Bespoke Cache Enclaves: Fine-Grained and Scalable Isolation from Cache Side-Channels via Flexible Set-Partitioning

Gururaj Saileshwar
gururaj.s@gatech.edu
Georgia Tech

Sanjay Kariyappa
sanjaykariyappa@gatech.edu
Georgia Tech

Moinuddin Qureshi
moin@gatech.edu
Georgia Tech

ABSTRACT

Cache partitioning is a principled defense against side-channel attacks on shared last-level caches (LLCs). Such defenses allocate isolated cache regions to distrusting applications and prevent a spy from monitoring the cache accesses of a victim. But current solutions have severe practical limitations. Way-partitioning is not scalable as the number of partitions is limited by cache associativity and page-coloring is inflexible as it requires coupled DRAM and LLC allocations in the same ratio. For cache partitioning to be practical, we need a scheme that can scale to a large number of fine-grained partitions and places no restrictions on DRAM allocations.

This paper proposes *Bespoke Cache Enclaves (BCE)*, a secure cache partitioning substrate that is scalable in supporting hundreds of isolated cache partitions and is flexible in allocating cache space independent of memory allocations. BCE allocates cache space at the granularity of a *cluster*, a group of a few sets (*e.g.*, 64 KB in size). The key insight of BCE is a configurable cache indexing function (determining the line to set mapping) that guides cache lines of a domain to only the allocated cache sets, enabling flexible set-partitioning independent of memory allocations. BCE achieves this by modifying the cache indexing hardware to include a *Cluster-Indirection Module (CIM)*, which maps logical-to-physical clusters of a domain and a *Load-Balancing Hash (LBH)*, which uniformly distributes lines of a domain among its clusters. Our implementation of BCE with a 32MB 16-way LLC scalably supports up to 512 isolated partitions while incurring negligible storage overheads (<2%) and slowdown (1% on average) compared to a non-secure unpartitioned LLC.

1. INTRODUCTION

Cache side-channel attacks allow malicious programs to leak sensitive information from victim programs by observing their data accesses and changes to the shared cache state. Last-level caches (LLCs) are particularly vulnerable as they are typically shared among several processor cores which may simultaneously run programs of distrusting users. Such attacks can cause serious security and privacy breaches, such as leaking secret AES/RSA keys [24], user keystrokes [36], user browsing history [44, 45], etc. Thus, there is a pressing need for designing secure yet high-performance LLCs.

Cache attacks fall under three broad categories: *conflict-based attacks* (*e.g.,* Prime+Probe [27]) that leak information via evictions from cache sets shared between a victim and spy process, *shared-memory-based attacks* (*e.g.,* Flush+Reload [55]) that leak information via hits to shared cache lines, and *cache-occupancy based attacks* [44, 45] that leak information based on changes to the space used by the victim in the shared LLC. The two main classes of defenses for cache attacks are randomization and partitioning. *Randomized LLCs* [31, 32, 38, 46, 51] randomize the mapping of addresses to cache-sets to make conflict-based attacks harder and duplicate shared lines across trust domains [38, 51] to mitigate shared-memory attacks. However, they still allow a victim and a spy to contend for the limited cache space and are vulnerable to cache-occupancy attacks [44, 45]. Such attacks [44] can perform remote website fingerprinting across browser tabs using just HTML+CSS in a malicious page and can subvert even advanced browser security measures like Chrome Zero [39], which limits JavaScript and timer access.

Cache partitioning [5, 12, 14, 21, 23, 34, 48, 49] provides a principled defense against all three classes of cache attacks, by fully isolating the cache usage of victim and spy programs and is the focus of our paper. These defenses achieve such isolation by allocating non-overlapping cache regions to distrusting domains (encompassing cores, VMs, processes, or enclaves in a process). Without loss of generality, in this paper, we assume domains to be at the granularity of processes.

An ideal LLC partitioning defense should have the following properties:

1. *Security:* The scheme should prevent the spy from monitoring hits and misses of the victim by disallowing any accesses outside its allocated cache space and also prevent inferences from shared state like replacement policy.
2. *Scalability:* The scheme should support hundreds of isolated LLC partitions. This is because the LLC could be shared between 64 – 128 cores on server systems; even on client systems, applications like Chrome Browser seeking isolation between web pages can spawn 50 - 100 processes, as characterized by Chrome Site Isolation [35].
3. *Fine-Grained Allocations and Flexibility:* The scheme should provide fine-grained allocations as sensitive programs like encryption algorithms can have small cache working sets (*e.g.,* AES T-Tables are 8KB). LLC allocations should also be independent of memory allocations and flexibly manageable, as the need for memory (footprint) and cache capacity (locality) may not be correlated.

Unfortunately, no existing cache partitioning defense satisfies all three properties. *Way-partitioning* [14, 21, 23, 49] schemes allocate cache-space at way granularity, with each domain getting one or more LLC ways. But this has limited scalability as the number of partitions is restricted by the cache associativity: *e.g.*, a 16-way cache only supports up to

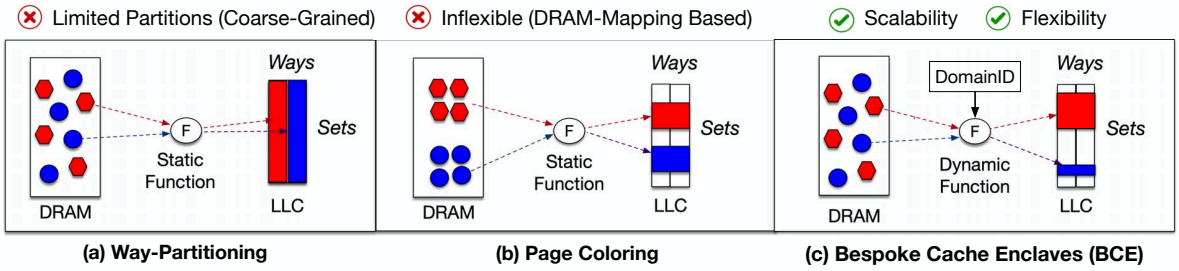


Figure 1: (a) Way-partitioning provides few partitions, restricted by the LLC associativity (b) Page-Coloring has finer allocations but does not allow flexible use of DRAM and LLC in different ratios. (c) BCE allows large number of fine-grained cache allocations and flexible memory usage, with dynamic indexing to guide lines to allocated cache regions.

16 partitions which is inadequate if the number of cores (or threads) increases beyond 16. Even if the number of ways equals number of desired partitions, such a design provides 1-way (direct-mapped) partitions with severe conflict-misses. Lastly, such solutions only provide coarse-grained allocations (*e.g.,* a 32MB 16-way cache provides 2MB partitions), which is inefficient for small working-set programs (*e.g.,* AES).

LLCs can also be partitioned along sets. As LLC sets allocated to a process depend on the locations of its physical pages, *Page-Coloring* [34] divides memory pages into different colors based on the cache sets they map to and allocates pages of only a subset of the colors to a process. MI6 [5] similarly achieves isolation by allocating physical memory to processes based on the sets the memory addresses map to. Such solutions are inflexible as they couple DRAM and cache allocations in the same ratio (*e.g.,* to obtain half the cache space, a process needs half the memory capacity), whereas memory (footprint) and cache capacity (locality) may not be correlated. Ideally, cache partitioning should not restrict virtual-to-physical mapping, keep memory and cache allocations independent, and scale to a large number of domains.

To that end, this paper proposes *Bespoke Cache Enclaves (BCE)*, a cache-partitioning defense that is scalable to hundreds of isolation cache partitions and has the flexibility to allocate cache resources independent of the memory mappings. BCE allocates cache space at the granularity of a *cluster*, a group of few contiguous cache sets (we use a cluster of 64 sets, *i.e.* 64KB). Each domain gets a configurable number of clusters isolated from other domains (each domain has its own "enclave" in the cache). The key insight enabling this is a flexible cache indexing function governing the line to set mapping, which ensures cache lines of a domain are only directed to the allocated clusters as shown in Figure 1(c).

To support configurable clusters per domain, the cache indexing in BCE needs to address several challenges:

1. *Non-Contiguous Clusters:* Over time, the available clusters could be fragmented all over the LLC and the clusters allocated to a new domain could be non-contiguous and at arbitrary LLC locations. The set-indexing needs intelligent indirection to map lines to arbitrary LLC locations.
2. *Non-Power-of-2 Clusters Allocation:* With different numbers of clusters per domain, some domains might be allocated a non-power-of-2 number of clusters to avoid leaving the clusters unallocated. The set-indexing needs intelligence to distribute lines uniformly among even non-power-of-2 clusters to minimize conflict misses.
3. *Constant Time Indexing:* As the set-indexing logic itself is shared by all domains, the indexing logic needs to be constant time to prevent any new timing side-channels.

BCE addresses these challenges with two hardware modules for set-indexing. First, a *Cluster-Indirection Module (CIM)*, which maps clusters of a domain (logical clusters) to physical clusters in the LLC. To ensure this mapping is constant-time, it uses two levels of indirection to perform the translation in 2-cycles. Second, to ensure uniform mapping of addresses within even non-power-of-2 clusters of a domain, it uses a *Load-Balancing Hash (LBH)* to randomize the mapping of addresses to logical clusters within a domain and minimize conflict misses.

