Prime+Scope: Overcoming the Observer Effect for High-Precision Cache Contention Attacks

Antoon Purnal
imec-COSIC, KU Leuven

Furkan Turan
imec-COSIC, KU Leuven

Ingrid Verbauwhede
imec-COSIC, KU Leuven

ABSTRACT

Modern processors expose software to information leakage through shared microarchitectural state. One of the most severe leakage channels is cache contention, exploited by attacks referred to as Prime+Probe, which can infer fine-grained memory access patterns while placing only limited assumptions on attacker capabilities.

In this work, we strengthen the cache contention channel with a near-optimal time resolution. We propose Prime+Scope, a cross-core cache contention attack that performs back-to-back cache contention measurements that access only a single cache line. It offers a time resolution of around 70 cycles (25ns), while maintaining the wide applicability of Prime+Probe. To enable such a rapid measurement, we rely on the deterministic nature of modern replacement policies and their (non-)interaction across cache levels. We provide a methodology to, essentially, prepare multiple cache levels simultaneously, and apply it to Intel processors with both inclusive and non-inclusive cache hierarchies. We characterize the resolution of Prime+Scope, and confirm it with a cross-core covert channel (capacity up to 3.5 Mbps, no shared memory) and an improved attack on AES T-tables. Finally, we use the properties underlying Prime+Scope to bootstrap the construction of the eviction sets needed for the attack. The resulting routine outperforms state-of-the-art techniques by two orders of magnitude.

Ultimately, our work shows that interference through cache contention can provide richer temporal precision than state-of-the-art attacks that directly interact with monitored memory addresses.

CCS CONCEPTS

• **Security and privacy → Software and application security**; *Systems security*.

KEYWORDS

Cache Attacks, Cache Side-Channels, Microarchitecture

ACM Reference Format:
Antoon Purnal, Furkan Turan, and Ingrid Verbauwhede. 2021. Prime+Scope: Overcoming the Observer Effect for High-Precision Cache Contention Attacks. In *Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS '21), November 15–19, 2021, Virtual Event, Republic of Korea.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3460120.3484816

CCS '21, November 15–19, 2021, Virtual Event, Republic of Korea

ACM ISBN 978-1-4503-8454-4/21/11…$15.00
https://doi.org/10.1145/3460120.3484816

1 INTRODUCTION

Modern processors comprise many hardware components that, transparently to the programmer, enhance average software performance. Several such components, with the cache hierarchy as the leading example, may be shared across software-defined security boundaries. Through their access patterns, programs may unwillingly encode secret information into shared cache state, which can be extracted by a co-located adversary using a timing side channel. In particular, she first prepares the cache state and afterwards measures it to infer the changes produced by other programs.

Several techniques have been proposed to prepare and measure the cache state. While some can only monitor accesses to shared memory between attacker and victim, or require specific processor features, other techniques have no such prerequisites and are purely based on contention for cache resources. By targeting core-shared cache levels, such as the last-level cache (LLC) [31, 37], or the coherence directory (CD) [65], an attacker can measure *cache contention* for victim programs running on other processor cores.

Known mostly as Prime+Probe, cache contention attacks are widely applicable. The contention channel leaks information across virtual machine boundaries [31, 37, 47], to and from sandboxed code (e.g., in the browser [41, 53]), or even over the network [33]. It has been used to extract sensitive information of various kinds, such as cryptographic keys [42], user input [33, 47], kernel information [29] and browsing behavior [53]. It also enables establishing covert channels [37–39] and, recently, transient execution attacks [22, 48].

An important metric of cache attack techniques is their *temporal resolution*, i.e., the precision with which they can localize victim memory accesses in the time domain. The finer the resolution of the attack, the greater the visibility into data accesses and control flow of victim applications. This is of special importance in the general setting, where the attacker monitors victim behavior asynchronously and victim accesses may occur at any given time.

In case of insufficient time precision, prior works slow down the victim application (e.g., [3, 4, 9, 21]), or interrupt it heavily (e.g., [27, 40]), to amplify secret-dependent time differences. Instead of such performance degradation of the victim, which may not always be possible, this work pursues the opposite direction and investigates whether the time precision of cache attacks can be improved (and optimized). In particular, we identify two key challenges to enhance the time resolution of state-of-the-art cache contention attacks.

First, the main challenge to improve the precision of cache attacks, in general, is the *observer effect*, i.e., the phenomenon where the act of measuring a system affects its state. Many techniques suffer from it, often requiring the state change of every measurement to be undone before the next one can be performed. To minimize the influence of this effect, cache attacks are often discretized along the time axis in *windows* of fixed duration (e.g., [3, 37, 65, 67, 69]). However, this places fundamental limits on their time resolution.

Second, the cache contention channel, in particular, faces another bottleneck. For Prime+Probe, each probe accesses as many cache lines as the associativity of the target structure, e.g., at least 11 ways for core-shared caches on modern Intel CPUs. Therefore, the time resolution is structurally bounded by the time it takes to access all these lines, even if the observer effect were to be overcome.

In this paper, we seek to optimize the resolution of Prime+Probe-style attacks. To this end, we ask the following main questions:

Is it possible to bypass the observer effect? Can contention be inferred by repeatedly measuring the access latency of a single cache line?

In this work, we make the surprising observation that the cache contention channel can have a higher time resolution than techniques that access monitored addresses directly. We propose Prime+Scope, a high-precision cross-core cache contention attack, whose measurement is both *repeatable* (i.e., the cache state does not need to be reinstated after every measurement), and essentially *optimal* (i.e., it performs a single memory access). Prime+Scope can monitor events asynchronously with a precision in the order of 25ns, significantly outperforming comparable techniques. At the same time, Prime+Scope inherits the general applicability of Prime+Probe.

Prime+Scope prepares the cache more precisely than traditional cache contention attacks. We obtain fast and effective Prime patterns using both an automated and a handcrafted methodology (resp. for inclusive and non-inclusive Intel LLCs). In the end, we find Prime+Scope to apply to all tested Intel CPUs of the last decade.

To confirm the superior time precision of Prime+Scope, we perform a quantitative comparison with state-of-the-art techniques. We also implement a cross-core covert channel on a last-level cache (LLC) and a coherence directory (CD). Symbols are encoded temporally in slots of no more than 80-120 processor cycles. The LLC/CD channels reach a capacity of 3.5 Mbps and 3.1 Mbps, respectively.

We evaluate Prime+Scope on a known-vulnerable AES implementation. With its fine temporal precision, it can extract the key material with 5x-25x fewer encryptions than Prime+Probe.

Finally, we bootstrap our newly discovered primitive to create a straightforward, portable and linear-time eviction set construction routine, which outperforms previous techniques by 100-600x.

Summarized, this paper makes the following main contributions:

- We present Prime+Scope, a generic cross-core cache contention primitive with near-optimal temporal resolution.
- To prepare the cache for continuous measurement, we propose PrimeTime, a methodology to find efficient Prime patterns.
- We evaluate Prime+Scope using micro-benchmarks, a high-capacity covert channel, and a high-precision attack on AES.
- Using the principles underlying Prime+Scope, we present fast and simple routines to construct LLC/CD eviction sets.

We have disclosed our findings to Intel. To facilitate reproduction of our research, artifacts are made available at

`https://www.github.com/KULeuven-COSIC/PRIME-SCOPE`

This article is organized as follows. Section 2 provides the necessary background. Section 3 explores the conditions for back-to-back cache measurements. Section 4 exposes Prime+Scope, our main result. Section 5 covers the efficient preparation of the cache state, and Section 6 evaluates Prime+Scope for micro-benchmarks and concrete examples. Section 7 positions our findings, Section 8 discusses limitations and countermeasures, and Section 9 concludes.

2 PRELIMINARIES

2.1 Caches

To overcome the comparatively high latency of memory lookups, caches are buffers that keep soon-to-be used data close to the CPU. Caches operate on fixed-size (e.g., 64 bytes) memory blocks called *cache lines*, and are typically set-associative, referring to their organization along *sets* and *ways*. Cache lines are mapped to sets based on their memory address, and addresses mapping to the same set are called *congruent*. The maximal number of congruent lines that can reside in the cache at any given time is determined by the number of ways W, also referred to as the *associativity*.

When caching a new line exceeds the associativity, one line in the set is *evicted*; in this paper, we refer to that line as the *eviction candidate (EVC)*. The EVC is determined by the *replacement policy*, which is implemented by a (complex) state machine at the set-level, for which the state transitions depend on the accesses to the set.

Contemporary Intel processors feature a three-level cache hierarchy, where the access latency increases along with the distance from the CPU. When a CPU core references a memory address, the cache line containing this address is retrieved from the closest cache level that has a valid copy. The first two levels (L1 and L2) are organized separately for every core, while the last-level cache (L3, or LLC) is shared among all the cores. The majority of Intel processors have *inclusive* LLCs, meaning that cache lines present in the L1 and L2 caches must also be present in the LLC. However, recent Intel servers feature higher core counts and larger private caches, prompting the adoption of *non-inclusive* LLCs. In such cache hierarchies, the LLC may or may not contain lines that are present in L1 or L2, reducing the storage overhead due to inclusion.

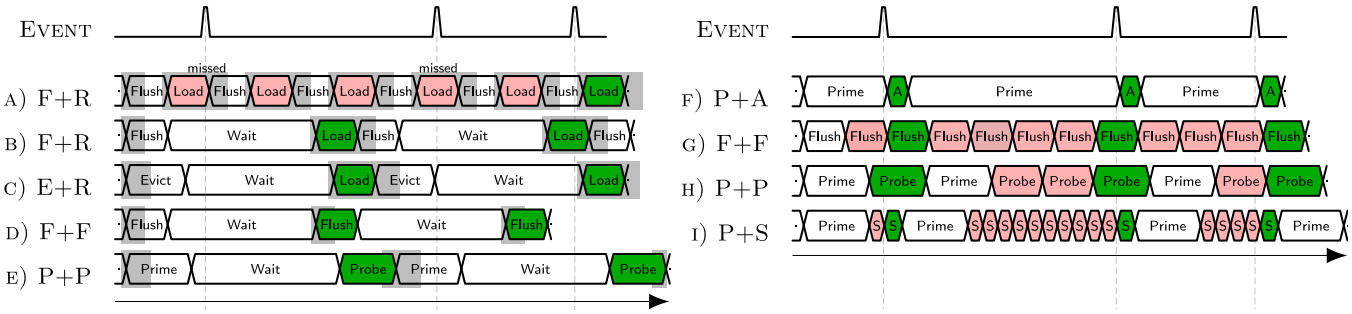
2.2 Cache Side-Channel Attacks

Over the last years, several techniques have been proposed to infer memory access patterns by other programs through observation of shared cache state. The two most prominent attack classes are represented, respectively, by Flush+Reload and Prime+Probe. In this overview, and in the paper, we focus on attacks across cores.

Flush+Reload-style techniques. If the memory address to be monitored exists in memory shared with the attacker, she is able to access it directly, allowing to infer memory activity by other processes at cache-line granularity. The representative technique for this attack class is Flush+Reload [27, 67], which removes (flushes) the cache line containing a target address from the cache, and later determines if the victim accessed it by its reload time. If cache flushing is not available, Evict+Reload [26] replaces it with eviction.

