RIC: Relaxed Inclusion Caches for Mitigating LLC Side-Channel Attacks

Mehmet Kayaalp
IBM Research
mkayaal@us.ibm.com

Khaled N. Khasawneh
University of California, Riverside
kkhas001@ucr.edu

Hodjat Asghari Esfeden
University of California, Riverside
hasgh001@ucr.edu

Jesse Elwell
Vencore Labs
jelwell@vencorelabs.com

Nael Abu-Ghazaleh
University of California, Riverside
naelag@ucr.edu

Dmitry Ponomarev
Binghamton University
dima@cs.binghamton.edu

Aamer Jaleel
Nvidia Corporation
ajaleel@nvidia.com

ABSTRACT

Recently, side-channel attacks on Last Level Caches (LLCs) were demonstrated. The attacks require the ability to evict critical data from the cache hierarchy, making future accesses visible. We propose Relaxed Inclusion Caches (RIC), a low-complexity cache design protecting against LLC side channel attacks. RIC relaxes inclusion when it is not needed, preventing the attacker from replacing the victim's data from the local core caches thus protecting critical data from leakage. RIC *improves* performance (by about 10%) and retains snoop filtering capabilities of inclusive cache hierarchies, while requiring only minimal changes to the cache.

1. INTRODUCTION

Side channel attacks represent a dangerous vulnerability that exploits weaknesses in the implementation of otherwise secure systems and algorithms. A particularly dangerous form of side channel attacks is the one targeting shared microprocessor resources on multi-core processors. Such attacks can be launched remotely without special privileges, removing the need for physical proximity present for many analog channels, and significantly lowering the barrier for launching side channel attacks. For example, such attacks can be used on cloud computing systems to allow a malicious application to exfiltrate sensitive data from other co-located applications [1, 2, 3, 4].

Initially, cache side-channel attacks were performed through L1 caches. To successfully perform an attack, the adversary needs to achieve co-residency with the victim process on the same core, which can be challenging [1, 5]. Moreover, a number of defenses have been proposed to mitigate the L1-based attacks [6, 7]. Consequently, the focus of recent attacks shifted from first-level caches to shared Last-Level Caches (LLC). Successful and fast secret key reconstruction from the LLC side channel has been shown under different assumptions about the attacker's capabilities [8, 2, 3, 9].

In light of serious threat posed by cache based side-channel attacks, it is important to protect cache hierarchies from these attacks, but without over-designing for security. Defenses developed for the L1 cache do not translate effectively to the new attacks on LLCs because of differences in the size, sharing, and complexity between these two levels of the cache. Recently, Liu et al. [10] proposed the first known solution for protecting against side-channel attacks on the LLC. Their design uses a combination of partitioning (via page coloring) and locking, which are individually known for protection against L1 attacks [11, 12], to allow these techniques to scale to the large number of threads that share the LLC. The solution requires support from the OS and the programming language to mark and allocate secure pages respectively. Locking and partitioning reduces the dynamic availability of cache space, which can cause performance degradation under certain workload combinations.

In this paper, we introduce *Relaxed Inclusion Caches* (RICs), a new low-complexity mitigation for LLC side-channel attacks that simultaneously improves performance relative to inclusive caches. We observe that PRIME+PROBE, the most general of side-channel attacks, on the LLC is possible because of the *inclusive nature* of modern cache hierarchies. Inclusion simplifies cache coherence because the LLC can serve as a snoop filter. However, inclusive hierarchies make systems vulnerable to the LLC attacks. Specifically, if an attacker can evict the victim's critical data from the shared LLC, the inclusive property guarantees that this data will also be evicted, using back-invalidations, from the local core caches of the core where the victim process executes. As

DAC 2017 Austin, Texas USA
© 2017 ACM. ISBN 123-4567-24-567/08/06...$15.00
DOI: 10.475/123_4

a result of the data being evicted from the private core caches, the next access to the critical data by the victim will miss into the private caches and thus will be visible to the attacker through the LLC. RIC avoids the eviction of critical data from the local caches, thus defeating the attacks. RIC also improves performance relative to inclusive caches, because it reduces data replication and uses the capacity of the cache hierarchy more efficiently. Compared to non-inclusive caches, RIC also retains snoop filtering thus simplifying cache coherence.

