Capturing and Obscuring Ping-Pong Patterns to Mitigate Continuous Attacks

Kai Wang*, Fengkai Yuan†, Rui Hou†, Zhenzhou Ji*‡ and Dan Meng†
*Department of Computer Science and Technology, Harbin Institute of Technology, Harbin, China
†State Key Laboratory of Information Security, Institute of Information Engineering, Beijing, China

***Abstract*—In this paper, we observed Continuous Attacks are one kind of common side channel attack scenarios, where an adversary frequently probes the same target cache lines in a short time. Continuous Attacks cause target cache lines to go through multiple load-evict processes, exhibiting Ping-Pong Patterns. Identifying and obscuring Ping-Pong Patterns effectively interferes with the attacker's probe and mitigates Continuous Attacks. Based on the observations, this paper proposes Ping-Pong Regulator to identify multiple Ping-Pong Patterns and block them with different strategies (Preload or Lock). The Preload proactively loads target lines into the cache, causing the attacker to mistakenly infer that the victim has accessed these lines; the Lock fixes the attacked lines' directory entries on the last level cache directory until they are evicted out of caches, making an attacker's observation of the locked lines is always the L2 cache miss. The experimental evaluation demonstrates that the Ping-Pong Regulator efficiently identifies and secures attacked lines, induces negligible performance impacts and storage overhead, and does not require any software support.**

***Index Terms*—security, side channel attacks, computer architecture, ping-pong regulator**

I. Introduction

Cache ***S***ide ***C***hannel ***A***ttack***s*** (CSCAs) observe the victim's cache access pattern to extract unauthorized security-critical information [1]–[5]. Among CSCAs, cross-core CSCAs are more dangerous as they not only are easier to be launched, but also possess higher bandwidth and lower noise features. This growing threat reveals the necessity to develop efficient countermeasures to mitigate such attacks [6]–[8].

Continuous attacks are one common kind of scenarios of cross-core CSCAs, where attackers frequently probe the target cache lines in a short timing window. The root cause for the attacks is that the attackers only extract limited information from each iteration, during which they infer whether the target cache lines are accessed by the victim. The information inferred from one iteration is not enough to reconstruct the cache access pattern essentially leaking the secret. For instance, the attack proposed by Liu et al. [1] repeats at least tens of thousands of iterations to recover a 3072-bit EIGamal key. Therefore, *Continuous Attacks* are necessary for obtaining complete information within the transient time window.

Under *Continuous Attacks*, the target cache lines including secrets of victims usually migrate back and forth between different cache (or memory) hierarchies, and we define such kind of memory access traffics as **Ping-Pong Patterns**. During iterations, an attacker first evicts the target lines to observe cache access time differences (hits or misses) determined by whether the victim accesses the lines. *Continuous Attacks* force the target lines to frequently swap in and out of CPU chips due to the attacker's evictions and the victim's accesses, thus exhibiting *Ping-Pong Patterns*.

Ping-Pong Patterns can be either cache-memory or L2-LLC (Last Level Cache). The former are migrations caused by cache evictions (cache-to-memory traffic) and cache lines refill (memory-to-cache traffic) [1], [3]. The latter are migrations caused by L2 evictions (L2-to-LLC traffic due to L2 directory conflicts) and cache lines refill (LLC-to-L2 traffic) [5].

Based on the above observations, we propose Ping-Pong Regulator (PPR) to mitigate *Continuous Attacks* by capturing *Ping-Pong Patterns* and obfuscating the probes launched by attackers. Specifically, this paper makes the following contributions:

1) We observe that the *Ping-Pong Patterns* are significant under *Continuous Attacks* and thus propose exploiting them as an effective indicator to countermeasure the attacks.

2) The micro-architecture design of Ping-Pong Regulator (PPR) is introduced, and PPR is deployed in the LLC to count the re-access number for each cache line. Upon reaching a specified threshold, the cache line is considered to exhibit a *Ping-Pong Pattern*.

3) Once capturing *Ping-Pong Patterns*, PPR launches two defensive actions to obfuscate attackers probes: **Preload** obscures cache-memory patterns by retrieving the attacked line from memory to the cache, causing an attacker to incorrectly infer the victim always accesses the line; **Lock** blocks the L2-LLC pattern by pinning the attacked line's directory entry to the LLC directory[1], inducing an attacker to believe the victim never accesses the line.