Provided that the time until completion of the `clflush` instruction depends on the presence of the target in the cache, it can be used to both prepare and measure the cache state. This technique, referred to as Flush+Flush [25], has a higher time resolution than Flush+Reload. However, the more subtle time-dependence of cache flushes results in a comparatively low cross-core accuracy.

The main drawback of Flush+Reload-style attacks is their structural dependence on shared memory with a victim, which is harder to obtain for an attacker than only co-location. Moreover, a cache flush instruction may also not be available in restricted contexts, e.g., in the browser [5, 25], or generally for unprivileged processes [34].



(i) Windowed techniques cope with blind spots by waiting between preparation and measurement, lowering their time precision

(ii) Windowless techniques measure the cache state continuously without waiting, and revert the state only after detecting an event

Figure 1: Cache Manipulation techniques in the windowed vs. windowless paradigms.

Prime+Probe-style techniques. Cache contention attacks, often synonymously referred to as Prime+Probe attacks, prime a full cache set and measure the time it takes. Activity in this cache set by other processes will evict one or more of the attacker's lines, which is reflected in a higher latency to complete the prime.

For a cross-core attack, an adversary generally targets an inclusive structure shared with the victim. The inclusive property guarantees the eviction of congruent addresses from the victim's private caches, ensuring that future victim accesses to them indeed generate contention on the measured set. In cache hierarchies with an inclusive last-level cache (LLC), this requirement is readily obtained [31, 37]. For non-inclusive Intel caches, a suitable structure has been found in the coherence directory (CD) [65], which keeps track of lines in all the L2 caches. It is organized in sets, like the LLC, with the same function to index memory addresses into sets and slices. Instead of priming the LLC, the attacker primes the CD, evicting the target address from the CD and, due to its inclusion property, also from the victim's private L1/L2 caches. When lines are evicted from the CD, they are moved to the LLC [65].

To measure contention on the LLC (or CD), an attacker needs to obtain memory addresses that are mapped to the same set. In the presence of unknown physical address bits or cache slices, these so-called *eviction sets* need to be obtained at runtime [37, 60, 65].

As a variant of Prime+Probe, Prime+Abort [18] is a contention-*informed* attack using Intel TSX. As TSX transactions are aborted upon eviction of certain lines from the LLC, it is amenable to measure LLC contention, as attack [18] or defense [23]. Intel TSX may not be exposed to an attacker (e.g., from the browser), may be disabled for security reasons [30], or may not be available at all.

Cache contention attacks offer a spatial granularity of sets, which is inferior to the cache-line granularity of shared-memory attacks. However, due to the large number of sets in modern LLCs/CDs, the spatial information encoded in cache contention is still quite large.

3 CACHE MANIPULATION PARADIGMS

Assume an attacker wants to spy on a cross-core event, i.e., one or more memory accesses by a victim program running on another CPU core. All cache attack techniques first *prepare* the cache state, and then *measure* it to infer the presence or absence of an event.

This section first revisits why some techniques need to allocate a waiting period between preparation and measurement, essentially partitioning the time axis into *windows*. Then, it examines the conditions under which a windowless paradigm can be adopted.

3.1 Windowed Paradigm

Blind Spots. To see why cache attacks are often organized in discrete time windows, consider the traces in Figure 1i. The first Flush+Reload trace (Figure 1i-A) continuously flushes and reloads a target. Such an application of Flush+Reload fails to detect many events. In particular, events that occur during the period slightly before the Reload, until the Flush has evicted the target, remain undetected [3, 67]. We refer to such a period as a *blind spot*.

To reduce the effect of blind spots on the detection rate, a wait stage may be inserted, i.e., a predetermined idle period between preparation and measurement. As in Figure 1i-B, such an organization detects the events that occur during the wait stage.

Other techniques, like Evict+Reload, Flush+Flush, and Prime+Probe, can also be instantiated like this (cf. Figure 1i-C-D-E). Evict+Reload behaves similarly to Flush+Reload, but has a larger blind spot as cache eviction is slower than flushing. In Section 3.2, we will see which techniques can be used without blind spots.

Resolution. The temporal resolution of windowed techniques is limited by the combined duration of the preparation, wait and measurement stages. In particular, the waiting period marks a trade-off between the accuracy and resolution of the attack. The larger the blind spot, the lower the resolution for the same detection accuracy.

Despite this limitation, windowed techniques such as Flush+Reload can be very powerful in practice. For instance, blind spots can be bypassed when the attacker controls the timing of the event (e.g., by synchronizing [12, 32, 42, 61] or interleaving [27, 68] with the victim). The limitation is also attenuated for infrequent events (e.g., user behavior [26, 41]), or when lower detection rates are tolerable (e.g., to profile a binary [26] or capture traces [28]). The miss probability may also be reduced by targeting events that reference the same line multiple times, e.g., loops [67] or function calls [8].

3.2 Windowless Paradigm

To understand how some techniques can increase the time resolution by avoiding windows [18, 56], we identify the two sources of blind spots. Both sources are an expression of the observer effect, i.e., the attacker perturbs the cache state by measuring it.

#1: Non-preserving. We refer to a cache measurement as *preserving* when, in the absence of an event, the relevant cache state before and after the measurement is equivalent. If the measurement is not preserving, monitoring cannot continue without undoing the changes caused by the measurement. Hence, non-preserving measurements trigger a repeated preparation phase, which naturally introduces a period of time in which victim events are missed (cf. Figure 1i). For instance, the Reload measurement in Flush+Reload is non-preserving, so it needs to be followed by a Flush, and events occurring at the beginning of the Flush are missed [3, 67]).

#2: Non-concurrent. We refer to a cache measurement as *concurrent* when it detects events that temporally overlap with it. Depending on the degree of overlap between event and measurement, an event coinciding with measurement j may be detected during measurement j or $j+1$, but will not be missed, roughly speaking. For instance, the Reload in Flush+Reload is non-concurrent, as events occurring right before or during the Reload are missed [67].

Non-preserving measurements cannot be concurrent, as the necessary preparation phase erases all relevant state changes, rendering them unobserved. Non-concurrent measurements, even if they are preserving, are a source of blind spots, resulting in the need for a waiting interval to obtain the desired detection accuracy. It should also be noted that measurements can be concurrent on one processor and non-concurrent on another (e.g., Flush, cf. Section 6.1).

Going Windowless. Cache measurements that are preserving and concurrent can be performed back-to-back while maintaining their detection accuracy. As a result, they enable a windowless paradigm that maximizes their time resolution. This paradigm first prepares the relevant cache state, and then continuously measures it until an event is observed. Only upon detection of an event, the preparation phase is repeated to continue monitoring for events.

In Prime+Abort [18], the cache measurement occurs implicitly, through the TSX abort. Hence, it is preserving and concurrent, and has a natural windowless instantiation (cf. Figure 1ii-F). Although it is advertised as a distinguishing feature for Prime+Abort, other cache attack techniques can also avoid intermittent wait stages.

Van Bulck et al. [56] demonstrate a windowless Flush+Flush [25] (cf. Figure 1ii-G). On some platforms, Flush measurements are non-concurrent (cf. Section 6.1). If the detection accuracy is unsatisfactory, one can resort to a windowed instantiation, as in Figure 1i-D.

We note that even Prime+Probe can be windowless [51] (cf. Figure 1ii-H), provided that the Probe measurement does not access more congruent addresses than the associativity W of the target structure. Indeed, it is preserving (if all W lines are simultaneously in the target structure, they will still be after a repeated access) and concurrent (an event will cause a miss on at least one of the attacker's lines at some point, regardless of overlap.)

Time Resolution. The advantage of windowless techniques is that their time precision is only fundamentally determined by the throughput of the measurement phase. Therefore, the duration of the preparation phase is of secondary importance for the resolution, as it only needs to be performed right after detecting an event.

3.3 This Work: Prime+Scope

This work sets out to optimize the resolution of cache-timing attacks, while maintaining only the basic requirements of cache contention to ensure that the technique is future-proof and suitable for restricted environments. In particular, we do not rely on shared memory between attacker and victim, or special ISA or processor features (e.g., `clflush` or Intel TSX). We achieve this by organizing the cache state such that the contention measurement is repeatable, i.e., it is preserving and concurrent, and optimally short, i.e., it consists of a single cache access. We call this technique Prime+Scope, and depict it in Figure 1ii-I. In the following section, we outline its core principles and instantiate it for different cache hierarchies.

4 PRIME+SCOPE

4.1 Threat Model

The adversary assumed in this work is able to execute unprivileged code on the same physical processor as a victim program. The attacker code need not be executed on the same core as the victim code, and the attacker is not assumed to be able to interrupt or otherwise control the victim program. Furthermore, we do not assume that attacker and victim have a shared memory region.

4.2 General Description

As described in Section 2.1, modern cache hierarchies comprise different levels. In what follows, C_S denotes the shared and inclusive cache structure in which contention is to be measured, and C_P denotes one of the attacker's private caches (e.g., the L1 cache).

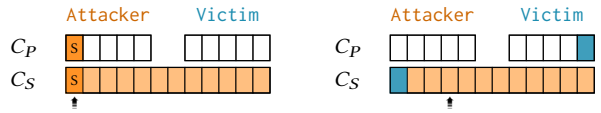
Compared to existing cache contention channels, Prime+Scope has two additional core requirements:

① The eviction candidate in the shared and inclusive target structure (C_S) can be accurately predicted.
② Reads served from a lower-level cache (C_P) do not influence the replacement state of the target structure (C_S).

① **Eviction Candidate.** When a new line is to be installed into a cache set, among all available lines (ways) in the set, a chosen one is replaced with the new line. In this paper, we call that chosen line the Eviction Candidate (EVC). The candidate is determined by the cache replacement policy, which is implemented at the cache-set level as a state machine. For instance, the eviction candidate for the LRU policy is the cache line that has least recently been used. Though modern processors implement more sophisticated replacement policies, they are often deterministic [1, 2, 11]. Therefore, specific access patterns can mold the replacement policy machinery into a state where a chosen cache line is the eviction candidate [11, 61].

Awareness of the EVC in C_S permits to observe contention by only measuring EVC latency, as a new cache line fill evicts the EVC by definition. However, the attacker suffers from the observer effect, i.e., measuring the access latency of the EVC may change it to another line. To make the measurement preserving, Prime+Scope relies on another common property of multi-level caches.

② **Low-Level Reads.** Prior work observed that the replacement state of inclusive Intel LLCs only depends on memory requests served by the LLC, not those served by the lower-level caches [2, 11, 59]. Instead of bypassing this filtering property (e.g., by enforcing



(i) Prime fixes S as the EVC in C_S, which remains the case for following Scope operations.

(ii) The victim access evicts S (=EVC) from C_S and C_P, resulting in high access latency for S.

Figure 2: Working principle of Prime+Scope

L1/L2 misses), our work explicitly relies on it to make the cache measurement preserving, and thus, overcome the observer effect.

