Dogo Rangsang Research Journal
ISSN : 2347-7180

UGC Care Group I Journal
Vol-08 Issue-14 No. 04, April 2021

An Attack with High Resolution on the Last-Level Cache

Mr. PRASANTA KUMAR MISHRA*, Ms. SWARNAKANTI SAMANTARAY
Dept. OF Computer Science and Engineering, NIT , BBSR
prasantkumar@thenalanda.com*,swarnakanti@thenalanda.com

ABSTRACT

Recent side-channel attacks on shared Last Level Caches (LLCs) have been proven, but their usefulness is limited by a variety of system and victim behaviour restrictions. This paper presents a new high-resolution LLC side channel attack that relaxes some of these presumptions and is demonstrated on a real system. We specifically introduce and use new methods to trace victim accesses with high resolution, enabling attacks on cyphers where crucial events have a small cache footprint. We contrast the side-channel quality of our attack with that obtained via FLUSH+ RELOAD attacks, which are substantially more accurate but only function when the victim and the attacker share the sensitive data. We demonstrate that our attack frequently achieves parity. We demonstrate that our attack commonly succeeds in establishing an equal-quality channel, and we moreover verified this by re-creating the victim's cryptographic key.

1. INTRODUCTION

Systems and applications extensively rely on cryptographic operations for security. Although a cipher may be cryptographically secure, it can still leak sensitive information through side channels. Recent side channel attacks target shared resources in conventional microprocessors, including shared caches [1–11]. Attacks on these resources are attractive because they do not require physical access; a spy is simply a co-located process that monitors and accesses shared resources without special privileges. Side-channel attacks can even be used to steal secrets from systems that support secure isolated execution environments [12–15].

Cache-based side-channel attacks exploit the fact that cache accesses performed by a victim process can be monitored by an attacker's spy process that shares the cache with the victim. The cache sets accessed by the victim as it performs encryption correlate with the indices to the cryptographic tables that are used by many ciphers for performance reasons (we attack the Advanced Encryption Standard, AES, in this paper). This information is often sufficient for reconstructing a secret key in a short amount of time [1].

Initially, cache-based attacks were performed through L1 caches. In this case, the attacker needs to achieve co-residency with the victim on the same core, which is not trivial to accomplish in practice [16, 17]. Moreover, a number of defenses are effective in mitigating such attacks [2–4]. For these reasons, the focus of recent attacks shifted from first-level caches to shared Last-Level Caches (LLC) [1,10,11,18,19]. In contrast to L1 caches, the LLC is shared by all cores. Therefore, the co-residency of the victim and the attacker from the standpoint of the LLC is achieved naturally, making the LLC an attractive attack target.

LLC Attack Strategies

FLUSH+RELOAD attacks [1,18,19] assume implementations where the cryptographic lookup tables, which are not a secret by themselves, are located in a memory region that is shared between the victim and the spy process. In such a setting, the spy can use a cache flush instruction (e.g., *clflush* in x86) to flush specific cache lines that contain the cryptographic tables from all cache levels, including the LLC [20]. Later, the attacker can determine if the victim accessed this data by re-accessing this memory line and timing the access. A miss indicates that the data has not been accessed by the victim, and a hit indicates that the access took place, and therefore the data is in the cache again. Although FLUSH+RELOAD was developed in the context of an L1 attack, it can be used to attack the LLC as well; a spy on a different core can observe victim accesses in the LLC because flush removes the data from all levels of the cache.

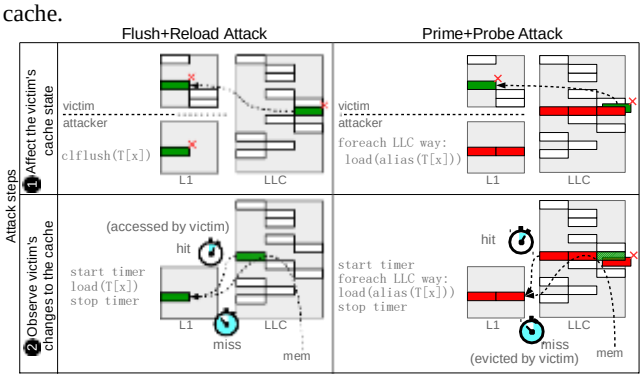

Figure 1: PRIME+PROBE vs FLUSH+RELOAD Attacks

Although extremely powerful, FLUSH+RELOAD does not work when the critical data is not shared. An alternative way to exploit the cache side channel in the absence of cryptographic data sharing is PRIME+PROBE attack [10, 11] (Figure 1). In order to evict victim's data from the cache, the attacker needs to populate all cache

ways, instead of surgically flushing only the cryptographic table cache lines as in FLUSH+RELOAD attack. The attacker times its accesses after it fills the cache; the presence of a cache miss indicates that the victim accessed the corresponding cache line causing the spy's data to be removed.