BCE also provides ISA extensions for software to request isolated LLC regions (at cluster granularity), which can be used for adding cache isolation to software sandboxing solutions like Google's NaCl or Chrome Site Isolation; the set-indexing and its security are provided by hardware.

We analyze the security of BCE and show it provides the strong isolation of secure cache partitioning schemes (such as DAWG [21]) while significantly increasing the flexibility and scalability of partitioning by providing hundreds of allocations and at fine-granularity. At its best, BCE can significantly improve performance with bespoke-sized cache allocations (by up to 40% compared to DAWG as per our case study in Section 5.6). On average, BCE's performance is functionally equivalent to Page-Coloring (within 1% slowdown).

Overall, this paper makes the following contributions:

- We propose *Bespoke Cache Enclaves (BCE)*, a cache substrate that can provide fine-grained cache allocations, support large number of domains, while not placing any restrictions on page mapping policies and page sizes.
- We propose an indirection scheme (CIM) that allows the logical clusters of a domain to be placed at arbitrary LLC locations and yet have constant-time low-latency lookup.
- We propose *Load Balancing Hash (LBH)*, a simple and effective way to distribute the lines of a domain uniformly among an arbitrary number of clusters.

We evaluate BCE for a 16-core system consisting of a 32MB 16-way LLC. BCE incurs a slowdown of 1.3% compared to a non-secure baseline without any partitioning while being scalable to hundreds of partitions compared to about 16 with the state-of-the-art DAWG [21]. BCE incurs a storage overhead of less than 2KB for newly added structures and additional 9 bits per tag-store entry (less than 2% of storage).

2. BACKGROUND AND MOTIVATION

We first describe the types of cache side-channel attacks that leverage shared caches and then discuss prior cache partitioning works and their limitations to motivate our solution.

2.1 Threat Model

Our threat model, shown in Figure 2, assumes a victim and spy running on different cores as resource-allocation often occurs at the granularity of a physical core in virtualized or cloud settings. Here, L1 and L2 caches are typically core-private and sharing of caches occurs at the LLC (e.g. L3-cache); hence we focus on side-channel attacks via LLCs. In case the L1 and L2 caches are time-shared or spatially shared among a few distrusting hyper-threads, we assume they are partitioned along ways or flushed on context-switches. Note that our proposal is also quite applicable to systems that share large L2 caches among many cores (*e.g.* Apple M1 CPUs).

Similar to prior cache partitioning works [21, 23, 34], we limit our focus to stateful cache attacks, where the spy observes latency variation due to LLC state changes by the victim (e.g., install, replacement-state update) which remain in the cache and leak information much after the victim execution completes. Like prior works, our threat model excludes attacks relying on bandwidth contention on NoCs [28, 47] or LLC ports as they are transient in nature, are quite susceptible to noise as they typically cause relatively small latency differences, and are thus less concerning. Note that our solution does not hinder any mitigations for these transient attacks such as bandwidth partitioning or not scheduling distrusting applications simultaneously.

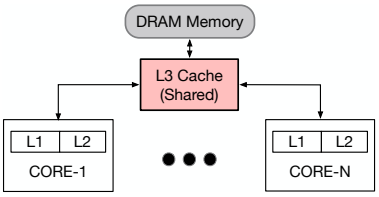


Figure 2: Threat model focuses on shared LLC attacks.

2.2 Types of LLC Side-Channel Attacks

There are three classes of LLC side-channel attacks capable of leaking confidential secrets from a victim process.

Conflict-Based Attacks. Such attacks rely on a spy being able to observe evictions of its lines based on addresses accessed by a victim due to set-conflicts. For example, in the Prime+Probe [27] attack, a spy primes cache-sets shared with a victim, allows the victim to execute and evict its lines, then observes which lines got evicted to infer addresses accessed by a victim; other attacks [6, 52] use changes to the order of evictions to infer a victim's replacement state updates. Such attacks require shared cache sets between a victim and a spy.

Shared-Memory Based Attacks. Such attacks rely on a spy being able to observe hits on addresses shared with a victim, that is typically read-only shared memory from shared-libraries or memory shared via de-duplication by the OS [1, 54]. A classic example is the Flush+Reload [55] attack, where a spy flushes a shared line from the cache, allows the victim to execute and install the line into the cache, then checks for a hit on the line to infer if the victim accessed it. Other variants include Evict+Reload [17], Thrash+Reload [40], etc. A key requirement of such attacks is the sharing of cache lines between distrusting processes.

Cache-Occupancy Based Attacks. Such attacks rely on a spy observing changes to its LLC working set due to cache usage by a victim, to infer sensitive information about a victim. A recent work [45] used such an attack to fingerprint websites visited on a web browser. In this attack, a spy web page periodically fills the entire LLC with its lines, allows the victim web page to execute, and then the spy counts the number of its lines that get evicted in each time period to develop a fingerprint of the website, which is matched with a repository of fingerprints to identify the website. A subsequent advancement [44] showed that such website fingerprinting attacks can be mounted with just HTML+CSS code rendered in a malicious web page on the same browser. Given such minimal requirements, these attacks are successful despite browser security measures like Chrome Site-Isolation [35], which renders web page content from different sources in separate processes, and Chrome Zero [39] which limits JavaScript and timers in web pages. All such attacks only require dynamic sharing of LLC space between a spy and the victim process, and hence are only mitigated if software explicitly requests isolated partitions of the cache for a spy and a victim.

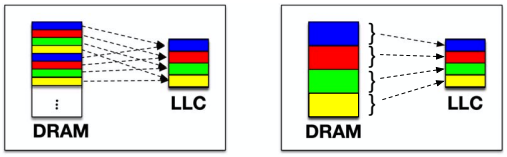
2.3 Cache-Partitioning Based Defenses

Partitioning the LLC across distrusting applications can prevent all three classes of cache attacks, as it provides a notion of strong isolation, i.e. the cache state observable by a program is solely a result of its own execution and is unaffected by any other program's cache-accesses. Current LLC-partitioning schemes broadly fall under two categories.

2.3.1 Way-Partitioning

Such solutions partition the LLC based on ways, allocating one or more non-overlapping ways to processes in different trust-domains. The state-of-the-art way-partitioning solution, DAWG [21], stores a software-configurable bit-mask for each domain in the cache-controller, which is used to determine the ways to be checked for cache-hit determination on LLC-accesses and eviction-decisions on LLC-misses. This allows DAWG to provide isolation across cache partitions of different trust-domains and prevent cross-domain hits or evictions or replacement policy updates. Along with OS support to ensure processes across different trust-domains only share read-only lines which are duplicated in each cache partition, this design prevents any sharing of cache-space or shared-lines or shared-sets between domains.

Pitfall: Unfortunately, the number of simultaneous trust-domains in such a design is limited to the LLC associativity (number of ways). With associativity being limited to 16 – 32 for LLCs as the number of cores continues to increase, way-partitioning does not scale to a large number of trust-domains. Moreover, the partitions available are coarse-grained (2MB for a 32MB 16-way LLC), which results in inefficient cache allocations for applications with small cache working sets (*e.g.* AES encryption). Ideally, LLC allocations should scale to a large number of domains and be fine-grained in size.



(a) Round-Robin Colors (b) MI6

Figure 3: Page-coloring based cache partitioning. (a) Typical schemes assign consecutive colors to consecutive physical pages. (b) MI6 [5] partitions both the DRAM and cache sets into 64 contiguous regions and assigns an entire DRAM and cache region a single color.

2.3.2 *Page Coloring*

Page Coloring [34, 43] defenses partition the LLC along sets by dividing memory pages into colors based on the cache sets they map to and allocating pages of different colors to distrusting processes. Typically, such schemes assign distinct colors to successive 4KB pages, as shown in Figure 3(a), and the OS allocates pages of as many colors to a process as the amount of cache space desired. This supports a large number of colors (a 4KB page of one color maps to 64 LLC sets) and up to 512 trust-domains with a 32MB, 16-way LLC. However, such solutions are not compatible with large pages (*e.g.,* a 2MB page covers the entire 32MB 16-way LLC with a single color), which can lead to significant TLB pressure.

MI6 [5] makes page-coloring friendly to large pages. It statically partitions the DRAM and LLC space into 64 contiguous regions and modifies the cache indexing to have a static 1-to-1 mapping of each DRAM and LLC region, as shown in Figure 3(b). As each LLC and DRAM region have one color, MI6 supports up to 64 colors and 64 trust domains. While domains may be assigned more than one color, to effectively utilize multiple cache regions, a domain requires its working-set to be spread by the OS uniformly across all allotted DRAM regions. However, MI6 observes that OS typically allocates pages contiguously causing clustering of hot data in one region and inefficient LLC utilization.

Pitfall: All page-coloring solutions cannot manage DRAM and cache space independently as they require allocations of both in the same ratio (*e.g.,* to claim half the cache, a domain must be given half the DRAM space). This results in inefficient use of cache space or memory space: an application with a large memory footprint but poor cache locality (*e.g.,* graph applications) needs significant DRAM space but not much cache space; meanwhile, a data-intensive kernel with high locality but small memory footprint (e.g. encryption or small matrix-multiply) may not need much DRAM space but benefits from larger cache space. Ideally, cache partitioning decisions should be independent of the memory allocations.

