Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It

Wei Song[*†], Boya Li[*†], Zihan Xue[*†], Zhenzhen Li[*†], Wenhao Wang[*†], Peng Liu[‡]

[*]State Key Laboratory of Information Security, Institute of Information Engineering, CAS, Beijing, China
[†]School of Cyber Security, University of Chinese Academy of Sciences, Beijing, China
[‡]The Pennsylvania State University, University Park, USA
{songwei, liboya, xuezihan, lizhenzhen1, wangwenhao}@iie.ac.cn, pxl20@psu.edu

Abstract—Cache randomization has recently been revived as a promising defense against conflict-based cache side-channel attacks. As two of the latest implementations, CEASER-S and ScatterCache both claim to thwart conflict-based cache side-channel attacks using randomized skewed caches. Unfortunately, our experiments show that an attacker can easily find a usable eviction set within the chosen remap period of CEASER-S and increasing the number of partitions without dynamic remapping, such as ScatterCache, cannot eliminate the threat. By quantitatively analyzing the access patterns left by various attacks in the LLC, we have newly discovered several problems with the hypotheses and implementations of randomized caches, which are also overlooked by the research on conflict-based cache side-channel attacks.

However, cache randomization is not a false hope and it is an effective defense that should be widely adopted in future processors. The newly discovered problems are corresponding to flaws associated with the existing implementation of cache randomization and are fixable. Several new defense ideas are proposed in this paper. Our experiments show that all the newly discovered problems are fixed within the current performance budget. We also argue that randomized set-associative caches can be sufficiently strengthened and possess a better chance to be actually adopted in commercial processors than their skewed counterparts because they introduce less overhaul to the existing cache structure.

I. INTRODUCTION

To reduce the latency of accessing memory, modern computers adopt a multi-level cache hierarchy where the last-level cache (LLC) is shared between all processing cores. Such sharing improves the utilization efficiency of the LLC as it can dynamically adapt its space allocation to the demand of different cores. However, it also allows a malicious software to trigger controlled conflicts in the LLC, such as evicting a specific cache set with attackers' data [1]–[3], to infer security-critical information of a victim program. This type of conflict-based cache side-channel attacks have already been utilized to recover cryptographic keys [4], break the sandbox defense [5], inject faults directly into the DRAM [6], and extract information from the supposedly secure SGX enclaves [7].

Cache partitioning [8]–[10] used to be the only effective defense against conflict-based cache side-channel attacks abusing the LLC. It separates security-critical data from normal data in the LLC; therefore, attackers cannot evict security-critical data by triggering conflicts using normal data. However, cache partitioning is ineffective when security-critical data cannot be easily separated from normal data [11] or normal data become the target [6]. It also reduces the autonomy of the LLC which might in turn hurt performance for some applications [12]. Finally, cache partitioning relies on specific operating system (OS) code to identify security-critical data, which means the OS must be trusted.

Recently, cache randomization [13]–[21] has been revived as a promising defense. Instead of cache partitioning, cache randomization randomizes the mapping from memory addresses to cache set indices. This forces attackers to slowly find eviction sets using search algorithms at run-time [3], [22]–[24] rather than directly calculating cache set indices beforehand. Even when eviction sets are found, attackers cannot tell which cache sets are evicted by them. However, cache randomization alone does not defeat conflict-based cache side-channel attacks but only increases difficulty and latency [16]. For this reason, dynamic remapping [16], [19] has been introduced to limit the time window available to attackers and skewed cache [17]–[19] has been proposed to further increase the attack difficulty.

As two of the latest implementations, CEASER-S [17] and ScatterCache [18] both claim to thwart conflict-based cache side-channel attacks using randomized skewed caches. ScatterCache even argues that dynamic remapping might not be necessary as the extra difficulty introduced by skewed cache is hard enough. Unfortunately, our experiments show that an attacker can easily find a usable eviction set within the chosen remap period of CEASER-S [17] and increasing the number of partitions without dynamic remapping, such as ScatterCache [18], cannot eliminate the threat. By quantitatively analyzing the access patterns left by various attacks in the LLC, we have newly discovered several problems with the hypotheses and implementations of randomized caches, which are also overlooked by the research on conflict-based cache side-channel attacks.

- *The possibility of using cache flush instructions in conflict-based attacks has been overlooked.* Our study shows, if attackers flush the eviction set after each probe, partial congruent eviction sets can be repeatedly used to drastically speed up attacks.
- *The concept of minimal eviction set no longer applies to randomized skewed caches.* Any group of cache blocks that can evict the target address with a reasonable probability should be considered as a usable eviction set.

- *Attackers do not have to use eviction sets with 99% eviction rate.* When finding such sets become too difficult, attackers will utilize eviction sets with low eviction rate but possible to find.
- *Measuring the remap period by LLC accesses is flawed,* since a significant portion of all the cache accesses might be filtered by the private level-one (L1) or level-two (L2) caches. The actual number of accesses observed by the LLC is much smaller than the total number of cache accesses. As a result, the remap period estimated by CEASER-S [17] is over-optimistic.

However, cache randomization is not a false hope. As researchers on the defense side, we strongly believe it is an effective defense strategy that should be widely adopted in future processors. The above-discovered problems are corresponding to flaws associated with the existing tactics towards accomplishing the "cache randomization" strategy. We believe that these problems are fixable, and that fixing these problems will make the strategy significantly more effective in defending conflict-based cache side-channel attacks. In particular, several new defense ideas are proposed in this paper:

- *Measure the remap period by LLC evictions rather than accesses* because the probability of successfully finding an eviction set is closely related to the number of evictions allowed between remaps.
- *Further reduce the period* to stop attackers from finding even small partially congruent eviction sets.
- *Adopt ZCache-like [25] multi-step relocation to minimize the number of cache blocks evicted during each remap.*
- *Promote the use of CEASER (randomized set-associative cache) rather than skewed caches* because CEASER introduces less overhaul to the existing cache structure than skewed caches and it can be made secure enough.
- *A simple attack detection method to strengthen CEASER.*

By utilizing these defense ideas, our experiments show that all the newly discovered vulnerabilities of existing randomized caches can be fixed within the current performance budget and the randomized set-associative caches can be made secure enough with reasonable performance overhead.

This paper is organized as follows: Section II introduces the necessary background information to understand this paper. Section III formulates the problems we try to answer in this paper. Section IV demonstrates the vulnerabilities of existing randomized caches by experiments. Section V shows how we can fix the randomized skewed caches and Section VI presents solutions to safely strengthen the randomized set-associative caches. The performance overhead is analyzed in Section VII. The limitations and related work are discussed in Section VIII. Section IX finally concludes the paper.

II. BACKGROUND

A. Caches

Modern processors use caches to store recently or frequently used data to reduce the memory access time. Most caches adopt a set-associative structure [26] as shown in Fig. 1.
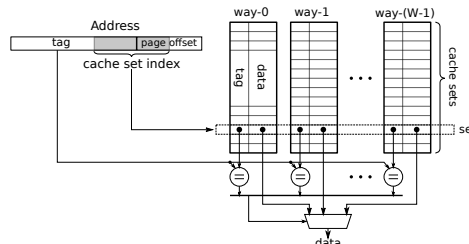

Fig. 1. A set-associative cache.

The cache space is divided into S cache sets and each set contains W ways of cache blocks. Cache sets are addressed by a cache set index which is typically a subset of the address bits shared by all cache blocks in the same set. If two addresses are mapped to the same cache set, they are congruent addresses [23]. When an address is accessed, the cache checks whether there is a match (hit) in the corresponding cache set by comparing tags. If no match is found (a miss), the cache block is fetched and stored in the cache set for future use. The specific position (way) to store this newly fetched cache block is chosen by a replacement policy and the old block is evicted. As a commonly used replacement policy, least-recently used (LRU) [27] retains the recently accessed cache blocks.

Multiple levels of caches are normally hierarchically organized. A processing core might have one or two levels of private caches (L1 and L2 caches) while all cores share a large LLC. An inclusive relationship between private caches and the LLC is usually adopted [26].[1] When a cache block is evicted from the LLC, it is also purged from all private caches. A hardware managed coherence protocol ensures data are correctly updated between caches.

B. Conflict-Based Cache Side-Channel Attacks

Conflict-based cache side-channel attacks [29] exploit the fact that cache blocks in the same set are congruent. This allows attackers to maliciously control the status of a target cache set using a group of at least W congruent addresses (cache blocks), namely an eviction set.

An attack normally occurs in two phases: *preparation phase* when the attacker collects enough number of eviction sets, and *exploitation phase* when the attacker infers sensitive information from a victim by controlling the status of certain cache sets using the collected eviction sets. Before cache randomization is applied, collecting eviction sets is relatively easy because attackers can deliberately construct an eviction set using addresses having the same cache set index bits [3]. This becomes unfeasible when caches are randomized. The exploitation phase normally contains numerous prime+probe cycles [1]–[3]. In each cycle, the attacker first *primes* a target set by filling it with cache blocks from a corresponding eviction set. If there were cache blocks belonging to the victim, they are likely evicted in the *prime* process. The attacker then tricks the victim into running a program segment related to the target cache set. If the victim indeed accesses data indexed to

---

[1]Some of the latest CPUs use non-inclusive LLCs but they might still suffer from conflict-based attacks when the directory is inclsuive [24], [28].
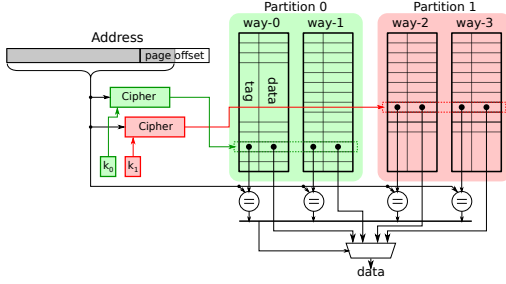
Fig. 2. A randomized skewed cache with two partitions over four cache ways.

the same cache set, it must have been fetched into the cache set and one block of the eviction set is consequently evicted. Finally the attacker *probes* the cache set by re-accessing all blocks of the eviction set. If the total access latency is longer than expected, the attacker learns that the victim should have accessed the target cache set, which might further infer other security-critical information.