Prime+Scope. Based on these two key ingredients, we propose Prime+Scope as a windowless technique to monitor cache contention. It allows an attacker to monitor victim accesses to a predetermined target address in two steps. The Prime step serves two purposes, as in Figure 2i. First, it evicts the target from the C_S using an eviction set. Second, it performs the eviction with a specific access pattern that fixes a chosen line from the eviction set, denoted as scope line (S), as the EVC in C_S (the shared and inclusive high-level structure), *while maintaining its presence in C_P (the lower-level caches).* Afterwards, the preserving and concurrent Scope step continuously fetches S from C_P, and measures the access latency. As it overcomes the observer effect, the relevant cache state remains intact both in C_P and C_S after each Scope.

The described cache state is destroyed when the victim accesses the target address. When this happens, as in Figure 2ii, the newly-allocated target replaces S, as it is the EVC. Because C_S is inclusive of C_P, the copy of S is also evicted from C_P. The next Scope will detect this event through a high access latency to S.

4.3 Instantiation

Cache Hierarchy. For processors with inclusive last-level caches (LLC), such as the majority of Intel's desktop CPUs or server CPUs until 2018, the core-shared and inclusive LLC itself can instantiate C_S, and the core-private L1 caches can instantiate C_P. Most Intel servers since 2018 have non-inclusive LLCs. For such processors, the CD is shared and inclusive [65], and can hence instantiate C_S.

Measurement: Scope. On all tested platforms (cf. Section 5), we found that requests served by C_P indeed preserve the EVC of C_S. The Scope continuously measures the access latency of the scope line S (=EVC), and terminates as soon as the access time exceeds a predetermined threshold, indicating the occurrence of the event. As in Figure 3, Prime+Scope measurements need to detect whether one cache line is served from L1, vs. from RAM (inclusive) or LLC (non-inclusive). In comparison, Prime+Probe measurements must distinguish "W lines in C_S" from "less than W lines in C_S".

Preparation: Prime. Prime+Scope is predicated on the existence and knowledge of a memory access pattern that prepares the cache state for repeated, single-access measurements. Concretely, we are looking for Prime patterns, consisting of accesses to W different addresses that satisfy the following requirements simultaneously:

$\mathbf{R}_A$. have high eviction rate ($> 99.5\%$)
$\mathbf{R}_B$. install a specific line S as the eviction candidate in C_S
$\mathbf{R}_C$. keep the line S in C_P

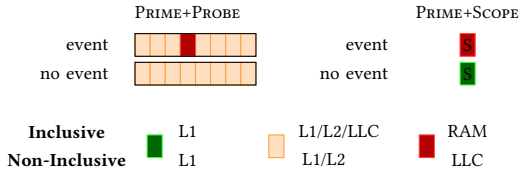


Figure 3: Prime+Probe monitors a full set in LLC (incl.) / CD (non-incl.) and detects eviction to RAM/LLC. Prime+Scope monitors one line in L1, and detects eviction to RAM/LLC.

Requirement $\mathbf{R}_A$ is a traditional requirement for cache contention attacks; otherwise, the victim access might not evict any of the attacker's lines. Requirements $\mathbf{R}_B$ and $\mathbf{R}_C$ are unique to Prime+Scope, so we cannot rely on patterns established in prior work. In particular, we identify the following challenges.

Challenge-LLC: Keeping the EVC in L1. Taken at face value, requirements $\mathbf{R}_B$ and $\mathbf{R}_C$ are contradictory. Assume we want to install line S as the EVC in C_S. While requirement $\mathbf{R}_B$ suggests to access S *less frequently* than the other lines in the eviction set, to ensure it becomes the EVC in C_S, requirement $\mathbf{R}_C$ suggests to access S *more frequently* than the others, to ensure it is kept in C_P.

Challenge-CD: Controlling the EVC. Prior work [65] observed that traditional eviction strategies do not perform well on the CD ($\mathbf{R}_A$). This poses a challenge for Prime+Scope, as controlling the EVC ($\mathbf{R}_B$) is strictly harder than only evicting the target.

5 FINDING EFFICIENT PRIME PATTERNS

This section covers the preparation of the cache state such that subsequent measurements can be performed with a single repeatable cache access. Although the Prime duration has limited impact on the time precision (cf. Section 3.2), we opt to implement the Prime step with fast and accurate access patterns. To find them, we propose PrimeTime, an automated gray-box search methodology.

To understand the nomenclature of Prime patterns, and how PrimeTime finds them, an example pattern is shown in Figure 4 together with its translation into a code snippet. It encodes the access sequence of lines in the eviction set, along with the stride (gap) between indices, and the amount of repetitions. This snippet uses the first line of the eviction set (`evset[0]`) as the scope line S, which becomes the EVC after a successful Prime with the snippet.

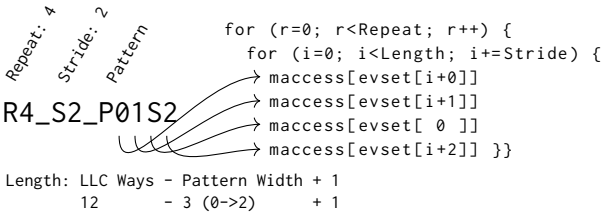


Figure 4: Translation of Prime access patterns into code snippets. After a successful prime, the scope line `S=evset[0]` is the EVC in the LLC, while remaining present in L1.

Table 1: Applicability of Prime+Scope to various CPU microarchitectures, along with a top-ranking access pattern as discovered by PrimeTime. Each pattern achieves (median) EVCr of $> 99.9\%$ at the indicated (median) cycle cost.

CPU	Year	Microarchitecture	LLC type	C_S	W_{C_S}	Prime+Scope	Prime Access Pattern	Cycles
Intel Core i7-9700K	2018	Coffee Lake	inclusive	LLC	12	✓	`R4_S4_P01SS2301233210`	1 332
Intel Core i7-7700K	2017	Kaby Lake	inclusive	LLC	16	✓	`R2_S4_P01SS2SS301230123`	1 255
Intel Core i5-7500	2017	Kaby Lake	inclusive	LLC	12	✓	`R3_S4_P32SS1SS00123`	1 074
Intel Core i7-6700	2015	Skylake	inclusive	LLC	16	✓	`R3_S4_P01SS2SS301230123`	1 694
Intel Core i5-6500	2015	Skylake	inclusive	LLC	12	✓	`R4_S4_P3SS2SS100123`	1 266
Intel Core i7-4790	2013	Haswell	inclusive	LLC	16	✓	`R3_S4_P3SS2SS100123`	1 149
Intel Core i5-4590	2013	Haswell	inclusive	LLC	12	✓	`R2_S1_P01S2S012`	1 221
Intel Core i7-3770	2012	Ivy Bridge	inclusive	LLC	16	✓	`R3_S4_P3SS2SS1032103210`	1 517
Intel Core i5-3450	2012	Ivy Bridge	inclusive	LLC	12	✓	`R2_S1_P2SS10012`	1 216
Intel Core i5-2400	2011	Sandy Bridge	inclusive	LLC	12	✓	`R5_S1_P0S12012`	3 708
Intel Xeon Platinum 8280	2019	CascadeLake-SP	non-incl.	CD	12	✓	alternating pointer-chase	2 970
Intel Xeon Platinum 8180	2017	Skylake-SP	non-incl.	CD	12	✓	alternating pointer-chase	2 750

5.1 Last-Level Cache (LLC)

Main Idea. The key idea of our solution to **Challenge-LLC** relies on property ②. Assume the scope line S to be the first line in the set (line 0). Then, the Prime patterns comprise accesses to W congruent lines, like other prime strategies, but accesses to lines 1 to $W-1$ are interleaved with accesses to S. Due to its frequent usage, S is always served from L1, so it keeps its insertion age in the LLC. The other lines, in contrast, evict each other from L1, and when they are read from the LLC, their age decreases, making them progressively younger. As soon as all other lines become younger than S, the latter is the EVC in the LLC without ever leaving the L1 cache.

Prime Properties. For each candidate Prime pattern, we assess the *eviction rate (EVr)*, i.e., the fraction of successful evictions of the target line. More importantly, we also record the *eviction candidate rate (EVCr)*, i.e., the fraction of attempts where the target line is evicted, *and* the line that will be evicted next is the intended S, and it is still in L1. Finally, we also record the *duration*, i.e., the number of cycles to complete the accesses indicated by the pattern.

It is clear that $EVCr \leq EVr$. While the EVr is the success rate of preparing the cache set for Prime+Probe, the EVCr is the success rate of preparing S for a continuous Scope. Understandably, prior efficient patterns for Prime+Probe typically have a low EVCr, because these patterns are oblivious to the EVC. Hence, good Prime patterns for Prime+Scope differ from those in prior work.

Methodology of PrimeTime. The high-level description of PrimeTime is shown in Algorithm 1. It starts with known access pattern templates, e.g., [24], and mutates them according to given directives. Mutations consist of repeated access to certain (sub-)patterns, permuting access orders, or interleaving accesses to S.
To limit execution time, PrimeTime tests patterns in stages, gradually becoming more restrictive on the patterns that pass to the next stage, both in EVCr and cycle count. In the first stage, we test each pattern with 10 000 repetitions, with loosely defined success criteria. Later stages perform up to a million repetitions, while filtering for the best-performing patterns. A run for a specific microarchitecture takes approximately one hour under our configuration, but this can be scaled in either direction (i.e., speed vs. accuracy). Furthermore, PrimeTime can be extended to cover a larger search space.

PrimeTime on Various Processor Generations. As shown in Table 1, PrimeTime is able to construct effective Prime access patterns on all tested generations of Intel CPUs, though their duration differs across microarchitectures. For each CPU, we indicate the target cache and one top-ranking pattern. To select this pattern, we consider EVCr, worst-case durations (99th percentile), and whether variants of the pattern are also successful. All patterns shown achieve $> 99.9\%$ EVCr. In fact, many patterns exist with similar EVCr.

For Sandy Bridge (2011), the necessary conditions for Prime+Scope still hold, but the Prime patterns we have found are less efficient. We hypothesize that this is because this generation of processors uses the MRU replacement policy in the LLC [2], for which the insertion age is already young to begin with (and the PrimeTime strategy works best when the insertion age is old).

Serialization. PrimeTime avoids processor-specific (reverse-) engineering work. As an alternative to PrimeTime, one can obtain Prime patterns by handcrafting patterns (e.g., [10, 11, 64]) that leverage on the knowledge of the exact cache replacement policy, and the interaction between cache levels. In the end, such a strategy may lead to efficient primes with minimal memory accesses.

Algorithm 1 PrimeTime

Output: Prime patterns with high EVCr and low cycle count

```
Patterns ← GenerateAccessPatterns()
Patterns ← Mutate(Patterns, with Repeated Access)
Patterns ← Mutate(Patterns, with interleaved S Accesses)
...
Measurements ← TestEviction(Patterns, 10 000 times)
Patterns ← Filter(Patterns, Measurements, Highest EVCr 7 000)
Patterns ← Filter(Patterns, Measurements, Fastest 5 000)
...
Measurements ← TestEviction(Patterns, 1 000 000 times)
Patterns ← Filter(Patterns, Measurements, Highest EVCr 150)
Patterns ← Filter(Patterns, Measurements, Fastest 100)
return Patterns
```

However, such handcrafted patterns generally need to serialize accesses [11, 64] to prevent out-of-order execution from destroying the intended effects. Such serialization is implemented with pointer-chasing or memory fences, rendering the Prime patterns slower. PrimeTime avoids serialization by executing on the target architecture to incorporate hard-to-predict runtime effects directly. Still, there may exist handcrafted patterns that are more effective than the unordered patterns found by PrimeTime. However, the patterns obtained with PrimeTime are sufficient for Prime+Scope.