4) Our in-depth evaluation shows that PPR can mitigate *Continuous Attacks* with negligible performance impacts and storage overhead.

II. Threat Model and Design Goals

We focus on *Continuous Attacks*, a common cache side channel attack scenario. *Continuous Attacks* contain an attacker and a victim running on different physical cores (i.e., cross-core attacks). Without losing generality, they can be

‡Corresponding Author: Z. Ji (E-mail: jizhenzhou@hit.edu.cn)

[1]Only the directory entry of the attacked line is pinned to the LLC directory, and the data is allowed to serve in L2 without introducing L2 directory conflicts.

978-3-9819263-4-7/DATE20/©2020 EDAA

two virtual machines, processes or threads with or without shared memory. The attacker can use any existing cache side channel attack method (Flush + Reload [4], Prime + Probe [1], etc.) to extract sensitive information in both inclusive and non-inclusive LLC architectures. Besides, we trust underlying operating systems.

The goal of this paper is to develop a feasible hardware approach exploiting *Ping-Pong Patterns* of the attacked cache lines to mitigate *Continuous Attacks*. The proposed solution not only provides strong protection against *Continuous Attacks* but also has (1) general solutions for inclusive and non-inclusive LLC architectures, (2) negligible performance overheads with acceptable hardware cost, (3) no reliance on any OS modification and software support.

III. Ping-Pong Regulator

A. Design Overview

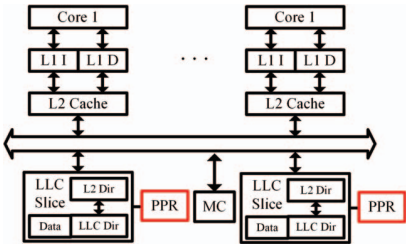


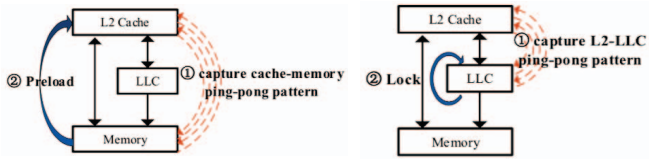
Fig. 1. One typical multi-core processor with non-inclusive caches and Ping-Pong Regulator (PPR).

Continuous Attacks cause attacked cache lines to exhibit *Ping-Pong Patterns*, i.e., they frequently experience the evict-load process at multi-level caches or between caches and memory. This paper proposes Ping-Pong Regulator (PPR) to distinguish the cache lines that exhibit these patterns, and trigger actions (Preload and Lock) on them to interfere with the attacker's probes thus mitigating *Continuous Attacks*.

In modern processors, L1 and L2 caches are generally inclusive, while the LLC is inclusive or non-inclusive. Our research takes the non-inclusive LLC as an example because this architecture exhibits all *Ping-Pong Patterns* (cache-memory and L2-LLC) under *Continuous attacks*. In section V we discuss how the solution works on the inclusive LLC architecture.

Fig. 1 shows a classic non-inclusive LLC architecture. The LLC is partitioned and physically distributed as multiple slices (each core has one slice). Each slice contains LLC data, L2 directory and LLC directory structures. The regulator is pinned to the directory structures in each slice to count the number of re-accesses for each cache line. If the value reaches a specified threshold, the corresponding cache line is considered to exhibit a *Ping-Pong Pattern*. Deploying the regulator on the directory is a good choice because such a position easily takes into account cross-core memory access related traffics and monitors all cache lines.

Once a cache line exhibiting Ping-Pong Patterns is captured, a related action needs to be triggered to interfere with an attacker's probes on the line. Here we design Preload and Lock to cope with *cache-memory* and *L2-LLC Ping-Pong Patterns*, respectively.



(a) Preload (b) Lock

Fig. 2. Defensive actions.

