This article has been accepted for publication in IEEE Computer Architecture Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LCA.2023.3297875

The Mirage of Breaking MIRAGE:
Analyzing the modeling pitfalls in emerging "attacks" on MIRAGE

Gururaj Saileshwar
University of Toronto & NVIDIA Research
gururaj@cs.toronto.edu

Moinuddin Qureshi
Georgia Institute of Technology
moin@gatech.edu

***Abstract*—This paper studies common modeling pitfalls in security analyses of hardware defenses to highlight the importance of accurate reproduction of defenses. We provide a case study of MIRAGE [1], a defense against cache side channel attacks, and analyze its incorrect modeling in a recent work [2] that claimed to break its security. We highlight several modeling pitfalls that can invalidate the security properties of any defense including (a) incomplete modeling of components critical for security, (b) usage of random number generators that are insufficiently random, and (c) initialization of system to improbable states, leading to an incorrect conclusion of a vulnerability, and show how these modeling bugs incorrectly cause set conflicts to be observed in a recent work's [2] model of MIRAGE. We also provide an implementation addressing these bugs that does not incur set-conflicts, highlighting that MIRAGE is still unbroken.**

***Index Terms*—cache side-channel attacks, randomized caches.**

I. Introduction

Security analysis of emerging defenses is challenging. Unlike performance analysis where approximations of a design are often acceptable, security analysis requires an exact reproduction of the original implementation. Even minor variations in implementations can lead to divergent security properties. Without an exact reproduction of the defense, there is a risk of a false discovery of a vulnerability due to incorrect modeling.

This work seeks to highlight the importance of accurate reproduction of defenses, by studying the pitfalls of incorrect modeling. We do this through a case study of MIRAGE [1], a defense against set-conflict based cache side channel attacks and analyze the implications of its incorrect modeling in a recent work [2] that analyzed the security of MIRAGE.

Set-conflict based cache side channels allow a spy process to leak sensitive data of a victim process through set-conflicts in a shared cache, which allow addresses accessed by the victim to be observed by the spy. To defend against these attacks, recent randomized caches [3]–[5], randomize the mapping of addresses to cache sets to obfuscate the pattern of set-conflicts. Subsequently, newer attacks [4], [6]–[8] emerged that succeeded despite such obfuscated set-conflicts.

MIRAGE [1] proposed to eliminate set-conflicts with a fully-associative design. It achieves this with a set-associative tag-store that over-provisions invalid tags in sets and with load-balancing that guarantees new addresses are always installed in invalid tags without set-conflicts. Cache installs result in *global* evictions, where a replacement candidate is selected randomly from the entire cache, leaking no address information about installed lines. MIRAGE guarantees global evictions for system lifetime and thus eliminates set-conflict based attacks.

From 2018 to 2021, half-a-dozen randomized cache defenses were broken within months whereas MIRAGE has been unbroken since 2021, until a recent work "Are Randomized Caches Truly Random? (ARCTR)" [2], claimed to observe set-conflicts in MIRAGE. However, on deeper inspection, we discovered ARCTR's set-conflicts were due to their inaccurate modeling of MIRAGE. Below, we describe modeling pitfalls that can break the security of any defense, including MIRAGE, and discuss how ARCTR suffers from these modeling issues.

Pitfall #1: Not modeling all the components in a defense. Even if a component does not affect the common case behavior (e.g., no visible performance impact), not modeling it can adversely affect system behavior in a pathological scenario and thus greatly reduce its security. As an example, we observe that ARCTR's [2] model of MIRAGE does not implement global evictions from the original work. This leads to more tags being valid than the cache capacity, and some sets being naively full, which leads to an incorrect observation of a set conflict. We describe this in detail in Section IV-B (Bug-1).

Pitfall #2: Using randomizers that are non-uniform. Randomization based defenses fundamentally rely on randomizing functions being correctly implemented. If the defense is unable to randomize uniformly, it can fail catastrophically. For example, the ARCTR [2] work's implementation uses a buggy cipher implementation for randomizing mapping of addresses to cache sets, which results in a biased mapping of more addresses towards a few sets. This causes a few sets to be fully occupied and results in set conflicts. We describe this implementation bug in Section IV-B (Bug-2).

Pitfall #3: Using non-independent randomizers. For defenses that rely on multiple randomizing functions, it is important that each of these be mutually independent. For example, if a defense uses two random functions, like MIRAGE, and both are correlated, address collisions in one function may result in collisions in the other as well, thus reducing the effectiveness of multiple functions. In the limit, this can cause set-conflicts as frequently as a single randomizing function.

Pitfall #4: Initializing the system to improbable states. The initial system state during security analysis determines the adversarial effort needed to defeat a defense. If a study starts from an almost broken state, an attack require negligible effort – but if such a state is improbable under natural operation, then such an attack is impractical. The ARCTR [2] work initializes tags as valid with 50% probability, leading to a few cache sets to start as already full, which is quite improbable in regular operation. We describe this bug in Section IV-B (Bug-3).

