Review of Secure Cache Designs against Conflict-based Cache Timing Attacks

Parangat Mittal
University of California, Los Angeles
parangat@ucla.edu

***Abstract*—Cache Side Channel attacks represent significant security threats in modern computing systems by exploiting a timing-based side channel in shared CPU caches. This review paper evaluates and compares MIRAGE, PhantomCache, ClepsydraCache, and SassCache – four secure cache designs that aim to mitigate these attacks through innovative architectural changes in the traditional caches. By examining the different cache aspects each design builds upon, this paper provides a comprehensive analysis of each approach to enhance security. The evaluation includes a discussion of the shared threat model, performance benchmarks, and security effectiveness, offering insights into the strengths and limitations of these strategies.**

***Index Terms*—Cache Side-Channel Attacks, Conflict-Based Cache Timing Attacks, Hardware Security, Last-Level Caches**

I. Introduction

CACHES in modern computing systems bridge the latency gap between main memory and CPU by keeping frequently-accessed data in a faster memory element closer to the processing element. However, this performance benefit comes with an added security vulnerability, where an attacker can extract information from the system through CPU microarchitecture. Typically, modern systems employ multi-level cache hierarchies to get maximum utilization, and performance-critical caches such as last-level caches (LLC) are often deployed as a shared resource between different CPU cores. When a victim application running on a core shares the LLC with an attacker application running simultaneously on a different core, cache side channels can leak memory access patterns of the first application.

A cache hit and a cache miss differ in access latency. This timing difference is observed by an attacker to infer the access behavior of a victim. The attacker forces a conflict in the shared cache concurrently when a victim is also utilizing it, thus creating a possible time difference in accessing certain addresses. These attacks on traditional caches are not very complex for the attacker to execute, and offer a low-cost solution to leak sensitive information from the user process. Among the most notorious such attacks are Prime+Probe, Flush+Reload and Evict+Reload, all of which are mitigated by secure cache designs reviewed in this paper. The modus operandi of these attacks is as follows. All these fundamental attacks operate on the deterministic nature of a cache.

A. *Prime+Probe*

In a Prime+Probe attack, the attacker primes the cache by filling a cache set with their own data, waits for the victim to access the cache, and then probes the cache to measure access times and identify which lines were evicted.

B. *Flush+Reload*

Flush+Reload Attack involves the attacker flushing a shared cache line using the Flush instruction available in certain ISAs, and later measuring access times to detect if the victim has reloaded the flushed cache line.

C. *Evict+Reload*

When a dedicated Flush instruction is not available, an Evict+Reload attack involves the attacker to evict a cache line and wait for the victim to reload it, while measuring the access time of the line to detect the reload.

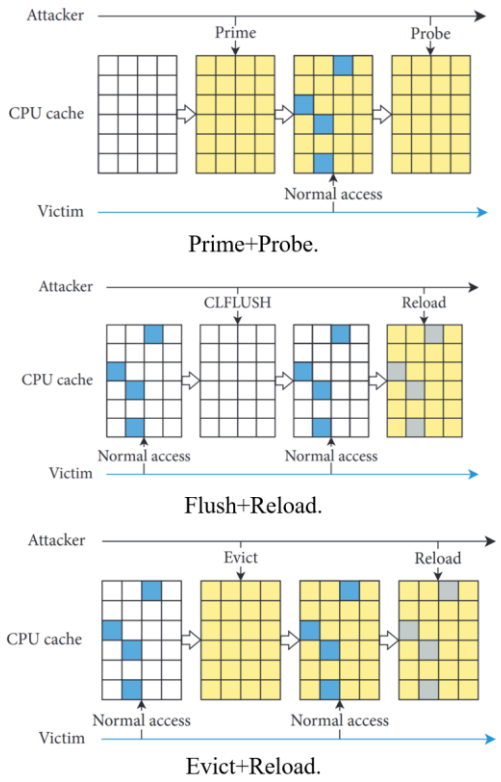

Fig. 1. Overview of Cache Timing Attacks

II. Solution

The primary design objective of a cache is to accelerate accesses to the main memory, the vulnerable timing side channel cannot be easily protected without any performance overhead. The traditional hardware defenses against these kinds of attacks rely on either cache partitioning or index randomization. Partitioning-based designs allocate parts of the cache per security domain, and Randomization-based designs obfuscate the mapping of addresses to cache lines. While both approaches are promising solutions to this problem, due to recent advanced attacks, they either fall short of strong security or pay a penalty in the form of system performance.