1) **Preload**. As shown in Fig. 2(a), the Preload handles the *cache-memory Ping-Pong Pattern*. This mechanism actively retrieves the attacked lines back to the L2 cache. Since the Preload causes the cache lines to migrate to caches, the attacker can not determine whether the victim has accessed these lines.
2) **Lock**. Fig. 2(b) shows that cache lines exhibiting an *L2-LLC ping-pong pattern* are captured and locked by the regulator. The locked lines' directory entries are fixed in the LLC directory without introducing L2 directory conflicts any more. The attacker's probes on L2 cache misses are unable to obtain any information.

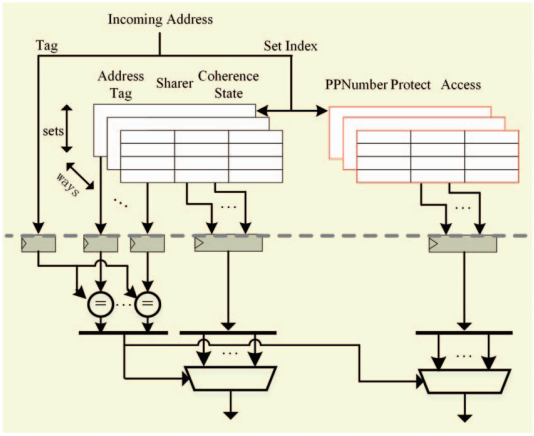
B. Micro-architecture implementation

Ping-Pong Regulator (PPR) is used to capture abnormal traffics and proactively obscure the traffics, preventing attackers from probing target cache lines. The corresponding PPR needs to be added to each slice of the LLC. Each PPR contains three portions, an extension of the L2 directory, an extension of the LLC directory, and a regulator directory.

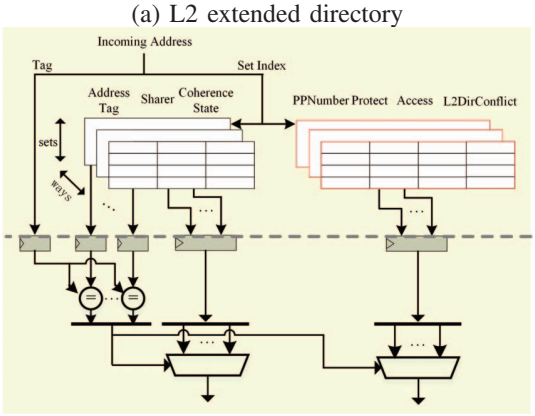
The regulator directory is based on an SRAM implementation. When a cache line is evicted from the cache, the corresponding directory entry along with re-access information is migrated to the regulator directory and is stored in the corresponding set according to the cache line's physical address. If the set suffers a conflict, the replacement policy will choose an entry to discard.

Fig. 3 depicts the micro-architectural implementation of *PPR*. The *Ping-Pong Number (PPNumber)* flag is used to count the number of re-accesses per cache line, thereby recognizing L2-LLC and cache-memory ping-pong mode accesses. The flag exists in the above three directory structures, but it migrates as the cache line moves, and is valid at most in one part at the same time. For example, when a cache line is in the L2 cache, the *PPNumber*, *Protect*, and *Access* flags are in the extension portion of the L2 directory. These flags (plus the *L2DirConflict* flag) are stored in the extension portion of the LLC directory when the cache line is available in the LLC. When a cache line is written back from the cache to memory, the *PPNumber* flag and the line's address tag are stored in the regulator directory. The *PPNumber* is updated and migrated according to the following rules:

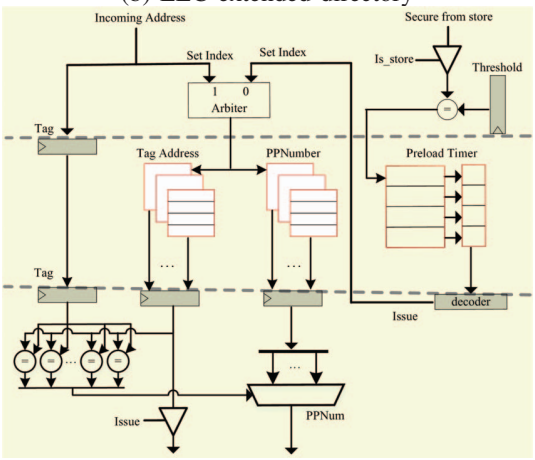
1) When a request hits the L1 or L2 cache, the corresponding L2 directory entry does not migrate and the extension flags do not change.