	S	LP	Sync	KC
FLUSH+RELOAD Attacks:				
Cache Games (S&P'11) [1]	Yes	N/A	No	No
Irazoqui et al. (RAID'14) [19]	Yes	N/A	Yes	Yes
Yarom et al. (USENIX'14) [18]	Yes	N/A	No	No
PRIME+PROBE Attacks:				
Liu et al. (S&P'15) [10]	No	Yes	No	No
S$A (S&P'15) [11]	No	Yes	Yes	Yes
OUR ATTACK	No	No	No	No

S: Attacker and Victim Share Memory
LP: Requires Large Pages
Sync: Assumes Synchronization
KC: Known Ciphertext

Table 1: Comparison of LLC Attacks

Contributions of this paper

Recent work [10] demonstrated the first PRIME+PROBE attack applied to the LLC. This attack was accomplished in the context of the El-Gammal cipher, which looks up large multiplier values in cryptographic tables. These multiplier values span a large number of cache sets, which results in substantial leakage that is detectable by the spy even in the presence of noise.

In this paper, we advance PRIME+PROBE LLC attacks such that they can be applied to ciphers that require high-resolution information of the cache access pattern; these include ciphers such as AES and Blowfish. The high-resolution attack uses the following relaxed conditions: 1) It does not rely on the use of large pages or cryptographic data sharing between the victim and the attacker; 2) It does not require synchronization between the victim and the attacker, and does not rely on the knowledge of the ciphertext; 3) It does not assume knowledge of virtual-to-physical page mappings neither for a victim nor for an attacker; and 4) It does not require any knowledge of the virtual address of victim's critical tables. Table 1 contrasts the attack to recently proposed LLC side channel attacks.

We pursue these improvements using two general techniques: (1) we develop accurate cache reverse-engineering algorithms to derive the detailed behavior of real LLCs. As a by-product, we relax the assumption of large memory pages present in the most recent attacks, and also more effectively probe the cache and more accurately interpret the results of the probes; (2) we use a concurrent attack on the cache sets containing victim's instructions. This allows the attacker to identify the progression of the victim within the cryptographic algorithm, improving the signal and reducing the noise.

In summary, this paper makes the following contributions:

- We propose a new PRIME+PROBE-style high-resolution LLC attack that works with arbitrary page sizes and does not rely on the sharing of cryptographic data between the victim and the attacker. In addition, our attack assumes no knowledge of virtual-to-physical page mappings neither for a victim nor for an attacker. A key component of our attack is a series of mechanisms to discover precise groups of addresses that map into the same LLC set in the presence of physical indexing, index hashing and varying cache associativity across the LLC sets.
- We demonstrate the new attack against AES cipher on an Intel SandyBridge processor with an 8MB shared LLC, and show (using side channel vulnerability metrics and key reconstruction) that our attack is practically as effective as previously proposed FLUSH+RELOAD attack on AES. Note that FLUSH+RELOAD is significantly less general than our attack because it relies on sharing of cryptographic libraries between the victim and the attacker to precisely identify the victim accesses; this sharing can be easily disabled to defeat FLUSH+RELOAD attacks.

2. HIGH-RESOLUTION LLC ATTACK

Figure 2 shows the main components of the attack. The pre-attack phase has three parts. The first part reverse-engineers the LLC hardware to discover its operation, including idiosyncrasies that make LLCs different from traditional textbook designs, but that affect the collection and interpretation of the side channel data. The next part uses this information to discover groups of memory addresses that map to each set of the LLC (we call these *collision groups*). The third part populates the LLC with the collision groups to discover the location of the victim's critical data based on the victim's access frequency and pattern.

