CTPP: A Fast and Stealth Algorithm for Searching Eviction Sets on Intel Processors

Zihan Xue
SKLOIS, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
xuezihan@iie.ac.cn

Jinchi Han
SKLOIS, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
hanjinchi@iie.ac.cn

Wei Song*
SKLOIS, Institute of Information Engineering, CAS
School of Cyber Security, University of Chinese Academy of Sciences
Beijing, China
songwei@iie.ac.cn

ABSTRACT

Eviction sets are essential components of the conflict-based cache side-channel attacks. However, it is not an easy task to construct eviction sets on modern Intel processors. As a promising defense against conflict-based cache side-channels, dynamic cache randomization makes the construction of eviction sets even more difficult by periodically randomizing the mapping between addresses and cache set indices. It forces attackers to develop fast search algorithms to find an eviction set at runtime with the lowest latency. Several fast search algorithms have been proposed in recent years. By using these algorithms, attackers regain the capability of launching useful attacks on dynamically randomized caches. Consequently, a detector was recently introduced to catch the fast search algorithms in action according to the uneven distribution of cache evictions. All existing fast search algorithms fail to work.

We present a new eviction set search algorithm called *Conflict Testing with Probe+Prune* (CTPP). Based on the evaluation on six Intel processors and a behavioral cache model, CTPP is found to achieve the lowest latency in finding an eviction set in all algorithms, potentially escape from the recently proposed detector, and present a strong tolerance to environmental noise.

CCS CONCEPTS

• **Security and privacy → Side-channel analysis and countermeasures**.

KEYWORDS

cache side-channel attack, eviction set, micro-architecture

ACM Reference Format:
Zihan Xue, Jinchi Han, and Wei Song. 2023. CTPP: A Fast and Stealth Algorithm for Searching Eviction Sets on Intel Processors. In *The 26th International Symposium on Research in Attacks, Intrusions and Defenses (RAID '23), October 16–18, 2023, Hong Kong, Hong Kong*. ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3607199.3607202

*Corresponding author.

RAID '23, October 16–18, 2023, Hong Kong, Hong Kong

ACM ISBN 979-8-4007-0765-0/23/10.
https://doi.org/10.1145/3607199.3607202

1 INTRODUCTION

Software cache side-channel and covert-channel have become an important tool for attackers to exploit modern computer systems. They have been used directly as an attack to leak security-critical information from the cache system [9, 17, 23], or indirectly as a tool to disarm existing defenses, such as the address space randomization [3, 5], before launching the actual attack. In some comprehensive attacks, cache side-channels have been used as an intermediate step, such as retrieving the leaked information at the end of a transient execution attack [12, 13], and constantly striking a row of the off-chip memory in a rowhammer attack [6].

Eviction sets, especially the minimal eviction sets [33], are essential components of the conflict-based cache side-channel attacks [37]. In such attacks, an attacker and her victim share the same cache space, typically certain cache sets in the last-level cache (LLC). The attacker needs to control the state of these shared cache sets to monitor the memory accesses of her victim, which are then used to infer security-critical information. To be specific, the attacker occupies (primes) a cache set by (repeatedly) accessing an eviction set in an optimized way [6]; therefore, her victim's access to this cache set must incur a cache miss, a refill of this missing cache block, evicting one of the attacker's addresses (the eviction set), and eventually a prolonged access. Both the eviction of the attacker's addresses and the prolonged access latency might be observable by the attacker and used to infer the access of her victim.

Reducing the size of an eviction set is important. Intuitively, a sufficiently large collection of random addresses is a generic eviction set capable of evicting any targeted address, because accessing this large collection of addresses primes the whole cache. Most cache attacks concentrate on a small number of targeted cache set and favor an environment with the minimal noise. A generic eviction set is obviously useless in this sense. Priming a targeted cache set using a minimal eviction set, a smallest collection of addresses just capable of occupying the targeted cache set, generates the least amount of noise and is normally faster than using non-minimal eviction sets [6]. It is important for attackers to construct and use minimal eviction sets. For simplicity, the term "eviction set" beyond this point refers to the minimal eviction set.

All addresses in an eviction set are *congruent* with the targeted cache set (mapping to the targeted cache set) [33]. At least W addresses are required for a W-way set-associative cache. Obviously, the key for constructing an eviction set is to find enough congruent addresses, which, unfortunately, is not an easy task on modern processors for several reasons: *LLC is indexed by physical addresses*

while only virtual addresses are observable by attackers. Without deciphering the virtual-to-physical page mapping, attackers cannot compute the mapped cache set of an address as cache indices are determined by physical addresses. As a way to get around this, some attackers may obtain partial control over the cache set index by exploiting huge pages. However, the *complex addressing scheme* utilized by modern Intel processors [15] randomized the mapping of physical addresses to LLC slices, which nullifies some of the benefits of using huge pages. At the end, attackers are often forced to blindly search for eviction sets.

Cache randomization has recently been proposed as a promising defense against conflict-based cache side-channels [21, 26]. By randomizing the mapping from addresses to cache set indices, it forcefully nullifies all existing ways to directly construct eviction sets by manipulating the mapping between addresses and cache set indices. An eviction set must be dynamically searched from a large amount of random addresses. Furthermore, the randomized mapping is remapped periodically to limit the time window left for attackers to utilize an eviction set even if it is found [2, 21, 26]. These reasons push attackers to develop fast search algorithms which can find an eviction set with the lowest latency.

Several fast search algorithms have been proposed in recent years, including *group elimination* (GE) [14, 27, 33], *prime, prune and probe* (PPP) [18, 20], *conflict testing* (CT) [22] and some CT derived algorithms, such as the optimized CT (CT-fast) utilized in the Prime+Scope attack [19] and write-after-write (W+W) [28]. With the help of these fast algorithms, the time required for finding an eviction set on current Intel processors has been reduced from hundreds of milliseconds [16] to several milliseconds (See Table 1). Evidences show that attackers may already have the capability to find eviction sets and launch useful attacks on dynamically randomized caches [2]. As a strong counter-measure, a detector was recently introduced in [26] to detect the action of fast search algorithms according to the uneven distribution of cache evictions across cache sets. Extra remaps are triggered when a dynamic search of eviction set is caught in action. In this scenario, all existing fast search algorithms fail to work. [26]

In this paper, we present a new eviction set search algorithm called *Conflict Testing with Probe+Prune* (CTPP), which combines the benefits of both CT and PPP algorithms. Comparing with existing algorithms, CTPP has three major advantages: (1) *It achieves the lowest latency in finding an eviction set on current Intel processors.* (2) *It can escape from the detector proposed in [26].* (3) *It presents a strong tolerance to environmental noise.* We have tested CTPP on six Intel processors, including i7 processors from the 3rd to the 11th generation and both the 1st and 2nd generation Xeon processors. The results show that CTPP can successfully find an eviction set in a couple of milliseconds with a high success rate on all processors. By re-implementing the detector proposed in [26] using the same behavioral-level cache model [27], we are able to verify the anti-detection capability of all algorithms. The results show that CTPP maintains working with a success rate ranging $10 \sim 70\%$ when all other algorithms fail (see Figure 12). We have also verified that CTPP may still work properly when another process (such as a SPEC CPU 2006 benchmark) is running in the background.

Overall, this paper makes the following contributions:

- Propose a new search algorithm, namely CTPP, which achieves the lowest latency in finding eviction sets, a unique capability to escape from the latest detector, and a strong tolerance to environmental noise.
- Analyze the causes of the slow speed of existing search algorithms.
- Analyze the behaviors of existing algorithms that make them exposed to detectors.
- Practical evaluation of existing algorithms along with CTPP on both real Intel processors and a behavioral-level cache model.

The paper is organized as follows: Section 2 introduces the necessary background for understanding this paper. Section 3 explains our motivation in designing a new algorithm for search eviction sets. The threat model is defined in Section 4. Section 5 presents the proposed CTPP algorithm, whose performance is compared with all existing algorithms in Section 6. The limitations of CTPP are discussed in Section 7. Finally, the paper is concluded in Section 8.

2 BACKGROUND

2.1 Cache Architecture

Current Intel processors are multicore processors adopting a three-level cache structure. Each processing core contains a pair of private level-one (L1) instruction and data caches, and a uniformed level-two (L2) cache. A large level-three (L3) cache is shared as the last-level cache (LLC) between all processing cores. The LLC is evenly divided into multiple slices, normally one per each processing core. The mapping between (physical) addresses and LLC slices is decided by an undisclosed hash function, namely the complex addressing scheme. The hash functions of several Intel processors have been reverse-engineered [15].

All cache levels use classic set-associative caches adopting a writeback allocation policy. The L1 caches are reported to use the pseudo-LRU (PLRU) replacement policy [1] while the L2 and LLC likely adopt some policies derived from RRIP [11, 32]. LLC acts as the coherence hub for the cache coherence management. LLC and the private L1/L2 caches maintain an inclusive relation on consumer processors (i3 to i9 processor families) as shown in Figure 1a, while the relation is non-inclusive on server-level processors (the Xeon processor family) starting from the SkyLake-SP architecture [38]. Cross-core cache side-channel attacks targeting the LLC depends on the attacker's capability to purge a cache block from another core's L1/L2 cache as an inclusion victim [10]. This is easily achieved on an inclusive LLC as evicting a cache block from the LLC also purges the cache block from the whole cache hierarchy. On the Xeon processors using non-inclusive LLCs, the directory adopts an inclusive structure as shown in Figure 1b. For cache blocks shared between cores, the block's metadata is stored in one of the traditional directory entries and can be evicted using an eviction set comprising of shared cache blocks [38], which also results in purging the cache block from all cache levels.