2. LLC SIDE CHANNELS: THREAT MODEL

To understand the attack principles, consider ciphers, such as AES, Blowfish, or Twofish where, for performance reasons, most implementations use precomputed lookup tables. The indices to these tables are used to perform the cryptographic functionality and are partially derived from the secret key. Therefore, by detecting the cache sets accessed by the cipher (through the LLC side channel), the attacker learns which entry of the table was accessed, thus obtaining information about the secret key.

There are two general approaches for cache side channel attacks, including the ones on LLC: FLUSH+RELOAD [8] and PRIME+PROBE [3, 9]. FLUSH+RELOAD LLC attacks [8] rely on cryptographic lookup tables, which are not a secret by themselves, being shared between the victim process and the attacker process. As a result, the attacker can use the *clflush* x86 instruction to flush specific cache lines that contain cryptographic tables from all cache levels, including the LLC. When the data is accessed again, the attacker can tell whether the victim accessed the same sets (cache hit) or not (cache miss). FLUSH+RELOAD can be defeated by disallowing sharing for critical data.

PRIME+PROBE is the most general cache side channel attack because it does not require sharing of the critical data between the attacker and the victim [3, 2, 9]. The attacker fills the cache with its own data during the prime stage. It later probes the cache with the same data, while timing the access. If it detects a cache miss, this indicates that the victim accessed the corresponding cache set, exposing the cache sets the victim is accessing. Recent commercial processors implement inclusive cache hierarchies to simplify cache coherence [13, 14]. With inclusive caches, the attacker can evict the victim's data from the LLC, and the inclusion property guarantees the data eviction from the local core cache of the victim, causing victim's next access to reach the LLC and exposing the access to the attacker.

Liu et al. [3] demonstrated an LLC PRIME+PROBE attack against ElGammal cipher. ElGammal cipher computes the critical data on-the-fly; the fact that the data is writable has implications on relaxed inclusion which we discuss later. Irazoqui et al. [2] demonstrated a similar attack on AES, albeit one that makes a number of assumptions on the synchronization between the victim and the attacker and assumes that attacker has access to the ciphertext. Kayaalp et al. [9] presented a PRIME+PROBE attack on LLCs that does not require large pages (as do the previous two attacks).

Consistent with these published attacks [8, 3, 2, 9], we assume an attacker and a victim are co-located on the same machine, but not necessarily on the same core. The attacker process has no special privileges.

3. RIC: RELAXED INCLUSION CACHES

LLC attacks rely on the inclusive property of modern cache hierarchies where a cache line that is replaced from the LLC is evicted from the private core caches (L1 and L2). To achieve security, retain snoop filtering, and improve performance in a low-complexity manner, we propose a side channel protection for inclusive caches that relaxes the inclusion property where it is safe to do so.

In particular, we relax inclusion in the following two cases:

- *Read-only data*, which includes the critical data for most ciphers (which is constant) as well as all instructions. It is safe to relax inclusion on read-only data since such data is never modified and therefore does not require cache coherence. Thus, the data can be safely cached in a non-inclusive way while retaining snoop filtering. In particular, when the data is present in the core caches but not in the LLC, there is no need to snoop the core caches since the data cannot be modified.

