similarly (not the same) to a fully associative cache, with normalized

TABLE V: Simulation parameters of the baseline system.

Core	8 cores, Out-of-order, hashed perceptron [20], 4 GHz with 6-issue width, 4-retire width, 512-entry ROB
TLBs	L1 ITLB/DTLB: 64 entries, 4-way, 1 cycle, STLB: 2048 entries, 16-way, 8 cycles
L1I	32 KB, 8-way, 1 cycle
L1D	48 KB, 12-way, 5 cycles, IPCP prefetcher [26]
L2	512 KB 8-way associative, 10 cycles, LRU, non-inclusive
LLC	2 MB/core, 16-way, 24 cycles, SRRIP [19], non-inclusive
MSHRs	8/16/32 at L1I/L1D/L2, 64/core at the LLC
DRAM controller	DDR4-3200, two channels/8-cores 4 KB row-buffer per bank, open page, burst length 16, $t_{RP, RCD, CAS}$: 12.5 ns

values of 0.996 and 0.992 for AES and modular exponentiation, respectively (Figure 9). Note that we normalize the number of encryptions to the number of encryptions required for a fully associative cache (10590 for AES and 94 for modular exponentiation). As expected, a 16-way associative cache makes it easier to mount an attack, with normalized encryptions of 0.85 (15% easier) and 0.63 (37% easier) for AES and modular exponentiation, respectively.

V. Performance Analysis

A. Methodology

We use the ChampSim [3] micro-architecture simulator to evaluate different cache designs. We use a non-secure 8-core 16MB, 16-way set-associative last-level cache with 64-byte cache lines as the baseline. Table V provides the simulated parameters of the baseline non-secure system configuration. We evaluate 42 homogeneous workloads created from 42 different sim-points from the SPEC CPU2017 benchmark suite [2] and 20 homogeneous workloads from 20 different sim-points from the GAP benchmark suite [4], with more than one LLC miss per kilo instruction (MPKI) for the baseline configuration. Note that we select benchmarks based on their LLC MPKI for a single core 2MB LLC. We also use a set of 21 heterogeneous mixes with randomly chosen benchmarks from the SPEC CPU2017 and GAP suites. Table VI shows the heterogeneous mixes representing the behavior of more than 1000 heterogeneous mixes. Table VII shows the average LLC MPKI for a 16MB LLC with eight cores of SPEC CPU2017 and GAP homogeneous mixes and heterogeneous mixes, respectively. Note that with Maya, there are tag-only misses at the LLC on top of tag+data misses.

We simulate 1.6B instructions for eight cores (200M instructions per core in the region of interest after a warmup of 200M instructions per core). We use the weighted speedup [37] performance metric to compare the performance of different cache designs for an 8-core multi-core system. We compare the performance of the Maya cache design with a non-secure baseline and the Mirage cache design. We also perform a sensitivity study on the LLC size per core and later analyze the performance of Maya for higher-core systems.

B. Performance

Homogeneous mixes. Figure 10 shows the performance for Maya and Mirage normalized to the baseline for various homogeneous SPEC and GAP workloads. For the SPEC benchmarks, on average, Maya outperforms Mirage marginally, with a marginal performance improvement over the baseline. For benchmarks such as `mcf`, `wrf`, `fotonik3d`, we observe a substantial increase in performance for Maya despite the higher LLC latency. We observe that for these benchmarks, Maya reduces the inter-core interference in the data store by more than 70% compared to the baseline due to the notion of storing only "useful" entries in the data store. This compensates for the higher access latency and results in a performance gain for Maya. For benchmarks such as `lbm`, `cactuBSSN`, and `cam4`, Maya incurs a performance slowdown compared to the baseline. The `cam4` and `cactusBSSN` workloads have a relatively low dead block percentage in the LLC and low inter-core interference, even for the baseline. Therefore, it benefits from the larger data store of the baseline and Mirage, and we observe a performance slowdown of with Maya. For `lbm`, which is a streaming workload with almost zero load hit rate in the LLC, Mirage incurs a slowdown of around 8% compared to the baseline because of the extra 4-cycle access latency.