2.2 Cache Side-Channel Attacks

Cache side-channel attacks refer to attacks using a cache as a medium for leaking security-critical information. Depending on how caches are manipulated by attackers, there are three types of

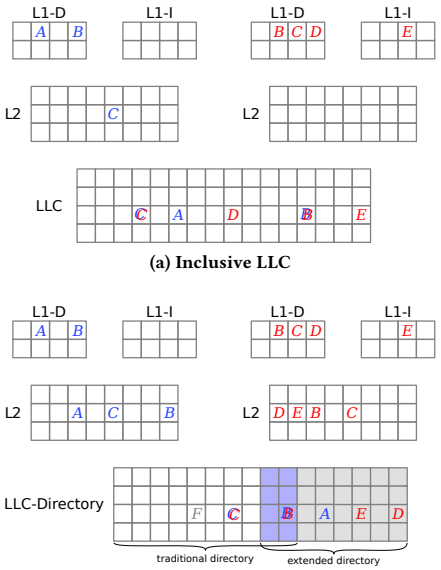


Figure 1: Examples of the inclusive LLC (a) used in Intel's consumer processors and the non-inclusive LLC with an inclusive directory (b) used in the Xeon processors. Cache blocks A, B and C are used by core 0; B, C, D and E are used by core 1; B and C are shared between cores.

cache side-channel attacks: cache occupancy attacks, flush-based attacks and conflict-based attacks. The cache occupancy attack [25, 31] is an untargeted cache attack which takes the whole LLC as the communication channel. An attacker uses a large collection of addresses to occupy (prime) most cache sets in the LLC and measures the interference caused to her victim as a way to leak information. Both flush-based and conflict-based attacks are targeted attacks concentrating on one or several cache sets. They both require an attacker to bring a targeted cache set into a controlled state but in different ways [37]. Flush-based attacks use explicit cache control instructions, such as `clflush` on x86 [39], to invalidate victim's cache blocks, while conflict-based attacks indirectly evict victim's cache blocks by occupying the targeted cache sets with attacker's own blocks [17]. Comparing with conflict-based attacks, flush-based attacks are fast and accurate (more information is leaked per attack); however, they require the flushed cache block to be shared between the attacker and her victim, which is a rather restricting requirement. In generic attack scenarios where an attacker and her victim are running on separated processes sharing no memory but only the cache space, conflict-based attacks are preferred over flush-based attacks. This paper focuses on the conflict-based cache attacks on current Intel processors.

An attack normally proceeds in two phases: a *preparation phase* and an *exploitation phase*. In the preparation phase, the attacker collects enough number of eviction sets. Each eviction set is a collection of congruent addresses mapping to a targeted cache set. Accessing this collection of addresses in a certain order is capable of occupying all ways of the targeted cache set, evicting any block belonging to the victim, and setting the cache set to a fully occupied

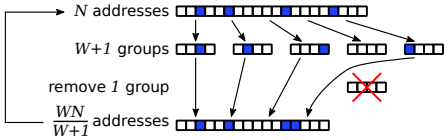


Figure 2: One prune round of GE in a 4-way cache.

state. In the *exploitation phase*, the attacker infers security-critical information by manipulating the targeted cache sets using the collected eviction sets. The exploitation phase normally contains numerous prime+probe cycles [14, 17]. In each cycle, the attacker first *primes* a target cache set using a corresponding eviction set. If there were cache blocks belonging to the victim, they are likely evicted. 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 this cache set, it must have been fetched and refilled into the cache set by eviction one of the attacker's cache blocks. Finally the attacker *probes* the cache set by re-accessing 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.

2.3 Fast Search Algorithms for Eviction Sets

Group elimination (GE) is an optimization of an old algorithm against the Intel's complex address scheme [22, 33]. It starts with a large eviction set composed of random addresses and quickly prunes it into a minimal one with only W addresses, where W is number of ways. In each prune round, as shown in Figure 2, the remaining N addresses are divided into $W+1$ groups. Since a minimal eviction set contains only W addresses, there is at least one removable group containing none of the W addresses and should be removed. By sequentially testing whether the address collection remains an eviction set without a certain group, the removable group is found and removed. The prune process continues with $N/(W+1)$ addresses removed in each round until a minimal set is produced. By average, it requires $O(SW^2)$ cache accesses to find a minimal eviction set in an S-set LLC [27].

Prime, prune and probe (PPP) is a search algorithm exploiting the LRU replacement [20, 22]. An attacker starts with accessing a large collection of random addresses (prime set) to prime the LLC. Since these addresses unavoidably cause self-conflicts, a prune process is used to remove conflicting addresses until all addresses remaining in the prime set are concurrently cached. To collect congruent addresses from the reduced prime set, the attacker makes a timed re-access of the target address and the prime set sequentially, as depicted in Figure 3. Addresses missing in the LLC are detected as they incur long access latency. Since all addresses in the reduced prime set are stored in the LLC, all missing addresses must be evicted by the target address X, such as E, or the previously evicted addresses in a chain reaction, such as G, J and L. It is easy to observe that all these addresses are congruent with X. In an ideal (noiseless) scenario, as shown in Figure 3, enough addresses are found in just one probe. The overall number of cache accesses is only $O(SW)$. This algorithm can be used with other types of replacement policies, such as the random replacement policy. However, the overall number of accesses rises to $O(SW^2)$, because the number of congruent addresses found in each search decreases significantly, leading to the need for extra search rounds [20].

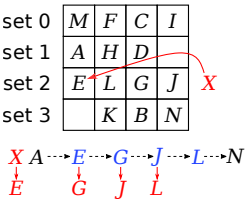


Figure 3: The probe step of PPP in a 4-way cache.

Conflict testing (CT) is an algorithm first proposed to find eviction sets in caches using random replacement [22], where an attacker can collect an eviction set by sequentially testing multiple random addresses whether any of them are 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 $1/W$ thanks to the random replacement. This condition is checked by a timed re-assess of the target address. An eviction set is produced when enough congruent addresses are collected. Overall, any random address might conflict with the target address by a probability of $1/(SW)$. The total number of cache accesses is estimated around $O(SW^2)$. This algorithm is also effective for permutation-based replacement, such as LRU and RRIP. Assuming the use of LRU, the probability of causing a conflict with the target address after accessing M random addresses is around:

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

This is equivalent to causing at least W conflicts in the target cache set. The average $\overline{M}$ is around SW. To find a minimal eviction set with W addresses, the number of cache accesses is $O(SW^2)$.

Two derived algorithms have been proposed based on the CT algorithm. An optimized version of CT was present in the Prime+Scope attack [19], which we call *CT-fast* in this paper. Whenever a congruent address is found (the target address is evicted), all the previously found congruent addresses are accessed after re-accessing the target to make the target easier to evict. The total number of memory accesses is roughly halved in CT-fast. Instead of checking congruence by evicting the target address, detecting the prolonged latency due to the LLC enforced serialization of parallel writes to the same cache set was also found effective [28]. The resulted algorithm, namely *W+W* in this paper, was claimed faster than the GE algorithm.

2.4 Randomized Caches

Cache randomization [21, 34] has recently been accepted as a promising defense. In a randomized cache as depicted in Figure 4, the mapping from memory addresses to cache set indices is randomized by a cipher, forcing attackers to slowly find eviction sets at runtime. Even when eviction sets are found, attackers cannot easily 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 [2, 21]. For this reason, periodical remapping [21] is used to limit the time window available to attackers, extra remaps can be triggered if a search of eviction sets is found in active [26], and cache skews [22] can be used to further increase the difficulty in finding eviction sets.

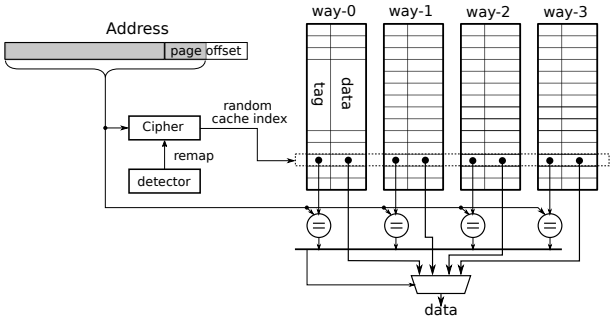


Figure 4: Randomized set-associative cache.

In this paper, we also consider finding eviction sets for the randomized set-associative caches [21, 26] (such as the results present in Figure 12) but leave the randomized skewed cache as one of our future works.

3 MOTIVATION

The proposal of CTPP is motivated by two findings on the existing fast search algorithms for eviction sets:

- Existing algorithms fail to exhaust all the available techniques to manipulate the LLC in Intel processors.
- Existing algorithms are easily detectable under the recently proposed detector [26].