C. Randomized Caches

The main objective of cache randomization is to deprive attackers from usable eviction sets [16]–[18]. The latest implementation of cache randomization is randomized skewed caches [17], [18], while randomized set-associative caches [16] can be considered as a special case with only one partition. Fig. 2 presents a randomized skewed cache whose four cache ways are evenly divided into two partitions independently indexed. Instead of using a subset of address bits, the cache set index is generated from a cipher taking the whole address and a hardware managed key as inputs. Assuming the encryption algorithm is unbroken and the key is not leaked, the cache set index is a random number unobservable to attackers. Therefore, they can no longer construct eviction sets simply by picking addresses but dynamically search for congruent addresses through run-time experiments, which was considered an intolerable long procedure [3], [22]–[24].

Another major benefit of randomized skewed caches is the reduced effectiveness of eviction sets. Two addresses are fully congruent when they are mapped to the same sets in all partitions while partially congruent when they are mapped to the same sets in some but not all partitions. A group of W addresses, where W is the number of ways, forms a fully congruent eviction set only when all of the W addresses are fully congruent. However, the probability that two random addresses are fully congruent in a K partitioned skewed cache is $\frac{1}{S^K}$, where S is the number of cache sets. This is an extremely small probability even with a moderate K. Finding such a fully congruent eviction set at run-time is unfeasible. Therefore, attackers have no choice but to use partially congruent eviction sets composed of partially congruent addresses. This has two drawbacks [18], [30]: The number of addresses needed is significantly increased and the eviction of the target address becomes a statistically random event.

D. Fast Algorithms for Searching Eviction Sets

At the time when CEASER was proposed, the fastest algorithm [3], [22] for finding a minimal eviction set with
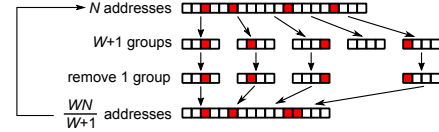

Fig. 3. Group elimination algorithm.

W addresses required $\mathcal{O}(N^2)$ cache accesses, where N is the number of addresses randomly collected to form a very large eviction set. As N is normally at the magnitude with the size of the LLC ($N \sim S \cdot W$) [24], $\mathcal{O}(N^2)$ cache accesses are just too long for any practical attacks. Soon afterwards, three fast search algorithms are proposed to drastically reduce the number of accesses.

Group elimination (GE) is an optimization of the original $\mathcal{O}(N^2)$ method [17], [23]. It still starts with a very large eviction set of N random addresses but it tries to remove multiple addresses in each cycle to quickly trim the set into a minimal one. Fig. 3 illustrates such a cycle targeting an LLC with four ways. The set of N addresses are divided into $W + 1$ groups. Since a minimal eviction set contains only W addresses (shadowed in red), there is at least one removable group containing none of the W addresses. By sequentially testing whether the set is still an eviction set without a certain group, the removable group is found and removed. Then the whole process starts again taking the remaining $\frac{WN}{W+1}$ addresses as the input set until a minimal set is produced. The whole process requires around $\mathcal{O}(WN)$ cache accesses, since $N \sim SW$, $\mathcal{O}(WN) = \mathcal{O}(SW^2)$.

Conflict testing (CT) is a new algorithm first proposed to find eviction sets in caches using random replacement [17]. Assuming an attacker has access to unlimited number of random addresses, she can collect an eviction set by sequentially testing each address whether it is congruent with the target address. The target address is accessed first to make it cached in the LLC. Then a random address is accessed. If this address is congruent with the target address, it might replace the target address by a chance of $\frac{1}{W}$ thanks to the random replacement. Overall, any random address might conflict with the target address by a probability of $\frac{1}{S \cdot W}$. To test the occurrence of such a conflict, the target address is re-accessed and timed. If the latency is longer than expected, the random address is considered congruent and put into the eviction set. The re-accessing of the target address also starts the test for the next random address. An eviction set is produced when enough congruent addresses are collected. The overall number of cache accesses is estimated around $\mathcal{O}(SW^2)$.

Note that this algorithm is also effective for permutation-based replacement (such as LRU). Assuming the use of LRU, the probability of causing a conflict with the target address after accessing M random addresses is around:

$$P = 1 - \sum_{i=0}^{W-1} \binom{M}{i} \frac{1}{S^i} (1 - \frac{1}{S})^{M-i} \tag{1}$$

This is equivalent to causing at least W conflicts in the target cache set. The average $\overline{M}$ is around SW. Note that re-accessing the target address is unlikely to cause an actual
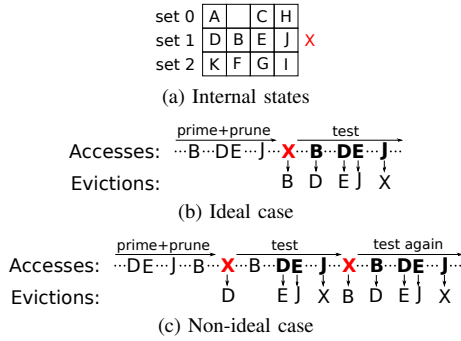
set 0	A		C	H
set 1	D	B	E	J	X
set 2	K	F	G	I

(a) Internal states

Accesses: ···B···DE···J···**X**··**B**··**DE**··**J**···
Evictions: B D E J X

prime+prune   test

(b) Ideal case

prime+prune   test   test again

Accesses: ···DE···J··B···**X**··B···**DE**··**J**··**X**··**B**··**DE**··**J**···
Evictions: D   E J X B D E J X

(c) Non-ideal case

Fig. 4. Prime, prune and then test on a 3-set 4-way LLC using the LRU replacement. (a) reveals the LLC internal states after prime and prune. (b) demonstrates a success test in an ideal case where all cache accesses are observed by the LLC. (c) shows a partial result in a non-ideal scenario when some cache accesses are filtered by the private caches and the observed orders of prime+prune and test are different.

access to the LLC because the target address is always cached in private caches (L1) until it is forcefully invalidated by a conflict in the LLC. As a result, the LRU replacer's internal state is unchanged for most re-accessing of the target address. To find a minimal eviction set with W addresses, the number of cache accesses is also around $\mathcal{O}(SW^2)$.

Prime, prune and then test (PPT) is an improved version of the search algorithm exploiting the LRU replacement [17], [30]. Let us consider an LLC using the LRU replacement. An attacker first accesses a large set of random addresses (prime set) to prime the whole LLC[2]. Since self-conflicts would naturally occur during the prime, a prune process is used to remove conflicted addresses until all addresses remaining in the prime set are simultaneously cached. Assuming Fig. 4a reveals the internal states of an LLC after prime and prune, the target cache set (set 1) is likely primed by the prime set. In an ideal scenario, the order of cache accesses observed by the LLC is the same order initiated by the attacker. As shown in Fig. 4b, if the attacker makes a timed re-access of the target address **X** and the prime set sequentially, all the addresses with long latency (miss in the LLC) are congruent with **X** and the number of them is just enough for an eviction set. However, the order seen by the LLC is normally different from the software order as many cache accesses are filtered by the private caches. In this scenario (Fig. 4c), the attacker collects some but less than W congruent addresses. She has to test again to force the order seen by the LLC equivalent to the software order. Regarding the total number of cache accesses, our experiments show that the prune process normally finishes in less than two rounds. Meanwhile, the size of the prime set after pruning is slightly less than the cache size, which means only one round of search is usually enough. The overall number of cache accesses is estimated around $\mathcal{O}(SW)$, which is the smallest in the three fast algorithms.

This algorithm can be used to find eviction sets in LLCs using other types of replacement policies. The estimated number of accesses for permutation-based replacement policies

[2]The attacker might choose to use a small set to prime a portion of the LLC but this will significantly reduce the success rate.

(including LRU) is normally the same ($\mathcal{O}(SW)$) [17] but it approaches to $\mathcal{O}(SW^2)$ for LLCs using random replacement for two reasons: One is the size of the prime set after pruning is much smaller than the cache size SW, which reduces the chance of finding congruent addresses in each round of search. The other one is that, even if the target cache set is primed, the number of congruent addresses found in each round of test is significantly less than W (only one in most cases). The attacker has to do multiple rounds of tests in multiple rounds of searches [30].

E. Attack Randomized Caches Using the Fast Algorithms

All the three search algorithms can easily defeat the static version of randomized set-associative caches, such as CEASE [16]. As a result, a randomized set-associative cache has to periodically remap its content by updating the hardware managed key (Fig. 2). This forces an attacker to dynamically search eviction sets and finish an attack both in the remap period. Short remap period increases the hardness to launch an attack [16]. However, frequent remaps lead to significant performance loss. During the remap process, all cache blocks in the LLC are sequentially relocated using the updated key. When there is no available space at the new location, a cache block is evicted to make space [16]. Our experiments show that 40% to 50% cache blocks are evicted for this reason.

To reduce the performance overhead while thwarting attacks, the remap period is carefully selected. For a 1024-set 16-way CEASER LLC, it has to remap around every 47K accesses (only three accesses per cache block) [17], which is an unbearably short period. This is why skewed caches are currently preferred. For a same sized CEASER-S LLC with two partitions, it is claimed that the remap period can be safely increased to 1.6M accesses (100 accesses per cache block) [17]. It becomes almost impossible to find fully congruent eviction sets in a randomized skewed cache remapped at the aforementioned rate [17]. In its current form, the *group elimination* algorithm simply fails in skewed cache due to the huge amount of false negative errors introduced by the randomly selected partitions. Both the *conflict testing* and the *prime, prune and then test* algorithms might still be able to find partially congruent eviction sets [30], which is the root of concern found in this paper.

III. PROBLEM FORMULATION

As researchers on the defense side, we would like to thoroughly examine the effectiveness of cache randomization and provide a strong defense against conflict-base side-channel attacks. To be specific, we plan to answer the following questions.

Problem Statement:

- Do the existing cache randomization schemes/techniques make any flawed hypothesis?
- If there are any flawed hypotheses, do they lead to broken defenses and discovery of new vulnerabilities?
- Whether the broken defenses, if any, can be fixed?

Before diving into the detailed analysis, let us first describe the threat model and the analysis platform.