5.2 Coherence Directory (CD)

To enable Prime+Scope on the coherence directory, we again need a suitable Prime pattern. Unfortunately, Yan et al. [65] showed that achieving a high eviction rate with known eviction patterns is hard, especially when limited to W addresses. For instance, they report that repeated accesses to $W = 12$ congruent lines require more than 10 iterations to fully prime the CD. This is **Challenge-CD**.

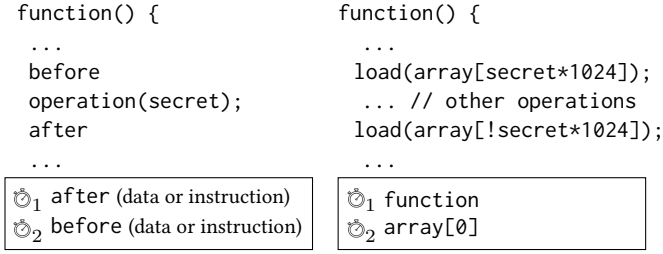
Slow Prime patterns, consisting of many accesses, are not a fundamental problem for Prime+Scope, as the ultimate time resolution is decoupled from the duration of the Prime (cf. Section 3.2). However, our PrimeTime tool indicates that such patterns fail to fix the EVC with high accuracy ($\mathbf{R}_B$), prohibiting Prime+Scope.

On the bright side, non-inclusive Intel caches have the advantage that lines in the CD always reside in one of the lower-level caches, satisfying $\mathbf{R}_C$ by design. Thus, what remains is to find a pattern that installs the desired eviction candidate in the CD ($\mathbf{R}_B$). We first cover a slow but universal solution. Then, we discuss our hypothesis for why traditional patterns do not work well on the CD, leading to a more efficient Prime pattern that leverages this information.

Fill-Flush-Fill. Prior work has used a fill-flush-fill approach to reset and simplify the replacement policy state [2, 11, 59]. Transposed to the CD, it would first fill the CD set, e.g., through many repetitions of an inefficient eviction pattern [65], flush all lines of the eviction set, and finally load them again in order. We confirm that such patterns successfully prime the CD set (with EVCr $> 99.9\%$), provided that the initial set filling is successful. However, such patterns are relatively slow. Moreover, the `clflush` instruction may not be available in restricted environments (cf. Section 4.1).

CD Replacement Policy. We believe that the poor performance of traditional eviction patterns on the CD is caused by property ②. The reason why this effect is more pronounced for the CD than for inclusive LLCs is the large associativity of private caches in current non-inclusive Intel hierarchies, and that lines in the CD are also cached in L1 and/or L2 [65]. If reading such lines does not influence the replacement state of the CD, many accesses are required for every attacker line to become younger than the lines to be evicted. Thus, for many access patterns, the CD *behaves* like a first-in-first-out (FIFO) queue, irrespective of the actual replacement policy.

Based on this hypothesis, a straightforward way to prime the CD is to access W congruent lines that are currently not in the CD. Indeed, we find such an access pattern to simultaneously achieve a near-perfect EVr and EVCr (the first element of the set being the scope line S), making it a suitable Prime pattern for Prime+Scope. On all non-inclusive platforms under consideration, our successive Primes alternate between two eviction sets of W addresses. As FIFO is very sensitive to ordering, and insensitive to repeated accesses, we enforce serialization by using a pointer-chasing approach [55].



(i) Variable-Time Operation **(ii) Variable-Time Access**

Figure 5: Uses of differential time: ⏲2 - ⏲1

6 CASE STUDIES

Micro-benchmarks. Prime+Scope bypasses the observer effect of the cache contention side channel, and reduces the cache measurement to a single memory access. Consequently, Prime+Scope is able to monitor victim behavior with high temporal precision, even asynchronously, without having to cope with missed accesses. Section 6.1 quantifies this precision and compares it to other techniques, and Section 6.2 characterizes the influence of noise.

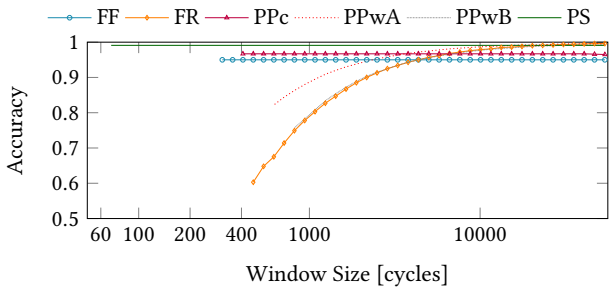
Differential Time. By scoping multiple sets simultaneously, Prime+Scope can estimate the temporal separation between two (or more) events with fine precision. Figure 5 shows two classes of timing leaks for which Prime+Scope is particularly well-suited.

The first class is that of *variable-time operations*, where the duration of an `operation` depends on a secret value. Such a code pattern encodes the secret in the time difference between memory accesses before and after `operation`, as in Figure 5i. Several attacks exploit leakage of this kind, e.g., for a secret-dependent number of loop iterations (e.g., [15]), or non-constant-time arithmetic (e.g., modular reduction [4, 21]). Cache attacks can only decode the secret if their precision is sufficient to detect the secret-dependent time difference of `operation`. Often, however, the resolution is too low, prompting the use of performance degradation of the victim [3, 4, 21].

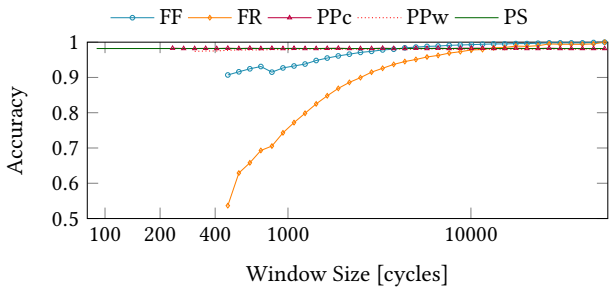
The second class is that of *variable-time accesses*, where memory accesses occur at a secret-dependent *time* (or, as a special case, in a secret-dependent *order* [7]). In Figure 5ii, the elements of `array` are always accessed, but the time relative to the start of `function` depends on a (binary) secret. Again, the attack needs sufficient precision to detect the secret-dependent time differences.

In this paper, we focus on *variable-time access* leakage. Section 6.3 demonstrates a high-capacity covert channel that works with such temporal encoding of the data. In Section 6.4, we show that AES T-tables, a well-studied cache attack target, also exhibits variable-time access leakage. Too fine-grained to be properly harnessed by prior techniques, with Prime+Scope, we exploit it to significantly reduce the number of traces needed for the attack.

Congruence Detection. The repeatable measurement of a single cache line is also useful to determine congruence in the target cache. In Section 6.5, we demonstrate this capability with a simple, efficient and portable eviction set construction methodology.



(i) Precision on LLC (Core i5-7500, Kaby Lake)



(ii) Precision on CD (Xeon Platinum 8280, Cascade Lake)

Tech.	WL	Prepare	Measure	$\text{Res}_{\texttt{min}}$	Res_{95}
FR	✗	`clflush T`	`load T`	420	4 350
PPwA	✗	`R1_S1_P0`	`R1_S1_P0`	580	2 500
PPwB	✗	`R3_S1_P012012`	`R1_S1_P0`	790	4 350
FF	✓	/	`clflush T`	300	300
PPc	✓	/	`R1_S1_P0`	390	390
PS	✓	/	`load EVC`	70	70

(iii) Techniques (LLC). WL denotes windowless; Res_{min} and Res_{95} denote max. resolution and resolution for 95% accuracy (cycles)

Tech.	WL	Prepare	Measure	$\text{Res}_{\texttt{min}}$	Res_{95}
FR	✗	`clflush T`	`load T`	440	5 000
PPw	✗	`simple ptr-chase`	`simple ptr-chase`	300	300
FF	✗	`clflush T`	`clflush T`	430	1 600
PPc	✓	/	`simple ptr-chase`	210	210
PS	✓	/	`load EVC`	80	80

(iv) Techniques (CD). WL denotes windowless; Res_{min} and Res_{95} denote max. resolution and resolution for 95% accuracy (cycles)

Figure 6: Accuracy and resolution as function of window size

6.1 Temporal Precision

We now quantify the time resolution of Prime+Scope (PS) to detect cross-core asynchronous events for an inclusive LLC and CD. For reference, we include the most prominent techniques; Prime+Probe (PP) for cache contention, i.e., the most comparable technique, and Flush+Reload (FR) and Flush+Flush (FF) for shared memory.

For Prime+Probe, the experiment includes windowed (PPw) and windowless (PPc) variants, where we consider two windowed versions for the LLC (PPwA and PPwB), as in Figure 6iii. For the CD, we use the accurate eviction patterns as discovered in Section 5.2, though they were unknown prior to this work.

On our non-inclusive processors, the Flush+Flush side channel also exists although inverted, i.e., lines present in the hierarchy have *lower* flush latency than those that do not. Moreover, the difference is quite large (200 vs. 330 cycles), unlike the subtle difference on our inclusive testbed. Prior work [13] reports that flushing an uncached line on multi-socket Intel systems triggers an access to memory for cross-socket coherence, which would clarify this behavior.

The measurement thresholds are calibrated dynamically and individually for every technique, based on timing histograms and the threshold selection regime with the best results. We note that some techniques (e.g., Flush+Reload, Prime+Scope) are less sensitive to the specific threshold value than others (e.g., Prime+Probe).

Methodology. We consider the following micro-benchmark for detecting asynchronous events. The event to be detected is an access to a specific cache line, by a process pinned to another core. To model an asynchronous event, the process first yields the CPU (`sched_yield`), before waiting for a randomly sampled number of `nops`. Then, the event is triggered with probability $1/2$.

We consider the instances listed in Figure 6iii and Figure 6iv. All instances start from an already-prepared state, using the top-ranking Prime from Table 1 for both Prime+Probe and Prime+Scope. The windowless instances (FF, PPc, PS) perform back-to-back measurements, so the preparation phase does not need to be repeated (indicated with /). In contrast, the windowed instances (FR, PPwA, PPwB) comprise a measurement, a preparation phase, and a waiting period until the end of the window. All instances run iteratively, and they terminate either when an event is detected, or when there was no event and the random process has terminated.

This experiment is repeated for 1 000 runs of 10 000 events for each window size and each technique, and the global accuracy (true positives and true negatives divided by total) is recorded. We also record the fundamental maximal resolution (i.e., the minimal window size that is able to contain one measurement iteration), as well as the maximal resolution that delivers an accuracy of 95%.

Note that this micro-benchmark serves to quantify, for each technique, the maximal probing resolution for reliable cross-core cache event detection. It should not be interpreted as a comparison of these techniques in a general setting, where more error sources are at play that are not captured here (e.g., noise). However, a poor resolution in this experiment implies a poor resolution in practice.