On average, Maya performs 5% better than the baseline for GAP workloads. The average improvement is influenced by 50% performance improvement with `pr`. For the `pr` workload, Mirage and Maya deliver 57% and 50% better performance than the baseline, respectively. This trend is contributed by a weak baseline for `pr`, where the IPCP prefetcher impacts the baseline performance as it behaves worse than no prefetching and LRU policy. The `cc` workload incurs a 5% performance slowdown compared to the baseline due to an increase in the inter-core interference in the data store.

Heterogeneous mixes. For the heterogeneous workloads (shown in Figure 11), Maya shows a 1.5% average performance improvement compared to the baseline, whereas Mirage incurs a marginal performance slowdown. Maya shows an improvement of more than 4% in performance for low-MPKI mixes because of the reduction in inter-core interference. Whereas medium-MPKI and high-MPKI mixes get a marginal performance slowdown because of their large working sets. In general, Maya helps improve performance for workloads with high inter-core interference and a high dead block percentage in the LLC. Note that in many mixes, Maya increases the miss rate at the LLC as it does not fill the cache line into the LLC on its first miss, providing tag-only hits. However, overall, it improves the performance as the useful entries are retained.

Performance of LLC fitting benchmarks. As Maya reduces the data store sizes, benchmarks that fit into the LLC may result in performance slowdowns. We simulate LLC fitting benchmarks from SPEC CPU2017 (LLC MPKI less than 0.5) and observe an average performance loss of 0.63% compared to a non-secure baseline.

Impact of random global tag eviction on performance. Random global tag evictions enhance the security provided by Maya. However, it can impact performance when a priority-0 entry that is yet to get reused (to be promoted to priority-1) gets invalidated by random global tag eviction. We quantify this event across all homogeneous mixes, and on average, less

TABLE VI: Heterogeneous mixes as per Table VII.

Mix	Composition	Bin
Mix-1	cactuBSSN(2)-wrf(1)-xalancbmk(1)-pop2(1)-roms(1)-xz(1)-sssp(1)	Low
Mix-2	bwaves(1)-mcf(1)-cactuBSSN(1)-wrf(1)-xalancbmk(1)-xz(1)-bfs(1)-sssp(1)	Low
Mix-3	mcf(1)-cactuBSSN(1)-omnetpp(1)-xalancbmk(1)-roms(1)-bfs(1)-cc(1)-sssp(1)	Low
Mix-4	perlbench(1)-bwaves(1)-mcf(3)-cam4(1)-xz(1)-bc(1)	Low
Mix-5	perlbench(1)-mcf(2)-cactuBSSN(1)-roms(1)-xz(1)-bc(1)-pr(1)	Low
Mix-6	gcc(1)-mcf(2)-cactuBSSN(1)-lbm(2)-fotonik3d(1)-roms(1)	Low
Mix-7	bwaves(1)-mcf(1)-cactuBSSN(1)-pop2(1)-xz(1)-bc(2)-sssp(1)	Low
Mix-8	gcc(2)-bwaves(1)-x264(1)-bc(1)-cc(1)-pr(1)-sssp(1)	Medium
Mix-9	gcc(1)-cactuBSSN(1)-lbm(1)-xalancbmk(1)-x264(1)-cam4(1)-pr(1)-sssp(1)	Medium
Mix-10	mcf(3)-lbm(1)-wrf(1)-fotonik3d(2)-sssp(1)	Medium
Mix-11	mcf(3)-lbm(1)-omnetpp(1)-pop2(1)-roms(1)-cc(1)	Medium
Mix-12	mcf(2)-cactuBSSN(1)-fotonik3d(1)-roms(2)-cc(1)-pr(1)	Medium
Mix-13	bwaves(1)-mcf(1)-xalancbmk(1)-fotonik3d(1)-roms(2)-bc(1)-sssp(1)	Medium
Mix-14	mcf(1)-lbm(1)-xalancbmk(1)-roms(1)-bc(1)-cc(1)-sssp(2)	Medium
Mix-15	bwaves(1)-cactuBSSN(1)-lbm(1)-roms(2)-bfs(1)-pr(1)-sssp(1)	High
Mix-16	mcf(3)-cactuBSSN(1)-lbm(1)-bfs(2)-cc(1)	High
Mix-17	mcf(1)-cactuBSSN(1)-wrf(1)-xalancbmk(1)-x264(1)-bc(1)-pr(2)	High
Mix-18	omnetpp(1)-wrf(1)-fotonik3d(1)-roms(1)-bc(2)-cc(1)-sssp(1)	High
Mix-19	bwaves(1)-mcf(2)-cactuBSSN(1)-xalancbmk(1)-bfs(1)-pr(1)-sssp(1)	High
Mix-20	perlbench(1)-mcf(2)-omnetpp(1)-fotonik3d(1)-pr(1)-sssp(2)	High
Mix-21	gcc(1)-bwaves(1)-mcf(2)-lbm(1)-bc(1)-pr(2)	High