(a) L2 extended directory



(b) LLC extended directory



(c) Regulator directory

Fig. 3. PPR micro-architecture.

2) When a request has an L2 cache miss and hits the LLC, the corresponding LLC directory entry will migrate to the L2 directory. The *PPNumber* and *L2DirConflict* flags of the entry will be checked out, and the (1) is performed:

$$PPNumber = PPNumber + L2DirConflict \quad (1)$$

The *L2DirConflict* flag indicates whether a cache line migrates to the LLC due to L2 directory conflicts, and if so, the value is 1. Besides, L2 cache replacements also make the cache line move from the L2 cache to the LLC, but the *L2DirConflict* is 0. The L2-LLC access pattern is caused by L2 directory conflicts and re-accesses [5]. Therefore, only when a cache line with *L2DirConflict* of 1 is accessed, PPR thinks it as an abnormal access, and the *PPNumber* increases.

3) When a request does not hit both L2 and LLC caches, if the regulator directory records the corresponding cache line address, it means that this is a cache-memory re-access, and the *PPNumber* value needs to increase by 1. Then, according to the address that needs to be refilled, a new entry needs to be allocated in the L2 directory, and the updated *PPNumber* value is recorded in the extension portion of the entry, and the *Access* and *Protect* flags are initialized to 0. Conversely, if the regulator directory does not record the address, an L2 directory entry needs to be populated with the *PPNumber* and other extension flags initialized to 0.

4) When a cache line is replaced from the L2 (by *clflush*) or LLC to memory, the *PPNumber* value needs to be updated and migrated to the regulator directory. The update rule (2) is performed:

$$PPNumber = PPNumber - Protect + Access \quad (2)$$

The *Protect* flag value of 1 indicates the cache line is preloaded to cache by an obscuring preload operation. The *Access* flag indicates whether the preloaded cache line was accessed before it left cache hierarchies, if accessed, the flag is set to 1.

As a directory entry migrates among directories, its *PPNumber* gradually increases. The threshold of *PPNumber* we used in the evaluation was 1, 2, and 3. However, we design a 3-bit *PPNumber* flag to support a larger threshold range. If the value reaches the threshold, the corresponding obscuring actions are triggered. The specific action depends on where the entry is currently located after the migration.

Preload. If the entry is in the regulator directory, this means that *cache-memory Ping-Pong Pattern* has occurred, and the obfuscation required is Preload. For simplicity, the existing cache preload mechanism can be reused, and PPR sends the preload request to the preload queue of the cache. The choice of preload timing also deserves considerations. On one hand, it must be longer than the memory access latency. Otherwise, preload and write back operations on the same line may compete for memory bandwidth. On the other hand, it should be shorter than an attacker's eviction interval to ensure that this action can interfere with the attacker.

Lock. If the entry is available in the LLC directory, this means that the *L2-cache Ping-Pong Pattern* has occurred, and the obscuring action is Lock. The lock only pins the directory entry of the attacked line to the LLC directory, and the line can still sever in private caches without triggering L2 directory conflicts. The locked line still follows the cache replacement policy and can be evicted out of caches.

IV. Evaluation

A. Methodology

We implemented the PPR mechanism based on gem5 [9], a cycle-accurate full-system simulator. The gem5 is configured

to model a multi-core processor, and the default parameters are shown in Table I. The logically shared non-inclusive LLC cache is physically distributed into as many slices as cores.

TABLE I
SIMULATOR CONFIGURATION.