2.4 Goal: Scalable & Flexible LLC Isolation

Our goal is to develop a secure cache partitioning substrate for LLCs that scalably supports a large number of isolated cache partitions (without being restricted by the associativity of the cache) and performs fine-grained cache allocations. At the same time, it should be flexible in making cache allocations to a domain without placing any restrictions on the memory capacity allocation, virtual-to-physical mapping, and page sizes. To that end, we propose *Bespoke Cache Enclaves (BCE)*, a design that achieves all these goals.

3. BESPOKE CACHE ENCLAVES

To develop a scalable and flexible LLC partitioning scheme, we focus on the cache indexing function which governs the mapping of addresses to cache sets. Prior cache partitioning defenses, using way-partitioning or page-coloring (including MI6), have fixed indexing functions configured statically at design time. The key insight of our work, *Bespoke Cache Enclave (BCE)*, is that the cache indexing can be made dynamic to guide addresses of a domain to only the isolated cache region (or cache "enclave") of the domain. We first provide an overview of BCE and then the design of its components.

3.1 Overview of BCE

BCE is a set-partitioning scheme that allocates cache space at the granularity of *clusters* (group of contiguous sets) and provides each security domain with an isolated cache partition consisting of one or more clusters. Without loss of generality, we use a cluster size of 64 sets (64KB) in our design to match the granularity of page-coloring (although our design can support clusters as small as a single set). Thus, our 32MB 16-way LLC is divided into 512 contiguous clusters (also referred to as *physical* clusters) and each of these is identified by a unique *Physical Cluster ID (PCID)*.

BCE allows each domain to be allocated a configurable number of clusters (also referred to as *logical* clusters), and these clusters could be located at any PCID in the LLC. For example, as shown in Figure 4, Domain-A (*e.g.,* a cache-sensitive workload like AES) could be allotted many LLC clusters and Domain-B (*e.g.*, a cache-insensitive graph application) could have only one LLC cluster, and these could be located at arbitrary physical locations in the LLC. Additionally, BCE allows independent LLC and memory allocations. For example, Domain-A could have a small memory footprint whereas Domain-B could have a large footprint, independent of their cache allocations. BCE allows complete flexibility in LLC management, supporting few domains with hundreds of clusters per domain or even hundreds of domains with few clusters per domain. The indexing hardware of BCE is responsible for mapping a given line address and Domain-ID into a cache set in one of the allocated clusters of the domain.

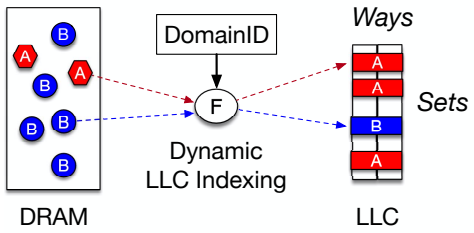


Figure 4: Overview of BCE Capabilities. BCE allows a configurable number of LLC clusters to be allocated to each domain, independent of memory allocations.

Challenges. As the number of clusters given to each domain can vary (from 1 to 512 in our design) and these clusters could be located anywhere in the cache, there are two key problems faced by BCE in mapping addresses to cache sets. First, how to store the mappings of where the clusters of each domain are physically located in the LLC while ensuring fast lookup, given that the clusters can be non-contiguous

and in arbitrary locations. Second, how to uniformly map addresses among the clusters of a domain to minimize set-conflicts, given that each domain can be configured with a different number of clusters. The first problem is a mapping problem that can be handled via indirection; the second is a load-balancing problem that can be handled via effective hashing of addresses. Lastly, the security of isolated cache regions must be guaranteed by the indexing logic in hardware, which itself needs to have constant-latency lookup to prevent new timing side-channels, as it is shared among all domains.

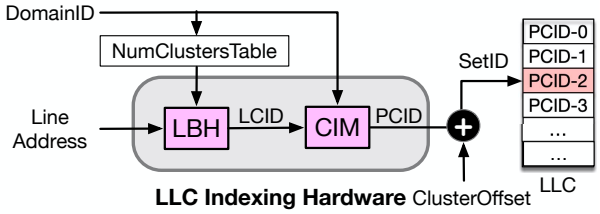


Figure 5: Overview of BCE Cache Indexing, which maps Line Address and Domain-ID to set-index. The Load-Balancing Hash (LBH) uniformly hashes addresses among *Logical Clusters* of a Domain (LCID) and the Cluster-Indirection Module (CIM) maps LCIDs of a domain to *Physical Cluster IDs (PCIDs)* of the LLC. PCID in conjunction with cluster-offset provides the set index.

Solutions. BCE's cache indexing hardware has two components, as shown in Figure 5. First, a *Cluster-Indirection Module (CIM)* to store and lookup the physical locations of the clusters of a domain (*i.e.,* logical clusters identified by LCID). Second, to determine which LCID an address maps to, a *Load-Balancing Hash (LBH)* is used, which maps addresses of a domain uniformly across all its clusters, regardless of the number of clusters. On an LLC access, the LBH maps a line address and DomainID to an LCID and the CIM lookup using this LCID provides the physical LLC location of the cluster (PCID), where the set corresponding to the line address is located. The set within the PCID is identified by the *cluster-offset bits* (the 6 least-significant bits of the line-address) and indexed by concatenating the PCID and cluster-offset bits. Note that BCE guarantees the isolation properties for security in hardware and is carefully designed to provide constant-time accesses to avoid new side-channels, while software is only responsible for deciding the number of clusters for each domain. We now describe each component of BCE.

3.2 Cluster-Indirection Module: Maps Clusters

The Cluster-Indirection Module (CIM) is responsible for locating the LLC clusters allocated to a domain. To access a cache set, line addresses are hashed to logical clusters private to a domain (by the LBH), which may be located anywhere in the LLC. To locate these clusters, each logical cluster of a domain, *i.e. Logical Cluster ID (LCID)*, must be translated to its allocated LLC cluster, *i.e. Physical Cluster ID (PCID)*.

Design Constraints. The design of the CIM that can translate domain LCIDs to LLC PCIDs has two key requirements: *flexibility* and *fast lookups*. With our LLC divided into 512 clusters, there can be up to 512 domains each with one cluster or one domain with all 512 clusters. The CIM data structures storing the cluster mappings need to be flexible enough to store these mappings for allocations of up to 512 clusters, for up to 512 domains. Additionally, the CIM needs to support fast lookup as it is in the critical path of LLC accesses. One option for storing cluster-mappings is a hash-table. However, such a design is prone to hash-conflicts and variable access-latency, which is a security vulnerability. As the CIM is shared by all domains, including mutually distrusting ones, the CIM itself can become the target of conflict-based attacks, where a spy in one domain observes variation in latency to access cluster-mappings due to changes in mappings a victim in another domain. To prevent such new side-channels, we need to ensure the CIM has constant access latency. To that end, we design the CIM with two tables and indirection.

Design. Figure 6 shows the two-level CIM design. The *Cluster Location Table (CLT)* keeps an ordered list of valid LCID to PCID mappings for each domain with at least 1 valid cluster. The mappings for each domain are kept contiguous in the CLT, starting with the base entry (LCID-0) for each domain. The *Domain Base Table (DBT)* tracks the location of the base entry (LCID-0) in the CLT for each domain. To translate an LCID to PCID, the DBT is accessed with the DomainID to obtain a Pointer to the Base Entry (BasePtr) of that domain in the CLT. Adding the LCID to the BasePtr provides the CLT location for the LCID of that domain. Accessing this CLT location provides the required PCID, which is concatenated with the cluster-offset bits to get the set-index.

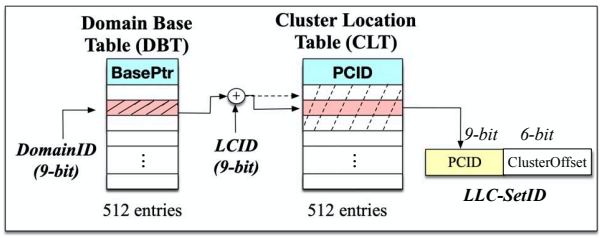


Figure 6: Design of the CIM. The DBT provides the base entry location (LCID-0) of a domain in the CLT. The base location added to the LCID, points to the CLT-entry containing the required PCID.

Isolation Guarantees. The CIM hardware, by design, provides the security guarantee of cache isolation, as only the LLC clusters allocated to a domain (those addressable by the CLT entries of the domain) are accessible to cache lookups from that domain. Only the CLT entries of a particular domain are accessible on cache lookups from that domain, as the LCID used to calculate the CLT index is always less than the number of clusters allocated to that domain (guaranteed by the LBH and the NumClustersTable in Figure 5). A more end-to-end security analysis is provided in Section 4.

Storage, Logic and Latency. To support up to 512 PCIDs, the CLT needs at most 512 valid entries. Similarly, the DBT requires at most 512 entries for 512 domains. As both tables have 9-bit entries and a 1-bit valid-bit per entry, the tables require a total of 640 bytes storage. Such small tables can be implemented as registers for fast lookup. Thus, the translation of LCID to PCID via CIM requires two register lookups and one combinational 9-bit adder, taking 2 CPU cycles.

3.3 Load-Balancing Hash: Maps Lines

While CIM ensures secure isolation between clusters of different domains, the performance of BCE relies on how uniformly addresses map to clusters of a domain (logical clusters) to minimize set-conflicts. The Load-Balancing Hash (LBH) is responsible for uniformly mapping line addresses to logical clusters of a domain (LCIDs).