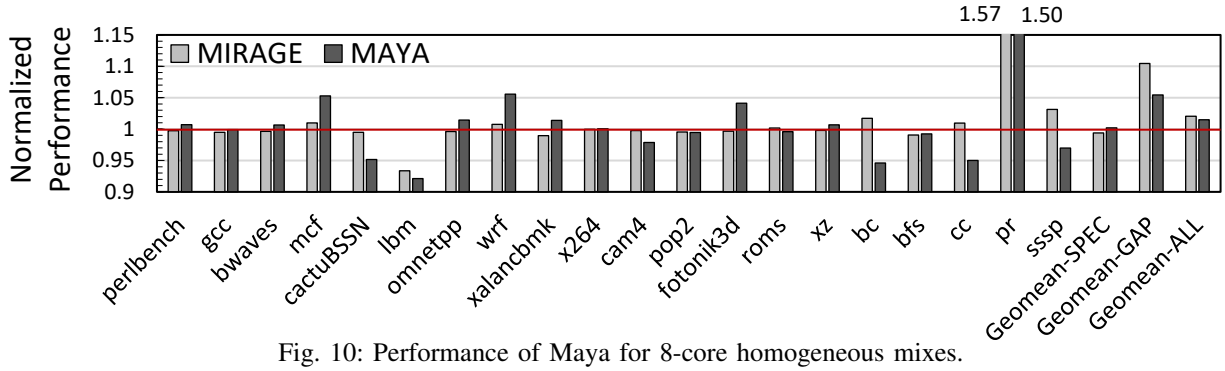

Fig. 10: Performance of Maya for 8-core homogeneous mixes.

TABLE VII: Average LLC MPKI with for a 16MB LLC. With Maya, there are tag-only LLC misses on top of tag+data.

Workloads		Baseline	Mirage	Maya
SPEC and GAP-RATE		13.9	12.5	12.5
HETERO	**LOW**	8.01	8.05	8.53
	MEDIUM	14.72	14.73	15.31
	HIGH	21.51	21.48	21.04

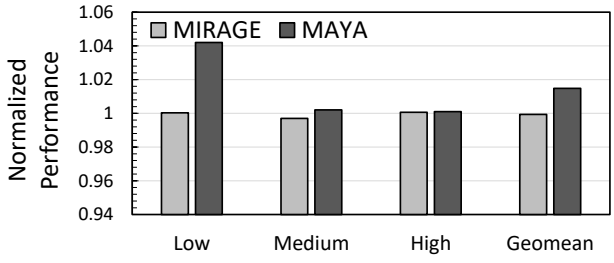

Fig. 11: Performance of Maya for 8-core heterogeneous mixes.

than 0.022% of the random global tag evictions to priority-0 entries would have gotten reused if we had used a non-random policy. We use the SRRIP [19] policy as the non-random policy. Overall, the negative impact of random global eviction of priority-0 entries has a marginal impact on performance.

Sensitivity to LLC size. For the Maya cache, we used an LLC data store size of 12MB (1.5MB per core). We now evaluate the performance of Maya with 6MB to 96MB data stores (baseline LLC size varying from 8MB to 128MB). Note that we also scale the tag store proportionately to the data store. Figure 12 shows the trend in normalized performance of homogeneous SPEC CPU2017 workloads as we increase the data store size from 6MB to 96MB. We observe that the 6MB Maya configuration shows the best performance compared to its counterpart baseline configuration. Performance decreases marginally as we increase the LLC size beyond 24MB as a large fraction of the working set starts getting LLC hits. The 1.9% increase in performance for the 6MB LLC per core can be attributed to the fact that the Maya cache design stores only "useful" entries in the data store and uses the smaller data store more efficiently than the baseline.