Baseline Parameters	
Architecture	4 cores at 2.0 GHz, using MOESI
Inclusive L1I/L1D	Private, 32 KB, 8-way, 2 cycles
Inclusive L2	Private, 1 MB/core, 16-way, 18 cycles
Non-inclusive L3	Shared, 1.5 MB/slice, 12-way, 35 cycles
Directory (per slice)	L2 Directory: 1024 sets, 16-way L3 Directory: 2048 sets, 12-way
DRAM	250-cycle latency
Ping-Pong Regulator Parameters	
Regulator directory	(2048 sets, 6-way)/slice
PPNumber threshold	3
Preload time	200 cycles

A typical attack is designed for security analysis. The victim and attacker processes run on different physical cores. The victim executes the Square-and-Multiply algorithm in RSA encryption. The algorithm iterates over secure keys from high to low. For each bit, it executes square-multiply operations if its value is 1. Otherwise, it performs an square operation. Obtaining the victim's access sequence can extract the keys. The attacker implements both Flush + Reload and Prime + Probe methods to extract the victim's access sequence, which leads to *cache-memory* and *L2-LLC Ping-Pong Patterns*, respectively.

The PARSEC and mixes of SPEC CPU 2006 with simlarge and reference input sizes respectively are used for performance evaluation on our proposed PPR. These SPEC benchmarks can be divided into high, middle and low cache miss rates [10]. Based on the above classification, we meticulously pick 2 workloads for each mix to cover all possible combinations. When running mixes on 4 cores, we run 2 copies of each application and assign them to different cores. For each workload, we use a representative slice of one billion instructions and record its execution time in full-system gem5 mode. Our baseline is the case which turns off the PPR mechanism, and all results are normalized to the given baseline.

B. Security Analysis

1) Defending Against the cache-memory Ping-pong Pattern: The first scenario exploits Flush + Reload. The attacker maps the victim's executable file into his virtual address space to build shared memory regions. The granularity of the probe operated by the attacker is 3000 cycles. In each probe phase, the attacker first uses the *clflush* instruction to evict two target lines out of caches. Then the victim continues its execution. After that, the attacker loads them back. According to the access time difference, the attacker can sense the execution path of the victim and thus infer the secret. These sets of operations cause on-chip traffics to exhibit the *cache-memory Ping-Pong Pattern*.

Fig. 4(a) shows the results in the baseline. A highlighted square represents a cache hit when the attacker loads the probe addresses. The pattern of the victim's accesses to execute the square and multiply (Fig. 4 top and bottom lines) leads to information leakage. When a multiply follows a square, the value of the bit in the secret key is 1. Otherwise, the value is 0. Fig 4(b) depicts the results with PPR. The regulator successfully captures the patterns and retrieves the attacked lines back to caches in advance. In this case, every load of the attacker will hit in the cache. Therefore, the attacker can not obtain any useful information.

2) Defending Against the L2-LLC Ping-Pong Pattern: The second scenario uses Prime + Probe. The attacker first constructs two eviction sets and then probes the victim at the granularity of 5000 cycles. In each probe phase, the attacker uses the eviction sets to make L2 directory conflicts, causing two target lines to be evicted from the L2 cache to the LLC. Then, the attacker re-accesses the eviction sets. If, in between, the victim accessed the target lines, cache lines of the eviction sets are evicted from L2 to LLC due to L2 directory conflicts, leading to L2 cache misses of the attacker. These sets of operations cause on-chip traffics to exhibit the *L2-LLC Ping-Pong Pattern*.

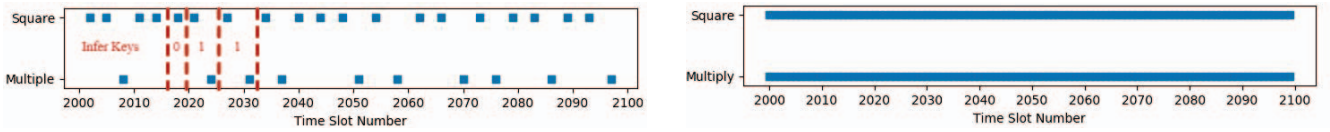
In Fig. 5(a), the highlighted square represents the result that re-accessing the eviction set experiences L2 misses. Apparently, the attacker obtains the victim's square and multiply (Fig. 5 top and bottom lines) access patterns and translates them into keys. When the architecture equipped with PPR, the lock pins the attacked lines' directory entries to the LLC directory. After that, the victim's accesses do not migrate the directory entry to the L2 directory and thus no longer cause L2 directory conflicts. As shown in Fig. 5(b), the L2-LLC pattern is eliminated (keeping blank) and the attacker's re-accesses always hit in the L2 cache (no L2 misses). Hence, there will be no information leakage.