A. Threat Model

The main objective of using cache randomization is to deprive attackers from usable eviction sets. We thus consider finding a usable eviction set targeting a specific address as a successful attack. Only conflict-based cache side-channel attacks targeting the LLC are considered in this paper.[3] In order to examine the effectiveness of defenses under hostile scenarios, we give attackers the following set of generous but still reasonable capabilities:

- She has fully reverse engineered the virtual to physical address mapping.
- She can access unlimited number of random addresses, make arbitrary memory accesses to her own data and accurately infer cache hit/miss status by measuring the access latency.
- She can flush a cache block from the whole cache hierarchy as long as it is her own data.
- She can accurately trick the victim into running a single memory access and there is no other active process during the attack.
- She has the full design details of the randomized cache, but the encryption algorithm and the key used for generating cache set indices are unbreakable.[4]

Note that we have explicitly allowed the attacker to flush her own data. This is different from flush-reload attacks [35] because no data is shared between the attacker and the victim. Although it is normally not required for conflict-based attacks targeting non-randomized caches, attackers do have such capability, such as a malicious user mode program running on an x86-64 processor [36] or a malicious kernel running on an ARM processor [37]. As described in Section IV-A, this enables attackers to launch fast attacks on the latest randomized caches using partially congruent eviction sets.

B. Analysis Platform

To quantitatively analyze the effectiveness of the latest randomized caches, we choose to implement CEASER-S and ScatterCache in a behavioral cache simulation model opensourced by [24], further extend the model with the defense ideas newly proposed in Section V and VI, and attack the randomized caches using the aforementioned fast search algorithms. All results revealed in Section IV, V and VI are obtained from these experimental attacks. To evaluate the impact of the new defense ideas on normal applications in Section VII, we run the SPEC CPU 2006 benchmark cases [38] on the RISC-V [39] instruction level simulator Spike [40] with its original cache model replaced with the extended cache model [24]. The use of Spike allows us to run benchmark cases at a speed around 1.5 million instructions

---

[3]Attacks utilizing the cache occupancy channel[31]–[33] are also out of the scope of this paper.

[4]we do not consider attacks targeting weak ciphers [34] or random number generators.
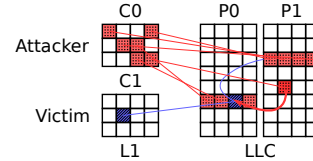

Fig. 5. When all cache blocks of the partially congruent eviction set (dot with red shadow) are cached in the LLC but failed to evict the target block (slash line with blue shadow), the eviction set become useless.

per second, which is ten times faster [41] than the Gem5 simulator [42] used in CEASER-S [17] and ScatterCache [18].

IV. DYNAMICALLY RANDOMIZED SKEWED CACHES ARE STILL VULNERABLE

By quantitatively analyzing the traces left by various attacks in the LLC, this section reveals the flawed hypotheses found in the latest randomized caches and uses experimental attacks to show that their defenses are indeed broken.

A. Flawed Hypothesis

Flawed hypothesis in CEASR-S: It is claimed in [17] that an attacker must find eviction sets composed of fully congruent cache blocks in order to evict the target address repeatedly. This is true for certain scenarios but not always true. In order to illustrate why this is not always true, let us first reflect on a "true" scenario, which is depicted in Fig. 5. An attacker wants to launch a cross-core attack from core zero (C0) to core one (C1). She has found an eviction set composed of seven fully congruent and one partially congruent cache blocks (dot with red shadow), which should have a 50% probability to evict the target address (slash line with blue shadow) in a skewed cache with two partitions. Assuming the attacker has successfully evicted the target address several times, she will fail eventually as the partially congruent cache block is randomly cached in the wrong partition (P1) as depicted in Fig. 5. The eviction set becomes useless afterwards.

From the attacker's viewpoint, the reason of the failure is the lack of enough self-conflicts to dislodge the misplaced partially congruent cache blocks during the re-accessing. To reuse an eviction set, attackers must find another way to purge the misplaced blocks from the LLC. Although one research [30] claims that it is still viable to construct covert channels by priming the LLC, this would cause significant amount of noise and noticeable performance degradation for normal prime-probe attacks. We argue that *attackers can accurately flush the eviction set using cache flush instructions* (such as `clflush` in x86-64), which is much cleaner and faster than priming the LLC. Our argument indicates that attacks using partially congruent eviction sets could enjoy big success. Note that using flush instructions here is fundamentally different with flush-reload attacks [35] where the target address being flushed is *shared* between the attacker and the victim. All blocks in an eviction set belong to the attacker's own address space.

We also argue this is a valid threat even for future computers because *the cache flush instructions will be here to stay.* We used to think Intel would eventually retire the `clflush`
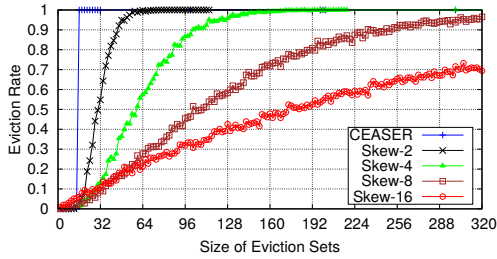
Fig. 6. The probability of evicting a target address (eviction rate) when a partially congruent eviction set is applied repeatedly. All LLCs are with the same size (1024-set 16-way). For skewed caches, the ways are divided into 2, 4, 8 and 16 partitions. Each result is averaged from 1000 independent experiments.

TABLE I
EXTRACTED FROM FIG. 6, THE ESTIMATED SIZES OF EVICTION SETS TO REACH THE EXPECTED EVICTION RATES (30%, 50% AND 80%).

| Cache Type	0.30	0.50	0.80
CEASER	16	16	16
Skew-2	25	30	39
Skew-4	45	59	87
Skew-8	68	108	190
Skew-16	90	172	400

instruction due to the threat of flush-related attacks [35], [43]–[45]. To our surprise, Intel not only continues to support `clflush` in their new architectures but also introduces new instructions with similar functionality, such as `clflushopt` and `CLWB`. As described in the ISA Reference [36], these instructions are added to reduce the performance overhead of accessing persistent memory [46]. Since persistent memory is a promising memory technology gradually adopted by almost all major computer architectures, cache flush instructions will remain in user land in the foreseeable future. Even if their usage is limited to the privileged software, prime-probe attacks from malicious kernels against other users/OSes [47], [48] or SGX enclaves [7] are still practical.

Assuming attackers (can use cache flush instructions to) flush their eviction sets after each probe, Fig. 6 reveals the probability of evicting a target address (eviction rate) when a partially congruent eviction set is applied repeatedly, which complies with the theoretical analysis done in ScatterCache (Fig. 5 in [18]): The eviction rate increases with the size of the partially congruent eviction set. *When enough addresses are collected, a partially congruent eviction set can be used just like a fully congruent one.*

Flawed hypotheses in ScatterCache: It is claimed that attackers must find eviction sets with 99% eviction rate and must use a separate prime set to prime the LLC after each probe (variant 1: single collision with eviction, Section 4.4 [18]). Both hypotheses are invalid. *An attacker can, and might be forced to, use eviction sets with low eviction rate.* As shown in Fig. 6, an eviction set with a lower eviction rate is much smaller than a set with a higher eviction rate. Since the time consumed in both finding and applying an eviction set is almost proportional to its size, reducing the required eviction rate can proportionally boost the attack frequency, which is essential for attacks demanding high temporal resolution [4],
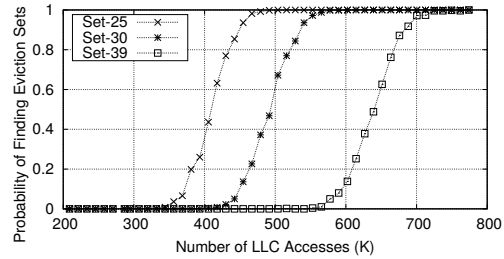
(a) Conflict testing (CT)
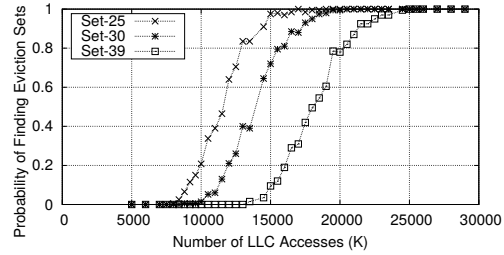
(b) Prime, prune and then test (PPT)

Fig. 7. The probability of finding eviction sets with 25, 30 and 39 partially congruent addresses in a skewed LLC (1024 sets, 16 ways, 2 partitions) within limited number of LLC accesses. Each result is averaged from 500 independent experiments.

[48]. Although utilizing eviction sets with low eviction rate does bring in high rate of false negative errors, these errors could be efficiently reduced by observing repeated victim events. Even worse, an attacker can force repeated replay of the victim's execution with microarchitectural replay attacks [49]. Furthermore, if remapping is adopted by ScatterCache, acquiring eviction sets with the 99% eviction rate might become unfeasible. Attackers would unavoidably resort to smaller eviction sets with lower eviction rates. For the hypothesis on priming the LLC, flushing the eviction set after each probe is much cleaner and faster than priming the LLC.

B. Broken Defense

Table I shows the number of partially congruent cache blocks needed to achieve a certain eviction rate. We consider 80% as a high eviction rate and 30% as a low but still threatening rate.[5] A defense is broken if it cannot stop attackers from finding eviction sets with the high eviction rate (80%), while it is relatively safe if attackers cannot acquire eviction sets with even the low eviction rate (30%). In the three fast search algorithms, only CT and PPT potentially work on randomized skewed caches. Let us consider a CEASER-S LLC with two partitions [17]. Fig. 7 demonstrates the probability of finding eviction sets with 25, 30 and 39 partially congruent addresses (corresponding to eviction rates of 30%, 50% and 80% respectively) using both CT and PPT. As shown by the result, although PPT is too long for any practical attacks (5M to 20M LLC accesses as shown in Fig. 7b), it is possible to find a small eviction set (30% eviction rate) in as low as 350K LLC accesses using CT (Fig. 7a), which is far less than the

---

[5]As researchers on the defense side, our intention is not to argue that 30% is a usable eviction rate in practical attacks but to use this low rate as a stress test to evaluate the strength of existing defenses.
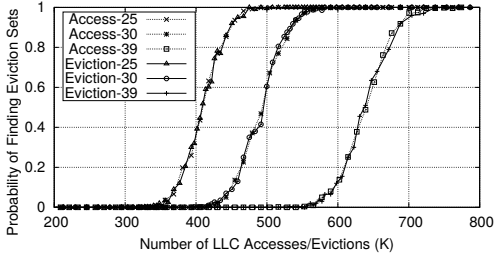
Fig. 8. The probability of finding eviction sets with 25, 30 and 39 addresses in a skewed LLC (1024 sets, 16 ways, 2 partitions) within limited number of LLC accesses or evictions. Each result is averaged from 500 independent experiments.
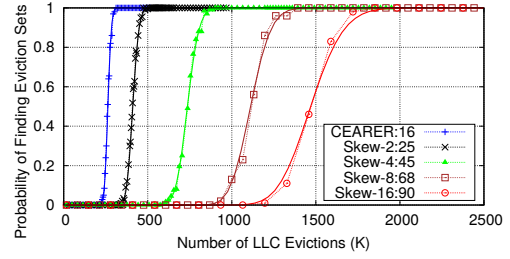

Fig. 9. The probability of finding the eviction sets with 30% evict rate in all types of LLCs (CEASER and skewed cache with 2 to 16 partitions). Experimental results (averaged from 500 independent experiments) are depicted in dots while the probability in theory (Equation 3) is drawn in solid lines.

preferred remap period of 1600K LLC accesses (100 accesses per cache block) [17]. In fact, 1600K LLC accesses are long enough to find partially congruent eviction sets with the high eviction rate. *CEASER-S is broken.*

The reasons for the failure of CEASER-S are twofold: One is its *neglect of the possibility of using partially congruent eviction sets*, which require much lower number of LLC accesses to find than fully congruent eviction sets. The other one is *measuring the remap period by LLC accesses while overlooking the filter effect of private caches*. Fig. 8 reveals the probability of using CT to find eviction sets within limited number of LLC evictions. Nearly all accesses observed by the LLC lead to misses (caused by visiting random addresses), indicating that all the re-accesses of the target address are filtered by private caches. The total number of cache accesses observed by the LLC is halved.