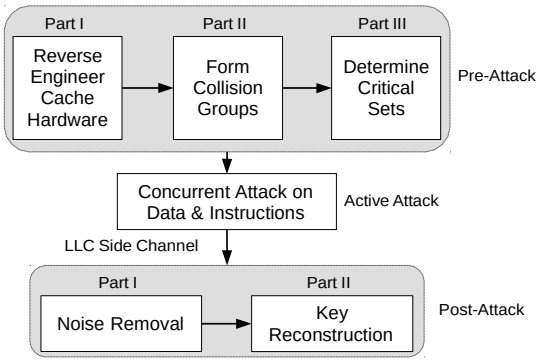

Figure 2: The New Attack Flow and Components

Pre-Attack I: Discovering LLC Details

The first step of the attack is the reverse engineering of the cache to derive some important characteristics that are necessary for the remainder of the attack.

Reverse-Engineering Index Hashing

LLCs have multiple banks that can be accessed concurrently. Index hashing is a technique used to increase the probability that nearby data will be placed in different banks, allowing for concurrent accesses, balanced cache utilization and heat distribution [21]. While the index hash functions are proprietary, they are not designed for security and therefore can be reverse-engineered. Deriving the index hashing is necessary to understand the mapping from memory accesses to cache sets.

We developed a systematic approach to reverse-engineer the hashing mechanism. Although the particular hash function that we discovered is specific to SandyBridge, the proposed method is applicable in general. The step-by-step reverse-engineering process and the resulting hash function are shown in Figure 3. The LLC of the SandyBridge processor used in our experiments has a cache line of 64 bytes, 8192 (2^{13}) sets, and 16 ways.

The *resulting* LLC index hashing function for SandyBridge processor was previously reported by Hund et al. [22] as part of their effort to reverse-engineer the kernel space layout using timing side