These make us to wonder whether it is possible to design an even faster search algorithm by exploiting extra ways to manipulate the caches of the current Intel processors and make it less likely to be detected. The rest of this section elaborates the two findings while the CTPP algorithm is proposed in Section 5.

3.1 Slowness of Existing Search Algorithms

The GE algorithm starts with a large eviction sets and recursively prunes the large set using the group elimination method. In the pruning process, no detailed controlling of the cache set state is applied to the targeted cache set. Each round of test is a full utilization of the assumed eviction set. This is why its success rate depends on the optimal traverse function of the eviction set [27]. This tolerance to the cache replacement policy makes the GE algorithm the most robust algorithm which works on almost all (non-skewed) cache architectures with a relatively high success rate (see Table 1); however, this is also the reason why its prune process is inefficient and slow.

The PPP algorithm also starts with a large prime set to occupy most of the LLC. The final probe step utilizes the RRIP replacement on Intel processors. It needs typically only one or two rounds of probes to collect enough congruent addresses for an eviction set. For this reason, PPP incurs the shortest latency in all existing algorithms (see Table 1). However, how to efficiently implement the prune step is unclear in existing literature. According to our own investigation, if the number of congruent addresses in the initial prime set is more than W, the prune step is likely to remove extra congruent addresses and reduce the total number to less than W in the pruned set, leading to a failed search. The success rate of PPP is low, which leads to a prolonged latency for successfully finding an eviction set.

The CT algorithm was originally proposed for LLCs using the random replacement policy but later found applicable to LLCs using LRU/RRIP policies as well. The problem for CT is that it actually tested a large number of congruent addresses (with the targeted cache set) and only a small portion of them are finally collected. For each time an addresses is successfully identified as a congruent address, $W-1$ congruent addresses are actually accessed (and tested) for pushing the target address to the LRU position, assuming the PLRU policy. For the original CT algorithm, W^2 congruent addresses are accessed for collecting W addresses. As an optimization, the CT-fast algorithm reduces the search time by pushing the target address towards the LRU position reusing the already collected congruent addresses before testing a new random address. However, CT-fast only reduces the total number of congruent addresses accessed by half, still significantly larger than W.

The W+W algorithm exploits the fact that parallel write accesses to the same cache set are serialized by the LLC, incurring a detectable latency overhead. Since causing such serialization condition does not require manipulating the cache state, W+W could, in theory, find an eviction set by accessing just W congruent addresses, potentially achieving a speed faster than CT-fast. However, detecting such latency overhead carries with significant amount of noise [28] and is feasible on some but not all Intel processors. On practical attacks, W+W is faster only than GE.

3.2 Detectable Footprint of Cache Evictions

A detector dedicated for detecting existing fast search algorithms for eviction sets has been proposed in [26]. It is claimed that both GE's recursive pruning and PPP's final probe step would unavoidably incur extra amount of cache evictions on the targeted cache set. By monitoring the distribution of cache evictions among cache sets, the unbalanced distribution caused by the extra evictions on the target cache set can be identified. Consequently, a remap is triggered to nullify the eviction set even if it were successfully found.

To be specific, the detector records the number of cache evictions on the i-th cache set during one monitoring period as e_i and calculate a z-score z_i, according to Equation 2, to single out the cache sets with outstanding amount of evictions.

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

However, directly using this z-score leads to false positive errors when only a small number of evictions happen in one monitoring period. As a solution, the z-score is weighted and moving averaged using Equation 3 and 4, respectively:

$$wz_i = (e_i - \bar{e}) \cdot z_i \tag{3}$$

$$az_i(t) = (1-\alpha) \cdot az_i(t-1) + \alpha \cdot wz_i(t) \tag{4}$$

where wz_i is the weighted score and $az_i(t)$ is the moving averaged score at the t-th monitoring period finally used for detection. For simplicity, let us call az_i as the detector score for i-th cache set. If any detector score surpasses a predefined threshold of 5.0, an attack is assumed in action and a remap is triggered.

We have reproduced the testing environment for the dynamically randomized LLC using the same cache model open-sourced by [27]. All existing fast search algorithms fail to work as shown in Figure 12. This result shows that this detector is indeed effective in identifying and thwarting existing search algorithms.

4 THREAT MODEL

Before diving into the details of the new search algorithm, we would like to specify our assumptions. For an eviction search algorithm, we define a successful attack as finding a sufficiently small (ideally minimal) eviction set. We assume that the search algorithm is run by an attacker in a restricted user model environment with the following capabilities and limitations:

- The targeted LLC is a non-skewed set-associative cache (potentially randomized) shared between the attacker and her victim.
- The amount of memory acquirable by the attacker is not limited by the system, so the attacker can access an arbitrarily large range of addresses.
- The attacker can occupy more than one processing cores and issue parallel memory accesses to the same LLC cache block.
- The attacker can flush her own data out of the LLC.
- The attacker can accurately trick her victim into accessing the target LLC cache set without incurring a large amount of noise.
- Some parameters regarding the cache system are made available, such as the number of sets and ways of each cache level, but neither the virtual to physical page mapping nor the Intel complex addressing scheme is reverse-engineered.

5 THE CTPP ALGORITHM

In this section, we propose a new search algorithm for eviction sets called CTPP: *Conflict Testing with Probe+Prune*. The behavioral description of CTPP is presented in Algorithm 1. It is a combination of the CT and the PPP algorithms. In all eviction set search algorithms, PPP achieves the lowest latency but suffers from a low success rate. This low success rate is caused by the ineffective prime and prune steps. By replacing them with a part of the CT algorithm, the success rate is improved significantly. In addition, the original probe step of the PPP algorithm is found easily detectable by the detector proposed in [26]. By swapping the order of prune and probe, we successfully reduce the likelihood of being detected.

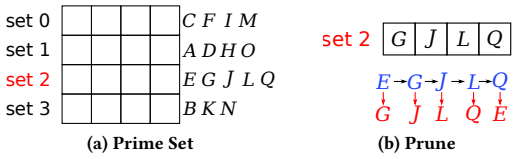
The algorithm accepts two inputs: the target address x and the number of ways W in the LLC. If success, the algorithm returns an eviction set $\mathcal{E}$ targeting x. The search proceeds in three steps: (1) A prime set C containing just W congruent addresses with the target x is obtained using the CT algorithm. This prime set is sequentially fed into a probe (2) and a prune (3) steps to reduce the large prime set into a minimal eviction set. Both the probe and the prune steps adopt the same method as in PPP but the order of the two steps are swapped. Depending on the noise level, extra rounds of probe+prune may be required. The rest of this section explains the logic behind the design of CTPP and why it is effective in finding eviction sets.

5.1 Obtain a Perfect Prime Set Using CT

First of all, let us revisit why the PPP algorithm is ineffective in obtaining a usable prime set after the prune step. In a common but

Algorithm 1: The CTPP algorithm

```
Input: x, target address.
Input: W, number of ways.
Output: E, an eviction set for x.
function ctpp(x, W)
    C ← ∅ // prime set
    R ← ∅ // probed set
    E ← ∅ // eviction set
    prime(LLC) // preparation
    // CT step
    access(x)
    while probe(x) do
        a ← random()
        access(a)
        C ⋃ {a}
    end
    while |C| > W do
        if R ≠ ∅ then
            R ← ∅
            flush(C), access(C)
        end
        // probe step
        access(x)
        foreach c ∈ C do
            if not probe(c) then
                R ⋃ {c}
            end
        end
        // prune step
        E ← ∅
        foreach c ∈ R do
            if probe(c) then
                E ⋃ {c}
            end
        end
        C ← E
    end
    return E
end
```



(a) Prime Set (b) Prune

Figure 5: PPP is ineffective in pruning an imperfect prime set. When an initial prime set (a) contains more than W congruent addresses, the prune step (b) is likely to remove the number of congruent addresses to less than W. Addresses colored in blue are found missing in the cache. Addresses colored in red are evicted from the cache. Cache set 2 is the target.

imperfect situation, a randomly chosen prime set contains more than W congruent addresses as shown in Figure 5a.[1] After priming the LLC, G, J, L and Q are left in the targeted cache set 2. By the default (also naive) prune method described in [20], the prime set is pruned by re-accessing all addresses in the prime set and removing all addresses missing in the LLC. Using the same accessing order as in the prime step, all accesses to the targeted cache set result

[1]We do not discuss the case where the number of congruent addresses is less than W as it always fails.

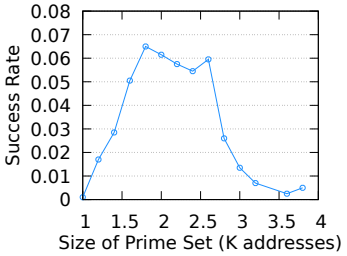


Figure 6: Success rate of PPP on i7-6700 using prime sets of different sizes. Each sample is averaged from 2000 tests.

in misses and the addresses are removed accordingly. The prime set after prune contains no address congruent with the targeted cache set and the search fails. The root cause of failure is that the ineffective prune fails to reduce a prime with more than W congruent addresses into one with exactly W congruent addresses. Unfortunately, no optimal prune method has been proposed yet.