Rather than periodically remapping the LLC, ScatterCache proposes to use extra partitions to further increase the hardness in finding eviction sets and assumes the extra hardness is enough to thwart attacks [18]. ScatterCache estimates that roughly 275 partially congruent addresses are needed to achieve the 99% eviction rate in a randomized skewed cache with eight partitions and finding such an eviction set requires approximately 33.5M victim accesses (equivalent to 33.5M LLC evictions), which is an intimidating large number. Fig. 9 demonstrates the number of LLC evictions required to finding a partially congruent eviction set with 30% eviction rate in all types of randomized caches. If an attacker tries to find a small eviction set (68 addresses for 30% eviction rate) instead of the large one, the total number of LLC evictions is reduced to 1.1M, which is only 3.3% of what the large eviction set needs.[6] Even if an attacker requires the 99% eviction rate, she can choose to re-access and flush the small eviction set 20 times. The total number of LLC evictions is around 2.7K, which is just a negligible fraction (0.2%) of the LLC evictions needed for finding the small set.[7] *ScatterCache is still unsafe if eviction sets with low eviction rate are considered.*

---

[6]We believe that ScatterCache has over-estimated the number of victim accesses required. Instead of measuring the latency of re-accessing the random address, an attacker can measure the latency of accessing the target address (by the victim). This reduces the number of victim accesses to $n_{ways} \cdot 2^{b_{indices}} \cdot t$, which is $\frac{1}{n_{ways}}$ of what ScatterCache estimates.

[7]This way of achieving the 99% eviction rate works only for the evict+time attacks [2] though.

V. Fix the Randomized Skewed Caches

Randomized skewed caches are still vulnerable to attacks using partially congruent eviction sets found by the CT algorithm. Several ideas are proposed in this section to strengthen the defense while retaining performance.

A. Count Cache Evictions Rather than Accesses

As analyzed in Section IV-B, the failure of CEASER-S is partially because the remap period is measured by LLC accesses but half of the supposed accesses are filtered by private caches. We propose to measure the remap period by LLC evictions.

In the CT algorithm, when the target address is cached in the LLC, the probability that a newly fetched random address is cached in the same set and partition with the target address can be described as:

$$P = \frac{1}{SK} \tag{2}$$

where K is the number of partitions. Assuming the LRU replacement is used, the target address is evicted from the LLC only when $\frac{W}{K}$ evictions occurred in the same set and partition. Therefore, the probability of collecting a partially congruent eviction set of L addresses in E LLC evictions can be estimated as:

$$Prob(X \geq L) = 1 - \sum_{i=0}^{\frac{LW}{K}-1} \binom{E}{i} P^i (1-P)^{E-i} \tag{3}$$

As shown in Fig. 9, the theoretical probability calculated using Equation 3 matches with the experiment result. We can use this equation to estimate the time of finding an eviction set (30% eviction rate) within different remap periods in various randomized caches. Assuming the highest frequency of LLC evictions is 800 MHz, Table II details the time estimation. If we consider one year as a secure time margin for thwarting potential attacks, the chosen remap periods along with its time estimation are listed in the final column. To safely thwart attacks, the remap period of a two partitioned CEASER-S LLC must be reduced to 14 LLC evictions per cache block (by average). Even a skewed cache with 16 partitions has to be remapped very 39 LLC evictions per cache block.

Such short remap periods might be considered intolerable. However, *remapping by counting LLC evictions is much more efficient than counting LLC accesses because the LLC miss rate of normal applications is much lower than attacks*. In

TABLE II
ESTIMATED TIME FOR SUCCESSFULLY FINDING AN EVICTION SET (30%
EVICT RATE) WITHIN DIFFERENT REMAP PERIODS (AVERAGE NUMBER OF
EVICTIONS PER CACHE BLOCK).

Remap Period	100	50	20	10	Chosen Period
CEASER	0.3ms	0.3ms	0.4ms	3.7y	10 (3.7y)
Skew-2	0.5ms	0.5ms	0.32s	>100y	14 (204y)
Skew-4	0.9ms	0.9ms	>100y	>100y	25 (40y)
Skew-8	1.4ms	2.8s	>100y	>100y	35 (12y)
Skew-16	1.8ms	1.2h	>100y	>100y	39 (12y)

(a)

(b)
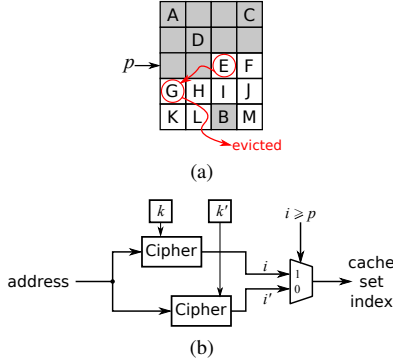
Fig. 10. The remap process of CEASER (CEASER-S). Cache sets are sequentially relocated as depicted in (a), where p points to the cache set that s currently be relocated. During the remap process, the cache set index for an incoming address is decided according to (b).

an ideal scenario, if the miss rate in the LLC is sufficiently low, remapping every 14 LLC evictions per cache block would trigger less remaps than remapping every 100 LLC accesses per cache block (preferred by CEASER-S). Our performance experiments in Section VII-A will analyze this effect in details.

B. Multi-Step Cache Relocation

Our experiment shows that 40% to 50% cache blocks in the LLC are evicted during the remap process, which is why frequent remaps can hurt performance significantly. Borrowing ideas from ZCache [25], we propose to use a multi-step relocation in the remap process, which reduces the eviction ratio to as low as 10%. This has two major benefits: One is the reduced performance loss as extra blocks remain in the LLC. The other one is the reduced damage from denial-of-service attacks [50]. An attacker can trigger frequent remaps by forcing a large amount of LLC accesses or evictions. Since the remap process cannot differentiate victim's data from the attacker's, the attacker can use remaps as a stealthy way to blindly evict victim's data. If the eviction ratio is reduced from 50% to just 10%, the return of such attacks becomes marginal.

In the remap process proposed by CEASER [16], cache sets are remapped sequentially as illustrated in Fig. 10a. Remapped blocks (shadowed in gray) are recorded in their metadata and a set-relocation pointer (p) always points to the cache set currently being remapped. The cache block E is currently being relocated to the next cache set chosen by the new key (k'). According to the replacement policy, G is evicted to make a room for E. By repeating this procedure, all blocks in the set are remapped and p moves to the next set. Since remapping is a gradual procedure, normal cache accesses might occur in
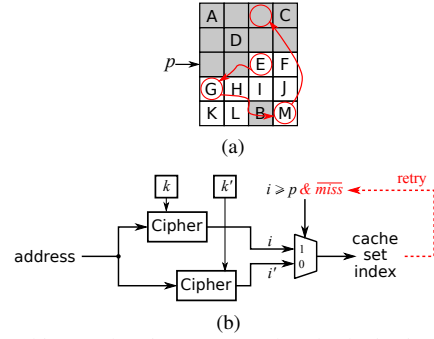

(a)

(b)

Fig. 11. A multi-step relocation process. When the destination of a relocation is taken by an unremapped cache block, this block is further relocated until an empty space is found as in (a) or a remapped cache block is found and evicted instead. The cache set index for an incoming address is decided according to (b). When using the old cache set index i results in a miss, retry using the new index i'.
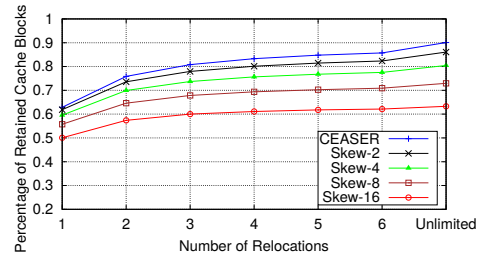

Fig. 12. The percentage of cache blocks retained during remapping by applying limited number of relocations. The maximum retaining percentage is achieved when 'infinite' trials of relocation are applied until a remapped cache block is found as the replacement (and evicted). Each result is averaged from 100 independent experiments.

parallel. Fig. 10b illustrates how the cache set index is decided. The old cache set index i and the new one i' are produced simultaneously by two independent ciphers using the old key k and the new key k' respectively. When $i \geq p$, denoting the cache set is not remapped yet, the old index i is used. Otherwise, the new index i' should be used.

The problem of the original remap process is the eviction of G. Whenever the target cache set for a relocated block is full, a cache block is evicted, which leads to a high number of evictions. Such evictions might be avoidable. The relocation procedure can keep on relocating the blocks to be evicted, such as G, in a chain until either a free space is found, as shown in Fig. 11a, or a remapped block is to be evicted. Note that using multi-step relocation does not increases the total number of relocated blocks. Once a block is relocated, it is recorded as remapped and will not be relocated again. The total number of blocks to be relocated is equal to the number of blocks in the LLC in both methods. As shown in Fig. 12, by allowing unlimited number of relocations, the percentage of blocks retained in the LLC grows. Randomized set-associative caches (CEASER) benefits the most as the percentage increases from 63% to 90%. The boost for randomized skewed caches drops gradually with the number of partitions.