3.3.1 The Problem of Load Balancing

In conventional LLCs, a modulo function is used as a uniform set-indexing function (`SetID = LineAddr % NumSets`), that is equivalent to selecting least significant bits of the line address for power-of-2 NumSets. However, in BCE, each domain could have a configurable number of clusters (from 1 to 512). These are not constrained to be a power-of-2, as that can result in under or over-provisioning of cache space for larger cache allocations. Ideally, we would like LBH to have the uniformity of the modulo function shown in Figure 7(a); but computing a modulo for non-power-of-2 divisors is expensive (requires tens of cycles)[1] for cache-indexing.

Imbalance with Simple Hashes. Figure 7(b) shows a simpler, single-cycle hash called *Linear-and-Invert*, which maps n-bits of a line-address to an n-bit LCID (where n is log of NumClusters, rounded up to the next integer). If the n-bit Line-Address value, say x, is less than the NumClusters, x is used as the LCID, else x is inverted and used as the LCID. Thus, either x or Invert(x) is guaranteed to be less than NumClusters and suitable to be used as the LCID. While this simple hash is low latency, it has significant non-uniformity. For example, if a partition has 3 clusters, using two bits of line address as input to the hash results in one cluster receiving 50% of the lines and other two clusters with 25% lines each.

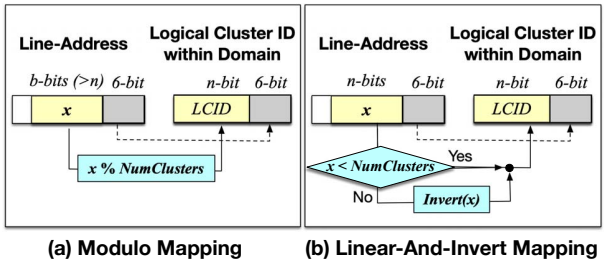


Figure 7: Candidates for LBH (mapping addresses to LCID): (a) Modulo mapping is uniform but requires multi-cycle hardware implementation. (b) Linear-And-Invert mapping is single-cycle but results in imbalance.

We quantify this imbalance with a metric called *Load Imbalance*, *i.e.* the ratio of maximum to average lines per cluster. Figure 8 shows the load imbalance for the Modulo and Linear-And-Invert hashes while streaming a large number of lines, as clusters per domain varies from 1 to 512. Modulo mapping is close to the ideal value of 100%, whereas linear-and-invert has imbalance of almost 200%, *i.e.* some clusters have 2x the average accesses.

[1]Hardware implementation of modulo with non-powers-of-2 divisors requires recursive division with $O(b)$ pipeline stages, [8] where b is the number of bits of dividend (line-address). For our design, this requires more than 10 stages and a latency of tens of cycles.

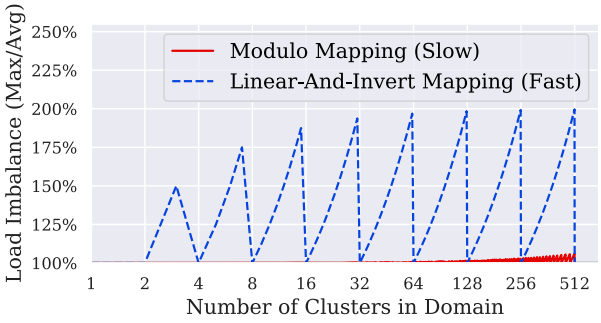


Figure 8: Load imbalance with Modulo and Linear-And-Invert mappings as the number of clusters in a domain varies from 1 to 512. Modulo is close to ideal but slow. Linear-And-Invert is fast but has up to 2x imbalance.

3.3.2 Solution: LBH with Randomizing Hashes

Design. To achieve near-ideal load imbalance with a single-cycle LBH, we construct it using multiple randomizing hashes as shown in Figure 9(a). To map a line address to LCID, the bottom n bits of the line-address (x) are checked to see if it is less than NumClusters (where n is log of NumClusters rounded to next integer) and can be directly used as LCID (like linear mapping). If not, the line address is transformed using a low-latency randomizing hash, $H_1(x)$, and the bottom n bits are checked again to see if they can be used as LCID. This process repeats k times with k randomizing hashes; if LCID is still not obtained, then the inverted value ($\sim x[n-1:0]$) is used as the LCID. As each hash is expected to produce the LCID with a probability greater than half, the likelihood of using the inverted mapping after 3 - 5 hashes is quite small and the overall mapping is balanced.

Implementation. To ensure $H_1(x)$ to $H_k(x)$ are sufficiently uniform and independent, while also having a fast hardware implementation, we construct the hash-functions using Random Binary Matrices (RBM). As shown in Figure 9(b), each $H_i(x)$ consists of $B \times n$ bits sized matrix (default: 24×9) with bit values statically populated from a uniform random distribution. Each bit of the output hash is computed as bit-wise AND of the input-vector and the corresponding row of the RBM, followed by an XOR reduction. Successive hashes (H_1 to H_k) are generated with different matrix bit values, all sampled from the same uniform random distribution. Such a construction of $H_1(x)$ to $H_k(x)$ has the property that any two functions selected at random only have a small probability of generating the same hash for the same input (such RBMs have been shown to create Universal Classes of Hashing Functions with this property [9]), and hence suited for our purpose.

Results - Reduced Imbalance. Figure 10 shows the load imbalance with LBH using multiple randomizing hashes. We choose RBM dimensions of 24×9, to support LCID of up to 9-bits to support up to 512 NumClusters. Using 1 hash reduces the worst-case imbalance to 112.5%, 3 hashes reduce it to 101.3%, and 5 hashes to 100.3% (in comparison, using 0 hashes, which is equivalent to the linear-and-invert, results in 200% imbalance as shown in Figure 8). With 3 – 5 hashes, the worst-case load imbalance is very close to ideal 100% and thus the mappings with LBH adds negligible conflict misses

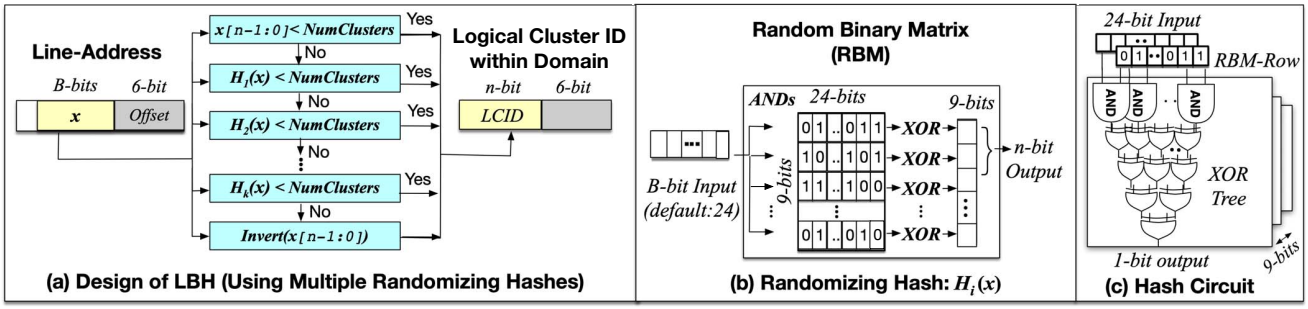


Figure 9: Design of Load-Balancing Hash (LBH). (a) Mapping of Line-Addresses to LCID via multiple randomizing hashes: If a linear-mapping ($x[n-1:0]$) does not work, n-bit hashes $H_1(x)$ to $H_k(x)$ are used to attempt different mappings; if none works, inverted-mapping ($\sim x[n-1:0]$) is used. (b) Implementation of randomizing hash $H_i(x)$: A static 24×9 bit Random-Binary-Matrix (RBM) is used, with each $H_i(x)$ matrix having different random values. Each output-bit computation involves bitwise-AND of input with an RBM-row followed by an XOR-reduction. (c) Hash Circuit Logic: Circuit has a critical path of 1 AND and 4 XORs (three 2 input and one 3-input in XOR Tree). As all $H_i(x)$ are computed in parallel, the critical path of LBH consists of $k+1$ n-bit comparators, 1 AND and 4 XORs.

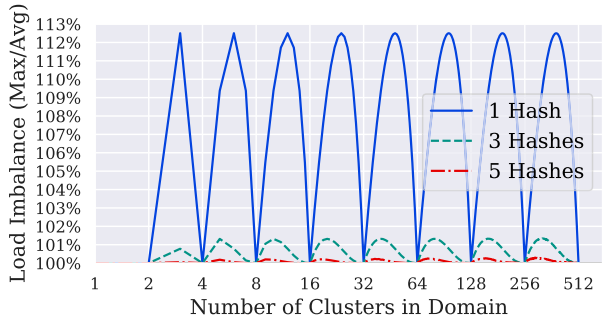


Figure 10: Load Imbalance with LBH using multiple randomizing hashes (24-bits of line-address as input) as the number of clusters in a domain varies from 1 to 512. With 3–5 hash functions, imbalance is within 1% of ideal.

due to non-uniformity. Note that the randomizing LBH mapping is statically provisioned at design-time, deterministic, and only for performance: it can be disclosed to software if needed for software to customize data-structure layout for further minimizing set-conflicts.