Figure 6 shows the success rate of PPP (without retry) on i7-6700 using prime sets of different sizes. The optimal size of the prime set is around 2 ~ 2.5 K addresses. When using a prime set smaller than the optimal size, the number of congruent addresses contained in the prime set is likely less than W and PPP is less likely to succeed. When using a prime set larger than the optimal size, the number of congruent addresses is likely more than W and the prune step is prone to reduce the number to less than W, which also results in a low success rate. Even when the prime set is with the optimal size, the success rate is still low (only around 6%) as the probability of containing just W congruent addresses in the prime set is small.

Instead of using PPP, we decide to use the CT algorithm to obtain a perfect prime set (C in Algorithm 1) with exactly W congruent addresses. When we successfully collect the first congruent address using CT in an LLC adopting the LRU replacement policy, all the random addresses accessed during the search process become a perfect prime set containing just W congruent addresses. The LLC in the Intel processors adopt variants of the RRIP replacement policy [32]. The attacker can degrade RRIP into LRU by enforcing duplicated accesses for each random address using two processing cores [22]. By feeding this perfect prime set collected by CT to the original PPP algorithm (CT+PPP), we significantly improve the success rate on i7-6700 from around 6% to 49% with a small latency overhead (prolonged from around 0.6 ms to 0.78 ms per test).

5.2 Stealth Probe

The PPP algorithm is found easily detectable by the detector proposed in [26] due to the unbalanced distribution of evictions among cache sets during the probe step.

Let us analyze this unbalanced distribution using a demonstrative example of running PPP on a 6-set 4-way cache as shown in Figure 7. A perfect prime set is first obtained (using the CT algorithm) as depicted in Figure 7a. In this perfect prime set, four addresses are congruent with the targeted cache set 2, exactly the number of ways. Assuming the address accessing order is $A \rightarrow X$ and the order observed by the cache matching with the order initiated by the attacker, the prune step removes all addresses missing during a re-accessing of the prime set. As shown in Figure 8b, after the prune

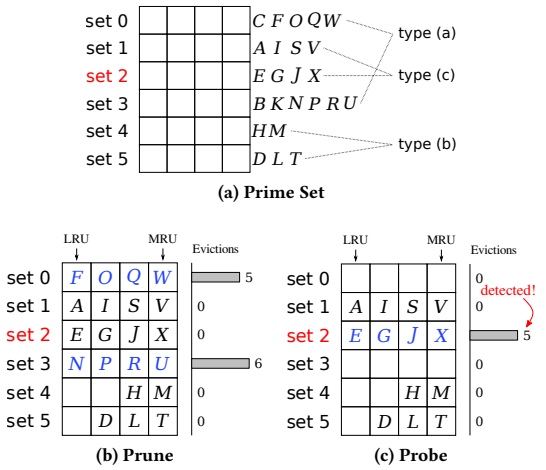


Figure 7: Explain the reason why PPP is easily detected. Cache set 2 is the target cache set. Addresses colored in blue are found missing in the cache. The perfect prime set collected by CT (a) is pruned using the default prune method (b) and then probed by re-accessing the reduced prime set after accessing the target address (c). In the probe step, all evictions unevenly occur on the target cache set.

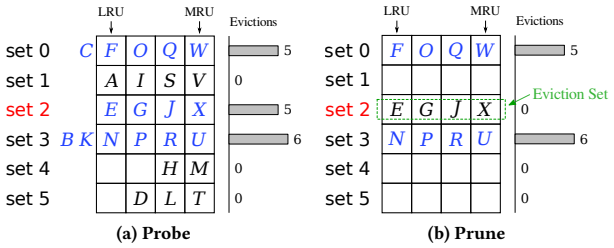


Figure 8: Explain the reason why CTPP is unlikely to be detected. Cache set 2 is the target cache set. Addresses colored in blue are found missing in the cache. The perfect prime set collected by CT is first probed by re-accessing it after accessing the target address (a) and then pruned (b). The addresses found hitting in the prune step form an eviction set. In both probe and prune, the eviction distribution is balanced.

step, all addresses colored blue are found missing in the cache and are removed while all addresses colored in black are concurrently stored in the cache and kept in the reduced prime set. The numbers of cache evictions happened on each cache set during the prune step are counted and presented in Figure 8b as well. Since two cache sets[2] demonstrate high amount of cache evictions, the distribution of cache evictions is relatively balanced among cache sets. However, such distribution become extremely unbalanced in the probe step. Since all addresses in the reduced prime set are concurrently stored in the cache, any addresses founding missing in the probe step, after re-accessing the target address, must be congruent with cache set 2 and cause an eviction on cache set 2. Consequently, only cache set 2 suffers from a large amount of cache evictions. Such unbalanced distribution is exactly the pattern constantly monitored by the detector and the search is therefore exposed.

Analyzing the whole procedure for the detector to detect PPP, we can observe two insights:

- Only cache evictions, rather than cache hits, are detectable.
- The eviction set is identified by evictions in the final probe.

Basically, the PPP algorithm starts with a large prime set containing three types of addresses (labeled in Figure 7a): type (a), addresses mapped to cache sets with more than W congruent addresses; type (b), addresses mapped to cache sets with less than W congruent addresses; and finally type (c) addresses mapped to cache sets (hopefully including the targeted cache set) with exactly W congruent addresses. The prune step reduces the prime set by removing all type (a) addresses using the default prune method. Later in the probe step, the eviction set is identified from type (c) addresses by cache evictions as all type (a) addresses are already removed. If we can reduce the initial prime set by removing all type (b) addresses rather than type (a) addresses, it is possible to identify the eviction set utilizing cache hits, which are not detectable. As the reduction of type (b) addresses is not set targeted, the distribution of cache eviction is relatively balanced. During the final identification of the eviction set, a large amount of cache evictions are caused by the remaining large amount of type (a) addresses, the distribution of cache eviction is relatively balanced as well. As a result, the whole search procedure becomes unlikely detectable. How can we do this? *Swap the order of prune and probe.*

As shown in Figure 8a, we first reduce the same perfect prime set used in the previous PPP example (Figure 7a) by a probe rather than a prune. The target address is first accessed (line 19 in Algorithm 1) and then the whole prime set is re-accessed $A \rightarrow X$ (line 20–24 in Algorithm 1). All addresses missing during the probe, type (a) addresses and the eviction set, are retained (colored in blue) while the addresses hitting in the cache, type (b) addresses and type (c) addresses except for the eviction set, are removed. Since a large amount of cache evictions are caused by re-accessing type (a) addresses and the eviction set, the distribution of cache evictions among cache set is balanced.

Addresses belonging to the eviction set are identified using a prune step as shown in Figure 8b. Note that the probe step removes not only all type (b) addresses but also the type (c) addresses not belonging to the eviction set. Addresses of the eviction set become the only type (c) addresses retained in the reduced prime set after probe. In the prune step, all remaining addresses are re-accessed in the same order $E \rightarrow X$ (line 27–31 in Algorithm 1). Assuming the access order observed by the cache matching with the attacker's re-accessing order, only addresses belonging to the eviction set are found hitting in the cache and are therefore identified. Similar to the probe step, a large amount of cache evictions are caused by re-accessing the type (a) addresses, the distribution of cache evictions among cache set is still balanced.

By swapping probe and prune, CTPP successfully avoids the unbalanced distribution of cache evictions among cache sets caused

[2]The number of cache sets experiencing evictions would be far larger in the LLC of a practical processor.

by the probe step in PPP. Consequently, CTPP has a much higher probability to succeed without being detected. Compared with PPP (or CT+PPP), no extra step is introduced and the total number of memory accesses increases only marginally. Testing on i7-6700, the performance of CTPP is almost unaffected by the swap of steps. It achieves a success rate of 42% and an average latency of 0.73 ms per test without retry, which lies in the same range with CT+PPP.

5.3 Match Access Order

The same access order initiated by the attacker and observed by the LLC is a prerequisite for the success of CTPP. In the probe step depicted in Figure 8a, CTPP expects to remove all type (b) addresses and retain all type (a) addresses along with the eviction set. If any type (a) address results in hitting in the cache in the probe, it is mistakenly retained and potentially becomes a type (b) address, because the total number of congruent addresses remaining in the reduced prime set and mapping to this cache set drops below W. As a result, it would be identified as a part of the eviction set, leading to a failed search. Such condition is likely to happen if the access order observed by the LLC is different with the order initiated by the attacker.[3]

Generally speaking, there are two issues need to be resolved: One is the filter effect of the private caches. If any access hits in the private L1/L2 caches, it is hidden from the LLC. The other one is the RRIP replacement policy adopted by modern Intel processors is anti-thrashing [11] while the accessing of a prime set is exactly a thrashing pattern [36].