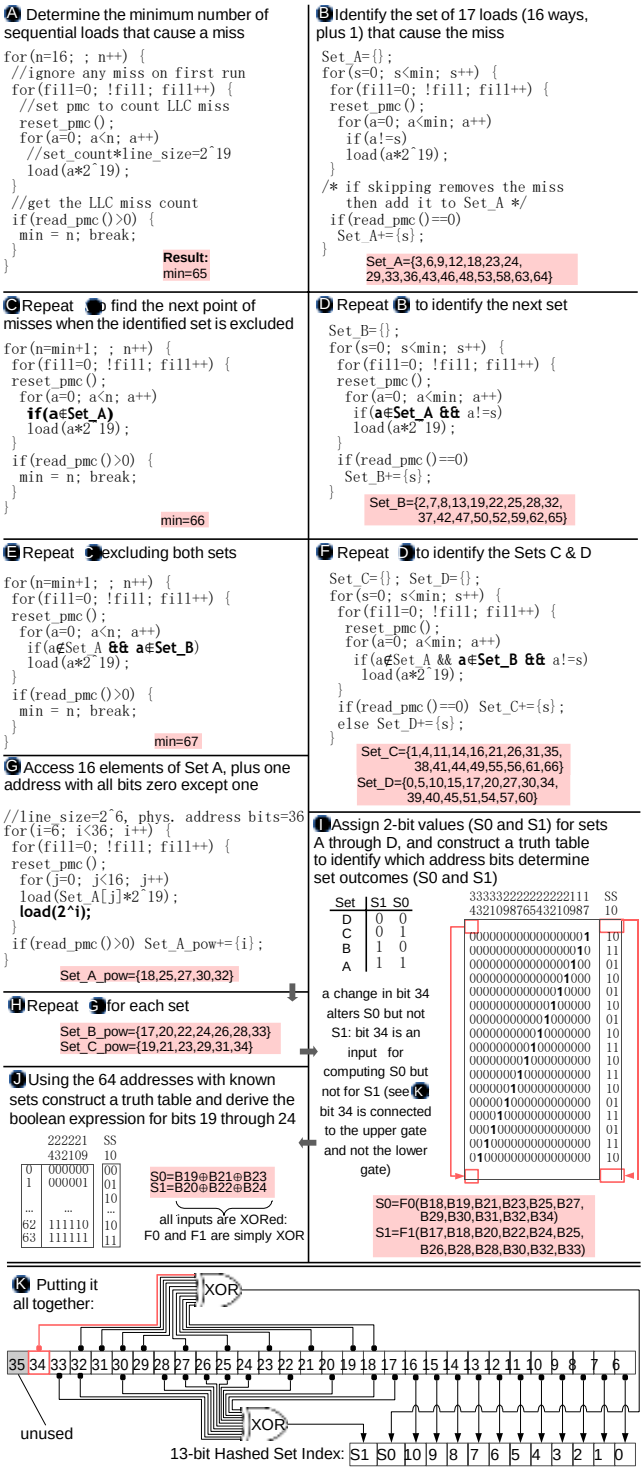

Figure 3: Reverse-Engineering the Index Hashing

channels. However, their reverse engineering procedure consists of hundreds of experiments followed by a manual effort to identify a hashing relationship consistent with the experiments. In contrast, we demonstrate a generalized step-by-step procedure that can be used to automate reverse-engineering of caches. Concurrent with our effort, reverse-engineering of cache indexing has also been demonstrated in two other recent works [23, 24].

Accounting for Cache Banks and Cavity Sets

Using the above approach and timing the cache accesses across each physical core, we also determined the hashing mechanism for individual cache banks. SandyBridge features four cores and four 2MB cache banks, aligned under the cores. The 13-bit hashed set index, X, is used in the following way to generate 2 bits, H_0 and H_1, that correspond to different cache banks.

$$H_0 = X_0 \oplus X_4 \oplus X_6 \oplus X_8 \oplus X_{10} \oplus X_{12}$$
$$H_1 = X_0 \oplus X_1 \oplus X_4 \oplus X_5 \oplus X_6 \oplus X_7 \oplus X_8 \oplus X_9 \oplus X_{10} \oplus X_{11}$$

Figure 4 shows the correspondence of hashed bits to banks with respect to the physical cores. Our experiments also showed that cache sets in the leftmost cache bank ($H_0 = H_1 = 0$) have only 15 ways instead of 16. We call these sets *cavity sets*. Identifying this effect was crucial to improving the precision of the attack as these sets produce false positive accesses if they are primed with collision groups designed for a 16-way cache. Making adjustments for cavity sets during the next step of discovering collision groups is necessary, as we demonstrate below.

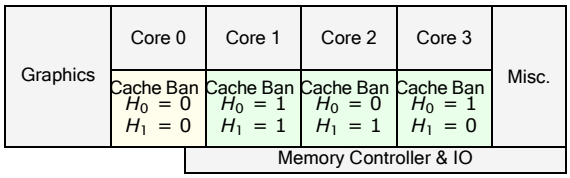

Figure 4: Identifying Cache Banks on SandyBridge Die Map

Pre-Attack II: Collision Groups

We now describe how an attacker can discover collision groups to populate the cache sets. Every virtual page can map into $\frac{S}{P}^{W}$ different physical locations in the LLC, where S is the LLC size, W is the number of LLC ways, and P is the size of a page. For the 16-way 8MB cache used in our experiments, 4KB pages can have 128 different mappings. Note that the two-bit output of the index hash is not affected by the offset within the page; thus, it is sufficient to do the mapping at the page level to learn the collision groups for all the cache lines within the page. Our objective in this part of the attack is thus to find collision groups of W pages, for each of the 128 locations, which are sufficient to cover the full cache.

First, we allocate a large chunk of memory [1]. Second, starting from an empty set Φ, we take one page (ρ) at a time from the pool of allocated pages, add it to Φ and measure $t[n]$—the number of cycles to access the n pages in Φ. If the difference in t caused by the addition of ρ, $d = t[n] - t[n-1]$, exceeds a predetermined number of cycles ($d > T_{DIFF}$), we conclude that the addition of this last page resulted in an LLC miss, because now there are $W+1$ pages in Φ that map to the same LLC set.

Third, similar to Step B in Figure 3, we remove one page at a time from Φ and measure the access time for the rest of the addresses. If the removed page is one of the $W+1$ addresses, the measured access time will be considerably smaller since a conflict miss is eliminated. At the end of this step, we identify $W+1$ pages from a single LLC set, and remove them from Φ.

[1]The physical memory corresponding to this allocated memory space must have sufficient addresses to form conflict groups for each set. Note that we can always allocate additional memory if the full set of 128 conflict groups could not be obtained from the first chunk. However, in all of our experiments, we were able to obtain sufficient coverage of the cache with a single allocation four times the size of the cache.