C. Performance Evaluation

Fig. 6(a) shows the performance impacts of PPR mechanism. The results are normalized to the baseline and the lower is better. Across all workloads, *cannel* sees the max improvement (3.1%), while *sjeng_libquantum* faces the max degradation (1.5%). Overall, PPR results in a 0.09% performance improvement on average.

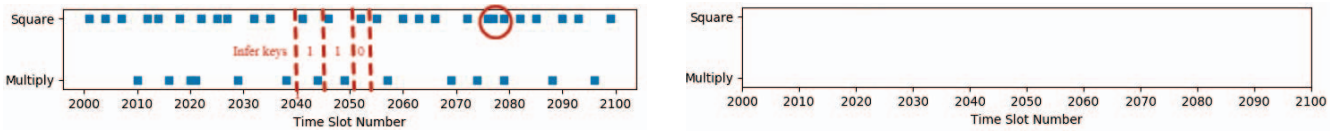
For most workloads, the performance impacts of PPR is negligible. It is because PPR has only a small number of false positives. Fig. 6(b) depicts the number of false positives (false locks and false preloads) in one million instructions per workload. *Cannel* triggers the most false positives (the number is 884). The number of false positives for most workloads is less than 30, which indicates that the extra cache operations introduced by PPR does not consume much bandwidth and has a little impact on performance.

To further understand the performance overhead of *PPR*, Fig. 6(c) illustrates the L3 misses per kilo instruction (MPKI). Intuitively, a smaller value of MPKI means better performance.



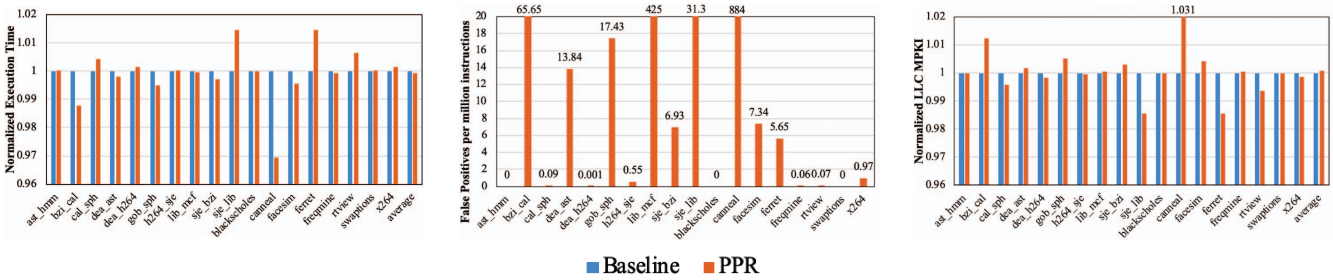
(a) Baseline without PPR (b) Baseline with PPR

Fig. 4. Cache usage patterns of probe addresses extracted by the attacker in Flush + Reload.



(a) Baseline without PPR (b) Baseline with PPR

Fig. 5. Cache usage patterns of probe addresses extracted by the attacker in Prime + Probe.



(a) Normalized Execution Time (b) Number of false positives with PPR (c) Normalized L3 MPKI

Fig. 6. Normalized performance of PPR for SPEC mixes and PARSEC.

Some workloads expose the impacts of PPR more than others. *Bzip2_calculix* is an exception. Its execution time has decreased, but the MPKI increases. Although PPR causes more cache misses, more requests hit the L2 cache due to preload. The latter accounts for more than the former and therefore improves performance.

D. Sensitivity analysis

Size of regulator directory. The capacity of the regulator directory reflects the number of cache lines that can be protected simultaneously. The larger size of the regulator directory brings much better security. However, people might have concerns about its performance overhead due to possible more false positives. We evaluated three different configurations (6 ways/2048 sets, 12 ways/2048 sets, and 24 ways/2048 sets), and found the performance loss are all less than 0.1%. Such results demonstrate the larger regulator directory does not have a significant performance overhead.