Also, the experiment assumes that the initial cache preparation is already successfully performed, which may paint an optimistic picture for windowed techniques. For instance, for the CD, the EVr of a single unordered probe of W lines is quite low [65]. Hence, a windowed Prime+Probe (PPw) has lower accuracy than in this experiment, due to false negatives incurred by the imperfect EVr, but its temporal resolution is adequately estimated by this experiment.

Results. For both the LLC (Figure 6i) and the CD (Figure 6ii), the resolution of Prime+Scope can be seen to tower above the other techniques, i.e., around 70 cycles or 25ns, while correctly detecting the majority of events (>98%). Figure 6iii and Figure 6iv indicate the maximal resolution (both fundamentally and for 95% accuracy).

As expected, windowed techniques have poor accuracy for small window sizes, with many events landing in blind spots (i.e., false negative errors). This is especially apparent for Flush+Reload, where small-window instances miss almost all events (cf. [3, 67]).

The resolution for windowless Prime+Probe (PPc) is already fairly high (390 cycles for the LLC, and 210 cycles for the CD). In contrast to typical applications of Prime+Probe [31, 37, 55], the windowless paradigm decouples Prime and Probe. This permits to optimize the Prime stage for high EVr, and Probe stage for speed.

The inflated time difference for Flush+Flush on the Cascade-Lake server makes it more accurate than Flush+Reload for all window sizes. However, the accuracy increases with the window size, indicating that the Flush measurement on this platform has a blind spot, i.e., it is not concurrent (cf. Section 3.2).

6.2 Susceptibility to Noise

Like Prime+Probe, Prime+Scope is susceptible to noise resulting from activity in the targeted cache set, other than the event which is to be monitored. This limitation is fundamental to the cache contention leakage mechanism. It is natural to ask whether the more precise Prime patterns for Prime+Scope make it more fragile in the presence of noise. We explore it in the following experiment.

We consider two threads pinned to different cores of an Intel Core i7-7700K (Kaby Lake, 16 ways), where one thread monitors the other's memory accesses under different levels of noise. The `stress` tool is used to generate heavy memory load on one or more other cores (e.g., as in [39]). One thread accesses a predetermined address periodically (every 10 000 cycles), as ground truth, while the other thread continuously monitors the cache set for events, and records the timestamps at which the events are detected. Timestamps are obtained using the CPU's time stamp counter, which is synchronized across cores. After execution, the collected timestamps are analyzed to evaluate the detection accuracy of the techniques.

In the ideal case, only one event is detected in each time slot, being the ground-truth periodic access. In the experiment, three cases are distinguished: *correct* when only one event is detected, and it was detected right after the event occurred; *miss* when the ground-truth event was not detected in the time slot (false-negative error); and *multi* when events were detected that did not correspond to the ground-truth access (false-positive error). If a time slot contains both error types, which is uncommon, it is classified as *multi*.

Prime+Scope (PS) is compared with two windowless Prime+Probe instances. As indicated in Figure 7, the first one (PP_{PS}) inherits the Prime access pattern of Prime+Scope. The second one (PP_{CST}) uses a custom Prime+Probe pattern, which is also obtained with PrimeTime, but optimized for EVr instead of EVCr.

For the Prime+Probe instances, the indicated Prime is repeated continuously and serves both as preparation (where *duration* is the number of cycles needed to prepare the cache after an event) and measurement (where *precision* is the number of cycles between successive measurements in the absence of an event). Note that

Method	Stress	Correct	Miss	Multi
PS	0	98.24	1.44	0.32
PP_{PS}		99.42	0.45	0.12
PP_{CST}		98.72	1.15	0.13
PS	1	79.71	2.35	17.94
PP_{PS}		83.83	0.54	15.63
PP_{CST}		81.74	0.56	17.70
PS	5	78.38	2.42	19.20
PP_{PS}		82.80	0.68	16.51
PP_{CST}		81.94	0.00	18.06

Figure 7: Distribution of time slots along *correct*, *miss* and *multi* categories for Prime+Scope and Prime+Probe (averages over 200 runs of more than 25 000 time slots). Stress indicates the amount of `stress` workers, pinned to different cores, that are active in the background. The properties of the Prime patterns are as follows:

	Pattern	EVr	EVCr	Duration	Precision
PS	`R3_S4_P01SS2SS301230123`	100%	99.9%	1810	70
PP_{PS}	`R2_S4_P01SS2SS301230123`	100%	99.9%	1255	1170
PP_{CST}	`R2_S1_P01`	100%	NA	1190	700

PP_{PS} performs much more accesses than PP_{CST}, which is almost completely hidden in the preparation stage in the shade of cache misses, but is clearly visible in the measurement precision. As in Figure 1ii-H, the Prime right after detection of an event is ignored, as its execution time may still be affected by that event.

A naive implementation of Prime+Scope performs the Prime just once for every detected event. However, suppose that the Prime is unsuccessful in fixing the EVC, e.g., due to noise. This will blind the following Scope operations, as they may be fast even if some elements of the cache set have been evicted. To overcome this issue, the Prime step is repeated when no events were detected within a chosen period (in this experiment, roughly 12 000 cycles).

Results. For each technique and noise level, Figure 7 indicates the distribution of time slots along *correct*, *miss* and *multi* rates. This micro-benchmark provides a rough indication of how noise translates to false-positive and false-negative errors for the different windowless techniques. We can draw the following conclusions:

- The *miss* rates of Prime+Scope are slightly (a few p.p.) higher than Prime+Probe. The main cause of such false-negative errors are accesses during the preparation phase of the attack, which may result in an imperfectly prepared set [11]. Hence, the observed behavior is clarified by imperfect preparation affecting the EVCr slightly more than the EVr. If high noise levels are to be expected, Prime+Scope fares well with an upwards correction of the Prime repetitions compared to the output of PrimeTime (e.g., as in this experiment, where `R2_*` → `R3_*`).
- In terms of *multi* rates, all instances are comparable. The main cause of such false-positive errors is noise during the measurement phase, evicting the EVC. As this leads to high access latencies for both Probe and Scope, this source of errors is expected to affect Prime+Scope and Prime+Probe equally.

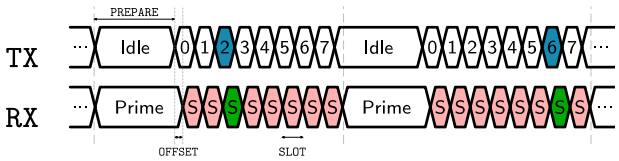


Figure 8: Covert Channel Operation ($m = 3$ bits per symbol).

6.3 Cross-Core Covert Channel

To show that Prime+Scope can discern fine-grained temporal cache activity, we build a high-capacity cross-core covert channel based on *variable-time access* leakage (cf. Figure 5ii). It temporally encodes m-bit symbols by performing a memory access in one of 2^m slots, where slots may be as short as 80 processor cycles.

As a representative sample, we implement it on the LLC of a Kaby Lake processor, and on the CD of CascadeLake-SP. For our proof-of-concept implementation, we assume a synchronized transmitter and receiver that have agreed on a contention set (e.g., as in [43, 44, 65]).

Figure 8 visualizes the working mechanism of the covert channel, as well as its defining parameters (duration of preparation stage, transmission slots, and transmitter-receiver offset). First, the receiver primes the set. Then, the transmitter sends an m-bit symbol M by accessing a congruent line in slot number $i = M$. At the same time, the receiver scopes the set every `SLOT` cycles, decoding M as the slot number in which the scope line `S` is evicted.

Optimizations. We perform a few modifications to improve the channel bandwidth. Instead of the canonical encoding, we encode the bitstream into m-bit symbols with reflected binary Gray codes to ensure that off-by-one symbol errors only lead to single-bit errors.

For the LLC channel, the receiver uses the Prime patterns of Section 5.1. We find that if the transmitter flushes the line right after accessing it, it slightly speeds up the prime for the receiver.

For the CD channel, the Prime stages consist of alternating pointer chases (cf. Section 5.2). To amortize the latency arising from serialization, four sets are primed simultaneously with their accesses interleaved (e.g., as in [16]). After the combined Prime, there are four rounds of 2^m slots, where each round encodes m bits.

Evaluation. Figure 9 gives capacity and error rate as a function of bandwidth, and summarizes the parameters for which the LLC- and CD-based channels obtain peak capacity. Respectively, the capacities are 3.5 Mbps and 3.1 Mbps, which is much higher than Prime+Probe on the LLC (e.g., 500 Kbps at 1% bit error rate [25]). Furthermore, they are in the same order of magnitude as state-of-the-art stateless channels without shared memory, such as Pessl et al. [44] (DRAM row buffer contention, 2.1 Mbps capacity) and Paccagnella et al. [43] (LLC ring contention, 4.1 Mbps capacity).

To our knowledge, the only other covert channel using the CD is due to Yan et al. [65], with a bandwidth of 0.2 Mbps (error rate not reported). The order-of-magnitude capacity improvement of our channel stems from both a fast and efficient Prime pattern (cf. Section 5.2), and the precision of Prime+Scope (cf. Section 6.1).

As the goal is to characterize the temporal precision of Prime+Scope, we limit the study of this covert channel to synchronized parties on idle systems. In practice, further engineering challenges need to be overcome (e.g., as undertaken in [37–39]).

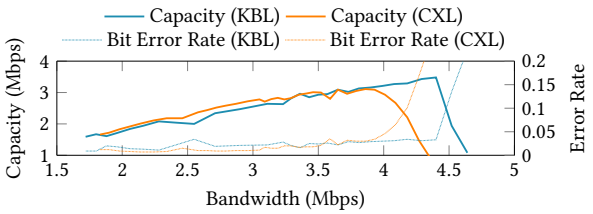


Figure 9: Covert Channel Capacities and error rates for the Kaby Lake (KBL) and CascadeLake-SP (CXL) platform. For the peak capacities, the configuration in the following table are used, where `PREPARE`, `OFFSET` and `SLOT` are in cycles.

Platform	C_S	m	Capacity	PREPARE	OFFSET	SLOT
Core i7-7500 (KBL)	LLC	4	3.5 Mbps	1 400	90	100
Xeon Pl. 8280 (CXL)	CD	3	3.1 Mbps	4 750	125	100

6.4 Side-Channel Attack on AES

We now revisit the seminal first-round known-plaintext attack on the T-table implementation of AES [42], a standard benchmark for cache attack techniques (e.g., [18, 25, 57]). The time precision of Prime+Scope allows a novel attack technique against AES, based on *variable-access time* leakage (cf. Figure 5ii), rather than traditional *access* leakage. As it can learn more information from each encryption, much fewer traces are needed to extract the secret. Although a windowless Prime+Probe can also absorb some of this information, Prime+Scope requires 10-70x fewer traces. We first give a high-level outline of the traditional attack (for details, refer to [42, 55]). Like prior work, we attack OpenSSL 1.0.1e (or similar).

Traditional Attack. The implementation features four precomputed tables Te_j, of 16 cache lines each. The attacker monitors accesses to such table lines $Te_j[M]$ which, on CPUs with 64-byte cache lines, leak the upper four bits (nibble) of every key byte k_i. We implement this attack with Prime+Probe (for comparison) and Flush+Reload (for reference), where the attacker prepares the cache, triggers an encryption with known plaintext, and measures afterwards. For plaintexts where $\lceil p_i \rceil^4 = \lceil k_i \rceil^4 \oplus M$, cache line $Te_{i \bmod 4}[M]$ is accessed in the first round, and hence, in 100% of encryptions. For other p_i, it is accessed in 92.5% of encryptions, so each monitored $Te_j[M]$ carries information in 7.5% of encryptions.