A conventional cache has three distinct stages of functionality which are coupled one after the another, which are line search, line placement and line eviction; the latter two are sometimes referred to as line replacement, which is the core vulnerability exploited in a conflict-based attack. When designing a secure cache, these two policies are improved, typically by employing one of the two approaches from partitioning and randomization. Randomness is typically cryptographically-induced, or in some cases through a hardware Linear Feedback Shift Register (LFSR).

For this review paper, we have chosen four recently-proposed cache architectures which aim to mitigate these attacks through a different design parameters and algorithms. A high-level overview of these papers is summarized in Table I.

TABLE I
HIGH-LEVEL SUMMARY OF THE DESIGNS

Design	Key Principle
MIRAGE	Decouple tag store and data store; provision of extra invalid tags
PhantomCache	Localized randomization; parallel search mechanism
ClepsydraCache	Time-based evictions; efficient randomized address mapping
SassCache	Invisibility of cache lines; new cache-specific cryptographic function

III. Design Details

A. MIRAGE

MIRAGE stands for Multi-Index Randomized Cache with Global Evictions. It addresses the conflict-based cache attacks by using a cache organization structure such that tag-store and data-store tables are decoupled, line install and line replacement decision is independent of each other, and there is randomness in the system to reduce predictability of access patterns. The main challenge remains to minimize additional logic overhead, and maintaining a practical cache lookup.

Decouple tag-store and data-store. The dissociation of tag-store and data-store in the cache adds an extra layer of complexity to access the actual data and address that has been requested by the victim. This is achieved through a pointer-based indirection approach. Each tag-store entry is appended with a Forward Pointer (FPTR) to the location in the data-store table where its metadata is stored. Similarly, each entry in the data-store has a Reverse Pointer (RPTR) back to the corresponding tag-store entry. This makes it possible to arbitrarily map any address tag to any available data location, thus breaking the *always-mapped-to-one* approach.

Decouple line install and line eviction. On a cache conflict, traditionally one of the existing lines is evicted, and is replaced with a new line, which is the vulnerability exploited in this attack. In order to have global evictions from the whole cache instead of a fixed location, the cache is overprovisioned to have extra entries, deemed as invalid. Since the size of tag-store is less than data-store, it is cheaper to have extra tag-store entries. On installing a new line, if any invalid tag is available, it is used to place that lines, eliminating the need for eviction. Instead of eviction, MIRAGE invalidates the older lines, and uses that space to install new lines completely unconnected to evicted ones.

Introduction of Randomness. While the previous two design techniques add a certain layer of security, it is still possible to break that through clever, though expensive, maneuvering. As like any other secure defense, MIRAGE also has a degree of randomness incorporated to completely obfuscate the cache. Tag-store is split into two equal partitions (called *skews*), with hash functions mapping addresses to sets in each skew. The choice of skew selection is based on a load-balancing policy, which chooses the one with maximum invalid tags present to install the new line, to minimize chances of eviction.

B. PhantomCache

PhantomCache addresses the problem of conflict-based attacks on larger last-level caches (LLCs), which suffers from increased access times and frequent misses if a global randomized address mapping strategy is used. This is because to efficiently get the security benefits, periodic remapping is required to conceal the mapping of addresses from potential discovery by the attacker. This remapping incurs a large cost for larger caches such as LLCs, and PhantomCache solves this problem by a localized remapping-free randomization scheme. By confining the randomization space to a limited number of cache sets, it ensures that lookups remain efficient while providing a level of security that is difficult to breach with advanced eviction-set discovery algorithms.

Localized Randomization. PhantomCache limits the mapping of an address within a limited number of cache sets instead of across the entire cache. For each address, a predefined number of candidate sets are selected randomly. The selection uses the address and a random mapping function to compute cache set indices. It uses the built-in Hardware Random Number Generator to generate certain random salts which are hashed with the index bits of the address to get the sets, and one of them is randomly chosen to accommodate the new line. Since the built-in hardware generator is leveraged, there is only one cycle overhead-per-access keeping the overall system performance optimized. Since index is used for hash

computation, only the tag is cached which eliminates the need for a wider cache.

Parallel Search Mechanism. Upon a cache access request, PhantomCache computes the indices of all the candidate sets associated with that address, and then compares the cached tag at those indices with the request to indicate a hit or a miss. Since the number of candidate sets is limited, it uses a parallel search mechanism to find the right member set. Upon a hit, for the address restoration both the tag and the index are needed. The tag is already cached, and the index is reverse computed using the virtual address and the cached random number used to generate the salts by leveraging the invertibility of the exclusive-OR function.

C. ClepsydraCache

ClepsydraCache mitigates conflict-based attacks on the cache by obfuscating the very side-channel attackers use, i.e. time. It introduces time-based evictions in the cache rather than capacity-based evictions, along with a novel index randomization. The overall aim of this secure cache design is to reduce the overall amount of cache conflicts and to remove the direct link between cache access patterns and cache evictions. The eviction strategy is based on timing instead of contention, which is called as *cache decay*.