Sensitivity to number of cores. Figure 13 shows the trend in normalized performance of homogeneous SPEC CPU2017 workloads as we increase the number of cores from 8 to 32.

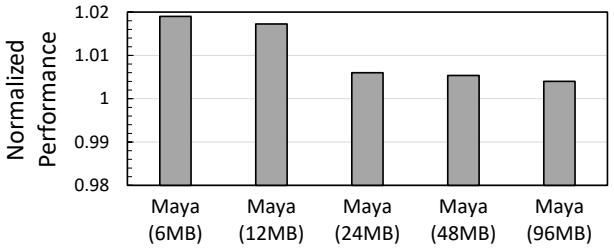

Fig. 12: Sensitivity study: LLC size on performance.

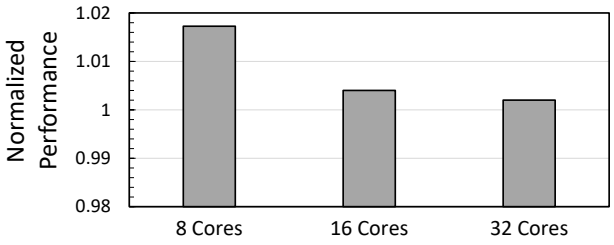

Fig. 13: Sensitivity study: Number of cores on performance.

Compared to an 8-core system, with 16 and 32-core systems, we observe marginal performance improvements over their respective baselines. Note that, the performance degradation for 32 cores compared to 16 cores is smaller than that of 16 cores compared to 8 cores, which signifies that the performance loss saturates as the number of cores increases. This shows that the Maya cache design can be extended to many-core systems.

VI. Storage, Area, and Power overheads

Storage. A self-contained Table VIII shows the storage requirements of Maya, Mirage, and the baseline.

Power consumption and energy. To estimate the static power and the dynamic access energy, along with the area required, we use the 7nm FinFET technology simulated using P-CACTI [1]. Table IX summarizes the observed dynamic energy and static power results for all three cache designs. We observe a reduction of 15.55% in dynamic read energy and 11.40% in dynamic write energy for the Maya cache compared to the baseline. On the other hand, Mirage shows a 3.81% increase in dynamic read energy and 4.52% increase in dynamic write energy compared to the baseline. We observe that the dynamic read/write energy is largely dominated by the energy required by the data store. Since Maya uses a smaller data store, we observe savings in dynamic energy for both reads and writes. Regarding the static power, Maya incurs 5.46% less power compared to the baseline. In contrast, Mirage incurs a power overhead of 18.16%, owing to the larger tag store and same-sized data store compared to the baseline.

Area. The data store largely takes up the area of the LLC. Because of this, the small data store design of the Maya cache can show savings of over 28.11% compared to the baseline, as seen in Table IX, whereas Mirage suffers a 6.86% area overhead due to the larger tag store. Note that Maya with

TABLE VIII: Storage comparison with the Baseline LLC (16MB) and Mirage (16MB). Storage of +20% and -2% denote 20% overhead and 2% savings, respectively.

Configurations		Baseline	Mirage	Maya
Tag Entry	Tag Bits	26	40	40
	Coherence	3	3	3
	Priority	-	-	1
	FPTR	-	18	18
	SDID	-	8	8
	Total bits	29	69	70
Tag Entries		262144	458752	491520
Tag Store Size		928 KB	3864 KB	4200 KB
Data Entry	Data Bits	512	512	512
	RPTR	-	19	19
	Total Bits	512	531	531
Data Entries		262144	262144	196608
Data Store Size		16384 KB	16992 KB	12744 KB
Total Storage		17312 KB	20856 KB (+20%)	16994 KB (-2%)

TABLE IX: Energy, power, and area overheads. Note that we use `fast` access mode: data and tag access happen in parallel. Maya ISO area is Maya with a similar area (16.085 mm^2) as Mirage (15.887 mm^2). Maya cache with 16.085 mm^2 is the closest implementation of Maya with the ISO area budget.