Fourth, we recheck the identified addresses to make sure that they do indeed form a collision group. Erroneous identification occasionally occurs for reasons such as cache accesses generated by the instructions. In addition, before adding the new page ρ to Φ, we check ρ against every collision group identified so far. If it matches, we discard it without adding it to Φ. This way, duplicate collision groups can be avoided. The process stops after all 128 collision groups have been discovered. The entire process takes 2 minutes and 35 seconds to complete on our Sandy Bridge platform. We repeated the experiment many times and were able to identify collision groups for all cache sets with 100% accuracy.

Pre-Attack III: Identifying Critical Cache Sets

Thusfar, we have been able to determine conflict sets corresponding to all cache lines in the cache. However, we have not established the mapping of these sets to the physical page addresses, or more importantly the cache sets in the cache being used for the sensitive data. In this next step, we identify this correspondence to allow the attacker to focus only on these critical collision groups rather than priming and probing the full LLC, significantly speeding up the attack.

Assuming that the critical pages of AES are aligned at the page boundary, the four 1KB tables fit into a single page. If the attacker accesses the collision groups that map to the same LLC sets as AES critical data, then the victim will experience many LLC misses for the table lookups and considerable slowdown. By measuring the time it takes for the victim to finish its computation, the attacker can determine whether the victim's critical table is in the probed location.

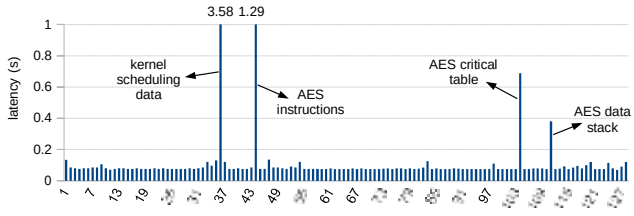

Figure 5: Timing Analysis to Determine Victim's Page Mapping

Figure 5 shows the latency (median of 100 measurements) of encrypting 320 bytes of data, for each of the 128 collision groups discovered in the previous step. When the probed location is used by the victim for the critical data, stack, or instruction memory, timing peaks are observed. The most significant slowdown is achieved when the evicted data is used by the operating system for context switching. From the remaining possibilities, the location of critical tables is determined by examining the memory access patterns. Indeed, while the accesses to the stack are sequential, the critical table lookups exhibit random access patterns, and the instruction accesses indicate a tight loop for every AES round. This pattern allows us to identify the instruction cache sets corresponding to the AES code, which we use to identify the start of encryption blocks, and to reduce noise.

When the tables are not aligned to the page boundary, two smaller peaks are observed for AES critical table in Figure 5. They can be identified easily, since the tables span two consecutive pages in this case, e.g., last 1KB of one page, and first 3KB of another page. Thus, there is no need to determine the virtual address of the victim's critical table: this also defeats defense approaches based on hiding or randomizing the layout of the victim's critical data.

Active Attack Phase: Concurrent Attackon AES Data and Instructions

The active attack phase consists of a concurrent attack on the LLC sets corresponding to critical data and the AES instructions. These sets are accessed using two synchronized threads. In each iteration, the attacker fills the critical LLC sets with appropriate collision groups and times its cache accesses. Some of the observed LLC misses are due to the victim's activity, while other misses correspond to noise. The concurrent attack which probes the sets containing the AES code instructions helps the attacker to identify which data accesses belong to the victim at what phase of the encryption the victim is executing.

Figure 6 shows a sample output of the side channel information obtained from observing the sets containing the AES instructions. The y-axis shows the cache sets that are being accessed that correspond to the location of the AES instructions we identified in Section 2.3. The x-axis shows the iterations of measurements, and black boxes show the cache misses, which indicate accesses to the cache sets by the victim. The instructions accessing the critical tables are between sets 3 and 14. Individual AES rounds are alternating between sets 3 to 6 and sets 7 to 10, and the instructions of the final round are in sets 11 to 14. From the pattern, we identify the execution of AES rounds which are marked with circles.