- *Thread private data*: We recognize that a few important ciphers, such as ElGamal [15], also compute and write the critical data before using it [3]. For ElGamal, in the sliding window implementation of modular exponentiation, the multipliers are critical data and they are computed on-the-fly, therefore requiring read and write access. Such ciphers would not be protected from by relaxing inclusion on read-only data. We observe that it is also safe to relax inclusion on data that is not shared such as thread-private data. For such data, even if it is updated, we know that it is only accessed by a single thread and coherence is not needed. Care must be taken if a thread is migrated from one core to another. In this case, any modified data must be flushed, or written through to the LLC.

Relaxing inclusion on read-only data provides side channel protection for most ciphers. The critical data is typically maintained in the form of precomputed tables that are only read during the execution of cryptographic codes, and never modified. Moreover, attacks on the instruction cache sets have also been proposed [3, 9]; these sets are also protected by this form of relaxed inclusion since instructions are read-only. Similarly, for ciphers that modify data, if the critical data is private, inclusion may be relaxed on such data also providing protection from PRIME+PROBE LLC side channel attacks.

To implement RIC, the caches are extended with a single bit per cache line to mark relaxed inclusion. Upon replacement, a relaxed inclusion cache line does not generate back-invalidations to enforce inclusion. *Snoop filtering is used as is for relaxed inclusion data because it is either never modified or never shared, correctness is preserved despite the relaxed inclusion property.*

With respect to read-only data, relaxing the inclusion property for all read-only memory pages avoids the need to mark critical data, and also increases the performance gains from this technique. Identifying thread-private data automatically is more difficult in general [16]. For applications that are not multi-threaded, the full memory image of the process may be treated as thread-private. However, for multi-threaded applications, protecting writeable

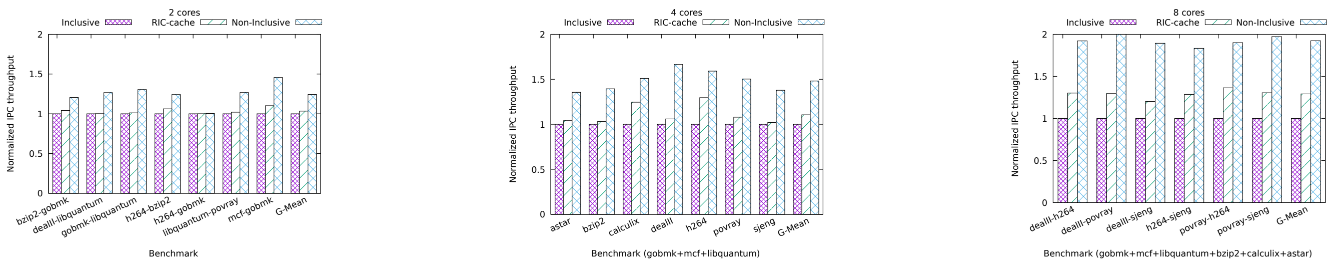


Figure 1: RIC Performance: 2MB LLC

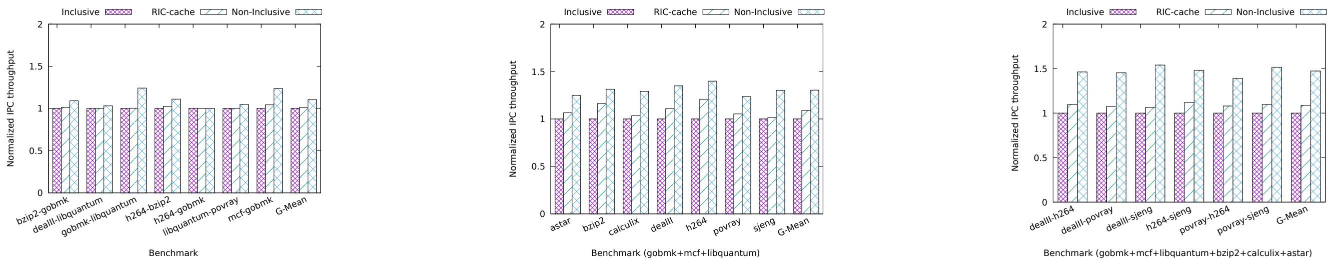


Figure 2: RIC Performance: 4MB LLC

data requires application support to mark the data as private/relaxed inclusion. A limitation of RIC is that it cannot protect writeable critical data that is not private (i.e., that is shared among threads); however, we are not aware of any ciphers that would benefit from such an implementation.