Let us first consider the filter effect in an inclusive cache hierarchy and pretend the replacement policy is LRU. We argue that the access order observed by the LLC matches with the access order initiated by the attacker as long as that the LLC is fully reset (by priming it with random and irrelevant addresses, at line 5 in Algorithm 1) and the prime set is always accessed using the same order. If the LLC is fully reset, each address accessed by the attacker during the CT step is taken as new by the LLC and occupies the MRU position in the cache set. After a prime set is found, the access order observed by the LLC, more importantly the replacement priority from MRU to LRU, matches with the order of random addresses issued by the attacker. By accessing the prime set using the same order during the probe step, all type (a) addresses and all addresses belonging to the eviction set are found missing in the LLC, since the accessing of these addresses presents a thrashing pattern where each address is accessed once and only once in a chain, and the chain is too long to fit in a cache set. The accessing of type (b) addresses does not present the same thrashing pattern but they are removed in the probe step. Later in the prune step, the re-accessing of type (a) addresses presents the same thrashing pattern and results in all misses, while all addresses belong to the eviction set hit in the LLC.

For the non-inclusive LLC use by Intel Xeon processors, a cache block shared between processing cores must occupy one of the traditional directory space, as reported by [38]. When a block becomes shared between cores, its metadata is moved from the extended directory to the traditional directory. If the traditional directory space is fully occupied, a room is made by evicting a shared cache block and purging it from the whole cache hierarchy, similarly as in an inclusive LLC. The size of the traditional directory space is the same with the number of LLC ways. By re-accessing all random addresses and the target by two processing cores, an attacker can enforce the use of traditional directory space and cause the same conflict as in inclusive LLCs. In this way, the thrashing access pattern remains effective.

The RRIP replacement policy is designed to be anti-thrashing but only effective when the thrashing accesses are initiated by a single core. If all accesses are duplicated, the RRIP algorithm is degraded into the traditional LRU [21]. Conveniently, the dual-core access method used for the non-inclusive LLC also enforces a duplicated access pattern for accessing each address.

In summary, the attacker can enforce the access order initiated by the attacker to be observed by the LLC by fully resetting the LLC before the search, and accessing all addresses always in the same order and by two processing cores.

5.4 Deal with Noise

Practical side-channel attacks rarely occur in a noiseless environment. The tolerance to noise is an important factor in considering the usefulness of an eviction set search algorithm. CTPP is designed with a strong tolerance to noise.

We consider two sources of noises: TLB noise and background noise. Let us first discuss the TLB noise. Since a large amount of random addresses are accessed by the attacker, the limited space of TLB (and L2-TLB) are stressed, increasing the probability of TLB misses. Accessing a cache block stored in the LLC may take an abnormally long latency and be mistakenly assumed missing due to a L2-TLB miss [8, 30]. This is especially problematic for the final prune step. If one of the addresses belonging to the eviction set is mistakenly assumed missing due to a TLB miss, the number of congruent addresses identified by the prune step is less than W. These addresses would fail to form an eviction set. To reduce this type of failure, we adopt the TLB preload method [3] commonly used in other eviction set search algorithms. In the prune step, before accessing an address a, $(a \oplus \texttt{0x0800})$ is first accessed to preload the TLB. For any set-associative cache with more than 32 sets, $(a \oplus \texttt{0x0800})$ is unlikely mapped to the same cache set with a but they share the same TLB entry.

The background noise is the unavoidable noise caused by other processes (including the kernel) running in the background, the code related memory accesses due to misses in attacker's own L1 instruction cache, and potentially the extra memory accesses issued by the victim when triggered to access the target. The attacker can neither control the amount of this noise nor predict the location or time when it strikes. A strong algorithm must tolerate such noise.

There are three scenarios that noise can fail the CTPP algorithm:

- *A noise-caused access to the targeted cache set occurs during the CT step.* This would cause the number of addresses in the prime set and congruent with the target to be less than W, leading to a failed search.
- *A noise-caused access to the targeted cache set occurs during the probe or the prune steps.* CTPP assumes that all the addresses belonging to the eviction set hit in the LLC in the prune step. When a noise-caused access strikes the target cache set, one

[3] An example leading to such condition is to reverse the access order in the probe step.

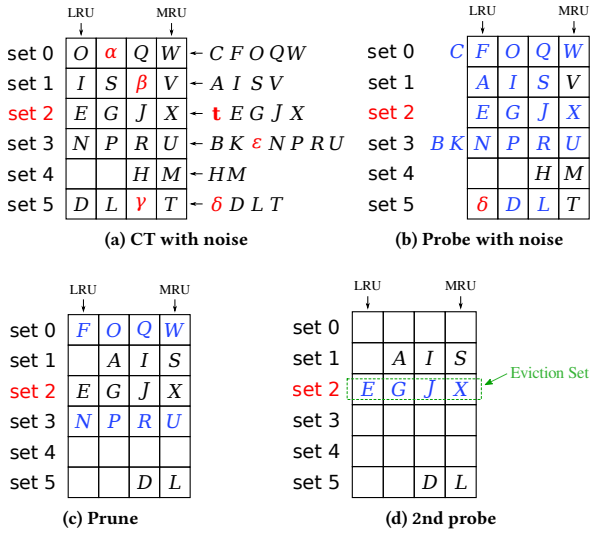


Figure 9: Explain the noise tolerance of CTPP. Cache set 2 is the targeted cache set. Addresses colored in blue are found missing in the cache. Addresses represented by red Greek alphabets are noise accesses. Re-accessing the target address is denoted by a red t. The cache is primed after a perfect prime set is obtained using CT (a). The order of accessing the prime set after re-accessing the target address is also depicted in (a). Due to noise, some type (b and c) addresses are kept after the probe (b), i.e., A, D, I, L and S. These addresses are obtained with the eviction set after prune (c). To remove them, a 2nd probe is required (d).

of the eviction set's address might be evicted and become missing in the final prune, leading to a failed search.

- *Noise-caused accesses to non-targeted cache sets during the CT and the probe steps.* Let's consider the perfect prime set shown in Figure 7a. As shown in Figure 9a, three noise-caused accesses occur during the CT step (α, β and γ) and two further noise-caused accesses occur during the probe step (δ and ϵ). As a result, five type (b and c) addresses, i.e., A, D, I, L and S, are mistakenly retained after the probe step, as shown in Figure 9b. These addresses would remain hitting in the cache during the prune step as shown in Figure 9c, leading to extra addresses being mistakenly identified as a part of the eviction set.

The first two noise-caused failures are unlikely to happen. The noise-caused accesses are randomly distributed among all cache sets. For modern processors with large LLCs, the number of cache set S is huge. As result, the probability that a noise-caused access hitting the targeted cache set is extremely small ($1/S$). The impact on success rate is negligible.

The third failing scenario is the common case that needs to cope with. As shown in Figure 9c, once a type (b) or type (c) address is found missing in the probe step due to noise, it is kept as a part of the eviction set obtained after prune. According to our estimation, the quantity of such addresses in practical tests is not small. On i7-6700, the average number of remaining addresses after each step is: ~2020 after CT, ~290 after probe, and ~80 after prune. As a result, ~60 type (b and c) addresses are mistakenly identified as a part of the eviction set.

Our solution to this problem is doing another round of probe. Since noise is not repeatable, a 2nd round of probe does not suffer from the same noise-caused accesses and all the mistakenly retained addresses should be removed, as shown in Figure 9d. In practical tests on all Intel processors, the number of addresses reduces to W after just one extra round of probe. For the simplicity in representation, we describe this rerun as a loop (line 13, 32, 33 in Algorithm 1) although a 2nd prune is actually unnecessary.

5.5 Work on Randomized Caches

CTPP works on randomized set-associative caches almost in the same way as on non-randomized caches, with only one extra operation: When an extra round of probe is needed, the remaining addresses (C in Algorithm 1) need to be deliberately flushed from the whole cache hierarchy (line 16 in Algorithm 1) and then accessed to enforce that the same access order initiated by the attacker is observed by the LLC.

This is not a problem for classic set-associative caches. Since the number of ways in the inner private caches are normally smaller than it in the LLC and multiple LLC cache sets are mapped to one cache set in the inner private caches, the accessing pattern observed by the targeted LLC cache set is always a thrashing pattern. Consequently the access order observed by the LLC matches with the order initiated by the attacker. However, since the cache set indices in a randomized LLC is randomized, addresses mapping to one LLC cache set is mapped with different cache sets in the inner private caches. As a result, the access order of the eviction set observed by the LLC in the prune step (Figure 9c) might be different with the access order initiated by the attacker as some addresses hit in the inner caches and are hidden from the LLC. If a 2nd probe is required, the cache block evicted by re-accessing the target might not be the first cache block (E in Figure 9d) accessed by the attacker and this block would result in a hit and be mistakenly removed. We need to restore the access order before the 2nd probe to avoid such errors. This is achieved by flushing and then accessing all remaining addresses after prune.

6 PERFORMANCE EVALUATION

In this section, we compare the performance of CTPP with all existing fast search algorithms on both modern Intel processors and on a dynamically randomized set-associative cache implemented in the same behavioral cache model used in [26].