Ping-Pong Number threshold. Lower thresholds might introduce more false positives. For example, when the threshold decreases from 3 to 1, the false positives of the mix of *bzip2_calculix* in every million instructions increase from 41 to 110. Although false positives have grown nearly twice, the base is still small and therefore has little impacts on performance. With different thresholds (3, 2, and 1), the average performance loss of all workloads is around 0.1%. However, we still suggest to carefully select the value of the threshold, especially when running memory-intensive programs.

Preload time. We evaluated different preload time (200 cycles, 300 cycles, 400 cycles). As the preload time increases, the average slowdown of all workloads is around 0.1%. Therefore, we recommend that the preload time choice to be as small as possible if the memory bandwidth competition is not severe.

E. Storage and Area Overhead

We compute the storage and area required by an LLC slice of the baseline architecture with and without PPR. The parameters of each structure are in Table I. PPR adds 5 bits (3-bit *PPNumber*, 1-bit Protect and 1-bit *Access*) and 6 bits (3-bit *PPNumber*, 1-bit *Protect*, 1-bit *Access*, 1-bit *L2DirConflict*) to each entry of the L2 and LLC directories, respectively. PPR also requires a regulator directory. Each entry in the regulator directory contains an address tag and a *PPNumber* flag, and each set is assigned a 10-bit preload timer. The area overhead estimation is given by CACTI 7 using 22 nm technology [11]. Overall, PPR requires an additional 78.5KB storage overhead per slice, which is 4.5% more than the baseline architecture. Also, the total area added to each slice is 0.07 mm^2, which is equivalent to 4.01% of the baseline.

V. Discussion

Defeating Continuous Attacks on Inclusive Caches. On the inclusive LLC architecture, the attacked cache lines only exhibit the *cache-memory Ping-Pong Pattern*. To mitigate *Continuous Attacks*, PPR requires even slighter extensions to

count re-access values just for cache-memory patterns and only summons Preload when capturing the patterns.

Defending against defense-aware Adversaries In the regulator directory, the location of an entry is statically determined by the cache line's physical address (the set index bits). A defense-aware adversary can evict the entry by crafting a group of addresses mapping to the same set. If an attacked cache line's entry is evicted from the regulator directory overwhelmingly fast, its critical re-access value is discarded and the defense cannot be activated. In response to this situation, our future work is to introduce a dynamic remapping mechanism that changes the mapping relationship between physical addresses and locations in the regulator directory. Periodically changing keys ensures uncertainty in this mapping relationship, making it impossible for an attacker to form an eviction set within a limited time. The mechanism can be accomplished by adding encryption and decryption hardware modules in the regulator directory, inspired by [12].

VI. Related Work

Many approaches have been proposed to defeat cache side channel attacks and they can be broadly divided into two categories, partition and randomization.

Partition. Partition relies on reserving cache space for sensitive data of the victim, making it harder for the attacker to evict and probe the data. Catalyst [6] uses Cache Allocation Technology to assign security-sensitive data to reserved cache ways. SecDCP [13] dynamically changes the partition size corresponding to the demands of each application.

Randomization. Randomization aims to introduce noise to victims' execution to make it hard for attackers to distinguish cache usage patterns. RPcache [7] utilizes a permutation table to remap a cache line to a new set. CacheGuard [14] defends against attacks by capturing and breaking cache abnormal traffics. However, it incurs unacceptable storage overhead. CEASER [12] uses the encryption module to translate the physical address into an encrypted address and accesses the cache with this translated address. SHARP [15] avoids evicting a cache line in the LLC that is available in L1 or L2 cache.

PPR is different from the above approaches. PPR mitigates *Continuous Attacks* by identifying and obscuring *Ping-Pong Patterns*. This is a general countermeasure on inclusive and non-inclusive LLC architectures. Furthermore, our solution is a pure hardware design and does not need any OS/software modifications.