Variable-Time Access: Prime+Scope. Consider the code snippet in Figure 10. Indeed, not only the access to a table encodes information, but also the encryption round in which it happens. We now show that, through its time precision, Prime+Scope is able to capture such

```
void AES_encrypt(...) {
 ... // s0-s3 contain p_i xor k_i
 // round 1:
 t0 = Te0[s0>>24] ^ Te1[(s1>>16) & 0xff]
    ^ Te2[(s2>>8) & 0xff] ^ Te3[s3 & 0xff] ^ rk[4];
 t1 = ... ; t2 = ...; t3 = ...; // similar to t0
 ... // rounds 2-10 (similar to round 1)
```

⏱1 AES_encrypt
⏱2 Te0/Te1/Te2/Te3

Figure 10: Variable-time access leakage for AES

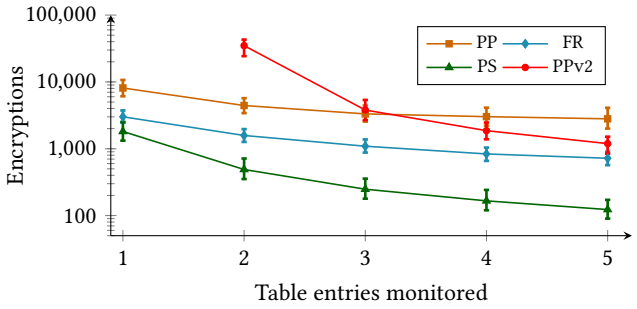


Figure 11: Median encryptions for AES T-tables (bars indicate 10-90th percentiles). Comparison of Prime+Scope (PS) with traditional Prime+Probe (PP) and Flush+Reload (FR), as well as differential-time Prime+Probe (PPv2).

leakage. Information is obtained through *differential time* between the start of the `AES_encrypt` function and one or more table entries.

We use the cache attack as an oracle for accesses to table entries during the first AES round. We spin up a thread for each monitored line (including the first instruction cache line of `AES_encrypt`). The adversary triggers encryptions, and each thread records the timestamp at which the access is detected (if any) for the monitored table entry. Then, the differential times are used to score the key nibble hypotheses. The larger the differential time, the larger the penalty for the key nibble, as the probability is lower that it corresponds to a first-round access. For a table access in the first round, we observe the differential time to be around 200-300 cycles.

An advantage of this attack is that every trace carries information for each monitored table entry, as opposed to only 7.5% for the traditional first-round attack. Note that a single-threaded Prime+Scope can also record differential times, but the temporal resolution decreases linearly with the number of lines scoped in one thread.

Variable-Time Access: Prime+Probe? For comparison, we explore whether Prime+Probe can also learn from the differential time. To capture the maximal performance of Prime+Probe, we consider an optimal, windowless configuration; the Prime is the same as for Prime+Scope, and the Probe is the simple, unordered traversal of the set (pattern `R1_S1_P0`). According to Figure 6iii), we expect a precision of approx. 400 cycles (cf. 70 cycles for Prime+Scope).

Results. Figure 11 presents the results on the LLC of an Intel Core i7-7700K (Kaby Lake, 16 ways). It shows the number of encryptions needed to mount the full first-round attack, which recovers 64 of the 128 key bits. We consider the key nibble found as soon as the hypothesis converges (i.e., it reaches the correct value and does not diverge from it). We perform 1 000 iterations and indicate the median and 10th and 90th percentiles to convey the variance. Note that these results are obtained without degrading victim performance (other than indirectly through the cache sets that are monitored).

Prime+Scope retrieves the secret information with fewer traces (between 5-25x) than the traditional Prime+Probe. The differential-time Prime+Probe is also able to capture some of the temporal information, but again more slowly, with more traces than Prime+Scope (10-70x). When only a single table entry is monitored in every encryption, we find that it fails to recover the secret even with as many as 100 000 traces, which may indicate that the timing differences are too small to be distinguished by Prime+Probe.

6.5 Finding Congruent Addresses

Cache contention attacks require the adversary to find eviction sets, i.e., sets of congruent addresses in the target cache. This practical challenge has been investigated thoroughly [18, 31, 37, 60, 65]. However, the principles underlying Prime+Scope enable an efficient congruence test, resulting in a faster and simpler routine that, counter-intuitively, requires fewer platform-specific parameters.

Algorithm: LLC. The foundation of the proposed LLC eviction set construction routine is given in Algorithm 2. It repeatedly measures the access latency of the `TARGET` address and, between each measurement, accesses a guess. As `TARGET` is continuously accessed, it is always served from the L1 cache, which does not influence its LLC replacement state. Guesses that turn out to be congruent with the `TARGET` are installed in the LLC, and each time this happens, the EVC in the LLC changes. After enough congruent guesses, the `TARGET` becomes the EVC. The next congruent guess then evicts `TARGET` from the LLC and, due to the inclusion property, also from the private caches. Therefore, the next access to `TARGET` is slow, indicating the congruence of the latest guess. The attacker repeats this procedure until she has obtained enough congruent addresses.

To speed up the routine, between lines 4 and 5 in Algorithm 2, we access already-obtained congruent addresses to accelerate `TARGET` becoming the LLC eviction candidate. Thus, the number of inner-loop iterations is expected to decrease as the algorithm proceeds.

To increase the robustness, we test whether the resulting set successfully evicts the `TARGET`. If not, an extra address is found, and the test is repeated. The number of failures until the test succeeds reveals the number of false positives in the set, which can be removed through a short reduction phase, akin to prior work [37, 60, 65].

The algorithm is identical for huge and small virtual memory pages, but the availability of huge pages speeds up the runtime significantly, as the guesses are more likely to be congruent due to the increased control over physical address bits [37, 60].

Algorithm: CD. On non-inclusive Intel caches, the set index mapping for the LLC and CD is identical. Hence, eviction sets constructed for one may be used for the other. Finding congruent addresses through contention on the CD is challenging, as congruence in the CD implies congruence in L2 [65], and `TARGET` may be

Algorithm 2 Eviction Set Construction

Input: `TARGET`: address for which an LLC eviction set is desired
Output: `ES`: eviction set

```
1:  ES ← empty list
2:  length ← 0
3:  while length < LLC_WAYS do
4:      access(TARGET)
5:      do
6:          GUESS ← a line possibly congruent to the TARGET
7:          access(GUESS)
8:      while access(TARGET) is fast
9:      ES[length++] ← GUESS
10: end while
```

Table 2: Runtime (median) and accuracy (%) for eviction set construction (1 000 runs for randomly selected targets)

Processor	Vila et al. [60]		Ours	
Cache	Huge*	Small*	Huge	Small
Skylake	165.2 ms	316.3 ms	0.25 ms	2.80 ms
12 Way LLC	99%	100%	99%	99%
Skylake	113.2 ms	643.8 ms	0.55 ms	4.03 ms
16 Way LLC	98%	100%	96%	100%
Skylake-SP	NA	NA	3.15 ms	35.40 ms
12 Way CD	NA	NA	100%	93%

* Initial set size for $^{12}_{16}$ Way LLC is $^{65}_{90}$ for huge pages, $^{3500}_{4000}$ for small.

evicted due to contention on L2, leading to false positives. Thus, like prior work [65], we perform the construction on the LLC.

The routine is similar to in Algorithm 2. However, recall that the LLC is non-inclusive, so the memory accesses on lines 4 and 7 do not guarantee the installation of the `TARGET` and the `GUESS`es in the LLC. We replace them with joint accesses by the attacker thread and a helper thread on another CPU core, as we observed that accesses from two cores place a copy of the line into the LLC[1].

Platforms. We tested the eviction set construction on all the machines in Table 1, as we had to obtain eviction sets for PrimeTime. To compare with other work, we perform a detailed comparison on the Skylake microarchitectures in Table 2. Apart from the differences for inclusive and non-inclusive LLCs and a parameter for LLC associativity, it requires no adaptation to the processor.

Comparison. Vila et al. [60] study eviction set construction in detail, and propose a linear-time algorithm that improves over the quadratic-time baseline [37]. These routines iteratively remove one or more lines from a big initial set, measuring whether the residual set still evicts the target. In contrast, Algorithm 2 starts from an empty set, and adds congruent lines to it. It overcomes practical problems identified by previous works, such as the dependence on replacement policies (and their adaptivity) [60], TLB thrashing [20], and hardware prefetchers [55]. Similar to prior techniques [37, 60, 65], it does not require knowledge of the slicing function.

Because our implementation does not require any preparation steps, such as organizing the memory space in a linked list, or selecting a suitable starting set, we take into account the *total* execution time of the construction routine, which includes the time spent for failed preparation steps in addition to the last successful reduction step. As shown in Table 2, our implementation executes up to 660x faster than the one by Vila et al. [60], while achieving the same success rate (where success is defined as a set of W addresses that consistently evicts the target). Furthermore, the default configuration is adequate for successful execution on all tested Intel processors, while containing only a few configuration parameters.

For non-inclusive caches, only the initial study by Yan et al. [65] describes how to find LLC/CD eviction sets. They adapt the congruence test of earlier work [37] to overcome the challenges provided by non-inclusive LLCs. Compared to ours, their routine has the advantage of being single-threaded. As performance metrics are not provided in [65], we are unable to directly compare our work with theirs. However, it has quadratic complexity, and is so far unsuccessful when huge pages are not available. Even if their routine is adapted to linear time (e.g., [60]), we expect our algorithm to outperform it, in accordance with the findings for inclusive caches.

[1]For more information, we refer the reader to `https://www.github.com/KULeuven-COSIC/PRIME-SCOPE/evsets`.

7 RELATED WORK

7.1 Classification of Attack Techniques

Complementing the quantitative study in Section 6, Table 3 positions Prime+Scope with respect to existing cross-core cache attack techniques on the basis of prerequisites and features.

Prerequisites. The most basic requirement is co-tenancy, where the attacker can run unprivileged code (native or otherwise) on the same physical machine as a victim. As long as both parties share at least one cache level, an attacker can measure contention on shared cache resources (as is done for Prime+Probe and Prime+Scope).

Some techniques are predicated on additional capabilities, such as shared memory with the victim, the presence of a `clflush` instruction, or special processor features like Intel's TSX. These extra capabilities can increase the power of the technique, e.g., in terms of spatial resolution or reliability. However, the additional prerequisites limit the applicability of these techniques. For instance, shared memory is discouraged in multi-tenant clouds, and `clflush` may not be available to code that is not running natively on the system (e.g., in the browser). Intel TSX is not available on all Intel CPUs and, for those where it is available, Intel has added support to disable it [30] in response to recent transient execution attacks [50, 58].

Features. The most relevant features to this work are whether a technique can be instantiated in a *windowless* paradigm, and the number of cache accesses for each measurement.