6.1 Search Speed

Table 1 demonstrates the speed performance of all existing fast search algorithms for eviction sets on six different Intel processors, including the 3rd to the 11th Gen Intel consumer processors and both the 1st and the 2nd Gen server-level processors. CTPP is clearly the winner in all algorithms. GE and W+W are significantly slower than other algorithms. According to our analysis in Section 3.1, GE suffers from its inefficient and slow prune process while W+W suffers from its sensitivity to noise in detecting conflicting write

Table 1: Speed performance of all existing fast search algorithms for eviction sets. Each test has been repeated for 2000 times.

Processor	LLC	Huge Page	Latency ($\mu \pm \sigma$) in Millisecond, Success Rate[a]					
			GE[b]	*PPP*[c]	*CT*[d]	*CT-fast*[d]	*W+W*[e]	***CTPP***[c]
i7-3770	inclusive,16-way	N	58 ± 32, 74%	1.07 ± 0.45, 26%	15 ± 1.3, 100%	14 ± 2.8, 100%	33 ± 41, 6.1%	0.98 ± 2.3, 96%
		Y	44 ± 21, 78%	0.21 ± 0.53, 50%	0.64 ± 0.16, 100%	0.68 ± 0.29, 100%	N/A	0.26 ± 1.2, 89%
i7-6700	inclusive, 16-way	N	82 ± 66, 79%	1.38 ± 2.9, 24%	18 ± 4.4, 73%	18 ± 7.4, 74%	10.41 ± 5.3, 5.3%	1.3 ± 3.6, 92%
		Y	46 ± 25, 85%	0.25 ± 0.06, 66%	2.4 ± 2.9, 100%	1.3 ± 0.88, 100%	N/A	0.29 ± 0.76, 100%
i7-9700	inclusive,12-way	N	115 ± 92, 85%	0.67 ± 0.78, 41%	54 ± 126, 88%	42 ± 100, 92%	159 ± 8.5, 2.0%	1.0 ± 2.2, 99%
		Y	87 ± 99, 83%	0.14 ± 0.14, 40%	2.2 ± 2.8, 100%	1.4 ± 1.6, 100%	N/A	0.24 ± 0.82, 100%
i7-11700	inclusive,16-way	N	642 ± 586, 24%	1.9 ± 0.82, 10%	23 ± 2.6, 45%	21 ± 8.0, 52%	3.0 ± 1.4, 0.4%	1.7 ± 0.85, 59%
		Y	446 ± 424, 47%	0.23 ± 0.041, 59%	1.6 ± 2.3, 100%	0.74 ± 0.15, 100%	N/A	0.18 ± 0.03, 86%
Xeon-4110[f]	non-inclusive,11-way	N	219 ± 203, 19%	2.9 ± 1.0, 71%	23 ± 7.3, 67%	23 ± 8.2, 75%	125 ± 133, 12%	1.5 ± 3.8, 98%
		Y	159 ± 205, 17%	0.40 ± 0.14, 86%	5.0 ± 2.4, 36%	4.7 ± 2.6, 44%	N/A	0.27 ± 0.025, 100%
Xeon-8280[f]	non-inclusive,11-way	N	380 ± 345, 28%	7.5 ± 7.8, 38%	49 ± 35, 58%	54 ± 76, 60%	441 ± 130, 20%	4.5 ± 5.5, 99%
		Y	299 ± 289, 26%	1.1 ± 0.48, 71%	3.5 ± 2.4, 64%	3.9 ± 2.7, 52%	N/A	0.52 ± 0.11, 100%

[a] The average latency and the standard deviation are estimated using the latency results of only the successful tests, because the average latency of the unsuccessful tests can be unreasonably affected by the retry parameter and suffers from large variance.
[b] Reuse the code open-sourced by [27] with manual parameter optimization without using the *reuse* technique.
[c] Each PPP or CTPP test is allowed to retry for four times in maximal if failing to find an eviction set.
[d] Reuse the code open-sourced by [19]. Each CT test is allowed to extend the search for eight extra addresses to cope with noise. The CT-fast algorithm is a full-blown one with an extended search of W addresses in maximum.
[e] The original code open-sourced by [28] fails to run on most of our test platforms. We implement our own version according to [28] and relax the condition for success. Each test collects 100 addresses and is counted as a success if these 100 addresses form an eviction set. Results with huge pages are not collected as huge pages are not supported in the original implementation.
[f] The eviction sets found for Xeon processors exploit the inclusive directory in the LLC.

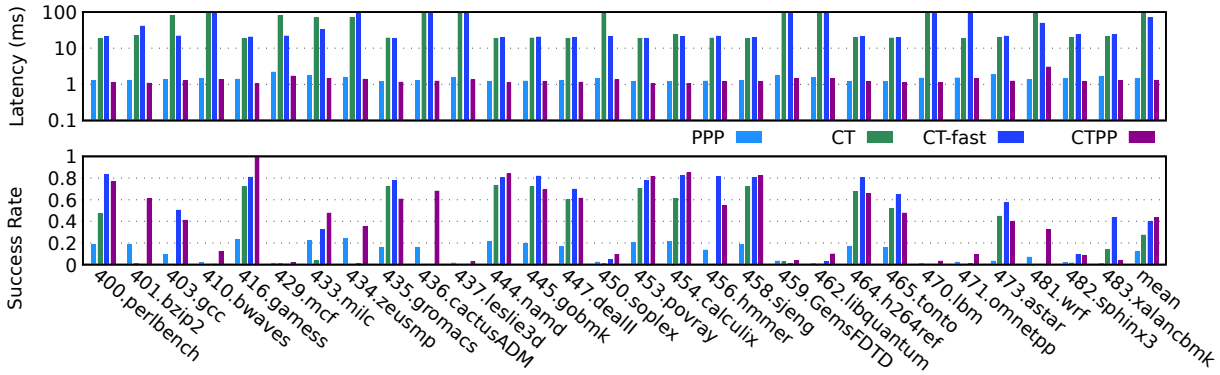


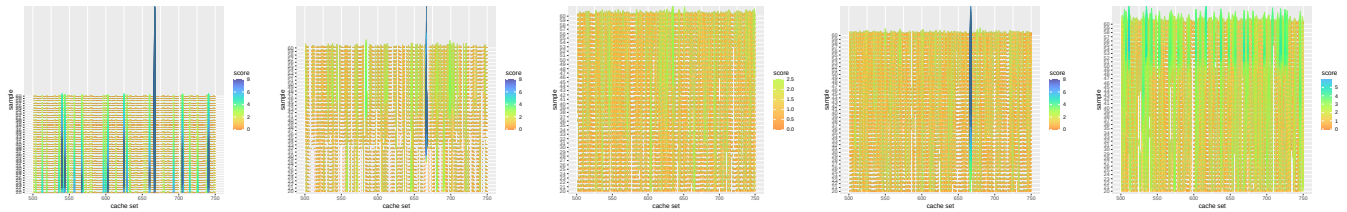
Figure 10: Find eviction sets on i7-6700 while running the SPEC CPU 2006 benchmark as background noise. Each test has been repeated for 10000 times. Huge pages are not used.

accesses. Both CT and CT-fast achieve high success rates in finding eviction sets. However, their search latency is almost ten times longer compared with PPP. As analyzed in Section 3.1, the CT variant algorithms suffer from the unnecessary accesses to extra number of congruent addresses while PPP achieves the low latency thanks to its manipulation of cache state according to the RRIP replacement policy. However, the inefficient prune of PPP leads to low success rate. Comparing with all the existing algorithms, CTPP achieves low search latency and high success rate at the same time. The low search latency comes from its exploitation of the replacement policy similar with the PPP algorithm. The high success rate thanks to the perfect prime sets obtained by CT.

Regarding the differences between architectures, older generations of consumer processors (before the 11th Gen) are easier to attack than the 11th Gen. All algorithms suffer from lower success rates on i7-11700 but CTPP still achieves the highest success rate of 59%. There is no significant difference between the two generations of Xeon processors. The CT variants (CT and CT-fast) and CTPP appear to achieve similarly high success rates while CTPP is best of providing a success rate approaching 100%. The search latency on Xeon-8280 is significantly longer than on Xeon-4110 due to its large LLC. Xeon-4110 is a 16-core processor with an 11MB LLC while the Xeon-8280 platform under test is a dual-socket server mounted with two 28-core Xeon-8280 processors and an LLC of accumulated ~38MB.

6.2 Noise Tolerance

Tolerance to background noise is important for the success of side-channel attacks in practical attacks. To examine the capability of noise tolerance, we run the four fastest search algorithms, i.e., PPP,



(a) GE (0–100) (b) PPP (0–30, 31–61) (c) CT (0–63) (d) CT-fast (0–32, 33-75) (e) CTPP (13–26, 27–41, 42–48, 49–60)

Figure 11: The detector scores of cache sets during the search for eviction sets on a 1024-set 16-way LLC using GE (a), PPP (b), CT (c), CT-fast (d) and CTPP (e). The score distribution is sampled (re-calculated) every 4 accesses per cache set. Samples 20–60 are depicted in the figure, omitting the initial 20 warm-up samples. For the best visibility of figures, the value of score is capped at 8.0. The same color scale is used for all sub-figures for visual comparison. The targeted cache set is fixed to cache set 666. The scores of cache set 500–750 are depicted including the targeted cache set. The samples corresponding to each search are labeled in the sub-figure caption in parentheses, e.g., 4 searches are finished in the 40 samples shown for CTPP (e).