Special considerations have to be given to memory page permission changes. A permission change in the PTE entry of a read-only page can occur for the following reasons: (a) the permissions are modified, for example via `mprotect` system call; (b) the page is torn down, for example via `munmap/exit` system call; (c) the page is swapped out of memory to secondary storage. For cases (b) and (c) above, the data corresponding to these pages cannot be accessed from the cache because the mappings are removed from the page table. For scenario (a) above, in order to avoid using stale data in the local caches (the data with old "read-only" permission), the OS can flush those lines from the cache using the same mechanisms already in place for cases (b) and (c).

Another scenario that requires attention is that of private data that is marked with relaxed inclusion when the thread that is operating on the data migrates after a context switch from one core to another. In this case, if the data has been replaced from the LLC, stale data may be read from memory. Thus, thread migration events must be either avoided or accompanied with a flush of the private caches.

Parameter	**Configuration**
Window Size	8-way issue, 128-entry ROB, 32-entry Issue Queue, 48-entry LSQ
L1 I-Cache	32 KB, 4-way, 64B line, 1 cycle hit
L1 D-Cache	32 KB, 4-way, 64B line, 1 cycle hit
L2 Unified Cache	256 KB, 8-way, 64B line, 10 cycle hit
L3 Unified Cache	2 MB/512 KB, 16-way, 64B line, 30 cycle hit
Memory latency	300 cycles

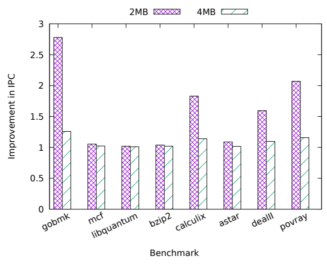
Table 1: Configuration of the simulated processor

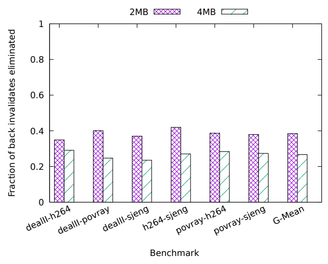
4. PERFORMANCE EVALUATION OF RIC

In this section, we present performance evaluation of RIC. We extend the MSim multicore simulator [17] which implements inclusive caches, to model non-inclusive caches as well as RIC. Unless otherwise stated, the configuration of the simulator is shown in Table 1. In the first study, we paired the SPEC 2006 benchmarks following the methodology described in [18] in terms of the selection of workloads, and evaluated the combined throughput of 2-, 4-, and 8-threaded workloads.

The RIC configuration in the experiments relaxes exclusion only on memory marked by the compiler to be read-only. For SPEC 2006, since the benchmarks are single-threaded, it is possible to treat all memory as private, obtaining using RIC the same performance as non-inclusive caches, but at a much lower complexity, and without any back-invalidate traffic.

Figure 1 and Figure 2 show the performance of RIC compared to inclusive and non-inclusive caches, for systems with a 2MB and 4MB LLC cache respectively. Each figure has 3 graphs corresponding to a 2 core, 4-core, and 8-core system respectively. The benchmark names shown in parenthesis at the bottom of each chart are included with every workload. For example, every 4-core workload contains: gobmk, mcf, libquantum and the fourth benchmark is denoted with the bar's label. As can be seen in the figure, non-inclusive caches noticeably outperform inclusive caches on average, and the difference increases as the ratio of the local core cache size to the LLC size is increased. Of course, this advantage comes at a cost of not having a snoop filter. RIC caches prevent invalidations of read-only data and increase the effective size of the cache, which leads to higher performance compared to inclusive caches. While non-inclusive cache has performance benefits, it is significantly more complex because it does not support snoop filtering. On the other hand, the RIC design is secure, retains snoop filtering advantages and outperforms inclusive caches. The advantage of RIC and non-inclusive caches is higher with a 2MB LLC because the