Time-based Evictions. Each value is assigned a randomly initialized TTL (Time-To-Live) value when it is installed in the cache. TTL value indicates how long the entry remains in the cache. It is steadily reduced and when expired (or becomes zero), the entry is evicted from the cache. The TTL-decrease rate is dynamically adapted to get high utilization with minimal conflicts.

On a cache miss, the data is loaded from the memory and the addressing function determines the target cache entry. Simultaneously, a uniformly random TTL is chosen and assigned to this entry, and is reduced steadily there on. As this value becomes zero, it is marked invalid, and written back to the main memory. If a cache hit occurs, the data is served to the requester and the TTL is set to another random value so that it stays longer in the cache.

Random Address Mapping. ClepsydraCache uses a low-latency randomization function which assigns pseudorandom cache lines in each way to a given address. A single address maps to same entry in each way at every access, the collection of which is called a dynamic set. It is assumed that each dynamic set has at least one invalid or expired entry due to time-based evictions, and one of them is selected to store the data without any conflict. In case, there exists a dynamic set with all valid entries, then a random replacement policy is used to replace any one entry.

D. SassCache

SassCache mitigates conflict-based cache attacks by making the victim lines invisible or hidden to the attacker, so that those cannot be probed or reloaded to backtrace the memory access patterns. It proposes a new randomization function, specifically designed for address-to-space mapping in caches, unlike other designs which repurpose existing cryptographic functions. It uses a two-layer cryptographic construction to limit the number of lines visible to the attacker, and prevent them to evict it. SassCache is a skewed set-associative design with an isolation mechanism such that each security domain only has access to a different and partially overlapping portion of the cache.

Index Generation Layer. Index Generation Layer (IGL) decouples the cache sets and physical addresses across multiple security domains. It makes the cache very difficult to be profiled to mount an attack through the side channels. It maps the address to W independent identifiers labelled as id_i, using a block cipher with Security Domain Identifier (SDID) and a key `<SDID, key>` as the key, and the address as the nonce to generate W intermediate identifiers. These identifiers are expected to be uniformly and independently distributed. These are unpredictable, not controllable and thus not observable for any attacker, and even learning them does not allow the recovery of the secret key in any form.

Index Spacing Layer. Index Spacing Layer (ISL) limits the accessibility of cache lines between security domains. As opposed to slicing the cache into static partitions and allocating it to each domain, this layer selects the cache lines pseudo-randomly and assigns them to different security domains. This pseudorandom mapping is based on a cryptographic tiny block cipher which is invoked in parallel for W intermediate identifiers generated by the previous layer. This gives the final indices for the particular address as idx_i.

To give a numerical context, the IDF (Index Derivation Function) of the SassCache maps a-bit addresses (e.g. a=48) into n-bit indices idx_i (e.g. n=11), with intermediate identifiers id_i of l = n+t bits (where t is used to control the coverage of the cache). The coverage comes with diminishing returns, a higher value provides more security but comes with storage overhead.

IV. EVALUATION

All the four cache designs reviewed in this paper are compared and evaluated from two aspects – security effectiveness and performance overhead. While there is no common single benchmark on which each design is run, there are overlapping tests and results which have been provided by the authors. For the purpose of this review, we present the evaluation of each design with respect to a traditional baseline cache, which is not secured against conflict-based timing side-channel attacks.

A. Security Effectiveness

Threat Model. All the designs are evaluated for their security effectiveness against a common threat model, which involves multiple processes running concurrently in a multi-core processor, with each core having access to a shared last-level cache (LLC). These processes can observe and influence cache behavior through timing analysis, aiming to extract sensitive information from co-resident victim applications. The attackers leverage cache contention to create timing discrepancies that reveal the victim's memory access patterns.

MIRAGE. This design is evaluated for a security effectiveness by a Bucket-and-Balls model, where the rate of Set-Associative Evictions is calculated for the cache. The buckets and balls are analogous with cache sets and new addresses, where spillage of balls is treated as a set-associative eviction. The set associative eviction rate is expected to be as least as possible, since that is the security violation this design is protecting. By an analytical analysis, MIRAGE has different eviction rates for different ways-per-set. The default design with 14 ways per set has an eviction rate of 10^{34} line installs, or once in every 10^{17} years, effectively securing the cache against the attacks during its lifetime.