Logic, Latency and Storage. The circuit for each hash, shown in Figure 9(c), has a critical path of 1 AND gate and 4-5 XOR gates (for XOR-reduction) from input to output. For LCID computation with LBH, the k hashes ($k = 3 \ldots 5$ is sufficient) can be computed in parallel, and hence the overall critical path has only one hash circuit (1 AND and 4 XOR gates) and $k+1$ n-bit comparators to check if hash-values are less than NumClusters. This computation incurs a latency of less than one CPU cycle (as cycle time in modern processors is typically designed for about 20 gate delays). The number of hash-functions and input-size (i.e. number of bits of the line-address used as the input to the hashes) determine the storage overheads of LBH. Using more input-bits considerably reduces imbalance as LBH generates more uniform mappings over a larger input space; we find that using 5 hash functions of 24×9 bits is sufficient for minimizing the load imbalance and requires negligible storage of 135 bytes.

3.4 Software Interfaces to Request Clusters

Like prior cache partitioning defenses [5, 21], BCE provides software interfaces to request isolated LLC partitions of a given size from the hardware. BCE has two new privileged instructions, `BCE_alloc` and `BCE_dealloc` for creation and deletion of LLC allocations for domains, usable by privileged software like OS responsible for resource allocation. The OS may provide system-calls that allow applications to request custom-sized LLC allocations (with appropriate fairness checks); this can spur research on real-world systems using customizable cache isolation (*e.g.,* extensions of Chrome Site Isolation [35] providing web pages bespoke cache isolation along with process isolation). We now describe the hardware operations performed on these instructions.

`BCE_alloc`: This instruction creates a new LLC partition for a trust domain. It takes two inputs: the DomainID and the number of clusters to be allocated to the domain. If the requested number of LLC clusters are available, then the instruction returns success, otherwise failure. If successful, the cache controller initializes a contiguous list of CLT entries (at the end of all valid CLT entries) with the PCIDs allocated for this domain. The DBT entry for the domain is initialized to point to the first CLT entry of the domain and the NumClustersTable is updated with the allocated clusters.

`BCE_dealloc`: This instruction deallocates an existing LLC partition. It takes a DomainID as an input, and invalidates the associated DBT and CLT entries. Subsequently, the lines in the physical LLC clusters of the domain are flushed and PCIDs of that domain become free to be allocated to other domains. The invalidated CLT entries are also compacted and moved to the end of the CLT to ensure that a subsequent cache allocation can obtain a contiguous cluster of free CLT entries. Note that the CLT compaction takes tens of cycles and is much faster than the latency to flush contents of the freed cache clusters. Fortunately, deallocations are quite infrequent (only on domain termination, resizing, etc.).

Note that if a domain requires LLC space but the LLC is fully allocated, partitions of inactive domains (that are context-switched out) can be deallocated, flushed, and then re-allocated to a new domain (similar to prior works [21]).

3.5 Putting it Together: BCE Operation

Process-Start. When a new program is launched, it can be assigned to an existing domain or a new domain. For programs not desiring security, a single large non-secure LLC partition can be perpetually reserved. For a process desiring a new security domain, the OS uses `BCE_alloc` to allocate an isolated LLC partition and embeds the DomainID in the process context. Note that for the security of L1 and L2 caches, if they are private to a domain, they are flushed before a new domain begins; if they are shared among a few hyper-threads/cores, they must be way-partitioned.

LLC-Indexing. All read and write accesses to the LLC in BCE are accompanied with the DomainID from the thread-context. For an LLC-access, the LBH hashes the line address into the logical cluster ID (LCID) and then CIM translates the LCID into the physical cluster ID (PCID). The PCID in conjunction with the cluster offset bits determines the set index for the LLC access and the access then proceeds by checking this set for a tag-match like a conventional LLC.

LLC-Hit/Miss. On LLC-Hits, replacement state updates proceed unchanged using any intelligent replacement policy, as the entire set belongs to a single domain. On an LLC-miss, the eviction candidate selection is unchanged and is chosen from the entire set. On dirty eviction, the line addresses for writebacks are generated by concatenating the tag with cluster offset of the set; tags are enlarged by 9 bits to support this.

Process-End. When a domain finally exits, the OS uses `BCE_dealloc` to reclaim the LLC partition allocated to it.

3.6 Support Required from System Software

Similar to prior secure cache partitioning works [5, 21], BCE requires additional support from system software for ensuring correctness and security.

Page-Sharing: Processes that trust each other are allotted the same DomainID; they share cache-sets with each other like conventional caches and have read-write sharing of pages. Page sharing between distrusting processes in different domains has to be regulated by the OS like in [51], to ensure no read-write sharing between them and to prevent direct information leakage. For read-only shared pages (whose shared cache lines get duplicated in each domain), the OS is responsible for flushing such pages from each of the LLC domains when such pages are swapped out or unmapped (just as the OS updates the page-tables for each of the processes).

Inter-Process-Communication: Data transfer between processes in a single trust-domain with read-write shared memory is unchanged. Data transfers between processes in different domains have to be marshaled by the OS with special system calls to ensure no secret-dependent data leaks via such transfers. Also, kernel memory-copying functions performing data transfer from user-space processes to kernel, like `copy_from_user`, need to be modified to ensure the data is read from user's cache-partition and written to kernel's cache-partition (vice-versa for `copy_to_user`), like in [21].

Coherence: For cache coherence within processes of the same domain, coherence protocol is unchanged, except that all coherence packets include the Domain-ID to ensure correct LLC sets are accessed on snoop/invalidation requests.

4. SECURITY ANALYSIS

Assumptions. The security focus of BCE is on preventing cross-domain side-channels relying on LLC state changes, where a spy observes timing differences due to cache state modified by secret-dependent activity of the victim. These include conflict-based attacks [6, 13, 27, 52] exploiting cross-domain evictions from shared cache-sets, shared-memory-based attacks [16, 18, 37, 54, 55] exploiting cross-domain hits on shared cache lines, and cache-occupancy based attacks [44, 45] exploiting changes to LLC space used per domain. While the analysis uses the cross-process setting for BCE (distrusting processes mapped to different domains) for its arguments, it is applicable for LLC-isolation between VMs or even enclaves within a process, if those map to a security domain. Our focus is the shared LLC, as L1 and L2 caches are typically private. If L1 or L2 are shared, BCE assumes they are way-partitioned [21] across domains; this is practically feasible as typically L1 or L2 caches are shared by no more than 2–4 threads at a time.

Security Guarantees. BCE promises strict isolation between LLC partitions of different security domains. The invariant it guarantees is: *the state of the LLC-partition of one domain is solely determined by its own LLC accesses and not influenced by any other domain's LLC accesses.* BCE achieves this by guiding addresses of different domains to disjoint LLC sets via its flexible LLC-indexing, duplicating lines of read-only shared addresses across domains, and disallowing read-write sharing of addresses across domains. As a result, all LLC operations of a domain only affect the LLC-state of its own domain and cannot affect other domains:

- LLC read or write accesses only have hits on lines within the LLC sets allocated to the domain; cache-flushes only impact cache lines within a domain and do not affect any duplicate lines for the same address in other domains; coherence-related invalidations are only allowed within the same domain – this prevents any cross-domain shared-memory attacks.
- Replacement state updates and victim-selections within a single set are only performed among the cache lines of a single domain – this prevents any cross-domain conflict-based attacks.
- LLC-partition sizes get allocated per domain by privileged software. The partition sizes are allocated at the domain creation time and are independent of any secret-dependent accesses of co-running domains – this prevents any occupancy-based attacks.

The BCE substrate itself does not introduce any new side-channels. While the metadata tables in BCE that maintain cluster-mappings (CIM tables) are shared across domains, the set-index computation using these tables has constant latency irrespective of the number of domains in use or their allocation sizes. Additionally, malicious processes cannot escape the LLC-partition sandbox enforced by BCE, by accessing or modifying these mappings as they are maintained in micro-architectural structures and inaccessible to software. This ensures the BCE-substrate is itself secure and free from any new attacks.

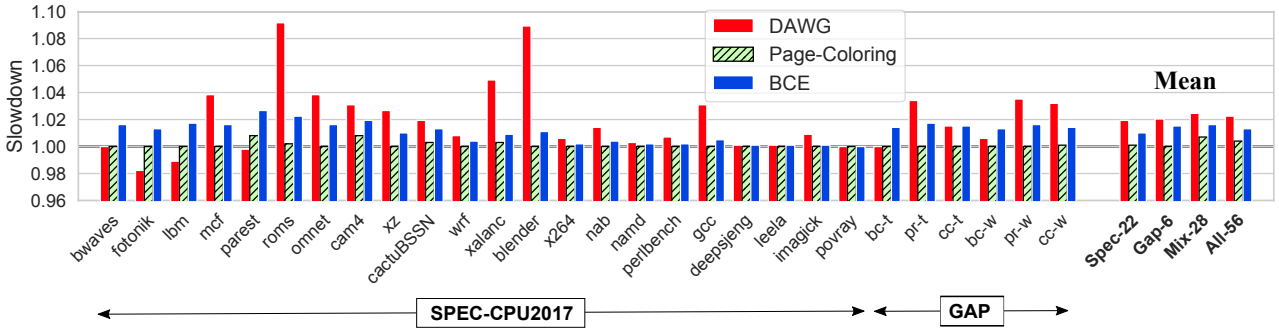


Figure 11: Slowdown of BCE compared with DAWG [21] and Page-Coloring [34]. All numbers are normalized to a Non-Secure baseline without partitioning. Across 56 workloads (sorted high to low by MPKI in SPEC and GAP suites), BCE has an average slowdown of 1.3%, while DAWG and Page-Coloring have slowdowns of 2.2% and 0.4% respectively.