The calculation of the cache set index needs a small change to support the multi-step relocation. As shown in Fig. 11b, when $i \geq p$, the old cache set index i should be used in the same way as in the original CEASER. However, if it results in
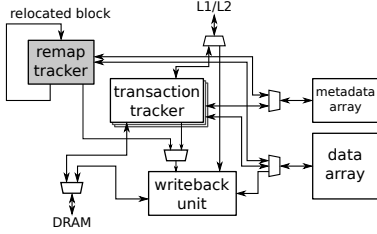
Fig. 13. Support multi-step relocation in the LLC of the Rocket-Chip.

a miss, the new cache set index i' should be used in a retrial as the block might have already be relocated. Since reading the metadata array and checking cache hit typically finish in one or two cycles, and retrials occur only occasionally during the relatively short remap process, the performance impact is trivial considering the significantly reduced eviction ratio.

Supporting multi-step relocation in the actual cache hardware should be straightforward. Fig. 13 demonstrates the internal structure of the LLC (L2) used in the Rocket-Chip SoC [51] (available from lowRISC v0.4 [52]), which is a widely adopted open processor design taped out for tens of times. To support multiple concurrent cache transactions initiated from the multiple L1 caches, each cache slice implements multiple transaction trackers sharing the same accesses to the metadata array, the data array and the writeback unit. When a transaction arrives without any race condition, a free tracker is allocated to serve it. To support remaps, a special remap tracker is added. During a remap, it tracks the set-relocation pointer and gradually relocates all cache blocks. In the case of multi-step relocation, when an unremapped cache block is swapped out, the remap tracker throws it back to itself as a prioritized writeback transaction. As long as unremapped blocks are swapped out, they are continuously relocated until a free space is found or a remapped block is swapped out instead (which is then evicted). This recursive procedure effectively implements the unlimited steps of relocation. The only hardware changes necessary to support multi-step relocation include adding an incoming port to the remap tracker and modifying its state machine accordingly.

VI. USE NORMAL INSTEAD OF SKEWED CACHES

Instead of advocating the use of randomized skewed cache like CEASER-S and ScatterCache, we argue that randomized set-associative caches can be sufficiently strengthened and possess a better chance to be actually adopted in commercial processors than their skewed counterparts. By a literature research with our best effort, we cautiously believe that skewed caches [53], [54] have not been adopted in the LLCs of any commercially available modern processors. Promoting them purely for security benefits might be a hard sale.

A. Issues with Skewed Caches

We agree that skewed caches can improve cache efficiency by reducing conflicts [53] and are natural candidates for compressed caches [54]. However, it seems that they are not yet embraced by the industry. One potential reason has already been pointed out by CEASER-S [17]: The benefit of skewed

caches diminishes when the cache associativity increases. As caches in modern processors are typically highly associative, the marginal gain in performance might not justify the extra hardware cost. For this reason, CEASER-S chooses to use only two partitions. Some of our experiments show excessive skewing (too many partitions) actually hurt performance as it reduces the efficiency of the LRU replacement. One example is already revealed in Fig. 12. The benefit of multi-step relocation drops with the increasing number of partitions.

From our own perspective in hardware designs, we also believe that skewed cache significantly complicates the design of modern LLCs which typically serve multiple cache transactions in parallel. Taking the LLC design of the Rocket-Chip (as shown in Fig. 13) as an example, before a tracker can accept a transaction, the LLC must ensure that this transaction would not conflict with the others currently being served. This typically means that no transaction served simultaneously should access the same cache set with others. Otherwise, one of the conflicting transactions should be blocked before it is accepted by a tracker (race condition). This is not a serious issue for set-associative caches as the cache set index of an incoming transaction can be calculated beforehand and compared with the indices of all active trackers simultaneously in a single cycle. For a skewed cache with K partitions and T trackers, the incoming transaction might access anyone of the K possible cache sets and it is not decided until the transaction hits on a set or a target set is chosen for replacement. In the worst scenario, $K \cdot TK$ parallel comparisons (rather than T for the set-associative cache) are required to check potential conflicts for an incoming transaction. Besides the obvious hardware cost in doing so, this significantly increases the probability of blocking an incoming transaction due to a conflict that might not occur eventually. It then prolongs the cache accessing latency.

Therefore, we would like to investigate potential means to strengthen the randomized set-associative caches.

B. Remap When Under Attack

Although randomized skewed caches are vulnerable only to the CT algorithm, randomized set-associative caches are vulnerable to all the three search algorithms introduced in Section II-D. To thwart the CT algorithm, a 1024-set 16-way CEASER LLC has to remap every 10 LLC evictions per cache block according to Table II, which allows for a total of 160K LLC evictions between remaps. However, our experiments show that the numbers of LLC evictions (accesses) needed for finding an eviction set are around 40.8K (168K) using the PPT algorithm and 81.3K (532K) using the GE algorithm. Both are valid threats.

Instead of shrinking the already short remap period, we propose to trigger a remap when an attack using the two algorithms is detected because both of them leave a unique pattern in the cache set distribution of evictions. Let us first consider the PPT algorithm. By periodically sampling the number of accesses and evictions occurred on individual cache sets during two consecutive attacks, Fig. 14a and 14b reveal

(a) Cache set distribution of accesses

(b) Cache set distribution of evictions

(c) Standardized distribution of accesses

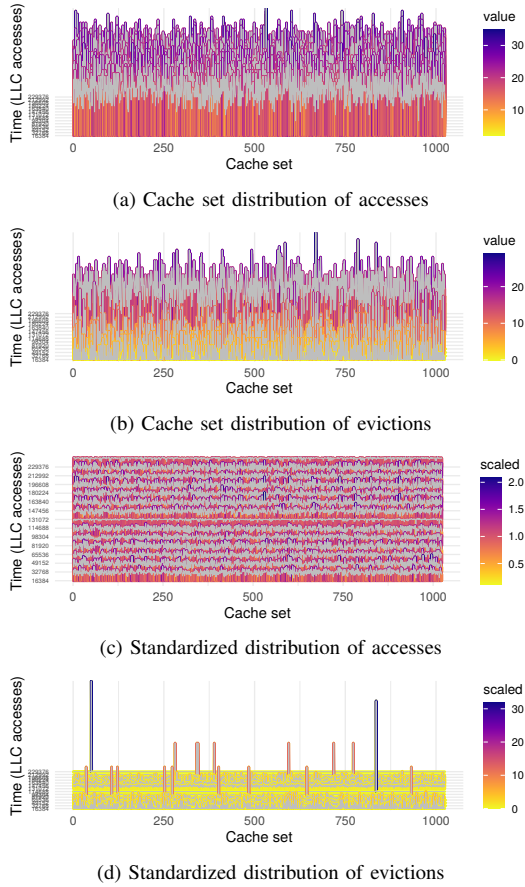(d) Standardized distribution of evictions

Fig. 14. Detect the PPT attack by analyzing the cache set distributions of accesses and evictions. The x-axis denotes the cache set index while the y-axis denotes simulation time measured in LLC accesses. The height and color of each data point denotes its value. One distribution is sampled every 16K LLC accesses. The standardization in (c) and (d) is done using the Z-Score method [55], [56].

the distribution of LLC accesses and evictions over all cache sets. Both distributions seem totally random. However, if we apply a Z-Score [55] standardization on the distributions, we can see two clear peaks in the standardized eviction distribution (Fig. 14d), although it is still random for the standardized access distribution (Fig. 14c). The two peaks appear only in the test phase of the PPT algorithm and would not show up in normal applications. After the prune phase, all blocks in the prime set are simultaneously cached in the LLC. If there is any eviction in the test phase, it must occur on the target cache set. As a result, the score of the target cache set reaches the maximum (32 for a 1024-set LLC) while it is zero on all other sets.

The GE algorithm presents a similar pattern as demonstrated by Fig. 15. Scores are small and randomly distributed at the early stage of the two simulated attacks but converge on a single cache set when the large eviction set is finally condensed into a minimal one.

Since PPT spends less time on testing the prime set than GE on trimming the eviction set, PPT is harder to detect than GE. We thus finalize our detector against PPT and it should work on GE as well. We start from a non-centered variant of
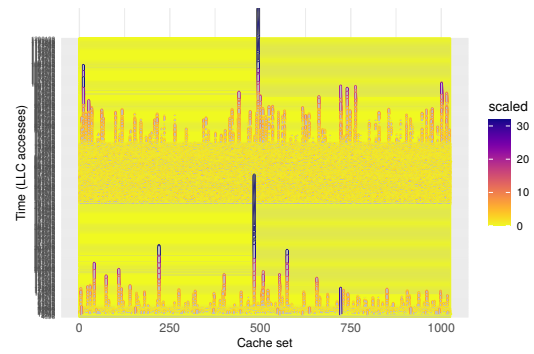

Fig. 15. Detect the *group elimination* attack by analyzing the standardized cache set distribution of evictions.

the Z-Score standardization [56] to avoid negative scores:

$$z_i = \frac{e_i}{\sqrt{\frac{\sum e^2}{S-1}}} \tag{4}$$

where e_i is the number of evictions on cache set i and z_i is the calculated score for cache set i. The score of the target cache set approaches to the maximum of $\sqrt{S}$ in an ideal attack. However, reporting an attack whenever a maximum score is detected leads to false positive errors. When the LLC miss rate is extremely low during normal operation, there might be only one eviction during the whole sample period, which also results in a maximum score. To avoid such errors, we introduce the number of evictions into Equation 4 as weight:

$$wz_i = (e_i - \bar{e}) \cdot z_i \tag{5}$$

where wz_i is the weighted score. Since an eviction set requires at least W addresses, the weighted score of the target cache set approaches to $W \cdot \sqrt{S}$ during the test phase of an ideal PPT attack.

An attacker can avoid detection if the detection threshold is simply set to $W \cdot \sqrt{S}$. She can hide her trace by spreading the test phase over multiple sample periods. In the extreme case, the attacker can collect only one congruent address in each round of PPT attack, which effectively caps the weighted score to $\sqrt{S}$.[8] To detect such behavior and improve the robustness of the detector, we apply an exponential moving average (EMA) [57], [58] on the weighted score:

$$az_i(t) = (1 - \alpha) \cdot az_i(t-1) + \alpha \cdot wz_i(t) \tag{6}$$

where α is a discount factor used to calculate $az_i(t)$, the EMA of wz_i at sample t. The use of EMA allows the detector to examine the history of wz because az is an infinite impulse response of wz. For normal applications, wz should be a zero-centered small number for all cache sets, but unavoidably raises to at least $\sqrt{S}$ for the target cache set during the test phase of PPT. By using a small α, the az of the target cache set effectively accumulates the large wz over the history, which makes it sufficiently significant for detection. We set the discount factor α to $\frac{1}{32}$ by a heuristic analysis.