Design	Read Energy / Access (nJ)	Write Energy / Access (nJ)	Static Power (mW)	Area (mm^2)
Baseline	3.153	4.652	622	14.868
Mirage	3.274	4.857	735	15.887
Maya	2.661	4.116	588	10.686
Maya ISO	3.276	4.862	760	16.085

ISO area budget consumes more static power as the ISO area implementation incurs a slight increase in area.

Summary. Table X shows a comparison of the Maya, Mirage, Mirage-Lite, and Maya ISO area cache designs with respect to the baseline. The Maya cache design provides an optimal balance between security, storage, and performance. Maya ISO area incurs a storage overhead of 26%, static power overhead of 22.1%, and improves performance by 1.84%, whereas Maya provides a performance improvement of 1.72% with a storage savings of 2% and static power savings of 5.46%. One can argue that the storage overhead of Mirage can be mitigated by removing the decoupled nature of the data store and extra FPTR/RPTR bits and ensuring that the total number of valid entries is below a particular threshold in the cache to ensure security. On a miss, the line can be installed into the set with fewer valid entries (load-aware selection), and random global evictions are enabled once the total valid entries in the cache reach a threshold. Using our bucket-and-balls model for 16MB LLC with 75% maximum cache entries (equivalent to a 12MB LLC), we observe that this method leads to an SAE after less than 10^9 cache installs (<1s) due to only 4 extra invalid ways per skew. Even if we enable cuckoo reallocation [32] (at most 3 re-allocations), the security guarantee will be similar to that

TABLE X: Storage and performance overheads. Performance is evaluated on SPEC CPU2017 homogeneous mixes.

Cache Design	Security (Installs per SAE)	Storage	Performance
Maya	10^{32} (10^{16} yrs)	-2%	$+1.72\%$
Mirage	10^{34} (10^{17} yrs)	$+20\%$	-0.55%
Mirage-Lite*	10^{21} (22,000 yrs)	$+17\%$	-0.55%
Maya ISO	10^{30} (10^{14} yrs)	$+26\%$	$+1.84\%$

TABLE XI: Performance and storage overheads with secure partitioning techniques for an 8-core system with 16MB LLC.

Technique	Performance	Storage
Page coloring [8]	-19%	$+0.5\%$
DAWG [22]	-16%	$+0.5\%$
BCE [13]	-9%	$+2\%$

of Mirage-Lite, which is much worse than Mirage or Maya's security guarantees (shown in Table X).

VII. Comparison with other related works

Secure LLC partitioning techniques. Dynamically Allocated Way Guard (DAWG) [22] at the LLC uses a software configurable mask to decide way allocations among multiple security domains running on a multi-core system. One of the limitations of DAWG is the upper limit on the isolated domains that are bounded by the number of LLC ways. Maya does not have this limitation. Page Coloring [8] at the LLC creates isolated regions at the LLC set level. LLC partitioning via page coloring creates different DRAM regions and uses DRAM region bits with LLC index bits to access LLC. One of the limitations of the page coloring technique is that it cannot manage LLC space and DRAM space independently. Bespoke Cache Enclave (BCE) [31] makes a case for flexible cache partitioning that provides isolation by creating partitions as small as 64KBs. One of the key benefits of BCE is that the number of partitions is not restricted by the number of LLC ways, and LLC space allocation is independent of DRAM space allocation, making it a scalable technique compared with DAWG and page coloring. Note that LLC partitioning techniques can mitigate both conflict and occupancy-based attacks. However, these techniques incur significant performance overheads. Table XI shows performance and storage overheads with three state-of-the-art secure LLC partitioning techniques for SPEC CPU2017 homogeneous mixes. Overall, partitioning techniques incur significant performance overheads (as compared to randomized caches) with marginal storage overheads.

Other approaches. HybCache [10] provides fully associative LLC but only mapping for a subset of the LLC (one to three ways). However, it is expensive to extend HybCache for the entire LLC. In contrast, Maya provides a fully associative LLC even for an extremely large LLC like 32MB LLC. Phantom-Cache [39] installs an incoming cache line in one of eight randomly chosen sets in the cache, each with 16 ways, increasing the LLC associativity to 128, which increases the energy overhead by more than 60%. On the other hand, Maya is energy efficient with energy savings and no overheads. Zcache [33] increases associativity and the pool of replacement candidates. However, the pool is limited in size (64 entries) making it vulnerable to conflict-based attacks.