5. EVALUATION RESULTS

In this section, we evaluate the performance and costs of our proposed BCE design and compare it against prior LLC-partitioning schemes DAWG [21] (that partitions along ways) and Page-Coloring [34] (that partitions along sets) and a non-secure baseline LLC without any partitioning.

5.1 Methodology

The system modeled in our study is shown in Table 1. We use a 16-core system with 32MB 16-way shared L3 cache. For performance evaluations, we use a trace-driven simulator running program execution traces (of length 1 billion instructions) generated using Intel-Pintool [25], drawn from a representative slice of a program using Simpoints [42]. We evaluate 56 workloads, including 22 SPEC-CPU2017 [7] and 6 GAP [2] (graph algorithms *bc*, *cc*, *pr* with *twitter* and *web* datasets) benchmarks (16-core rate-mode), and 28 mixed workloads (each has 16 randomly chosen SPEC or GAP benchmarks). Our baseline is a non-secure shared LLC. For BCE, we use a 3 cycle overhead for LLC lookup, based on the latency estimated in Sections 3.2 and 3.3.2.

Table 1: Baseline System Configuration

Processor and Private Caches	
Core	16-cores, In-order Execution, 3GHz
L1, L2-Cache / core	L1-32KB, L2-256KB, 8-way, 64B linesize
Last-Level Cache and Main-Memory	
LLC (shared)	32MB, 16-way, 64B linesize, Non-Inclusive SRRIP Repl [19], 24 cycle latency
DRAM	45 ns latency

5.2 Impact on Cache Misses

Table 2 shows the impact of partitioning on the LLC-misses per thousand instructions (MPKI) averaged across the different workload suites for BCE, Page-Coloring and DAWG, compared to a non-secure LLC without partitioning. Each of the 16 benchmarks in a workload runs on a separate core and only the LLC is shared: for the LLC-partitioning schemes, we assume each benchmark is allocated an equal-sized LLC-partition. All the partitioning schemes incur a higher number of misses than the non-secure baseline due to the capacity sharing restrictions imposed.

Table 2: Average LLC MPKI for Non-Secure, DAWG, Page-Coloring and BCE.

Workloads	Non-Secure	DAWG	Page-Coloring	BCE
Spec-22	8.42	8.73	8.44	8.44
Gap-6	41.64	42.61	41.63	41.63
Mix-28	27.56	29.82	28.53	28.59
All-56	**21.55**	**22.91**	**22.04**	**22.07**

On average, partitioning the LLC with BCE increases the LLC MPKI by 2.4% which is near-identical to the increase with Page-Coloring, compared to non-secure LLC. This is because both Page-Coloring and BCE allocate similar-sized set-partitions under an equi-partitioning policy.

DAWG has 6% higher MPKI than the non-secure design and this increase is more than double that of BCE. This is due to increased conflict-misses, as with equal partitions, DAWG allocates each core with a 2MB direct-mapped partition. Among high MPKI workloads, *roms* is the worst affected with DAWG, incurring a 20% increase in LLC MPKI.

5.3 Impact on Performance

Figure 11 compares the slowdown of BCE, Page-Coloring and DAWG (normalized to non-secure LLC). Across 56 workloads, BCE has an average slowdown of 1.3%. The main drivers of slowdown in BCE are the increase in both the LLC-misses and the LLC-access latency. The LLC-misses increase by less than 3% due to the cache space restrictions with partitioning, while the cache latency increases by 3-cycles. In comparison, Page-Coloring incurs a slowdown of 0.4%, as it incurs similar misses as BCE, but does not increase the latency for LLC-accesses. However, note that Page-Coloring requires memory allocation in the same ratio as LLC allocations, which can lead to memory pressure. Ideally, LLC and DRAM allocations should be independent like in BCE.

DAWG incurs a higher slowdown of 2.2%, mainly due to its higher LLC conflict-misses, which outweighs the fact that it has an identical access-latency as a non-secure LLC. The higher conflict-misses cause DAWG to incur a worst-case slowdown of 8% slowdown for *roms*, with multiple workloads suffering slowdowns of 4 - 8%. In contrast, BCE incurs a worst-case slowdown of 2.5% for *parest* and 1 - 2% for other high MPKI workloads, which are sensitive to the increase in LLC latency in BCE.

5.4 Sensitivity to Increase in LLC Latency

We use a latency overhead of 3 cycles for BCE set-index computation. Figure 12 shows the slowdown of BCE (normalized to the non-secure baseline) as the latency overhead is varied from 1 cycle to 6 cycles. As the latency of BCE increases, the slowdown increases from 0.8% (for 1-cycle latency) to 2.3% (for 6-cycle latency). In comparison, page-coloring, which has a 0-cycle additional latency, incurs a 0.4% slowdown. Thus, our 3-cycle index-calculation (with 1.3% slowdown) adds less than 1% extra slowdown on average due to the added lookup LLC latency, which is negligible.

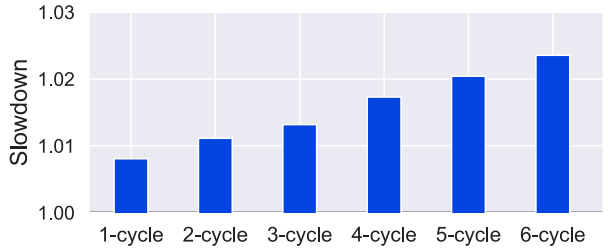


Figure 12: BCE slowdown as the latency overhead of set-index computation varies (default: 3 cycles)

5.5 Sensitivity of Performance to LLC Size

As LLC size increases from 8MB to 64MB, slowdown of BCE compared to non-secure LLC varies from 1.1% to 1.8%. At larger sizes, application working sets start to fit in. Therefore, at higher LLC hit-rates, the impact of the increased access-latency and misses is more pronounced; yet, BCE sustains a slowdown <2%. Similarly, Page-Coloring has slowdown varying between 0.2% to 0.9%; slowdown for DAWG varies between 2.2% to 2.7% with increasing cache-size as its higher conflict-misses result in higher slowdowns.

5.6 Benefits of BCE's Fine-Grained Allocations

With DAWG, the cache partitioning scheme is constrained to making coarse-grained allocations at way-granularity. For a 16-core 16-way, 32MB LLC system, DAWG results in inflexible LLC allocations where each core gets 1-way 2MB partition regardless of its requirement. This is quite limiting if one core needs LLC space and the other 15 cores do not benefit from the allotted LLC space. As BCE has fine-grained allocations (at the granularity of 64KB), it allows resource allocation policies the flexibility to allocate few LLC clusters to programs not requiring much LLC space and more clusters to programs that require and benefit from LLC allocations.

Figure 13 shows such a scenario where *roms* is running with 15 copies of *PageRank* (*pr*). *roms* needs cache space, whereas *pr* does not benefit much from it. However, DAWG-Eq is still forced to equally allocate 2MB LLC (1-way) to *roms* and each copy of *pr*. With BCE, the LLC space allocated per *pr* copy can be 2MB if distributed equally among all 16 cores (BCE-Eq), or smaller allocations of 1.5MB, 1MB or 0.5MB per copy of *pr* are possible (with allocation policies: BCE-1, BCE-2, BCE-3) allowing *roms* LLC allocations to go up to 9.5MB, 17MB, 24.5MB respectively; with such BCE allocations, relative IPC of *roms* versus DAWG increases by up to 40%, while IPC of *pr* is unaffected (<2% change).

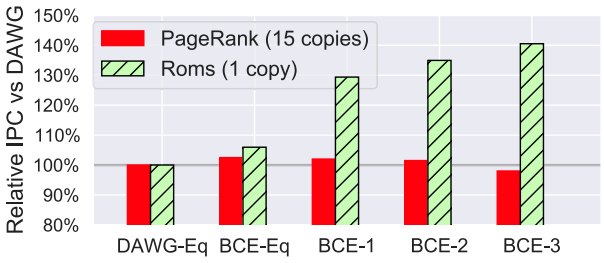


Figure 13: Flexibility of BCE vs DAWG. BCE allows smaller allocations to cache-insensitive *PageRank* while cache-friendly *roms* can have larger allocations, unlike DAWG with equal allocations of 1-way (2MB) for each of the 16 cores. BCE-Eq, BCE-1, BCE-2, BCE-3 allocate 2MB, 1.5MB, 1MB, 0.5MB LLC to each copy of *PageRank*, leaving 8MB, 9.5MB, 17MB, 24.5MB LLC for *roms*.

5.7 Storage Overheads

Implementing BCE requires additional structures to support the flexible indexing. Table 3 summarizes the storage overheads for the newly added structures with BCE. The index-computation module in BCE includes the Load-Balancing-Hash requiring 5 RBM hashes and three tables (Num-Clusters-Table, Domain-Base-Table, and Cluster-Location-Table), each with 512 entries. The newly added structures require a storage overhead of 2KB to support 512 clusters. To support 256 clusters, these structures would need an overhead of 1KB, whereas for 1024 clusters they would need 4.4KB.