VII. Conclusion

This paper finds that Continuous Attacks are a common cross-core cache side channel attack scenario and observes that the attacked cache line usually exhibits Ping-Pong Patterns in such attacks. Capturing and obscuring Ping-Pong Patterns can effectively interfere with an attacker's observation of the target cache line, mitigating Continuous Attacks. Based on the above observations, this paper proposes Ping-Pong Regulator. The regulator is deployed to the LLC slice to capture Ping-Pong Patterns and obscure them with two defensive actions (Preload and Lock), making it impossible for attackers to extract the victim's cache usage pattern. Using the full-system simulator gem5, our work presents the implementation of Ping-Pong Regulator. The evaluation indicates that our solution can effectively defend against Continuous Attacks only introducing a negligible performance impact and hardware overhead.

VIII. Acknowledgment

This work was supported by the National Natural Science Foundation of China under grant No.61472100 and the Strategic Priority Research Program of Chinese Academy of Sciences under grant No.XDC02010000. We also thank the reviewers for their valuable comments and suggestions.

References

[1] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. Lee, "Last-level cache side-channel attacks are practical," In 2015 IEEE Symposium on Security and Privacy, pp. 605–622, IEEE, 2015.

[2] Y. Zhang, A. Juels, M. Reiter, and T. Ristenpart, "Cross-vm side channels and their use to extract private keys," In Proceedings of the 2012 ACM conference on Computer and communications security, pp. 305–316, ACM, 2012.

[3] D. Gruss, C. Maurice, K. Wagner, and S. Mangard, "Flush+ flush: a fast and stealthy cache attack," In International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment, pp. 279–299, Springer, 2016.

[4] Y. Yarom and K. Falkner. "Flush+ reload: a high resolution, low noise, l3 cache side-channel attack," In 23rd USENIX Security Symposium, pp. 719–732, 2014.

[5] M. Yan, R. Sprabery, B. Gopireddy, C. Fletcher, R. Campbell, and J. Torrellas, "Attack directories, not caches: Side channel attacks in a non-inclusive world," In 2019 IEEE symposium on security and privacy, IEEE, 2019.

[6] 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 Proceedings of 22nd HPCA, pp. 406–418, IEEE, 2016.

[7] Z. Wang and R. Lee, "New cache designs for thwarting software cache-based side channel attacks," ACM SIGARCH Computer Architecture News, 35(2):494–505, 2007.

[8] D. Meng, R. Hou, G. Shi, B. Tu, A. Yu, Z. Zhu, et al, "Security-first architecture: deploying physically isolated active security processors for safeguarding the future of computing," Cybersecurity, 2018.1.5, 1(2): 1-12.

[9] N. Binkert, B. Beckmann, G. Black, S. Reinhardt, A. Saidi, A. Basu, et al., "The gem5 simulator," ACM SIGARCH Computer Architecture News, 39(2):1–7, 2011.

[10] A. Jaleel, E. Borch, M. Bhandaru, Steely J., and J. Emer, "Achieving noninclusive cache performance with inclusive caches: Temporal locality aware (tla) cache management policies," In Proceedings of 43rd MICRO, pp. 151–162, IEEE Computer Society, 2010.

[11] R. Balasubramonian, A. B. Kahng, N. Muralimanohar, A. Shafiee, and V. Srinivas, "CACTI 7: New tools for interconnect exploration in innovative off-chip memories," ACM Transactions on Architecture and Code Optimization (TACO), 2017.

[12] M. Qureshi, "CEASER: Mitigating conflict-based cache attacks via encrypted-address and remapping," In proceedings of 51st MICRO, pp. 775–787, IEEE, 2018.

[13] Y. Wang, A. Ferraiuolo, D. Zhang, A. Myers, and G. Suh, "Secdcp: secure dynamic cache partitioning for efficient timing channel protection," In Proceedings of the 53rd Annual Design Automation Conference, pp. 74, ACM, 2016. Security symposium, pp. 189–204, 2012.

[14] K. Wang, F. Yuan, R. Hou, J. Lin, Z. Ji, and D. Meng, "Cacheguard: a security-enhanced directory architecture against continuous attacks," In Proceedings of the 16th ACM International Conference on Computing Frontiers, pp. 32–41, ACM, 2019.

[15] M. Yan, B. Gopireddy, T. Shull, and J. Torrellas, "Secure hierarchy-aware cache replacement policy (SHARP): Defending against cache-based side channel attacks," In Proceedings of 44th ISCA, pp. 347–360. IEEE, 2017.