In terms of spatial granularity, techniques based on shared memory can infer accesses to specific cache lines, whereas cache contention attacks are fundamentally limited to set-granularity. Prime+Scope belongs to the latter category. Table 3 also indicates which techniques have been shown on CDs of non-inclusive LLCs [65].

Measuring *multiple events* enriches the information content of the channel and is an essential requirement to record differential times (cf. Section 6). Prime+Abort cannot monitor multiple events while maintaining the ability to distinguish between them [18]. Other techniques can do so, but may have to take the influence of spatial hardware prefetching into account [26, 61, 66].

Other Properties. Some techniques are tailored to overcome specific system-level constraints. Cache occupancy attacks [53] forego the search for congruent addresses and instead measure contention on a cache-sized buffer. This makes them amenable for deployment in (very) restricted environments [52], at the cost of all spatial granularity and significant time precision. Some techniques offer stealth against runtime detection [11, 25], or bypass software-based countermeasures with indirect cache accesses [57]. Exploring Prime+Scope in these system models is beyond the scope of this work.

Table 3: Classification of cross-core cache attack techniques in terms of prerequisites and features

Attack Technique	Mechanism		Prerequisites			Features			
	Leakage Source	Spatial Granularity	No Shared Mem.	No `clflush`	No TSX	Window-less	Measure Size	Multi-Target	Shown on CD
Flush+Reload [67]	load latency	line	✗	✗	✓	✗	$1_{✓}$	✓	✓
Flush+Flush [25]	load latency	line	✗	✗	✓	✓	$1_{✓}$	✓	✓
Evict+Reload [26]	load latency	line	✗	✓	✓	✗	$1_{✓}$	✓	✓
Reload+Refresh [11]	repl. state	line	✗	✗	✓	✗	$2_{✓}$	✓	✗
Prime+Probe [31, 37]	contention	set	✓	✓	✓	✓	$W_{✗}$	✓	✓
Occupancy [53]	contention	none	✓	✓	✓	✗	huge$_{✗}$	✗	✓
Prime+Abort [18]	TSX abort	set	✓	✓	✗	✓	$\varnothing_{✓}$	✗	✗
Prime+Scope	**contention**	**set**	✓	✓	✓	✓	**$1_{✓}$**	✓	✓

7.2 Cache Attacks and Replacement Policies

Cache replacement policies were long perceived as obstacles, leading to techniques that minimize their influence (e.g., double pointer-chasing [55] or black-box eviction strategies [24]). However, enabled by reverse-engineering advances [1, 2, 11, 59], some works use replacement properties to the advantage of the attacker.

Same-Core. Xiong and Szefer [64] use the PLRU policy of the L1 cache to leak information between processes through LRU states. Recently, Röttger and Janc [48] use it to amplify the time difference between presence and absence of a speculative memory access.

Cross-Core. Reload+Refresh [11] detects accesses to a shared address by monitoring changes in the EVC. In this context, our efficient Prime patterns may be useful to prepare the EVC. Wang et al. [61] probe the L2 EVC to limit the impact of the aggressive hardware prefetcher on low-end, in-order Intel CPUs. Their Prime pattern consists of $2W$ ordered accesses (all cache misses), making it comparatively slow. Briongos et al. [10] detect the start of a victim routine and exploit LLC replacement to evict prefetched lines at the right time. As Prime+Probe lacks the required precision, they rely on Prime+Abort for detection. Future work should investigate the use of Prime+Scope to remove the dependency on Intel TSX.

To enable Rowhammer attacks without flushing, Gruss et al. [24] find efficient eviction strategies for unknown replacement policies. Aweke et al. [5] develop a pattern predicated on the Sandy Bridge MRU policy, which De Ridder et al. [16] modernize and improve for browser-based Rowhammer in the presence of DRAM mitigations.

8 LIMITATIONS AND COUNTERMEASURES

Requirements. Prime+Scope does not work on processors for which the key properties (cf. Section 4) do not hold. For instance, it fails when the shared structure C_S has a random replacement policy (as it eliminates predictability of the EVC), or if the lower-level caches do not act as a filter for C_S (as it eliminates repeatability of the measurement). We believe these two properties are the only anchor points for the deployment of countermeasures to reduce Prime+Scope to Prime+Probe. However, invalidating these properties may adversely affect multi-level cache performance.

Leakage Types. As demonstrated in Section 6, Prime+Scope is able to extract information from fine-grained timing leaks. However, if time differences are more coarse-grained (e.g., RSA square-and-multiply [37, 67]), the increased precision of Prime+Scope does not directly lead to a more efficient attack. Still, we note that the windowless nature of Prime+Scope eliminates false-negative errors due to overlap between measurement and event, which may help to reduce the number of required observations to retrieve the secret.

High-Frequency Events. Recall that for Prime+Scope (and Prime+Abort and Prime+Probe), even for windowless instances, the cache state needs to be prepared after every detected event. If the event rate is very high, i.e., when the temporal separation of accesses to the same address is in the order of the Prime duration, the preparation step becomes dominant for the time precision. We note that, although Prime+Scope places more demands on cache state preparation than its counterparts, the Prime patterns obtained with PrimeTime are still fairly competitive, with most of them in the range of 1000-1300 cycles (cf. Table 1).

Generic Countermeasures. Flush+Reload and Flush+Flush can be thwarted by disallowing shared memory across security boundaries, but countermeasures to mitigate the cache contention channel are far more invasive. However, in recent years, this defensive avenue has attracted attention in the research community. The main lines of work are based on *isolation*, i.e., partitioning the cache along isolated portions (e.g., [6, 14, 17, 19, 35]), or *randomization*, i.e., obfuscating interference by modifying the set index mapping (e.g., [36, 45, 46, 49, 54, 62, 63]). By strengthening cache contention attacks, our work motivates further research in this direction.

9 CONCLUSION

This paper introduced Prime+Scope, a high-resolution primitive to measure contention on shared cache resources. It can target last-level caches and directories alike, and we found it to apply to all tested Intel processors of the last decade. Roughly speaking, Prime+Scope is a high-resolution successor to Prime+Probe, assuming the same attacker capabilities that make the latter so widely applicable. The fast and repeatable Scope measurement essentially optimizes the resolution of cache contention attacks, delivering a cross-core time precision that even flush-based techniques cannot provide.

We believe that Prime+Scope is a valuable addition to the microarchitectural attack toolbox. We quantitatively evaluated its properties, and illustrated them with a high-bandwidth covert channel, a new fine-grained attack on AES T-tables, and a simple, efficient, and portable routine to construct eviction sets.

ACKNOWLEDGMENTS

We thank the anonymous CCS 2021 reviewers, as well as Frank Piessens and Márton Bognár, for their valuable feedback. This research is partially funded by the European Research Council (ERC - #695305) and the Flemish Government through the FWO project TRAPS. It was also supported by the CyberSecurity Research Flanders (#VR20192203). Additional funding was provided by a generous gift from Intel. Antoon Purnal is supported by a grant of the Research Foundation - Flanders (FWO).

REFERENCES

[1] Andreas Abel and Jan Reineke. 2013. Measurement-based Modeling of the Cache Replacement Policy. In *IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS)*.

[2] Andreas Abel and Jan Reineke. 2020. nanoBench: a Low-overhead Tool for Running Microbenchmarks on x86 Systems. In *IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*.

[3] Thomas Allan, Billy Bob Brumley, Katrina Falkner, Joop Van de Pol, and Yuval Yarom. 2016. Amplifying Side Channels Through Performance Degradation. In *Annual Conference on Computer Security Applications (ACSAC)*.

[4] Diego F Aranha, Felipe Rodrigues Novaes, Akira Takahashi, Mehdi Tibouchi, and Yuval Yarom. 2020. Ladderleak: Breaking ECDSA With Less Than One Bit of Nonce Leakage. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[5] Zelalem Birhanu Aweke, Salessawi Ferede Yitbarek, Rui Qiao, Reetuparna Das, Matthew Hicks, Yossi Oren, and Todd Austin. 2016. ANVIL: Software-based Protection Against Next-generation Rowhammer Attacks. *ASPLOS* (2016).

[6] Raad Bahmani, Ferdinand Brasser, Ghada Dessouky, Patrick Jauernig, Matthias Klimmek, Ahmad-Reza Sadeghi, and Emmanuel Stapf. 2021. {CURE}: A Security Architecture with CUstomizable and Resilient Enclaves. In *USENIX Security Symposium*.

[7] Mohammad Behnia, Prateek Sahu, Riccardo Paccagnella, Jiyong Yu, Zirui Zhao, Xiang Zou, Thomas Unterluggauer, Josep Torrellas, Carlos Rozas, Adam Morrison, Frank Mckeen, Fangfei Liu, Ron Gabor, Christopher W. Fletcher, Abhishek Basak, and Alaa Alameldeen. 2021. Speculative Interference Attacks: Breaking Invisible Speculation Schemes. *ASPLOS* (2021).

[8] Naomi Benger, Joop Van de Pol, Nigel P Smart, and Yuval Yarom. 2014. "Ooh Aah... Just a Little Bit": A Small Amount of Side Channel can go a Long Way. In *Cryptographic Hardware and Embedded Systems (CHES)*.

[9] Daniel J Bernstein, Joachim Breitner, Daniel Genkin, Leon Groot Bruinderink, Nadia Heninger, Tanja Lange, Christine van Vredendaal, and Yuval Yarom. 2017. Sliding Right into Disaster: Left-to-right Sliding Windows Leak. In *Cryptographic Hardware and Embedded Systems (CHES)*.

[10] Samira Briongos, Ida Bruhns, Pedro Malagón, Thomas Eisenbarth, and José M. Moya. 2021. Aim, Wait, Shoot: How the CACHESNIPER Technique Improves Unprivileged Cache Attacks. In *IEEE European Symposium on Security and Privacy (EuroS&P)*.

[11] Samira Briongos, Pedro Malagon, Jose M. Moya, and Thomas Eisenbarth. 2020. RELOAD+REFRESH: Abusing Cache Replacement Policies to Perform Stealthy Cache Attacks. In *USENIX Security Symposium*.

[12] Leon Groot Bruinderink, Andreas Hülsing, Tanja Lange, and Yuval Yarom. 2016. Flush, Gauss, and Reload–a Cache Attack on the BLISS Lattice-based Signature Scheme. In *Cryptographic Hardware and Embedded Systems (CHES)*.

[13] Lucian Cojocar, Jeremie Kim, Minesh Patel, Lillian Tsai, Stefan Saroiu, Alec Wolman, and Onur Mutlu. 2020. Are We Susceptible to Rowhammer? An End-to-End Methodology for Cloud Providers. In *IEEE Symposium on Security and Privacy (S&P)*.

[14] Victor Costan, Ilia Lebedev, and Srinivas Devadas. 2016. Sanctum: Minimal Hardware Extensions for Strong Software Isolation. In *USENIX Security Symposium*.

[15] Luca De Feo, Bertram Poettering, and Alessandro Sorniotti. 2021. On the (in) security of ElGamal in OpenPGP. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[16] Finn de Ridder, Pietro Frigo, Emanuele Vannacci, Herbert Bos, Cristiano Giuffrida, and Kaveh Razavi. 2021. SMASH: Synchronized Many-sided Rowhammer Attacks from JavaScript. In *USENIX Security Symposium*.