When the az of a certain cache set reaches a threshold ($az \geq th$), the detector triggers a remap. The value of th

---

[8]In practice, the number of rounds is limited because remaps will be triggered due to the accumulation of LLC evictions.
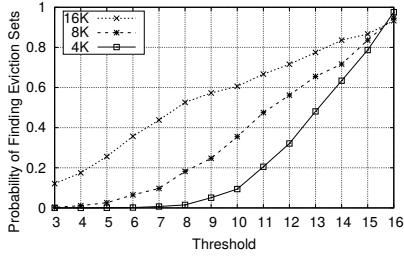
Fig. 16. The probability of finding an eviction set under active detection. Three sample periods are chosen: 16K, 8K and 4K LLC accesses. Each result is averaged from at least 200 independent experiments.
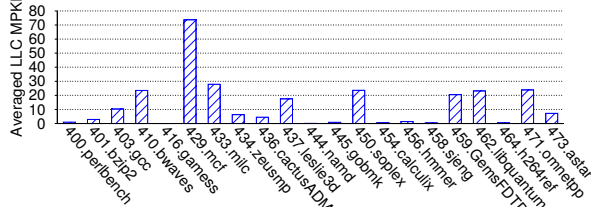

Fig. 17. MPKI of SPEC CPU 2006 benchmark cases using a static 1024-set 16-way CEASE LLC.

is crucial to the speed and the correctness of the detector. The detector might leave a small window for a quick attack if a remap is late due to a large th. However, normal programs might trigger remaps if th is too small. To choose a proper th, we run PPT attacks detected by different combinations of threshold and sample period (4K, 8K and 16K LLC accesses). As shown in Fig. 16, sampling every 4K LLC accesses and triggering a remap whenever $az \geq 5$ is enough to reduce the probability of finding eviction set to almost nil. Although not shown in the paper, we have verified that GE attacks cannot escape detection with the same parameters.

VII. PERFORMANCE

As the performance overhead of randomized caches has been shown to be acceptable [16]–[18], we analyze only the performance impact of the newly proposed ideas.

A. Impact on Normal Applications

The SPEC CPU 2006 benchmark suite [38] is used to evaluate the impact on normal applications. Similar to Scatter-Cache, performance results are measured without concurrent processes [18]. As described in Section III-B, we use a modified Spike simulator [40] as the evaluation platform. A processing core has two private L1 data and instruction caches (16KB, 64-set, 8-way, 64B cache block). A 1024-set 16-way L2 cache is used as the LLC where all randomized caches are implemented. All cache levels use the LRU replacement. Thanks to the fast simulation speed of Spike, we are able to run 100 billion instructions for each benchmark case, which is 100 and 400 times of the instructions simulated in CEASER[16] and ScatterCache [18]. Fig. 17 shows the number of misses per K instructions (MPKI) using a static CEASE LLC, which is used as the baseline for other performance results. The MPKI figures match with the result provided in CEASER[16].

Fig. 18 demonstrates the performance overhead of different remap strategies on a CEASE LLC. The remap period is
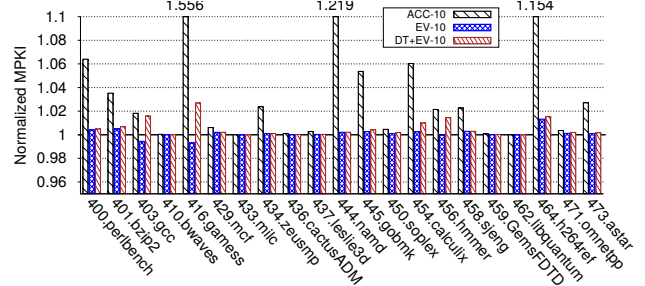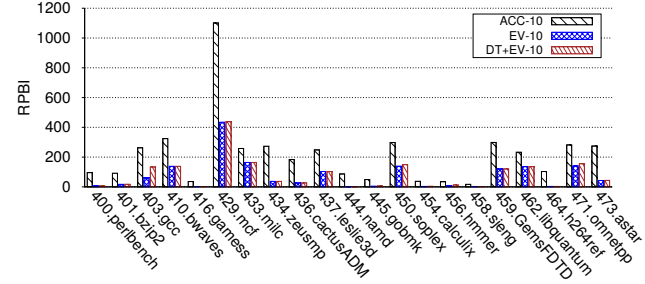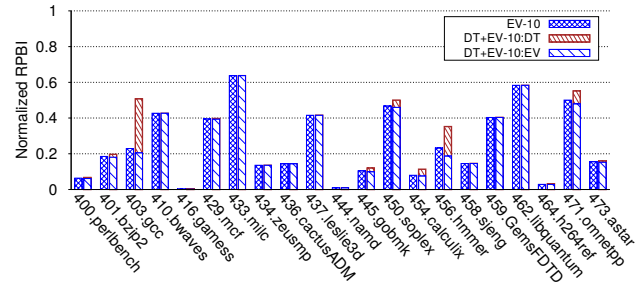

Fig. 18. Normalized MPKI of SPEC CPU 2006 benchmark cases running on a CEASER LLC. ACC-10: remap every 10 accesses per cache block; EV-10: remap every 10 evictions per cache block; DT: attack detection. The static CEASE is used as the baseline.

(a) Remaps per billion instructions (RPBI)

(b) Normalized RPBI using ACC-10 as the baseline

Fig. 19. (a) Remaps per billion instructions (RPBI) of SPEC CPU 2006 benchmark cases running on a CEASER LLC. In (b), DT and EV denote the remaps triggered by attack detection and reaching remap period respectively.

increased to 10 accesses/evictions per cache block to thwart the CT attack. The average overhead is 0.61%, 0.077% and 0.19% for ACC-10 (remapping by accesses), EV-10 (remapping by evictions) and DT+EV-10 (EV-10 plus attack detection) respectively. Measuring the remap period by evictions rather than accesses reduces MPKI by 69% with attack detection or 87% without.

Such significant performance boost comes from two reasons: One is the reduced number of remaps as shown in Fig. 19a. The average reduction is 72% with attack detection or 74% without. The other one is the reduced impact for each remap. To explain this effect, Fig. 20 depicts the run-time MPKI and miss rate curves extracted from a representative window of the 403.gcc (expr2) benchmark case. Note that for the LLC remapped by accesses, the remap period is increased to 100 accesses per cache block to avoid excessive remaps (a totally pink colored background). Remapping by accesses inclines to remap when both MPKI and miss rate are low, such as the time segments of (11–13), (19–20), and (26–30) billion instructions, while there is nearly no remaps for the same

(a) MPKI (remap period: 100 accesses per cache line)

(b) Miss rate (remap period: 100 accesses per cache line)

(c) MPKI (remap period: 10 evictions per cache line)

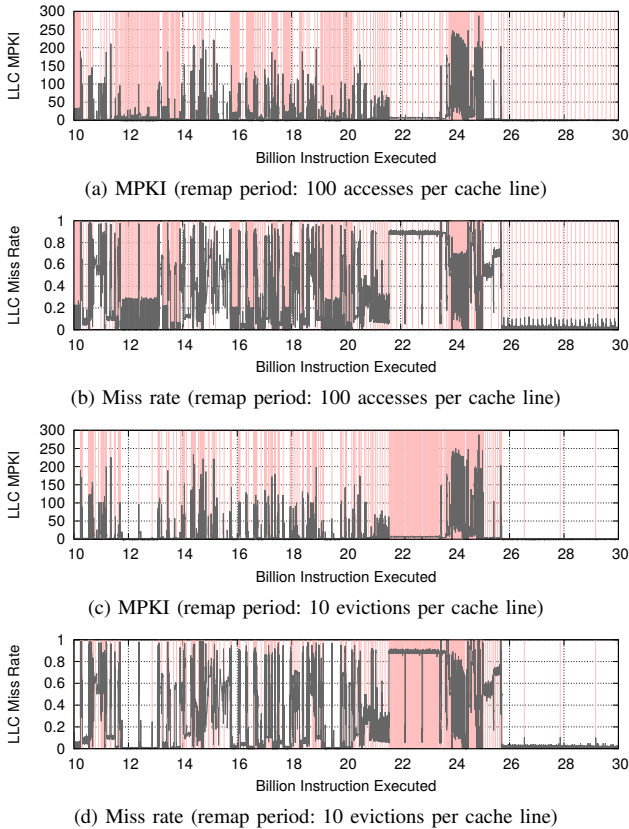(d) Miss rate (remap period: 10 evictions per cache line)

Fig. 20. Compare the triggered remaps of running the SPEC CPU 2006 case 403.gcc (expr2). Each remap is depicted as a vertical pink line in the background.

period in the case of remapping by evictions. What is worse, these remaps lead to unnecessary block evictions which in turn raise the miss rate. On the contrary, remapping by evictions inclines to remap when the miss rate is high, such as the time segments of (22–23) and (25–26). During these segments, the utilization efficiency of the LLC is already reduced by the high miss rate. The performance impact of the unnecessarily evicted blocks in each remap is thus weakened.

The cost of enabling attack detection in CEASER is relatively small compared with the performance boost from remapping by evictions. Fig. 19b provides a detailed breakdown the remaps when attack detection is enabled. On average, attack detection introduce 16% extra remaps (false positive errors) in addition to the remaps triggered by evictions but together they cause only 28% of the remaps if remapping by accesses is used. For most benchmark cases, including the memory intensive cases [59] such as 429.mcf, 462.libquantum, 464.h264ref and 471.omnetpp, the rate of remaps triggered by detection (related to the rate of false positive errors) is tiny. Only cases like 403.gcc and 456.hmmer have high rates of errors. Since the absolute number of remaps for 456.hmmer is extremely low (RPBI ≈ 8 in Fig. 19a), the high error rate does not actually hurt performance. As for 403.gcc, the absolute number of MPKI increased from 10.25 to 10.47, leading to an extra 2.1% performance loss. Considering the MPKI is relatively low, this 2.1% loss should be tolerable.
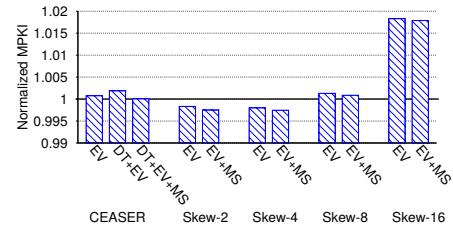

Fig. 21. Normalized MPKI of SPEC CPU 2006 benchmark cases using the static CEASE as the baseline. EV: remapping by evictions; DT: attack detection; MS: multi-step relocation.