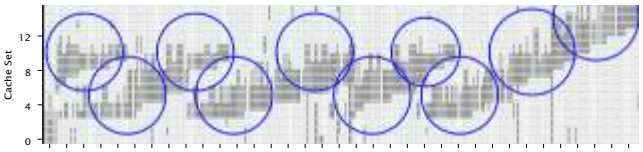

Figure 6: Results of the Attack on the LLC Sets Containing Instructions (AES rounds are marked)

Post-Attack: Noise Removal and Key Reconstruction

Once the data is collected, we apply a number of heuristics to remove noise patterns. Figures 7(a) and 7(c) show the initial results for the two types of attacks. We developed a script to remove recurring noise patterns and to group consecutive loads: Figures 7(b) and 7(d) show the results after removing this noise. Finally, we reconstruct the key using the mechanism described by Gullasch et al. [1]. We modified the reconstruction procedure slightly to account for the differences between the two AES implementations: the authors used one 2KB table each for encryption and decryption, while our implementation uses four 1KB tables. Consequently, the number of observable bits for every critical byte is reduced from five to four in our case. As a result, our attack has to consider 16 times more possibilities when reconstructing each key column candidate, only introducing bounded amount of extra work.

3. ATTACK ANALYSIS

We compare the side channels obtained by *Cache Games* attack [1] and our attack using three metrics. The first metric is the True Positive Rate (TPR)—the number of true critical accesses observed by the attacker, divided by the number of all critical accesses of the victim. The second metric is the False Discovery Rate (FDR)—the number of false critical access observations by the attacker, divided by the number of all measurements of the attacker. The third metric is Cache Side-channel Vulnerability (CSV) that was proposed in [25].

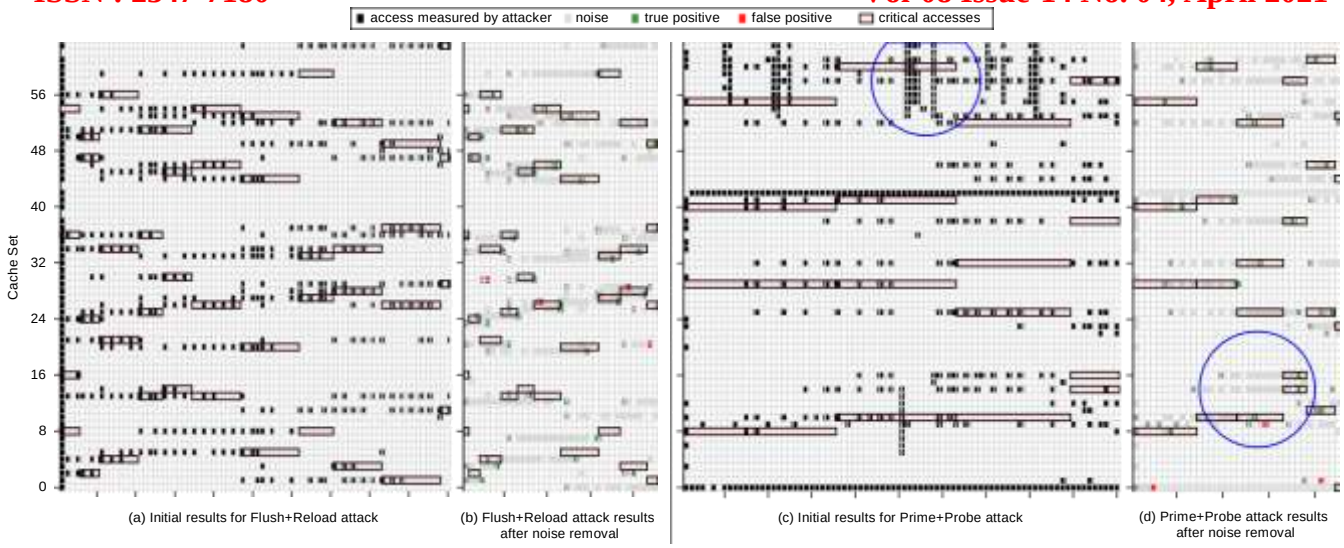

Figure 7: Sample Noise Removal Results for Two Attack Types

CSV is computed using an oracle trace that is perfectly aligned with the attacker trace. Even though the trace of critical accesses for the victim is available, aligning the time series is a challenge. To approximate, we captured minimal timing information for the victim and assumed that any attacker measurement of the same critical access that occurred immediately before or after is a correct measurement. To align the two traces, we placed a "hole" value (-1) to pair with missed and incorrect measurements.