[17] Ghada Dessouky, Tommaso Frassetto, and Ahmad-Reza Sadeghi. 2020. HybCache: Hybrid Side-Channel-Resilient Caches for Trusted Execution Environments. In *USENIX Security Symposium*.

[18] Craig Disselkoen, David Kohlbrenner, Leo Porter, and Dean M. Tullsen. 2017. Prime+Abort: A Timer-Free High-Precision L3 Cache Attack using Intel TSX. In *USENIX Security Symposium*.

[19] Leonid Domnitser, Aamer Jaleel, Jason Loew, Nael Abu-Ghazaleh, and Dmitry Ponomarev. 2012. Non-Monopolizable Caches: Low-Complexity Mitigation of Cache Side Channel Attacks. *ACM Transactions on Architecture and Code Optimization (TACO)* (2012).

[20] Daniel Genkin, Lev Pachmanov, Eran Tromer, and Yuval Yarom. 2018. Drive-by Key-extraction Cache Attacks from Portable Code. In *Applied Cryptography and Network Security*.

[21] Daniel Genkin, Luke Valenta, and Yuval Yarom. 2017. May the Fourth be With You: A Microarchitectural Side Channel Attack on Several Real-world Applications of Curve25519. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[22] Enes Göktas, Kaveh Razavi, Georgios Portokalidis, Herbert Bos, and Cristiano Giuffrida. 2020. Speculative Probing: Hacking Blind in the Spectre Era. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[23] Daniel Gruss, Julian Lettner, Felix Schuster, Olga Ohrimenko, Istvan Haller, and Manuel Costa. 2017. Strong and Efficient Cache Side-channel Protection Using Hardware Transactional Memory. In *USENIX Security Symposium*.

[24] Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2016. Rowhammer.js: A Remote Software-Induced Fault Attack in JavaScript. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*.

[25] Daniel Gruss, Clémentine Maurice, Klaus Wagner, and Stefan Mangard. 2016. Flush+Flush: A Fast and Stealthy Cache Attack. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*.

[26] Daniel Gruss, Raphael Spreitzer, and Stefan Mangard. 2015. Cache Template Attacks: Automating Attacks on Inclusive Last-level Caches. In *USENIX Security Symposium*.

[27] David Gullasch, Endre Bangerter, and Stephan Krenn. 2011. Cache Games–Bringing Access-based Cache Attacks on AES to Practice. In *IEEE Symposium on Security and Privacy (S&P)*.

[28] Berk Gülmezoglu, Mehmet Sinan Inci, Gorka Irazoqui Apecechea, Thomas Eisenbarth, and Berk Sunar. 2015. A Faster and More Realistic Flush+Reload Attack on AES. In *Constructive Side-Channel Analysis and Secure Design (COSADE)*.

[29] Ralf Hund, Carsten Willems, and Thorsten Holz. 2013. Practical Timing Side Channel Attacks against Kernel Space ASLR. In *IEEE Symposium on Security and Privacy (S&P)*.

[30] Intel. 2019. Intel Transactional Synchronization Extensions (Intel TSX) Asynchronous Abort. https://software.intel.com/security-software-guidance/deep-dives/deep-dive-intel-transactional-synchronization-extensions-intel-tsx-asynchronous-abort.

[31] Gorka Irazoqui, Thomas Eisenbarth, and Berk Sunar. 2015. S$A: A Shared Cache Attack That Works Across Cores and Defies VM Sandboxing – and Its Application to AES. In *IEEE Symposium on Security and Privacy (S&P)*.

[32] Gorka Irazoqui, Mehmet Sinan Inci, Thomas Eisenbarth, and Berk Sunar. 2014. Wait a minute! A fast, Cross-VM attack on AES. In *Research in Attacks, Intrusions, and Defenses (RAID)*.

[33] Michael Kurth, Ben Gras, Dennis Andriesse, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2020. NetCAT: Practical Cache Attacks From the Network. In *IEEE Symposium on Security and Privacy (S&P)*.

[34] Moritz Lipp, Daniel Gruss, Raphael Spreitzer, Clémentine Maurice, and Stefan Mangard. 2016. ARMageddon: Cache Attacks on Mobile Devices. In *USENIX Security Symposium*.

[35] Fangfei Liu, Qian Ge, Yuval Yarom, Frank Mckeen, Carlos Rozas, Gernot Heiser, and Ruby B Lee. 2016. Catalyst: Defeating Last-level Cache Side Channel Attacks in Cloud Computing. In *IEEE International Symposium on High Performance Computer Architecture (HPCA)*.

[36] Fangfei Liu and Ruby B. Lee. 2014. Random Fill Cache Architecture. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[37] Fangfei Liu, Yuval Yarom, Qian Ge, Gernot Heiser, and Ruby B. Lee. 2015. Last-Level Cache Side-Channel Attacks Are Practical. In *IEEE Symposium on Security and Privacy (S&P)*.

[38] Clémentine Maurice, Christoph Neumann, Olivier Heen, and Aurélien Francillon. 2015. C5: Cross-Cores Cache Covert Channel. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*.

[39] Clémentine Maurice, Manuel Weber, Michael Schwarz, Lukas Giner, Daniel Gruss, Carlo Alberto Boano, Stefan Mangard, and Kay Römer. 2017. Hello from the Other Side: SSH over Robust Cache Covert Channels in the Cloud. In *Network and Distributed System Security Symposium (NDSS)*.

[40] Ahmad Moghimi, Gorka Irazoqui, and Thomas Eisenbarth. 2017. CacheZoom: How SGX Amplifies the Power of Cache Attacks. In *Cryptographic Hardware and Embedded Systems (CHES)*.

[41] 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 *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[42] Dag Arne Osvik, Adi Shamir, and Eran Tromer. 2006. Cache Attacks and Countermeasures: The Case of AES. In *Cryptographers' Track at the RSA Conference on Topics in Cryptology (CT-RSA)*.

[43] Riccardo Paccagnella, Licheng Luo, and Christopher W. Fletcher. 2021. Lord of the Ring(s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical. In *USENIX Security Symposium*.

[44] Peter Pessl, Daniel Gruss, Clémentine Maurice, Michael Schwarz, and Stefan Mangard. 2016. DRAMA: Exploiting DRAM Addressing for Cross-cpu Attacks. In *USENIX Security Symposium*.

[45] Moinuddin K. Qureshi. 2018. CEASER: Mitigating Conflict-based Cache Attacks via Encrypted-address and Remapping. In *IEEE/ACM International Symposium on Microarchitecture (MICRO)*.

[46] Moinuddin K. Qureshi. 2019. New Attacks and Defense for Encrypted-address Cache. In *International Symposium on Computer Architecture (ISCA)*.

[47] 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 *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[48] Stephen Röttger and Artur Janc. 2021. A Spectre proof-of-concept for a Spectre-proof web. https://github.com/google/security-research-pocs/tree/master/spectre.js.

[49] Gururaj Saileshwar and Moinuddin Qureshi. 2021. MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design. In *USENIX Security Symposium*.

[50] Michael Schwarz, Moritz Lipp, Daniel Moghimi, Jo Van Bulck, Julian Stecklina, Thomas Prescher, and Daniel Gruss. 2019. ZombieLoad: Cross-Privilege-Boundary Data Sampling. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[51] Michael Schwarz, Samuel Weiser, Daniel Gruss, Clémentine Maurice, and Stefan Mangard. 2017. Malware Guard Extension: Using SGX to Conceal Cache Attacks. In *Detection of Intrusions and Malware, and Vulnerability Assessment (DIMVA)*.

[52] Anatoly Shusterman, Ayush Agarwal, Sioli O'Connell, Daniel Genkin, Yossi Oren, and Yuval Yarom. 2021. Prime+Probe 1, JavaScript 0: Overcoming Browser-based Side-Channel Defenses. In *USENIX Security Symposium*.

[53] 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 *USENIX Security Symposium*.

[54] Qinhan Tan, Zhihua Zeng, Kai Bu, and Kui Ren. 2020. PhantomCache: Obfuscating Cache Conflicts with Localized Randomization. In *Network and Distributed System Security Symposium (NDSS)*.

[55] Eran Tromer, Dag Arne Osvik, and Adi Shamir. 2010. Efficient Cache Attacks on AES, and Countermeasures. *Journal of Cryptology* (2010).

[56] Jo Van Bulck, Nico Weichbrodt, Rüdiger Kapitza, Frank Piessens, and Raoul Strackx. 2017. Telling Your Secrets Without Page Faults: Stealthy Page Table-based Attacks on Enclaved Execution. In *USENIX Security Symposium*.

[57] Stephan Van Schaik, Cristiano Giuffrida, Herbert Bos, and Kaveh Razavi. 2018. Malicious management unit: Why stopping cache attacks in software is harder than you think. In *USENIX Security Symposium*.

[58] Stephan Van Schaik, Alyssa Milburn, Sebastian Österlund, Pietro Frigo, Giorgi Maisuradze, Kaveh Razavi, Herbert Bos, and Cristiano Giuffrida. 2019. RIDL: Rogue In-flight Data Load. In *IEEE Symposium on Security and Privacy (S&P)*.

[59] Pepe Vila, Pierre Ganty, Marco Guarnieri, and Boris Köpf. 2020. CacheQuery: Learning replacement policies from hardware caches. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*.

[60] Pepe Vila, Boris Köpf, and José F. Morales. 2019. Theory and Practice of Finding Eviction Sets. In *IEEE Symposium on Security and Privacy (S&P)*.

[61] Daimeng Wang, Zhiyun Qian, Nael Abu-Ghazaleh, and Srikanth V Krishnamurthy. 2019. Papp: Prefetcher-aware Prime and Probe Side-channel Attack. In *Design Automation Conference (DAC)*.

[62] Zhenghong Wang and Ruby B. Lee. 2007. New Cache Designs for Thwarting Software Cache-based Side Channel Attacks. In *International Symposium on Computer Architecture (ISCA)*.

[63] Mario Werner, Thomas Unterluggauer, Lukas Giner, Michael Schwarz, Daniel Gruss, and Stefan Mangard. 2019. SCATTERCACHE: Thwarting Cache Attacks via Cache Set Randomization. In *USENIX Security Symposium*.

[64] Wenjie Xiong and Jakub Szefer. 2020. Leaking Information Through Cache LRU States. In *IEEE Symposium on High Performance Computer Architecture (HPCA)*.

[65] 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 *IEEE Symposium on Security and Privacy (S&P)*.

[66] Yuval Yarom and Naomi Benger. 2014. Recovering OpenSSL ECDSA Nonces Using the FLUSH+ RELOAD Cache Side-channel Attack. *IACR Cryptol. ePrint Arch. 2014/140* (2014).

[67] Yuval Yarom and Katrina Falkner. 2014. FLUSH+RELOAD: A High Resolution, Low Noise, L3 Cache Side-channel Attack. In *USENIX Security Symposium*.

[68] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2012. Cross-VM Side Channels and their use to Extract Private Keys. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.

[69] Yinqian Zhang, Ari Juels, Michael K Reiter, and Thomas Ristenpart. 2014. Cross-tenant Side-channel Attacks in PaaS Clouds. In *ACM SIGSAC Conference on Computer and Communications Security (CCS)*.