Fig. 21 shows the normalized MPKI of all types of randomized caches using the static CEASE as the baseline. In general, skewed caches with a moderate number of partitions indeed reduce MPKI but such reduction is marginal (less than 0.5%). When more than eight partitions are used, MPKI begins to rise and hurt performance. This is why we believe randomized set-associate caches (CEASER) should be used if they are safely strengthened. Utilizing the multi-step relocation (MS) reduces MPKI roughly by 0.05% and the skewed cache with only two partitions benefits the most (0.08%). This complies with our estimation in Fig. 12. For CEASER LLCs, periodically remapping by evictions (EV) introduces 0.08% extra MPKI and enabling attack detection (DT) adds another 0.11%, but adopting the multi-step relocation (MS) would reduce the overhead back to a trivial 0.007%. This result shows that, *when all the newly proposed ideas are applied to CEASER (DT+EV+MS), the randomized set-associate cache is safe enough without significant performance loss.*

B. Logic and Memory Overhead

The memory overhead of randomized caches has been analyzed in [16], [17]; therefore, we estimate only the extra cost using the new ideas. We use a single core Rocket-Chip (lowRISC ver. 0.4) [52] as the base. Using the same configuration as in Spike, the LLC (L2 cache) consumes around 22% logic and 99% SRAM of the processor (without outer AXI buses and devices). To support remaps, a remap tracker is added to the LLC which originally has two acquire (access) trackers and one release (writeback) tracker. The extra area overhead would be round 7.6% logic of the processor (34% logic of the LLC). This overhead is relatively high but unavoidable. Remapping by evictions rather than accesses introduces no area overhead. The overhead of supporting multi-step relocation is also marginal because the only changes required are adding a port to the remap tracker and modifying its state machine. To estimate the overhead of attack detection, we made a prototype of the detector in hardware. The hardware detector finishes each round of detection in 2K cycles (less than the sample period of 4K LLC accesses). By shrinking the precision of the intermediate results and reducing multiplier/divider to adder/shifter, the detection error is within 5% compared with the software implementation while the area overhead (after place and route) is around 0.8% logic and 0.4% SRAM of the processor (3.5% logic and 0.4% SRAM of the LLC), both of which are marginal.

VIII. Discussion

New cache designs: Since the introduction of randomized skewed caches, two new designs [19], [21] have been proposed and both of them promote the use of set-associative caches. The two level dynamic randomization (TLDR) [19] tries to strengthen CEASER by another layer of randomization using an indirection table (iTable). An address is first randomly mapped to an iTable entry and then the entry is mapped to a random cache set. It is claimed that the extra iTable provides higher level of randomness than randomized skewed caches and gradually remapping iTable entries reduces the remap-related performance loss. PhantomCache [21] proposes to place an incoming cache block in one of the multiple randomly selected cache sets rather than partitions as in skewed caches. This increases the level of randomness and allows the use of LRU for the whole cache set. Both designs can safely defeat the GE attack but their effectiveness against CT and PPT attacks needs further investigation. Finally Doblas [20] extends the cache randomization from LLC to the L1 caches by using simple randomization functions.

Performance evaluation: The performance results of all existing cache randomization designs come from various Gem5 simulations [16]–[18], [21], [42], whose slowness limited the total number of instructions that can be simulated in reasonable time, which further constrains the coverage on representative workloads [60], [61]. Our choice of using the fast (event-driven and timeless) Spike simulation allows us to boost the number of simulated instructions by $100 \sim 400$ times, which significantly increases the coverage on representative workloads but limits the performance evaluation in miss rate only, leaving the overhead on CPU execution time unstated. We believe this is a reasonable trade-off. After the encryption algorithm used in CEASER has been found problematic [34], there is no consensus on which encryption algorithm should be adopted but this choice has significant impact on the extra delays introduced by cache randomization. It is still an open challenge to choose a strong and fast encryption algorithm for randomized caches. As a result, the estimation on CPU execution time is already inaccurate for any comparison between designs even if the slowest Gem5 OoO model [42] is used. Cache miss rate is the only frequently used and unbiased metric available.

Attack detection: Run-time detection of cache side-channel attacks using the existing performance counters (pfc) [58], [62]–[65] has shown to be effective to detect persistent attacks by software. Some of them adopt machine learning to increase the detection accuracy [62], [64] but they are always constrained by the limited information available from pfc. The concentration of cache accesses on the target cache sets during the exploitation phase has long been discovered [22], [58], [66]. Recent hardware detectors with set level granularity begins to utilize this pattern [67], [68]. Most of them exploit the cyclic pattern between an attacker and her victim [65], [67], [68]. To the best of our knowledge, we are the first to exploit the unique set distribution of cache evictions during the search for eviction sets in randomized caches; however,

whether an attacker can evade detection by slowing down and hiding behind the background noise is still an open question.

New attacks: Purnal *et al.* improve the original PPT attack [17] by introducing the prune phase and correctly points out it is possible to use partially congruent eviction set to launch covert channel attacks on ScatterCache [30]. Our simulation and analysis on PPT are based on Purnal's work but with our own optimized prune method as it is not clearly described in [30]. Our experiments show that PPT attacks would fail on randomized skewed caches because the accumulated number of LLC evictions always surpasses the proposed remap period. Very recently, Purnal *et al.* have further improved PPT by optimizing the pruning and profiling method [69]. We need to evaluate these new optimisations and decide whether attack detection is also needed for randomized skewed caches. In another concurrent work [70], Bourgeat *et al.* analyze the end-to-end security impact of utilizing partially congruent eviction sets. It is found that attackers may decide to use eviction sets with lower eviction rate in return for better chance of information leakage and reducing the remap period can signifcantly increase the cost of attacks. These findings complement well with this paper.

IX. Conclusion

We have newly discovered several problems with the hypotheses and implementations in the latest randomized skewed caches: The possibility of using cache flush instructions in conflict-based attacks has been overlooked. The concept of minimal eviction set no longer applies to randomized skewed caches. Attackers do not have to use eviction sets with 99% eviction rate. Measuring the remap period by LLC accesses is flawed. As a result, existing randomized skewed caches are still vulnerable to conflict-based cache side-channel attacks.

We proposed several defense ideas to fix the newly discovered problems: Measure the remap period by LLC evictions rather than accesses while further reduce the period. Adopt ZCache-like multi-step relocation to minimize the number of cache blocks evicted during the remap process. Our experiments show that all the newly discovered vulnerabilities are fixed within the current performance budget. We also claim that randomized set-associative cache can be sufficiently strengthened with reasonable overhead using a simple attack detection mechanism. Compared with randomized skewed caches, randomized set-associative caches are better candidates for future commercial processors.

REFERENCES

[1] C. Percival, "Cache missing for fun and profit," 2005.
[2] D. A. Osvik, A. Shamir, and E. Tromer, "Cache attacks and counter-measures: The case of AES," in *Topics in Cryptology – CT-RSA 2006*. Springer, 2006, pp. 1–20.
[3] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, May 2015.
[4] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$A: A shared cache attack that works across cores and defies VM sandboxing – and its application to AES," in *Proceedings of the Symposium on Security and Privacy (S&P)*. IEEE, May 2015, pp. 591–604.
[5] D. Genkin, L. Pachmanov, E. Tromer, and Y. Yarom, "Drive-by key-extraction cache attacks from portable code," in *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS)*. Springer, 2018, pp. 83–102.
[6] D. Gruss, C. Maurice, and S. Mangard, "Rowhammer.js: A remote software-induced fault attack in JavaScript," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*. Springer International Publishing, 2016, pp. 300–321.
[7] M. Hähnel, W. Cui, and M. Peinado, "High-resolution side channels for untrusted operating systems," in *Proceedings of the USENIX Annual Technical Conference (UTC)*. USENIX Association, 2017, pp. 299–312.
[8] D. Page, "Partitioned cache architecture as a side-channel defence mechanism," *IACR Cryptology ePrint Archive*, vol. 2005, 2005, http://eprint.iacr.org/2005/280.
[9] T. Kim, M. Peinado, and G. Mainar-Ruiz, "STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud," in *Proceedings of the USENIX Security Symposium (Security)*. USENIX Association, 2012, pp. 189–204.
[10] F. Liu, Q. Ge, Y. Yarom, F. McKeen, C. V. Rozas, G. Heiser, and R. B. Lee, "CATalyst: Defeating last-level cache side channel attacks in cloud computing," in *Proceedings of the International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, 2016, pp. 406–418.
[11] D. Gruss, C. Maurice, A. Fogh, M. Lipp, and S. Mangard, "Prefetch side-channel attacks: Bypassing SMAP and kernel ASLR," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2016, pp. 368–379.
[12] N. El-Sayed, A. Mukkara, P. Tsai, H. Kasture, X. Ma, and D. Sánchez, "KPart: A hybrid cache partitioning-sharing technique for commodity multicores," in *Proceedings of the IEEE International Symposium on High Performance Computer Architecture (HPCA)*. IEEE, February 2018, pp. 104–117.
[13] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*. ACM, June 2007, pp. 494–505.
[14] ——, "A novel cache architecture with enhanced performance and security," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2008, pp. 83–93.
[15] F. Liu and R. B. Lee, "Random fill cache architecture," in *Proceedings of the International Symposium on Microarchitecture (Micro)*. IEEE, 2014, pp. 203–215.
[16] M. K. Qureshi, "CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2018, pp. 775–787.
[17] ——, "New attacks and defense for encrypted-address cache," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*. ACM, 2019, pp. 360–371.
[18] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "ScatterCache: Thwarting cache attacks via cache set randomization," in *Proceedings of the USENIX Security Symposium (Security)*. USENIX Association, 2019, pp. 675–692.
[19] K. Ramkrishnan, A. Zhai, S. McCamant, and P. C. Yew, "New attacks and defenses for randomized caches," *arXiv: abs/1909.12302*, 2019, https://arxiv.org/abs/1909.12302v1.
[20] M. Doblas, I.-V. Kostalabros, M. Moretó, and C. Hernández, "Enabling hardware randomization across the cache hierarchy in Linux-class processor," in *Proceedings of the Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2020, p. 7.