VIII. Conclusion

We presented Maya, a randomized fully associative last-level cache that uses additional tag entries and fewer data entries. Overall, Maya guarantees that it will take 10^{16} years for one set associative eviction to initiate a conflict-based attack, which is more than the system's lifetime. Maya is energy-efficient (5.46% less static power) and area-efficient (28.11% savings) thanks to a smaller data store. Maya provides a strong security guarantee with storage savings (and not overhead) compared to a non-secure baseline cache. Overall, Maya provides the sweet spot in terms of security, performance, area, and energy overhead.

IX. Acknowledgements

We would like to thank all the anonymous ISCA-24 reviewers for their helpful comments and suggestions. Special thanks to Pratik for the initial implementation of the PRINCE cipher and randomized caches. We would also like to thank members of the CASPER research group, Gururaj, Moin, and Sayandeep for their feedback on the initial draft. This work is supported by the Trust Lab Research Grant 2023.

Appendix

A. Abstract

This artifact contains all the information necessary to reproduce the main experiments in the paper. We describe how the required software and the elements that compose it can be obtained, and how to run the artifact.

B. Artifact check-list (meta-information)

- **Program:** ChampSim
- **Compilation:** GNU GCC 7.5.0
- **Data set:** SPEC CPU2017 traces from 3^{rd} Data Prefetching Championship (https://dpc3.compas.cs.stonybrook.edu/champsim-traces/speccpu/), GAP traces (https://utexas.app.box.com/s/2k54kp8zvrqdfaa8cdhfquvcxwh7yn85/folder/132804598561)
- **Run-time environment:** an Intel x86_64 processor
- **Hardware:** tested on an Intel Xeon Gold 5220R
- **Metrics:** Weighted Speedup
- **Output:** four PDF files with graphs
- **How much disk space required (approximately)?:** 35 GB
- **How much time is needed to prepare workflow (approximately)?:** 30 minutes
- **How much time is needed to complete experiments (approximately)?:** 5-6 days
- **Publicly available?:** yes
- **Archived (provide DOI)?:** 10.5281/zenodo.10894886

C. Description

1) How to access: The software can be obtained from GitHub: https://github.com/AnubhavBhatla/maya-cache
Use the following command to clone the repository:

```
$ git clone
https://github.com/AnubhavBhatla/maya-cache
```

2) Software dependencies: We test the artifact on a system with these features:

- Ubuntu 20.04.4 LTS
- Linux Kernel 5.15.0
- Python 3.8.10
- Bash 5.0.17
- GCC 7.5.0
- **Python3 Packages**
  - matplotlib 3.7.3
  - numpy 1.24.4
  - pandas 2.0.3

The Python3 packages can be downloaded using the command:

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

3) Data sets: For this artifact the SPEC CPU2017 traces from the 3^{rd} Data Prefetching Championship (https://dpc3.compas.cs.stonybrook.edu/champsim-traces/speccpu/) and the GAP traces (https://utexas.app.box.com/s/2k54kp8zvrqdfaa8cdhfquvcxwh7yn85/folder/132804598561) are needed. These traces are automatically downloaded by the artifact.

D. Installation & Experiment workflow

The overall flow for running the artifact is as follows:

1) Clone the repository:
   ```
   $ git clone
   https://github.com/AnubhavBhatla/maya-cache
   ```
2) Enter the performance-analysis directory:
   ```
   $ cd maya-cache/performance-analysis
   ```
3) Download the required traces:
   The zip file for the required GAP traces can be downloaded from https://utexas.app.box.com/s/2k54kp8zvrqdfaa8cdhfquvcxwh7yn85/folder/132804598561. Extract it in the `traces` directory. To download the required SPEC CPU2017 traces, run the command: `$ ./traces.sh` This also validates if the GAP benchmarks have been extracted in the correct directory.
4) Generate the required binaries:
   ```
   $ ./compile.sh
   ```
5) Run the performance simulations:
   ```
   $ ./run.sh
   ```
   This step will take a large amount of time to complete (5-6 days).
6) Generate the performance plots:
   Once all the performance simulations have been completed, the plots can be generated using
   ```
   $ ./plot.sh 0
   ```
   We have also provided our simulation results which can be used to generate the plots using:
   ```
   $ ./plot.sh 1
   ```
7) Enter the security-analysis directory:
   ```
   $ cd ../security-analysis
   ```
8) Generate the required binaries:
   ```
   $ make
   ```
9) Run the security simulations:
   ```
   $ ./run.sh
   ```
10) Generate the security plots:
    Once all the security simulations have been completed, the plots can be generated using
    ```
    $ python3 plot.py
    ```