(a) IPC of individual cores


(b) Reduction in back invalidations

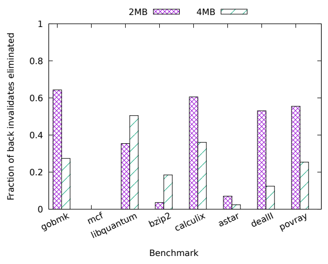

(c) Back invalidate rate

Figure 3: Performance of a Selected Benchmark Mix

cache size is more constrained; the effective increase in the cache size that comes from relaxing inclusion has a larger effect. The advantage also increases with the number of cores due to the higher pressure on the shared cache.

In order to explain the reasons for the performance advantages, we first show the individual IPC (committed Instructions per Cycle) of the cores in one of the 8-core experiments in Figure 3a. First, we observe that some applications have almost no benefit from RIC, while others (e.g., gobmk) benefit significantly, especially when the LLC size is small. Figure 3b shows that the percentage of back-invalidations eliminated by RIC is fairly constant across the benchmarks. However, Figure 3c shows that the rate of back-invalidations varies significantly between the applications, explaining the difference in impact for RIC relative to inclusive caches.

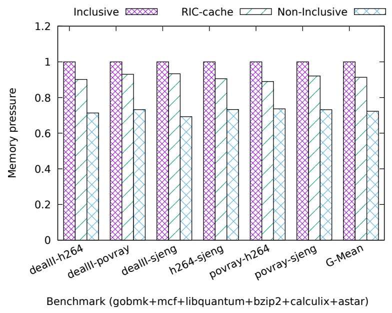


Figure 4: Memory pressure

Finally, we examine the impact of RIC on the memory pressure experienced by the system in Figure 4. Non-inclusive caches reduce the memory pressure in this configuration (8 core, 4MB LLC) by about 30%, while RIC reduces the memory pressure by about 10%, roughly corresponding to the improvement in IPC for this configuration.

5. RIC HARDWARE COMPLEXITY

To evaluate the impact that RIC has on hardware complexity, we modeled it using Cacti [19] version 6.5. The cache that we modeled is based on an L3 cache that is consistent with recent Intel Core i7 series processors, which represents the LLC in these designs. The configuration parameters of this cache are shown in Table 2. The format of each cache line is shown in Figure 5 which depicts the width (in bits) of the various fields of each cache line, including the newly added relaxed inclusion bit that is added to support RIC.

Parameter	**Value**
Total Size	8MB
Line Size	64 bytes
Associativity	16-way
Sets	8192
Tag Bits	29
Physical Address Width	48 bits
Process Technology	32 nm

Table 2: RIC cache model configuration

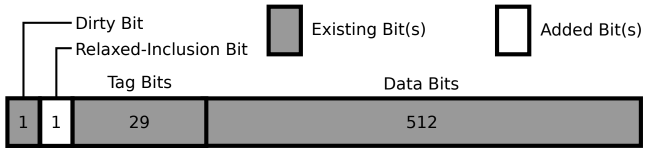


Figure 5: RIC cache line

We conservatively assumed that the only status bit necessary is a ”dirty” bit, although more status bits might be necessary for a specific implementation (i.e. to support cache coherency). Given this assumption, the total size of each cache line is 542-bits, thus our hypothesis was that RIC’s addition of a single read-only bit would have a negligible impact on the complexity of the cache as a whole. Our evaluation of the RIC cache focuses on three key aspects of the design: the cache’s access time, its hardware cost in terms of die area, and its energy consumption. Due to the addition of the read-only bit, the access time of the modeled cache is increased by 0.018%, the total area of the cache is increased by 0.176%, and the total dynamic energy per read access is increased by 0.122%. Moreover, RIC does not affect the primary operation of the cache; the bit is only used to filter out back-invalidations upon replacement of a cache line.