[21] Q. Tan, Z. Zeng, K. Bu, and K. Ren, "PhantomCache: Obfuscating cache conflicts with localized randomization," in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*. Internet Society, 2020.
[22] Y. Oren, V. P. Kemerlis, S. Sethumadhavan, and A. D. Keromytis, "The spy in the sandbox," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2015.
[23] P. Vila, B. Köpf, and J. Morales, "Theory and practice of finding eviction sets," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, 2019.
[24] W. Song and P. Liu, "Dynamically finding minimal eviction sets can be quicker than you think for side-channel attacks against the LLC," in *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*. USENIX Association, 2019, pp. 427–442.
[25] D. Sánchez and C. Kozyrakis, "The ZCache: Decoupling ways and associativity," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2010, pp. 187–198.
[26] M. Cekleov and M. Dubois, "Virtual-address caches. Part 1: Problems and solutions in uniprocessors," *IEEE Micro*, vol. 17, no. 5, pp. 64–71, 1997.
[27] C. Berg, "PLRU cache domino effects," in *Proceedings of the International Workshop on Worst-Case Execution Time Analysis (WCET)*. Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2006, http://drops.dagstuhl.de/opus/volltexte/2006/672.
[28] M. Yan, R. Sprabery, B. Gopireddy, C. W. Fletcher, R. H. Campbell, and J. Torrellas, "Attack directories, not caches: Side-channel attacks in a non-inclusive world," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*. IEEE, May 2019, pp. 888–904.
[29] M. Yan, B. Gopireddy, T. Shull, and J. Torrellas, "Secure hierarchy-aware cache replacement policy (SHARP): Defending against cache-based side channel attacks," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*. ACM, 2017, pp. 347–360.
[30] A. Purnal and I. Verbauwhede, "Advanced profiling for probabilistic Prime+Probe attacks and covert channels in ScatterCache," *arXiv: abs/1908.03383*, 2019, https://arxiv.org/abs/1908.03383v1.
[31] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, November 2009, pp. 199–212.
[32] D. Cock, Q. Ge, T. C. Murray, and G. Heiser, "The last mile: An empirical study of timing channels on seL4," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, November 2014, pp. 570–581.
[33] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust website fingerprinting through the cache occupancy channel," in *Proceedings of the USENIX Security Symposium (Security)*. USENIX Association, August 2019, pp. 639–656.
[34] R. Bodduna, V. Ganesan, P. SLPSK, K. Veezhinathan, and C. Rebeiro, "BRUTUS: Refuting the security claims of the cache timing randomization countermeasure proposed in CEASER," *IEEE Computer Architecture Letters*, vol. 19, no. 1, pp. 9–12, January 2020.
[35] Y. Yarom and K. Falkner, "FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack," in *Proceedings of the USENIX Security Symposium (Security)*. USENIX Association, 2014, pp. 719–732.
[36] Intel, "Intel® architecture instruction set extensions programming reference," Intel, Tech. Rep. 319433-023, August 2015, https://software.intel.com/sites/default/files/managed/07/b7/319433-023.pdf.
[37] X. Zhang, Y. Xiao, and Y. Zhang, "Return-oriented flush-reload side channels on ARM and their implications for Android devices," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2016, p. 858870.
[38] J. L. Henning, "SPEC CPU2006 benchmark descriptions," *SIGARCH Computer Architecture News*, vol. 34, no. 4, pp. 1–17, 2006.
[39] *The RISC-V Instruction Set Manual. Volume I: Unprivileged ISA Version 20191213*, Editors Andrew Waterman and Krste Asanović, RISC-V Foundation, December 2019, https://riscv.org/specifications/isa-spec-pdf/.
[40] A. Waterman, T. Newsome, C.-M. Chao, Y. Lee, S. Beamer, and others, "Spike RISC-V ISA simulator," https://github.com/riscv/riscv-isa-sim.

[41] T. Ta, L. Cheng, and C. Batten, "Simulating multi-core RISC-V systems in gem5," in *Proceedings of the Workshop on Computer Architecture Research with RISC-V (CARRV)*, 2018, p. 7.

[42] N. L. Binkert, B. M. Beckmann, G. Black, S. K. Reinhardt, A. G. Saidi, A. Basu, J. Hestness, D. Hower, T. Krishna, S. Sardashti, R. Sen, K. Sewell, M. Shoaib, N. Vaish, M. D. Hill, and D. A. Wood, "The gem5 simulator," *SIGARCH Computer Architecture News*, vol. 39, no. 2, pp. 1–7, 2011.

[43] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+flush: A fast and stealthy cache attack," in *Proceedings of the International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*.   Springer, 2016, pp. 279–299.

[44] M. Lipp, M. Schwarz, D. Gruss, T. Prescher, W. Haas, A. Fogh, J. Horn, S. Mangard, P. Kocher, D. Genkin, Y. Yarom, and M. Hamburg, "Meltdown: Reading kernel memory from user space," in *Proceedings of the USENIX Security Symposium (Security)*.   USENIX Association, 2018, pp. 973–990.

[45] P. Kocher, D. Genkin, D. Gruss, W. Haas, M. Hamburg, M. Lipp, S. Mangard, T. Prescher, M. Schwarz, and Y. Yarom, "Spectre attacks: Exploiting speculative execution," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.   IEEE, May 2019, pp. 19–37.

[46] A. Rudoff, "Persistent memory programming," *;login:*, vol. 42, no. 2, pp. 34–40, 2017.

[47] N. Zhang, K. Sun, D. Shands, W. Lou, and Y. T. Hou, "TruSpy: Cache side-channel information leakage from the secure world on ARM devices," *IACR Cryptology ePrint Archive*, vol. 2016, p. 16, 2016, http://eprint.iacr.org/2016/980.

[48] M. S. Inci, B. Gulmezoglu, G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cache attacks enable bulk key recovery on the cloud," in *Proceedings of the Conference on Cryptographic Hardware and Embedded Systems (CHES)*.   Springer, 2016, pp. 368–388.

[49] D. Skarlatos, M. Yan, B. Gopireddy, R. Sprabery, J. Torrellas, and C. W. Fletcher, "Microscope: Enabling microarchitectural replay attacks," *IEEE Micro*, vol. 40, no. 3, pp. 91–98, 2020.

[50] J. Kong, O. Aciiçmez, J. Seifert, and H. Zhou, "Deconstructing new cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the ACM Workshop on Computer Security Architecture (CSAW)*.   ACM, October 2008, pp. 25–34.

[51] K. Asanović, R. Avizienis, J. Bachrach, S. Beamer, D. Biancolin, C. Celio, H. Cook, D. Dabbelt, J. Hauser, A. Izraelevitz, S. Karandikar, B. Keller, D. Kim, J. Koenig, Y. Lee, E. Love, M. Maas, A. Magyar, H. Mao, M. Moreto, A. Ou, D. A. Patterson, B. Richards, C. Schmidt, S. Twigg, H. Vo, and A. Waterman, "The Rocket chip generator," EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, April 2016, http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-17.html.

[52] J. Kimmitt, W. Song, and A. Bradbury, "lowRISC with tagged memory and minion core (version 0.4)," June 2017, https://www.lowrisc.org/docs/minion-v0.4/.

[53] A. Seznec, "A case for two-way skewed-associative caches," in *Proceedings of the Annual International Symposium on Computer Architecture (ISCA)*.   ACM, May 1993, pp. 169–178.

[54] S. Sardashti, A. Seznec, and D. A. Wood, "Skewed compressed caches," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.   IEEE, 2014, pp. 331–342.

[55] M. Sugiyama, "Section 2.5 – transformation of random variables," in *Introduction to Statistical Machine Learning*.   Boston: Morgan Kaufmann, 2016, pp. 22–23.

[56] "Scale: Scaling and centering of matrix-like objects," https://www.rdocumentation.org/packages/base/versions/3.6.2/topics/scale.

[57] J. S. Hunter, "The exponentially weighted moving average," *Journal of Quality Technology*, vol. 18, no. 4, pp. 203–210, 1986.

[58] Y. Zhang and M. K. Reiter, "Düppel: Retrofitting commodity operating systems to mitigate cache side channels in the cloud," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS)*.   ACM, 2013, pp. 827–838.

[59] A. Phansalkar, A. Joshi, and L. K. John, "Analysis of redundancy and application balance in the SPEC CPU2006 benchmark suite," in *Proceedings of the International Symposium on Computer Architecture (ISCA)*.   ACM, June 2007, pp. 412–423.

[60] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically characterizing large scale program behavior," in *Proceedings of the International Conference on Architectural Support for Programming Languages and Operating Systems (ASPLOS)*.   ACM Press, 2002, pp. 45–57.

[61] A. A. Nair and L. K. John, "Simulation points for SPEC CPU 2006," in *Proceedings of the International Conference on Computer Design (ICCD)*.   IEEE Computer Society, 2008, pp. 397–403.

[62] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Applied Soft Computing*, vol. 49, pp. 1162–1174, 2016.

[63] M. Payer, "HexPADS: A platform to detect "stealth" attacks," in *Proceedings of the International Symposium on Engineering Secure Software and Systems (ESSoS)*.   Springer, 2016, pp. 138–154.

[64] T. Zhang, Y. Zhang, and R. B. Lee, "CloudRadar: A real-time side-channel attack detection system in clouds," in *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*.   Springer, 2016, pp. 118–140.

[65] J. Chen and G. Venkataramani, "CC-Hunter: Uncovering covert timing channels on shared processor hardware," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.   IEEE, 2014, pp. 216–228.

[66] A. Fuchs and R. B. Lee, "Disruptive prefetching: Impact on side-channel attacks and cache designs," in *Proceedings of the ACM International Systems and Storage Conference (SYSTOR)*.   ACM, 2015, pp. 14:1–14:12.

[67] F. Yao, H. Fang, M. Doroslovacki, and G. Venkataramani, "Towards a better indicator for cache timing channels," *arXiv: abs/1902.04711*, 2019, http://arxiv.org/abs/1902.04711.

[68] A. Harris, S. Wei, P. Sahu, P. Kumar, T. M. Austin, and M. Tiwari, "Cyclone: Detecting contention-based cache information leaks through cyclic interference," in *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.   IEEE, 2019, pp. 57–72.

[69] A. Purnal, L. Giner, D. Gruss, and I. Verbauwhede, "Systematic analysis of randomization-based protected cache architectures," in *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*.   IEEE, May 2021.

[70] 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 *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*.   IEEE, October 2020.