E. Evaluation and expected results

In the `performance-analysis` directory, two graphs are generated, namely, `fig1.pdf`, `fig10.pdf`. In the `security-analysis` directory, two graphs are generated, namely, `fig7.pdf`, `fig8.pdf`.

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, Feb. 2019.

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

[4] "GAP traces for champsim," https://utexas.app.box.com/s/2k54kp8zvrqdfaa8cdhfquvcxwh7yn85/folder/132804598561, Mar. 2021.

[5] J. Albericio, P. Ibáñez, V. Viñals, and J. M. Llabería, "The reuse cache: Downsizing the shared last-level cache," in *2013 46th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2013, pp. 310–321.

[6] J. Borghoff, A. Canteaut, T. Güneysu, E. B. Kavun, M. Knezevic, L. R. Knudsen, G. Leander, V. Nikov, C. Paar, C. Rechberger, P. Rombouts, S. S. Thomsen, and T. Yalçin, "PRINCE - A low-latency block cipher for pervasive computing applications (full version)," *IACR Cryptol. ePrint Arch.*, p. 529, 2012. [Online]. Available: http://eprint.iacr.org/2012/529

[7] T. Bourgeat, J. Drean, Y. Yang, L. Tsai, J. Emer, and M. Yan, "Casa: End-to-end quantitative security analysis of randomly mapped caches," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1110–1123.

[8] T. Bourgeat, I. A. Lebedev, A. Wright, S. Zhang, Arvind, and S. Devadas, "MI6: secure enclaves in a speculative out-of-order processor," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture, MICRO 2019, Columbus, OH, USA, October 12-16, 2019*. ACM, 2019, pp. 42–56. [Online]. Available: https://doi.org/10.1145/3352460.3358310

[9] S. Briongos, P. Malagon, J. M. Moya, and T. Eisenbarth, "RELOAD+REFRESH: Abusing cache replacement policies to perform stealthy cache attacks," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 1967–1984. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/briongos

[10] G. Dessouky, T. Frassetto, and A.-R. Sadeghi, "HybCache: Hybrid Side-Channel-Resilient caches for trusted execution environments," in *29th USENIX Security Symposium (USENIX Security 20)*. USENIX Association, Aug. 2020, pp. 451–468. [Online]. Available: https://www.usenix.org/conference/usenixsecurity20/presentation/dessouky

[11] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+Abort: A Timer-Free High-Precision l3 cache attack using intel TSX," in *26th USENIX Security Symposium (USENIX Security 17)*. Vancouver, BC: USENIX Association, Aug. 2017, pp. 51–67. [Online]. Available: https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/disselkoen

[12] D. et al., "Non-monopolizable caches: Low-complexity mitigation of cache side channel attacks," vol. 8, no. 4, jan 2012. [Online]. Available: https://doi.org/10.1145/2086696.2086714

[13] S. et al., "Bespoke cache enclaves: Fine-grained and scalable isolation from cache side-channels via flexible set-partitioning," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*. IEEE, 2021, pp. 37–49.

[14] W. et al., "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*, ser. Proceedings - IEEE Symposium on Security and Privacy. United States: Institute of Electrical and Electronics Engineers Inc., May 2021, pp. 955–969.

[15] L. Giner, S. Steinegger, A. Purnal, M. Eichlseder, T. Unterluggauer, S. Mangard, and D. Gruss, "Scatter and split securely: Defeating cache contention and occupancy attacks," in *44th IEEE Symposium on Security and Privacy, SP 2023, San Francisco, CA, USA, May 21-25, 2023*. IEEE, 2023, pp. 2273–2287. [Online]. Available: https://doi.org/10.1109/SP46215.2023.10179440

[16] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing smap and kernel aslr," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[17] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *Proceedings of the 13th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*, 2016.