Table 3: Storage Overheads for BCE Structures

New Structure	Bits/Entry	NumEntries	Storage
Random-Binary-Matrix Hash (RBM)	24×9	5	135B
Num-Clusters-Table (NCT)	9	512	576B
Domain-Base-Table (DBT)	10	512	640B
Cluster-Location-Table (CLT).	10	512	640B
Total Storage for New Structures			**2 KB**

BCE also needs the tag-entry of each line to be increased by 9-bits to allow line-address of a cacheline to be correctly regenerated for writebacks. While conventional LLCs generate the line-address by concatenating the tag with 15-bit set-index, BCE only has a 6-bit cluster-offset that it can use for this purpose. Hence, the tag size is increased by 9-bits to allow concatenation of the tag and cluster-offset to regenerate the line-address. This increases LLC-SRAM area by 1.8%.

6. RELATED WORK

We first discuss cache-partitioning solutions for security and performance, and then other cache side-channel defenses.

6.1 Cache-Partitioning for Security

Page-Coloring [20, 34, 43] was an early OS/VMM-based approach for LLC partitioning that guides sensitive program memory to specific pages to ensure the usage of isolated LLC sets. But this requires memory-allocations in the same ratio

as LLC-allocations, and is incompatible with large pages; this can result in high overheads on memory or cache constrained systems. **MI6** [5] extends page-coloring to allow support of large pages with static changes to the set-indexing; however the LLC allocations are still in the same ratio as DRAM allocations causing under-utilization of LLC or DRAM; additionally, this design only supports up to 64 domains. In contrast, BCE allows independent LLC and DRAM allocations providing the flexibility needed by real-world applications such as graph workloads, and is scalable to hundreds of domains.

DAWG [21] is the state-of-the-art way-partitioning defense providing isolation from cross-domain hits and evictions. Other prior solutions such as PLCache [49], and CATalyst [23] prevent cross-domain evictions and not hits, and thus are vulnerable to hit-based replacement policy attacks. **NoMo cache** [14] and **SecDCP** [48] allow increasing way-allocations via a security-performance trade-off (NoMo) or by allowing one-way information leakage (SecDCP). All such way-partitioning-based solutions only support as many domains/partitions as the cache associativity (typically limited to 16-32 for LLCs). Our work provides equivalent security as DAWG, the state-of-the-art, while supporting up to 512 flexible partitions and with better performance.

Jumanji [41] proposes LLC partitioning at the granularity of LLC-Banks and allocating different LLC-banks to different VMs to prevent cross-VM cache side-channels. However, it does not provide any security between distrusting processes within a VM, as it uses non-secure utility-based way-partitioning of the LLC between processes of a VM which is vulnerable to cross-process Flush+Reload [55] and cache-occupancy [44, 45] attacks. Moreover, it cannot support LLC isolation between enclaves of a single process as it caches the LLC-bank mappings in a per-core "VTB" cache to enable fast indexing, which is vulnerable to new conflict-based attacks in this setting. In comparison, BCE is *more secure*, as it can isolate the cache state between any two trust-domains: enclaves within a process, processes within a VM or VMs themselves. Moreover, BCE's 2-level indexing is carefully designed to be secure (constant-time) yet fast, unlike Jumanji's LLC-bank mappings that need to be cached leaving them vulnerable. Lastly, BCE can support hundreds of trust-domains and is *more scalable* than Jumanji, which only supports few tens of trust-domains (as many as LLC-Banks) and requires flushing LLC-Banks on domain-switches to support more trust-domains causing high slowdown.

SHARP [53] modifies the replacement policy to prefer same-domain evictions on cache installs which do not leak information unlike cross-domain evictions. However, cross-domain evictions are still needed and performed when same-domain lines are unavailable in an indexed set, allowing conflict-based attacks to continue. Additionally, SHARP's restrictions on flush instructions prevent Flush+Reload attacks, but not Thrash+Reload attacks [40] (without flushes). BCE's principled cache-isolation mitigates all such attacks.

6.2 Cache-Partitioning for Performance

Utility-Based Cache Partitioning (UCP) [33] monitors the cache utility for each application and decides the number of ways to dedicate to each application. While this may be beneficial for performance, it leaves the cache vulnerable to occupancy-based attacks. Furthermore, the UCP scheme suffers from the scalability issues of way-partitioning. **K-Part** [15] and **PriSM** [26] overcome the granularity restrictions of way-partitioning by allowing multiple applications to co-reside in a single way; however, these schemes are insecure as applications can still evict each other's lines. Prior studies [22] [56] have also used **Dynamic Page-Coloring** to dynamically partition the cache space. However, these schemes have the same short-coming as Page-Coloring in that memory and cache allocations are coupled. **Jigsaw** [3] uses page-mapping algorithms to share the capacity and reduce the latency of a banked cache. This design does not take security into account (the allocation and placement decisions based on utility monitoring are vulnerable to occupancy-based attacks). Finally, all cache partitioning schemes described in this sub-section are vulnerable to shared memory attacks.

6.3 Alternative Cache Side-Channel Defenses

Defenses such as **CEASER** [31], **CEASER-S** [32], **RP-Cache** [49], and **NewCache** [50] adopt randomization of cache locations for defending against conflict-based side-channel attacks. **ScatterCache** [51] and **MIRAGE** [38] use randomization along with duplication of lines across domains to also make shared-memory based attacks harder. However, recent works [4,30] have shown some of these randomization-based defenses vulnerable to newer conflict-based attacks and none of these guard against cache-occupancy based attacks.

HybCache [12] prevents conflict-based attacks by providing applications fully-associative cache regions and prevents shared-memory-attacks by duplicating shared lines across applications. However, it allows dynamic sharing of cache-space between multiple processes and is thus vulnerable to cache-occupancy attacks [45]. Also, authors of HybCache note that *"applying it to LLC or larger caches can be expensive"* [12]: a fully-associative lookup over thousands of LLC lines can be slower than a DRAM access and is impractical.

Attack-detection techniques proposing profiling attacks using performance counters [11, 29] or dedicated hardware [10] suffer from either false positives or false negatives.

In contrast, BCE provides secure isolation between domains and security against all three classes of cache-attacks (conflict, shared-memory, and occupancy-based attacks).

7. CONCLUSION

Cache-partitioning is a principled defense against cache side-channel attacks. However, existing partitioning schemes relying on page-coloring and way-partitioning face scalability and flexibility challenges that limit their adoption. We enable *Bespoke Cache Enclaves* (BCE), which provides fine-grained isolated LLC partitions to programs that are flexibly managed independent of memory allocations, and which scalably supports several hundred security-domains. BCE incurs negligible storage overheads ($<$2%) and slowdown ($<$1%).

8. ACKNOWLEDGMENTS

We thank the reviewers for their feedback. This work was partially supported by SRC/DARPA Center for Research on Intelligent Storage and Processing-in-memory (CRISP). Gururaj Saileshwar was supported in part by a Georgia Tech IISP Cybersecurity PhD Fellowship.

REFERENCES

[1] A. Arcangeli, I. Eidus, and C. Wright, "Increasing memory density by using KSM," in *Proceedings of the Linux Symposium*, 2009.

[2] S. Beamer, K. Asanović, and D. Patterson, "The gap benchmark suite," *arXiv preprint arXiv:1508.03619*, 2015.

[3] N. Beckmann and D. Sanchez, "Jigsaw: Scalable software-defined caches," in *Proceedings of the 22nd International Conference on Parallel Architectures and Compilation Techniques*, 2013, pp. 213–224.

[4] T. Bourgeat, J. Drean, Y. Yang, L. Tsai, J. Emer, and M. Yan, "CaSA: End-to-end Quantitative Security Analysis of Randomly Mapped Caches," in *53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*, 2020, pp. 1110–1123.

[5] T. Bourgeat, I. Lebedev, A. Wright, S. Zhang, and S. Devadas, "MI6: Secure enclaves in a speculative out-of-order processor," in *Proceedings of the 52nd Annual IEEE/ACM International Symposium on Microarchitecture*, 2019, pp. 42–56.

[6] S. Briongos, P. Malagón, J. M. Moya, and T. Eisenbarth, "Reload+ refresh: Abusing cache replacement policies to perform stealthy cache attacks," in *USENIX Security*, 2020.

[7] J. Bucek, K.-D. Lange, and J. v. Kistowski, "SPEC CPU2017: Next-generation compute benchmark," in *Companion of the 2018 ACM/SPEC International Conference on Performance Engineering*, 2018, pp. 41–42.

[8] J. T. Butler and T. Sasao, "Fast hardware computation of x mod z," in *2011 IEEE International Symposium on Parallel and Distributed Processing Workshops and Phd Forum*. IEEE, 2011, pp. 294–297.

[9] J. L. Carter and M. N. Wegman, "Universal classes of hash functions," *Journal of computer and system sciences*, vol. 18, no. 2, pp. 143–154, 1979.

[10] J. Chen and G. Venkataramani, "CC-Hunter: Uncovering Covert Timing Channels on Shared Processor Hardware," in *2014 47th Annual IEEE/ACM International Symposium on Microarchitecture*, 2014.

[11] M. Chiappetta, E. Savas, and C. Yilmaz, "Real time detection of cache-based side-channel attacks using hardware performance counters," *Appl. Soft Comput.*, 2016.