6. SECURITY ANALYSIS OF RIC

We measured the leaked critical information through the side channel to establish the security properties of RIC. We simulate encryption of 15MBs of random data using core AES implementation of OpenSSL 1.0.1j, where the critical data size is 4KB. Table 3 shows the number of evictions of the lookup table entries from the L1 cache for different cache sizes (which would be visible to the attacker). Even with a 12KB 3-way set associative cache, the data is completely protected. Compare this to core cache sizes which are over 256K even for mobile processors.

L1 Size	Number of evictions	Percentage of all critical accesses
4KB 1-way	7,831,093	4.99%
8KB 2-way	163,262	0.10%
12KB 3-way	0	0.00%

Table 3: Evictions of L1 Critical Data under RIC

Consider now that the LLC is protected from the attacks using RIC. However, to provide a complete protection, the possibility of a coordinated attack on multiple cache levels also has to be considered. If an attacker simultaneously targets the LLC and the local core caches, then not all defenses work synergistically with RIC under such a threat model, and the defenses for the local caches have to be chosen carefully.

Specifically, only the techniques that guarantee that the critical data remains in the local caches and cannot be evicted by the attacker's accesses to the local caches themselves would provide security. These include cache line locking [12], static partitioning, or secure dynamic partitioning [11]. On the other hand, techniques that do not have such property, such as randomized victim selection [12, 6] or fuzzy timers [20] do not complement RIC against coordinated attacks. The vulnerability occurs because these defenses allow the critical data to be evicted from the core caches, thereby exposing victim's accesses to the attacker through the LLC.

On the other hand, if the LLC is protected through partitioning or randomization (and not RIC), then any technique for protecting the local core caches will be secure. However, those techniques have substantial performance and complexity implications when applied at the LLC level as discussed in Section 7.

7. RELATED WORK

RIC and Alternative Cache Hierarchies: From a performance perspective, RIC fits in the category of relaxing cache coherence protocols [16, 21]. Most similar to our work, Cuesta et al. [16] propose eliminating the overhead of coherence tracking of private data in Distributed Shared Memory (DSM) architectures. Alisafaee [21] improves on this solution by allowing coherence to be relaxed for data that is private temporarily or shared across a subset of the processors. Both of these approaches do not consider read-only data or coherence relaxation in the context of on-chip/snooping bus protocols.

One simple solution from the security standpoint is to completely move away from the inclusive property to retain critical data in the core caches. However, this approach eliminates the substantial performance and complexity benefits of snoop filtering. One could ask whether solutions in the space between inclusive and non-inclusive caches, which were proposed for performance reasons, are sufficient for security. As a representative of this class of work, we consider TLA caches [18] and NCID caches [22].

TLA caches [18] have been proposed to bridge the performance gap between inclusive and non-inclusive hierarchies, while retaining the snoop filtering capability. TLA achieves this by ensuring that data lines with high temporal locality are not back-invalidated from the local core caches upon the LLC eviction. However, TLA caches are still vulnerable to advanced side channel attacks, because the attacker can carefully control the temporal locality in its access patterns to effectively degenerate TLA caches into traditional inclusive caches.

The NCID design [22] is a non-inclusive cache architecture with inclusive directory. It allows the data in the LLC to be non-inclusive or exclusive, but retains tag inclusion in the LLC directory to support complete snoop filtering. The NCID design is not secure against side-channel attacks; an attacker can oversubscribe the NCID directory by streaming through large amounts of data, causing the secure data to be discarded as in the baseline inclusive cache. Enlarging the directory to a secure configuration significantly increases the overhead. Besides the area overhead, NCID changes the core circuitry of the caches.