The CSV metric was computed as the Pearson correlation of attacker and victim traces, which are constructed as:

$$M_{i,j} = \begin{cases} 1, & \text{if accessed } set_i \text{ at } time_j \\ 0, & \text{otherwise} \end{cases}$$

We converted the aligned traces of attacker and victim into such matrices A and V, where each step has either one access or zero accesses for hole values. Then we computed CSV as follows:

$$CSV = \frac{\sum_{i,j}(A_{i,j} - \bar{A})(V_{i,j} - \bar{V})}{\sqrt{\sum_{i,j}(A_{i,j} - \bar{A})^2 \sum_{i,j}(V_{i,j} - \bar{V})^2}}$$

While CSV is also a relative metric, *the absolute value of CSV* is a good indicator of the attacker's capability of reconstructing the secret key. The reason is that for the traces used in our analysis, CSV is a direct metric of the percentage of critical accesses that are captured by the attacker, the percentage of critical accesses that are missed, and the percentage that are measured incorrectly. If all measurements are correct, the CSV value is 1. Each critical access that has not been captured by the attacker and each incorrect measurement brings the CSV value down.

Figure 8 compares the two attacks in terms of these three metrics. Recall that *Cache Games* is a FLUSH+RELOAD attack which requires shared critical data enabling precise eviction of victim data and detection of its accesses. Thus, it has the best obtainable channel quality. We executed 100 experiments for both attack types and sorted them for each metric. Our attack has inferior side channel quality in some of the experiments. However, for a substantial number of experiments, the quality of the channels is comparable. To present the likelihood of success in our attack, Figure 8 shows the mean value for each metric for *Cache Games* attack along with the percentage of our attacks that achieve the same mean. According to all considered metrics, at least one in five instances of our attack has the same side channel quality. For TPR, this number is two out of three. These results demonstrate that using the methodology for *Cache Games* attack described in [1], the secret key reconstruction can also be performed for our attack with comparable effort; on average, our post-attack phase including noise removal and key reconstruction took less than 3 minutes to complete.

The main reason for the inferior side-channel quality in some PRIME+PROBE experiments is the collision of pages that contain critical data with other memory pages in the LLC. For some experiments, most of the cache sets are constantly accessed, as they are used by either the scheduler, or the system calls used for the attack on CFS scheduler. However, the attacker can force the critical data to be mapped elsewhere if the memory is put under pressure and the critical page is evicted out of physical memory. In our experiments, we found that the kernel swap daemon `kswapd` can be forced to run by allocating all available memory to evict victim's critical data. Upon reload, the critical page is likely to get a different mapping and therefore a different hashed index in the LLC. Our estimates showed that claiming 16GB memory can force a new hash in under 3 minutes.

4. CONCLUDING REMARKS

This paper contributes a new high-resolution attack on LLCs that extracts leakage at high precision from the cache and does not require the use of large pages. To enable the attack, we first reverse engineer the cache to identify the critical cache sets without relying on large pages. We also identify and account for idiosyncrasies in the cache organization to allow a more accurate attack and interpretation of the results. In addition, we carry out a concurrent attack on the instruction cache to identify when the victim is executing encryption to distinguish relevant memory accesses from noise. We showed that the attack allows to recover a secret key within a few minutes, accounting for both the active probe phase and the post-attack analysis.