[18] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive Last-Level caches," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015.

[19] A. Jaleel, K. B. Theobald, S. C. S. Jr., and J. S. Emer, "High performance cache replacement using re-reference interval prediction (rrip)," in *37th Int'l Symp. on Computer Architecture (ISCA)*, Jun. 2010, pp. 60–71.

[20] D. A. Jiménez and C. Lin, "Dynamic branch prediction with perceptrons," in *7th Int'l Symp. on High-Performance Computer Architecture (HPCA)*, Jan. 2001, pp. 197–206.

[21] S. M. Khan, Y. Tian, and D. A. Jiménez, "Sampling dead block prediction for last-level caches," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 175–186.

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

[23] D. J. Lilja, *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, 2000.

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

[25] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and countermeasures: The case of aes," in *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, ser. CT-RSA'06. Berlin, Heidelberg: Springer-Verlag, 2006, p. 1–20. [Online]. Available: https://doi.org/10.1007/11605805_1

[26] S. Pakalapati and B. Panda, "Bouquet of instruction pointers: Instruction pointer classifier-based spatial hardware prefetching," in *2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA)*, 2020, pp. 118–131.

[27] A. Purnal, L. Giner, D. Gruss, and I. Verbauwhede, "Systematic analysis of randomization-based protected cache architectures," in *2021 IEEE Symposium on Security and Privacy (SP)*, 2021, pp. 987–1002.

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

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

[30] M. K. Qureshi, "New attacks and defense for encrypted-address cache," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 360–371. [Online]. Available: https://doi.org/10.1145/3307650.3322246

[31] G. Saileshwar, S. Kariyappa, and M. Qureshi, "Bespoke cache enclaves: Fine-grained and scalable isolation from cache side-channels via flexible set-partitioning," in *2021 International Symposium on Secure and Private Execution Environment Design (SEED)*, 2021, pp. 37–49.

[32] G. Saileshwar and M. Qureshi, "MIRAGE: Mitigating conflict-based cache attacks with a practical fully-associative design," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[33] D. Sanchez and C. Kozyrakis, "The zcache: Decoupling ways and associativity," in *2010 43rd Annual IEEE/ACM International Symposium on Microarchitecture*, 2010, pp. 187–198.

[34] A. Seznec, "A case for two-way skewed-associative caches," in *20st Int'l Symp. on Computer Architecture (ISCA)*, May 1993, pp. 169–178.

[35] I. Shah, A. Jain, and C. Lin, "Effective mimicry of belady's min policy," in *2022 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, 2022, pp. 558–572.

[36] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust website fingerprinting through the cache occupancy channel," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 639–656. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/shusterman

[37] A. Snavely and D. M. Tullsen, "Symbiotic jobscheduling for a simultaneous multithreaded processor," *SIGOPS Oper. Syst. Rev.*, vol. 34, no. 5, p. 234–244, nov 2000. [Online]. Available: https://doi.org/10.1145/384264.379244

[38] W. Song, B. Li, Z. Xue, Z. Li, W. Wang, and P. 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*, 2021.

[39] Q. Tan, Z. Zeng, K. Bu, and K. Ren, "Phantomcache: Obfuscating cache conflicts with localized randomization," *Proceedings 2020 Network and Distributed System Security Symposium*, 2020. [Online]. Available: https://api.semanticscholar.org/CorpusID:211483077

[40] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "ScatterCache: Thwarting cache attacks via cache set randomization," in *28th USENIX Security Symposium (USENIX Security 19)*. Santa Clara, CA: USENIX Association, Aug. 2019, pp. 675–692. [Online]. Available: https://www.usenix.org/conference/usenixsecurity19/presentation/werner

[41] M. Yan, J.-Y. Wen, C. W. Fletcher, and J. Torrellas, "Secdir: a secure directory to defeat directory side-channel attacks," in *Proceedings of the 46th International Symposium on Computer Architecture*, ser. ISCA '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 332–345. [Online]. Available: https://doi.org/10.1145/3307650.3326635

[42] Y. Yarom and K. Falkner, "Flush+reload: A high resolution, low noise, L3 cache side-channel attack," in *Proceedings of the 23rd USENIX Conference on Security Symposium*, 2014.