Existing L1 Defenses: Most defenses proposed for L1 caches do not apply directly at the LLC level. One simple technique to make caches immune to side channel attacks is static partitioning to create isolation [23]. Unlike the L1 cache which is only shared when a core is hyperthreaded, on a many-core system the number of threads sharing the LLC can exceed the number of ways available in the cache, making cache way-based partitioning impossible. These limitations also apply to designs that provide a mixture of exclusive and shared cache ways [11]. Locking of critical data [12, 24] in the cache prevents it from being replaced by the attacker's prime operations. The solution requires support from the OS, programming language and compiler to mark the critical data, in addition to a bit for each cache line to indicate whether it is locked. Randomization, exemplified by NewCache [6], randomizes the victim selection process on cache replacements, so that the attacker cannot glean useful information from its cache misses. The solution requires an index remapping table, extra bits in the cache to indicate which lines are subject to the random victim selection, and also support from the software layers to mark such critical data [6].

LLC Defenses: Zhou et al. [25] introduced a software based solution for mitigating leakage through the LLC. In particular, the solution has two components: (1) To defeat FLUSH+RELOAD attacks, they use a copy-on-access to duplicate pages shared across VMs when they are being accessed concurrently; and (2) To defeat PRIME+PROBE attacks, they manage the cacheability of pages to limit the number of ways in each set that each VM can occupy. These techniques result in significant slowdown, up to 25% for some workloads, although usually significantly lower.

Most related to our work is CATalyst: a recent defense targeted towards protecting the LLC [10]. CATalyst uses Intel's recent Cache Allocation Technology (CAT) to partition the cache into an unrestricted insecure partition and a secure partition (similar to page coloring). In addition, within the secure partition, the critical data may not be evicted since it is pinned in the cache and therefore cannot be replaced by the attacker. CATalyst requires limited changes to the programming language and run-time, in addition to the architecture, to mark the sensitive data and to differentiate allocation of secure and unrestricted memory pages.

8. CONCLUDING REMARKS

Shared LLCs have become a target of recent software-based side-channel attacks. We proposed the Non-Inclusive Read-Only (RIC) cache as a mechanism to efficiently protect caches against side channel attacks. The key idea of RIC is to relax the inclusion property where cache coherence is not needed (e.g., read-only data). As a result, RIC retains the security-critical data in the local core caches and makes accesses to it invisible to the attacker through the LLC side-channel, thus closing the vulnerability to side channel attacks in principle. The key benefit of RIC is that security is achieved with performance gain, snoop filtering capability, low design complexity and no modifications to the software. As a result, RIC represents an attractive design point in the domain of secure and high performance caches.

9. ACKNOWLEDGEMENT

This material is based on research sponsored by the National Science Foundation grant CNS-1422401.

10. REFERENCES

[1] 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 *16th ACM Conference on Computer and Communications Security (CCS)*, pp. 199–212, 2009.

[2] G. Irazoqui, T. Eisenbarth, and B. Sunar, "S$a: A shared cache attack that works across cores and defies vm sandboxing and its application to AES," in *IEEE Symposium on Security and Privacy (SP)*, 2015.

[3] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, "Last-level cache side-channel attacks are practical," in *IEEE Symposium on Security and Privacy (SP), San Jose, CA, US*, 2015.

[4] D. Evtyushkin, D. Ponomarev, and N. Abu-Ghazaleh, "Jump over ASLR: Attacking branch predictors to bypass ASLR," in *49th International Symposium on Micrarchitecture (MICRO)*, 2016.

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

[6] Z. Wang and R. Lee, "A novel cache architecture with enhanced performance and security," in *Proc. International Symposium on Microarchitecture (MICRO)*, Dec. 2008.

[7] F. Liu and R. Lee, "Random fill cache architecture," in *International Symposium on Microarchitecture, Cambridge, UK*, 2014.