CT, CT-fast and CTPP, on an i7-6700 processor while running one SPEC CPU 2006 benchmark [7] as the background noise. As shown in Figure 10, the success rate of all algorithms is similarly reduced by half while CTPP achieves the best of 41%. The search latency of PPP and CTPP is almost the same with or without the background noise but it is prolonged by around four times for CT and CT-fast. CTPP incurs the lowest search latency of 1.3 ms in average (with or without noise).

Regarding the performance when different background noise is applied by individual SPEC CPU 2006 benchmarks, the fast search algorithms are sensitive to memory heavy benchmarks, such as 410.bwaves, 437.leslie3d, 450.soplex, 459.GemsFDTD, 462.libquantum, 470.lbm, 471.omnetpp, 482.sphinx3. The success rate of all algorithms drops to almost zero. One exception is 433.milc, where PPP, CT-fast and CTPP still work with a success rate higher than 10%, and CTPP achieves a much higher rate of 45%. For benchmarks with a moderate amount of memory accesses, such as 401.bzip2, 403.gcc, 434.zeusmp, 436.cactusADM, 481.wrf, CTPP's success rate outperforms other search algorithms by a large margin.

In summary, all fast search algorithms for eviction sets are unavoidably sensitive to background noise. CTPP presents the strongest tolerance to background noise and outperforms other algorithms by a large margin when the memory accesses generated by the background noise is moderate.

6.3 Escape Detection

To evaluate the capability of escaping from the detector proposed in [26], we port all fast search algorithms for eviction sets to the same behavioral cache model used in [26] (except for W+W because serialization of write accesses incur no latency penalty in the behavioral cache model). The cache model is configured into a cache hierarchy with two levels of set-associative caches. The L2 cache acts as the LLC, which is randomized and dynamically remapped using EV10+DT.[4] As a visual demonstration, we extract the detector score $az_i(t)$ described in Equation 4 during the active search

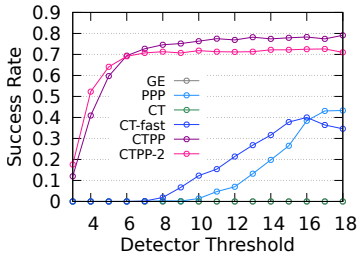


Figure 12: Success rate of finding eviction sets when the EV10+DT detector is enabled. The detector threshold ranges from 3.0 to 18.0. Each test is repeated for 1000 times.

for eviction sets using different algorithms. The score distribution among cache sets for all algorithms is shown in Figure 11.

GE, PPP and CT-fast are clearly exposed by the detector. A high score (actually much larger than 8 as the score is capped) is found on the targeted cache set 666, since too many cache evictions are unevenly incurred on it. For GE, this unbalanced eviction is clearly visible during the whole prune process. The PPP algorithm incurs the unbalanced eviction distribution in the final probe step. A similarly unbalanced distribution is produced by the extended search introduced by the CT-fast algorithm.

CT seems to be the most stealth algorithm. Since congruent addresses are collected by testing a huge number of random addresses, the eviction distribution is statistically even. No detector score ever surpasses 2.5. However, it suffers from the slow speed. It finishes just one search in 64 sample periods (0–63 as labeled in the caption of Figure 11c). Although it can escape from the detector, it is easily defeated by the periodical remap triggered by EV10.

The eviction distribution incurred by CTPP generates higher scores than CT. The highest score can rise to around 5.5, surpassing the default detector threshold of 5.0. However, the speed of CTPP is much faster than all other algorithms. In the same 60 sample periods, CTPP has finished 5 rounds of search. Although it would be eventually detected, it is very likely that an eviction set has already been successfully found and utilized.

[4] EV10: periodically remap every 10 evictions per cache block. DT: trigger extra remaps when an eviction set search algorithm is detected in action. [26]

By applying the full remap scheme (EV10+DT), Figure 12 reveals the drop of success rate of all search algorithms when the detector threshold shrinks from 18.0 to 3.0. Matching with the conclusion in [26], all existing search algorithms fail to work when the threshold is set to the default value of 5.0, but CTPP achieves a high success rate around 60%. Even when the detector threshold is decreased to 3.0, CTPP still works with a 12% success rate. CTPP obviously outperforms all existing algorithms in escaping form detection.

Algorithm 2: CTPP-2, search an eviction set targeting two addresses simultaneously.

Input: x_0, the 1st target address.
Input: x_1, the 2nd target address.
Input: W, number of ways.
Output: $\mathcal{E}$, an eviction set for x_0 or x_1.

```
1  function ctpp-2(x0, x1, W)
2      C ← ∅ // prime set
3      R ← ∅ // probed set
4      E ← ∅ // eviction set
5      prime(LLC) // preparation
6      // CT step
7      access(x0), access(x1)
8      h0 ← true, h1 ← true
9      while h0 or h1 do
10         a ← random()
11         access(a)
12         C ⋃ {a}
13         h0 ← h0 and probe(x0)
14         h1 ← h1 and probe(x1)
15     end
16     // probe step
17     access(x0), access(x1)
18     foreach c ∈ C do
19         if not probe(c) then
20             R ⋃ {c}
21         end
22     end
23     // prune step
24     E ← ∅
25     foreach c ∈ R do
26         if probe(c) then
27             E ⋃ {c}
28         end
29     end
30     return E
31 end
```

By a further investigation, we find that the success rate can be improved by searching an eviction set targeting multiple target addresses. A revised algorithm that targets two addresses (x_0 and x_1) simultaneously is described by Algorithm 2, where the revised parts are highlighted in red. Instead of collecting addresses just enough to form a perfect prime set for a single target address, extra addresses are collected for evicting two target addresses. The following probe and prune steps remain the same with the original algorithm except for accessing both targets at the beginning of probe. If successful, the collection of addresses returned by the algorithm, $\mathcal{E}$, is an eviction set for either x_0 or x_1,[5] or even both by a 15% probability. Labeled as "CTPP-2" in Figure 12, when the detector threshold drops to lower than 6.0, CTPP-2 achieves a higher success rate than CTPP. For the lowest threshold of 3.0, CTPP-2 raises the success rate from 12% to 18% (50% improvement). The underlying reason is the increased number of type (a) addresses when a prime set is collected for two target addresses. These extra number of type (a) addresses increase the number of cache sets experiencing with evictions, leading to a more even distribution of evictions than the original algorithm. Detector scores rise slower and becomes less likely to surpass the detector threshold.

[5] An eviction set that works with a 50% probability is still better than no eviction set, because the former can be used to leak information in a statistical manner. [2]

7 DISCUSSION

Similar with many other eviction set search algorithms [27, 28], we have tested CTPP on two AMD processors (Ryzen-7 5700G and Threadripper 3955WX) and verified that CTPP fails to work. Recent AMD processors adopt a non-inclusive LLC with a snooping coherence protocol, lacking the inclusiveness required by CTPP.

Randomized skewed caches [4, 22, 24, 29, 35] have recently been considered as a promising defense against conflict-based cache side-channel attacks, although none of the existing commercial processors adopts a skewed cache in the cache hierarchy. Directly applying CTPP to randomized skewed caches does not work in its current form. There are at least two reasons: One is the re-accessing of the target address (line 7 in Algorithm 1) may fail to strike on the correct skew occupied by the prime set, resulting in the removal of most addresses partially congruent with the target as they hit in the cache. The other one is the thrashing access pattern required by CTPP for retaining the eviction set during probe. Even if the re-accessing of the target address strikes on the correct skew and one of the partially congruent addresses is evicted, it may not cause a conflict on the same (correct) skew when this partially congruent address is re-accessed later. Theoretically, an attacker may still get a chance by enforcing conflicts on the correct skew by repeated flushing and re-accessing, but we cannot easily predict whether CTPP remains effective. We leave this research exploration to one of our future works.

Regarding potential counter-measures on set-associative LLC utilized by existing Intel processors, some hardware changes may significantly reduce the success rate of CTPP. For inclusive LLCs, introducing randomness in the replacement policy breaks the matching access order observed by the LLC, which leads to potential removal of some addresses belonging to the eviction set in the probe step. This is also effective for the non-inclusive LLC using an inclusive directory, but relaxing the strict inclusive condition for shared cache blocks [38] would potentially nullify the usefulness of eviction sets.

8 CONCLUSION

A new eviction set search algorithm called *Conflict Testing with Probe+Prune* (CTPP) is proposed in this paper. It combines the benefits of both CT and PPP algorithms. Comparing with existing algorithms, CTPP achieves the lowest latency of less than five milliseconds in finding an eviction set on current Intel processors, escapes from the detector proposed in [26] by a 60% probability when all other algorithms fail to work, and presents the strongest tolerance to environmental noise as verified by running the SPEC CPU 2006 benchmark in the background.

AVAILABILITY

CTPP is available at https://github.com/comparch-security/ctpp.

ACKNOWLEDGMENTS

This work was partially supported by the National Natural Science Foundation of China under grant No. 61802402 and 62172406, the CAS Pioneer Hundred Talents Program. Any opinions, findings, conclusions, and recommendations expressed in this paper are those of the authors and do not necessarily reflect the views of the funding parties.