5. ACKNOWLEDGEMENT

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

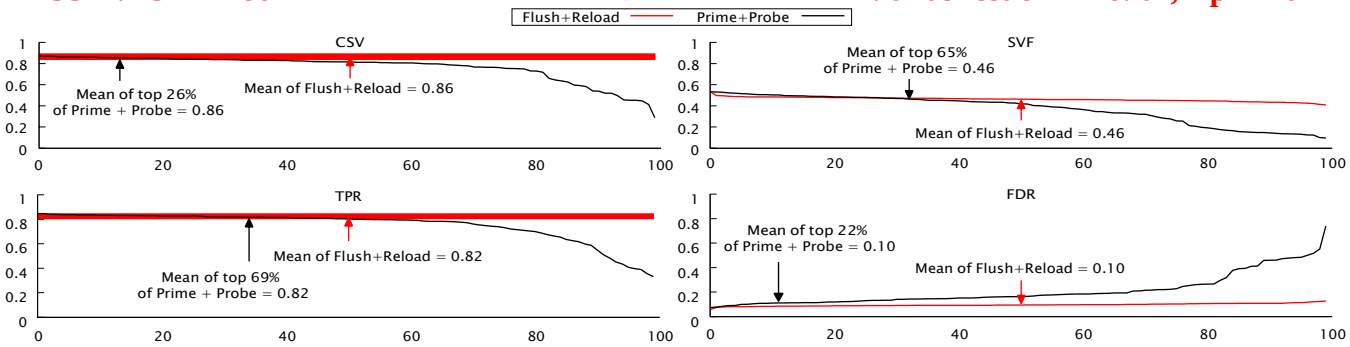

Figure 8: Comparison of the Two Attacks Using Different Metrics for 100 Runs

6. REFERENCES

[1] 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.

[2] 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.

[3] 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.

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

[5] 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)*, 2009.

[6] E. Tromer, A. Shamir, and D. Osvik, "Efficient cache attacks on AES, and countermeasures," in *Journal of Cryptology*, 2009.

[7] C. Percival, "Cache missing for fun and profit," 2005. http://www.daemonology.net/papers/htt.pdf.

[8] D. Bernstein, "Cache-timing attacks on AES," 2005. Available at: http://cr.yp.to/antiforgery/cachetiming-20050414.pdf.

[9] J. Bonneau and I.Mironov, "Cache-collision timing attacks against AES," in *CHES Workshop*, 2006.

[10] 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)*, 2015.

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

[12] F. McKeen, I. Alexandrovich, A. Berenzon, C.Rozas, H. Shafi, V. Shanbhogue, and U. Svagaonkar, "Innovative instructions and software model for isolated execution," in *Wkshp. on Hardware and Architectural Support for Security and Privacy, with ISCA'13*, 2013.

[13] "Intel Software Guard Extensions Programming Reference," 2014. Accessed Mar. 2015 at https://software.intel.com/sites/default/files/managed/48/88/329298-002.pdf.

[14] D. Evtyushkin, J. Elwell, M. Ozsoy, D. Ponomarev, N. A. Ghazaleh, and R. Riley, "Iso-x: A flexible architecture for hardware-managed isolated execution," in *Microarchitecture (MICRO), 2014 47th Annual IEEE/ACM International Symposium on*, pp. 190–202, IEEE, 2014.

[15] Y. Xu, W. Cui, and M. Peinado, "Controlled-channel attacks: Deterministic side channels for untrusted operating systems," 2015.

[16] 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 (CCS)*, pp. 199–212, 2009.

[17] 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.

[18] Y. Yarom and K. Falkner, "Flush+reload: a high resolution, low-noise, l3 cache side-channel attack," in *Proc. USENIX Security Symposium*, Aug. 2014.

[19] G. Irazoqui, M. Inci, T. Eisenbarth, and B. Sunar, "Wait a minute! a fast, cross-vm attack on aes," in *Research in Attacks, Intrusions and Defenses (RAID)*, pp. 299–319, 2014.

[20] I. Intel, "Intel 64 and ia-32 architectures software developer's manual," 2010.

[21] G. F. Grohoski, M. Shah, J. D. Davis, A. Saulsbury, C. Fu, V. Iyengar, J.-Y. Tsai, and J. Gibson, "Level 2 cache index hashing to avoid hot spots," 2007. US Patent No. 7,290,116.

[22] R. Hund, C. Willems, and T. Holz, "Practical timing side channel attacks against kernel space aslr," in *IEEE Symposium on Security and Privacy (SP)*, pp. 191–205, 2013.

[23] C. Maurice, N. L. Scouarnec, C. Neumann, O. Heen, and A. Francillon, "Reverse-engineering intel last-level cache complex addressing using performance counters," in *Proc. of 18th International Symposium on Research in Attacks, Intrusions and Defenses (RAID)*, 2015.

[24] Y. Yarom, Q. Ge, F. Liu, R. Lee, and G. Heiser, "Mapping the intel last-level cache," in *Cryptology ePrint Archive: Report 2015/905*, 2015.

[25] T. Zhang, S. Chen, F. Liu, and R. Lee, "Side channel vulnerability metrics: the promise and the pitfalls," in *Workshop on Hardware-Assisted Security, held in conjunction with HPCA-13*, 2013.