PhantomCache. The degree of security of this design is evaluated on three security subgoals – security of eviction addresses, hardness of eviction set minimization and hardness of salt cracking. The degree of security is dependent on the number of candidate cache sets used to build the PhantomCache. The general observation that more the cache sets, more the security holds true for this context as well, but this comes with storage overhead. The default and recommended PhantomCache design has 8 candidate sets, comes with a high eviction set minimization complexity, which would take 9,583,986 years for the attacker to break.

ClepsydraCache. Security evaluation involves assessing the reduction in cache side-channel leakage through time-based eviction strategies. ClepsydraCache shows a significant reduction in side-channel leakage, with a decrease in attack success rates by approximately 98%. The time-based eviction mechanism effectively randomizes eviction patterns, making it challenging for attackers to correlate cache accesses with specific memory operations.

SassCache. SassCache's security is evaluated by analyzing the number of successful contention and occupancy attacks. The evaluation metrics include the difficulty of forming eviction sets and the time taken to perform successful attacks. achieves a reduction in attack success rates to less than 1%, with attackers requiring over 10x the amount of time to identify victim cache lines. The two-layered cryptographic construction ensures that cache lines remain hidden and inaccessible, providing robust security against advanced attacks. It prevents full eviction of the target cache line in 99.99997% of cases, and the attacker can try to construct an eviction set for 1 in 3,000,000 cache lines, which makes it 64B in 185MB memory, increasing the cost of attack significantly.

B. Performance Overhead

MIRAGE incurs a modest logic overhead due to the pointer-based indirection and additional invalid tags, but this is offset by its efficient handling of cache entries. The storage overhead is due to the overprovisioned tag store, estimated at around 17-20% additional storage compared to traditional caches. Despite these overheads, the performance impact remains minimal, with an average slowdown of only 2%.

PhantomCache maintains low logic overhead by leveraging built-in hardware random number generator (HRNG) for localized randomization, resulting in only one cycle overhead per access. Its storage overhead is minimal, given the limited number of candidate sets for each address and the use of efficient parallel search mechanisms. The impact is limited to an average slowdown of 1.20%.

ClepsydraCache introduces additional logic for managing TTL values and dynamic set assignments. The time-based eviction mechanism adds complexity, but it is designed to be hardware-efficient. Storage overheads are relatively low, attributed mainly to the randomized address mapping function and the dynamic set management. The performance overhead is measured at 1.38%, making it a viable security enhancement with acceptable costs.

SassCache has higher logic overhead due to the cryptographic functions employed for index generation and spacing. The two-layered approach involves complex computations, but these are optimized for parallel execution. Storage overhead is driven by the need for intermediate identifiers and the additional indices for secure spacing. The performance impact remains manageable, with an average slowdown of 1.75%.

V. Conclusion

In conclusion, in this review paper, we explored four innovative cache designs—MIRAGE, PhantomCache, ClepsydraCache, and SassCache—that address the significant security threats posed by conflict-based cache timing attacks. Each design introduces unique architectural modifications to mitigate these attacks effectively. While each design offers distinct advantages and trade-offs in terms of security and performance overhead, a hybrid approach that incorporates the best elements of these designs could provide an optimal solution. This hybrid design could leverage MIRAGE's indirection, PhantomCache's localized randomization, ClepsydraCache's temporal eviction strategy, and SassCache's cryptographic security to achieve robust protection against cache side-channel attacks with minimal performance impact.

References

[1] F. Liu, Y. Yarom, Q. Ge, G. Heiser, and R. B. Lee, “Last-level cache side-channel attacks are practical,” in *S&P*, 2015, pp. 605–622.

[2] Y. Yarom and K. Falkner, “Flush+ reload: A high resolution, low noise, l3 cache side-channel attack.” in *USENIX Security*, vol. 1, 2014, pp. 22–25.

[3] D. Gruss, R. Spreitzer, and S. Mangard, “Cache template attacks: Automating attacks on inclusive last-level caches.” in *USENIX Security Symposium*, 2015, pp. 897–912.

[4] Gururaj Saileshwar and Moinuddin K. Qureshi. MIRAGE: mitigating conflict-based cache attacks with a practical fully-associative design. In *USENIX Security Symposium*, 2021.

[5] Qinhan Tan, Zhihua Zeng, Kai Bu, and Kui Ren. PhantomCache: Obfuscating Cache Conflicts with Localized Randomization. In NDSS, 2020.

[6] J. P. Thoma *et al.*, ‘ClepsydraCache -- Preventing Cache Attacks with Time-Based Evictions’, in *32nd USENIX Security Symposium (USENIX Security 23)*, 2023, pp. 1991–2008.

[7] L. Giner et al., “Scatter and Split Securely: Defeating Cache Contention and Occupancy Attacks,” in 2023 IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA: IEEE, May 2023, pp. 2273–2287. doi: 10.1109/SP46215.2023.10179440.