REFERENCES

[1] Christoph Berg. 2006. PLRU cache domino effects. In *Proceedings of the International Workshop on Worst-Case Execution Time Analysis (WCET'06)*.

[2] Thomas Bourgeat, Jules Drean, Yuheng Yang, Lillian Tsai, Joel Emer, and Mengjia Yan. 2020. CaSA: End-to-end quantitative security analysis of randomly mapped caches. In *Proceedings of the IEEE/ACM International Symposium on Microarchitecture (MICRO'20)*. 1110–1123.

[3] Daniel Genkin, Lev Pachmanov, Eran Tromer, and Yuval Yarom. 2018. Drive-by key-extraction cache attacks from portable code. In *Proceedings of the International Conference on Applied Cryptography and Network Security (ACNS'18)*. Springer, 83–102.

[4] Lukas Giner, Stefan Steinegger, Antoon Purnal, Maria Eichlseder, Thomas Unterluggauer, Stefan Mangard, and Daniel Gruss. 2023. Scatter and Split Securely: Defeating Cache Contention and Occupancy Attacks. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'23)*. IEEE.

[5] Ben Gras, Kaveh Razavi, Erik Bosman, Herbert Bos, and Cristiano Giuffrida. 2017. ASLR on the line: Practical cache attacks on the MMU. In *Proceedings of the Network and Distributed System Security Symposium (NDSS'17)*. Internet Society.

[6] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer.js: A remote software-induced fault attack in JavaScript. In *Proceedings of the Conference on Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA'16)*. Springer, 300–321.

[7] John L. Henning. 2006. SPEC CPU2006 benchmark descriptions. *SIGARCH Computer Architecture News* 34, 4 (2006), 1–17.

[8] Ralf Hund, Carsten Willems, and Thorsten Holz. 2013. Practical timing side channel attacks against kernel space ASLR. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'13)*. IEEE, 191–205.

[9] Mehmet Sinan İnci, Berk Gulmezoglu, Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2016. Cache attacks enable bulk key recovery on the cloud. In *Proceedings of the Conference on Cryptographic Hardware and Embedded Systems (CHES'16)*. 368–388.

[10] Aamer Jaleel, Eric Borch, Malini Bhandaru, Simon C. Steely Jr., and Joel Emer. 2020. Achieving non-inclusive cache performance with inclusive caches: Temporal locality aware (TLA) cache management policies. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'10)*. IEEE.

[11] Aamer Jaleel, Kevin B. Theobald, Simon C. Steely Jr., and Joel S. Emer. 2010. High performance cache replacement using re-reference interval prediction (RRIP). In *Proceedings of the International Symposium on Computer Architecture (ISCA'10)*. ACM, 60–71.

[12] Paul Kocher, Daniel Genkin, Daniel Gruss, Werner Haas, Mike Hamburg, Moritz Lipp, Stefan Mangard, Thomas Prescher, Michael Schwarz, and Yuval Yarom. 2019. Spectre attacks: Exploiting speculative execution. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'19)*. 19–37.

[13] Moritz Lipp, Michael Schwarz, Daniel Gruss, Thomas Prescher, Werner Haas, Anders Fogh, Jann Horn, Stefan Mangard, Paul Kocher, Daniel Genkin, Yuval Yarom, and Mike Hamburg. 2018. Meltdown: Reading kernel memory from user space. In *Proceedings of the USENIX Security Symposium (Security'18)*. USENIX Association, 973–990.

[14] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-level cache side-channel attacks are practical. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'15)*. IEEE.

[15] Clémentine Maurice, Nicolas Le Scouarnec, Christoph Neumann, Olivier Heen, and Aurélien Francillon. 2015. Reverse Engineering Intel Last-Level Cache Complex Addressing Using Performance Counters. In *Proceedings of the International Symposium on Research in Attacks, Intrusions, and Defenses (RAID'15)*. Springer, 48–65.

[16] Yossef Oren, Vasileios P. Kemerlis, Simha Sethumadhavan, and Angelos D. Keromytis. 2015. The spy in the sandbox: Practical cache attacks in JavaScript and their implications. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'15)*. ACM Press.

[17] Colin Percival. 2005. Cache missing for fun and profit.

[18] Antoon Purnal, Lukas Giner, Daniel Gruss, and Ingrid Verbauwhede. 2021. Systematic analysis of randomization-based protected cache architectures. In *Proceedings od the IEEE Symposium on Security and Privacy (S&P'21)*. IEEE, 987–1002.

[19] Antoon Purnal, Furkan Turan, and Ingrid Verbauwhede. 2021. Prime+Scope: Overcoming the observer effect for high-precision cache contention attacks. In *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security (CCS'21)*. ACM, 2906–2920.

[20] Antoon Purnal and Ingrid Verbauwhede. 2019. Advanced profiling for probabilistic Prime+Probe attacks and covert channels in ScatterCache. (2019). arXiv:1908.03383v1 [cs.CR]

[21] Moinuddin K. Qureshi. 2018. CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'18)*. 775–787.

[22] Moinuddin K. Qureshi. 2019. New attacks and defense for encrypted-address cache. In *Proceedings of the International Symposium on Computer Architecture (ISCA'19)*. ACM, 360–371.

[23] Thomas Ristenpart, Eran Tromer, Hovav Shacham, and Stefan Savage. 2009. Hey, you, get off of my cloud: Exploring information leakage in third-party compute clouds. In *Proceedings of the ACM Conference on Computer and Communications Security (CCS'09)*. ACM, 199–212.

[24] Gururaj Saileshwar and Moinuddin K. Qureshi. 2021. MIRAGE: Mitigating conflict-based cache attacks with a practical fully-associative design. In *Proceedings of the USENIX Security Symposium (Security'21)*. USENIX Association, 1379–1396.

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

[26] Wei Song, Boya Li, Zihan Xue, Zhenzhen Li, Wenhao Wang, and Peng Liu. 2021. Randomized last-level caches are still vulnerable to cache side-channel attacks! But we can fix it. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'21)*. IEEE, 955–969.

[27] Wei Song and Peng Liu. 2019. 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'19)*. USENIX Association, 427–442.

[28] Jan Philipp Thoma and Tim Güneysu. 2022. Write me and I'll tell you secrets — Write-after-write effects on Intel CPUs. In *Proceedings of the International Symposium on Research in Attacks, Intrusions and Defenses (RAID'22)*. ACM.

[29] Thomas Unterluggauer, Austin Harris, Scott Constable, Fangfei Liu, and Carlos V. Rozas. 2022. Chameleon cache: Approximating fully associative caches with random replacement to prevent contention-based cache attacks. In *Proceedings of the IEEE International Symposium on Secure and Private Execution Environment Design (SEED'22)*. IEEE, 13–24.

[30] Stephan van Schaik, Kaveh Razavi, Ben Gras, Herbert Bos, and Cristiano Giuffrida. 2017. RevAnC: A framework for reverse engineering hardware page table caches. In *Proceedings of the European Workshop on Systems Security (EuroSec'17)*.

[31] Tarunesh Verma, Achilleas Anastasopoulos, and Todd M. Austin. 2022. These aren't the caches you're looking for: Stochastic channels on randomized caches. In *Proceedings of the IEEE International Symposium on Secure and Private Execution Environment Design (SEED'22)*. IEEE, 37–48.

[32] Pepe Vila, Pierre Ganty, Marco Guarnieri, and Boris Köpf. 2020. CacheQuery: Learning replacement policies from hardware caches. In *Proceedings of the ACM SIGPLAN Conference on Programming Language Design and Implementation (PLDI'20)*. ACM.

[33] Pepe Vila, Boris Köpf, and Jose Morales. 2019. Theory and practice of finding eviction sets. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'19)*. IEEE, 39–54.

[34] Zhenghong Wang and Ruby B. Lee. 2008. A novel cache architecture with enhanced performance and security. In *Proceedings of the Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'08)*. IEEE Computer Society, 83–93.

[35] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. 2019. ScatterCache: Thwarting cache attacks via cache set randomization. In *Proceedings of the USENIX Security Symposium (Security'19)*. USENIX Association, 675–692.

[36] Henry Wong. 2013. Intel Ivy Bridge cache replacement policy. http://blog.stuffedcow.net/2013/01/ivb-cache-replacement/.

[37] Mengjia Yan, Bhargava Gopireddy, Thomas Shull, and Josep Torrellas. 2017. Secure hierarchy-aware cache replacement policy (SHARP): Defending against cache-based side channel atacks. In *Proceedings of the Annual International Symposium on Computer Architecture (ISCA'17)*. ACM, 347–360.

[38] Mengjia Yan, Read Sprabery, Bhargava Gopireddy, Christopher W. Fletcher, Roy H. Campbell, and Josep Torrellas. 2019. Attack directories, not caches: Side-channel attacks in a non-inclusive world. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P'19)*. IEEE, 888–904.

[39] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A high resolution, low noise, L3 cache side-channel attack. In *Proceedings of the USENIX Security Symposium (Security'14)*. USENIX Association, 719–732.