[12] G. Dessouky, T. Frassetto, and A.-R. Sadeghi, "HybCache: Hybrid Side-Channel-Resilient Caches for Trusted Execution Environments," in *29th USENIX Security Symposium (USENIX Security 20)*, 2020.

[13] C. Disselkoen, D. Kohlbrenner, L. Porter, and D. Tullsen, "Prime+ abort: A timer-free high-precision L3 cache attack using Intel TSX," in *26th USENIX Security Symposium (USENIX Security 17)*, 2017.

[14] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable Caches: Low-complexity Mitigation of Cache Side Channel Attacks," *ACM Trans. Archit. Code Optim.*, vol. 8, no. 4, Jan. 2012.

[15] N. El-Sayed, A. Mukkara, P.-A. Tsai, H. Kasture, X. Ma, and D. Sanchez, "KPart: A hybrid cache partitioning-sharing technique for commodity multicores," in *HPCA*, 2018.

[16] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ Flush: a fast and stealthy cache attack," in *DIMVA*, 2016.

[17] D. Gruss, R. Spreitzer, and S. Mangard, "Cache template attacks: Automating attacks on inclusive last-level caches," in *24th USENIX Security Symposium (USENIX Security 15)*, 2015, pp. 897–912.

[18] G. Irazoqui, T. Eisenbarth, and B. Sunar, "Cross processor cache attacks," in *Proceedings of the 11th ACM on Asia conference on computer and communications security*, 2016.

[19] A. Jaleel, K. B. Theobald, S. C. Steely Jr, and J. Emer, "High performance cache replacement using re-reference interval prediction (RRIP)," *ACM SIGARCH Computer Architecture News*, vol. 38, no. 3, 2010.

[20] T. Kim, M. Peinado, and G. Mainar-Ruiz, "STEALTHMEM: System-level protection against cache-based side channel attacks in the cloud," in *USENIX Security*, 2012.

[21] V. Kiriansky, I. Lebedev, S. Amarasinghe, S. Devadas, and J. Emer, "DAWG: A Defense Against Cache Timing Attacks in Speculative Execution Processors," in *MICRO*, 2018.

[22] J. Lin, Q. Lu, X. Ding, Z. Zhang, X. Zhang, and P. Sadayappan, "Gaining insights into multicore cache partitioning: Bridging the gap between simulation and real systems," in *2008 IEEE 14th International Symposium on High Performance Computer Architecture*. IEEE, 2008, pp. 367–378.

[23] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. B. Lee, "CATalyst: Defeating last-level cache side channel attacks in cloud computing," in *HPCA 2016*, 2016.

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

[25] C.-K. Luk, R. Cohn, R. Muth, H. Patil, A. Klauser, G. Lowney, S. Wallace, V. J. Reddi, and K. Hazelwood, "Pin: Building customized program analysis tools with dynamic instrumentation," *ACM SIGPLAN Notices*, vol. 40, no. 6, pp. 190–200, 2005.

[26] R. Manikantan, K. Rajan, and R. Govindarajan, "Probabilistic Shared Cache Management (PriSM)," in *Proceedings of the 39th Annual International Symposium on Computer Architecture*, ser. ISCA '12, 2012, p. 428–439.

[27] D. A. Osvik, A. Shamri, and E. Tromer, "Cache Attacks and Countermeasures: The Case of AES," in *Proceedings of the 2006 The Cryptographers' Track at the RSA Conference on Topics in Cryptology*, ser. CT-RSA'06, 2006.

[28] R. Paccagnella, L. Luo, and C. W. Fletcher, "Lord of the Ring (s): Side Channel Attacks on the CPU On-Chip Ring Interconnect Are Practical," *arXiv preprint arXiv:2103.03443*, 2021.

[29] M. Payer, "HexPADS: A Platform to Detect "Stealth" Attacks," in *Engineering Secure Software and Systems*. Springer International Publishing, 2016.

[30] A. Purnal, L. Giner, D. Gruss, and I. Verbauwhede, "Systematic analysis of randomization-based protected cache architectures," in *42th IEEE Symposium on Security and Privacy*, vol. 5, 2020, p. 2021.

[31] M. K. Qureshi, "CEASER: Mitigating Conflict-Based Cache Attacks via Dynamically Encrypted Address," in *Proceedings of the 51st Annual IEEE/ACM International Symposium on Microarchitecture*, 2018.

[32] M. K. Qureshi, "New attacks and defense for encrypted-address cache," in *Proceedings of the 46th International Symposium on Computer Architecture (ISCA)*, 2019.

[33] M. K. Qureshi and Y. N. Patt, "Utility-based cache partitioning: A low-overhead, high-performance, runtime mechanism to partition shared caches," in *39th Annual IEEE/ACM International Symposium on Microarchitecture (MICRO'06)*. IEEE, 2006, pp. 423–432.

[34] H. Raj, R. Nathuji, A. Singh, and P. England, "Resource management for isolation enhanced cloud services," in *Proceedings of the 2009 ACM Workshop on Cloud Computing Security*, 2009, pp. 77–84.

[35] C. Reis, A. Moshchuk, and N. Oskov, "Site Isolation: Process Separation for Web Sites within the Browser," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 1661–1678.

[36] T. Ristenpart, E. Tromer, H. Shacham, and S. Savage, "Hey, you, get off of my cloud: exploring information leakage in third-party compute clouds," in *Proceedings of the 16th ACM Conference on Computer and Communications Security*, 2009, pp. 199–212.

[37] G. Saileshwar, C. W. Fletcher, and M. Qureshi, "Streamline: a fast, flushless cache covert-channel attack by enabling asynchronous collusion," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 1077–1090.

[38] G. Saileshwar and M. Qureshi, "MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[39] M. Schwarz, M. Lipp, and D. Gruss, "JavaScript Zero: Real JavaScript and Zero Side-Channel Attacks," in *NDSS*, 2018.

[40] M. Schwarz, M. Schwarzl, M. Lipp, J. Masters, and D. Gruss, "Netspectre: Read arbitrary memory over network," in *ESORICS*, 2019.

[41] B. C. Schwedock and N. Beckmann, "Jumanji: The Case for Dynamic NUCA in the Datacenter," in *2020 53rd Annual IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2020, pp. 665–680.

[42] T. Sherwood, E. Perelman, G. Hamerly, and B. Calder, "Automatically

characterizing large scale program behavior," *ACM SIGPLAN Notices*, vol. 37, no. 10, pp. 45–57, 2002.

[43] J. Shi, X. Song, H. Chen, and B. Zang, "Limiting cache-based side-channel in multi-tenant cloud using dynamic page coloring," in *2011 IEEE/IFIP 41st International Conference on Dependable Systems and Networks Workshops (DSN-W)*, 2011, pp. 194–199.

[44] A. Shusterman, A. Agarwal, S. O'Connell, D. Genkin, Y. Oren, and Y. Yarom, "Prime+ Probe 1, JavaScript 0: Overcoming Browser-based Side-Channel Defenses," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[45] A. Shusterman, L. Kang, Y. Haskal, Y. Meltser, P. Mittal, Y. Oren, and Y. Yarom, "Robust website fingerprinting through the cache occupancy channel," in *28th USENIX Security Symposium (USENIX Security 19)*, 2019, pp. 639–656.

[46] W. Song, B. Li, Z. Xue, Z. Li, W. Wang, and P. Liu, "Randomized Last-Level Caches Are Still Vulnerable to Cache Side-Channel Attacks! But We Can Fix It," in *IEEE Symposium on Security and Privacy (SP) 2021*, 2021.

[47] J. Wan, Y. Bi, Z. Zhou, and Z. Li, "Volcano: Stateless Cache Side-channel Attack by Exploiting Mesh Interconnect," *arXiv preprint arXiv:2103.04533*, 2021.

[48] Y. Wang, A. Ferraiuolo, D. Zhang, A. C. Myers, and G. E. Suh, "SecDCP: secure dynamic cache partitioning for efficient timing channel protection," in *Design Automation Conference (DAC)*, 2016.

[49] Z. Wang and R. B. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proceedings of the 34th Annual International Symposium on Computer Architecture*, 2007, pp. 494–505.

[50] Z. Wang and R. B. Lee, "A Novel Cache Architecture with Enhanced Performance and Security," in *International Symposium on Microarchitecture (MICRO)*, 2008.

[51] M. Werner, T. Unterluggauer, L. Giner, M. Schwarz, D. Gruss, and S. Mangard, "ScatterCache: Thwarting Cache Attacks via Cache Set Randomization," in *USENIX Security*, 2019.

[52] W. Xiong and J. Szefer, "Leaking Information Through Cache LRU States," in *HPCA*, 2020.

[53] M. Yan, B. Gopireddy, T. Shull, and J. Torrellas, "Secure hierarchy-aware cache replacement policy (SHARP): Defending against cache-based side channel attacks," in *ISCA*, 2017.

[54] F. Yao, M. Doroslovacki, and G. Venkataramani, "Are Coherence Protocol States Vulnerable to Information Leakage?" in *HPCA*, 2018.

[55] Y. Yarom and K. Falkner, "FLUSH+ RELOAD: A High Resolution, Low Noise, L3 Cache Side-Channel Attack," in *USENIX Security*, 2014.

[56] Y. Ye, R. West, Z. Cheng, and Y. Li, "Coloris: A dynamic cache partitioning system using page coloring," in *PACT*, 2014.