[8] D. Gullasch, E. Bangerter, and S. Krenn, "Cache games – bringing access-based cache attacks on aes to practice," in *Security and Privacy (SP), 2011 IEEE Symposium on*, pp. 490–505, 2011.

[9] M. Kayaalp, N. Abu-Ghazaleh, D. Ponomarev, and A. Jaleel, "A high-resolution side-channel attack on last-level cache," in *Proc. of the ACM Design Automation Conference (DAC)*, 2016.

[10] F. Liu, Q. Ge, Y. Yarom, F. Mckeen, C. Rozas, G. Heiser, and R. Lee, "Catalyst: Defeating last-level cache side channel attacks in cloud computing," in *Proc. 22nd IEEE Symposium on High Performance Computer Architecture (HPCA)*, 2016.

[11] L. Domnitser, A. Jaleel, J. Loew, N. Abu-Ghazaleh, and D. Ponomarev, "Non-monopolizable caches: Low-complexity mitigation of cache side-channel attacks," in *ACM Transactions on Architecture and Code Optimization, Special Issue on High Performance and Embedded Architectures and Compilers*, Jan. 2012.

[12] Z. Wang and R. Lee, "New cache designs for thwarting software cache-based side channel attacks," in *Proc. International Symposium on Computer Architecture (ISCA)*, June 2007.

[13] P. H. et al., "Haswell: The fourth-generation intel core processor," in *IEEE Micro Magazine*, Apr. 2014.

[14] D. Bouvier, B. Cohen, W. Fry, S. Godey, and M. Mantor, "Kabini: An amd accelerated processing unit system on a chip," in *IEEE Micro Magazine*, Apr. 2014.

[15] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," in *Advances in cryptology*, pp. 10–18, Springer, 1984.

[16] B. A. Cuesta, A. Ros, M. E. Gómez, A. Robles, and J. F. Duato, "Increasing the effectiveness of directory caches by deactivating coherence for private memory blocks," in *International Symposium on Computer Architecture*, pp. 93–104, 2011.

[17] "M-sim version 3.0, code and documentation," 2005. Available at: http://www.cs.binghamton.edu/˜msim.

[18] A. Jaleel, E. Borch, M. Bhandaru, S. Steely, and J. Emer, "Achieving non-inclusive cache performance with inclusive caches - temporal locality aware (TLA) cache management policies," in *Proc. International Symposium on Microarchitecture (MICRO)*, 2010.

[19] P. Shivakumar and N. P. Jouppi, "Cacti 3.0: An integrated cache timing, power, and area model," tech. rep., Technical Report 2001/2, Compaq Computer Corporation, 2001.

[20] R. Martin, J. Demme, and S. Sethumadhavan, "Timewarp: Rethinking timekeeping and performance monitoring mechanisms to mitigate side-channel attacks," in *International Symposium on Computer Architecture (ISCA)*, June 2012.

[21] M. Alisafaee, "Spatiotemporal coherence tracking," in *Proceedings of the 2012 45th International Symposium on Microarchitecture (MICRO)*, MICRO-45, pp. 341–350, 2012.

[22] L. Zhao, R. Iyer, S. Makineni, D. Newell, and L. Cheng, "Ncid: A non-inclusive cache, inclusive directory architecture for flexible and efficient cache hierarchies," in *Proc. ACM International Conference on Computing Frontiers*, May 2010.

[23] D.Page, "Partitioned cache architecture as a side-channel defense mechanism," in *Crypt. ePrint Arch.*, 2005.

[24] J. Kong, O. Aclicmez, J. Seifert, and H. Zhou, "Hardware-software integrated approaches to defend against software cache-based side channel attacks," in *Int. Symp. on High Performance Comp. Architecture (HPCA)*, February 2009.

[25] Z. Zhou, M. K. Reiter, and Y. Zhang, "A software approach to defeating side channels in last-level caches," in *Proc. ACM CCS*, 2016.