After fixing the modeling bugs in the ARCTR [2] work, we observed NO set conflicts, highlighting that MIRAGE is not broken. Our code with bug fixes is at https://github.com/gururaj-s/refuting_HPCA23_randCache.

In the rest of the paper, we formalize the conditions for a set conflict in MIRAGE [1] in Section III, show a case study of the modeling bugs in the ARCTR paper [2] in Section IV and Section V, and discuss future attacks in Section VI.

Authorized licensed use limited to: Nvidia Corp. Downloaded on August 11,2023 at 23:24:00 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Computer Architecture Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LCA.2023.3297875

II. Background on Mirage

MIRAGE [1] is a randomized cache that provides security against conflict based attacks by guaranteeing the abstraction of a fully associative cache. It relies on three key design components. First it provisions *extra invalid tags* in the LLC to allow new addresses to be installed in the tag-store without set-conflicts. Second, it uses *load-balancing indexing* based on Power of 2 Choices [9] to guarantee the availability of invalid tags. As shown in Figure 1, the tag-store is split into two skews or partitions. When a new address is installed, it indexes two sets (one in each skew) using a randomized mapping, and is installed in the set with fewer valid tags. This ensures balanced availability of invalid tags across sets and new cache line installs do not incur a set-associative eviction.

The third component is cache line replacement via *global evictions*. When a new line is installed, it evicts an entry randomly selected from among all the data-store entries. The resident data in the entry is evicted along with its tag (identified based on a Reverse Pointer or RPTR lookup), and the new line's tag is associated with this entry using a Forward Pointer (FPTR). Thus, evictions occur in a fully associative manner and do not leak information about installed addresses.

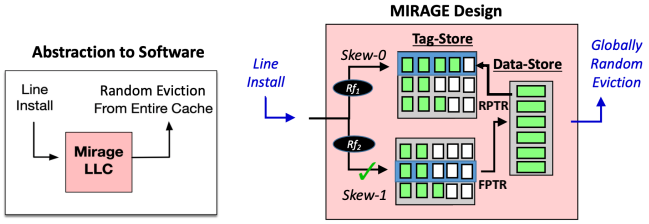


Fig. 1: Design of MIRAGE and its Abstraction to Software.

Security Guarantee: MIRAGE [1] suggests that over-provisioning an 8-way cache with 6 extra ways (75% extra) in the tag-store guarantees the probability of a set-associative eviction is once in 10^{34} cache installs, which would take almost 10^{17} years. MIRAGE ensures set-conflicts are unlikely in system lifetime and eliminates conflict-based attacks.

III. Analyzing Mirage's Security Guarantees

A recent work, ARCTR [2], claimed to be able to observe frequent set associative evictions in MIRAGE. To that end, we first reason about the conditions required for a set-conflict in MIRAGE, and then analyze the ARCTR's implementation to understand why they observe set-conflicts.

A. Conditions Required for Set-Conflicts in MIRAGE [1]

A set conflict in MIRAGE occurs only when a new line to be installed finds the two indexed sets where it may be installed are both full. At that point, a set-conflict occurs (instead of a global eviction) and the cache invalidates an existing address from these sets to free up a tag for the new address.

Intuitively, the Power-of-2-Random-Choices indexing coupled with global evictions, makes such an occurrence quite improbable. For a set to grow its occupancy by 1 ($w \rightarrow w+1$), both indexed sets need to have at least w lines. Thus, the probability of high loaded sets becomes smaller ($p_{w+1} = p_w^2$, if p_w is the probability of a set with at least w lines). At the same time, global evictions reduce the occupancy of sets. As the probability of a way in a set being occupied squares with each additional way, MIRAGE with 6 extra ways makes the probability of two random sets being fully occupied extremely small (within 10^{-34}). We refer the reader to the original work [1] for a more formal analysis.

B. Implications of an Observed Set-Conflict

If a set-conflict is observed in MIRAGE, the root-cause is likely to be one of the following:

1) **Implementation Bug in the Randomizing Functions.** A key assumption of the Power-of-2-Choices indexing in MIRAGE is that randomizing function is uniformly random and the functions in each skew are independent. If either of these assumptions are broken, *e.g.,* due to a bug in the cipher, it would result in incorrect functioning of load-balancing. Or the global evictions might not be sufficiently random due to incorrect implementation.
2) **Design Flaw in Power-of-2-Choices Indexing:** If there are no bugs, frequent set-conflicts are indicative of a deeper algorithmic flaw in the power-of-2-choices indexing and global evictions. This would be an interesting discovery given the rich body of theoretical results highlighting the load-balancing properties of Power-of-2-Choices [9].

IV. Case Study of Bugs in "Are Randomized Caches Truly Random?" [2]

A recent work "Are Randomized Caches Truly Random? (ARCTR)" [2], published at HPCA'23, claimed to observe set-associative evictions in MIRAGE once every 100,000 cache installs, implying a breach in its security. However, in the process of identifying the root-cause of set conflicts, we inspected their code[1] and found fundamental bugs in their MIRAGE implementation to be the cause of these set-conflicts.

We identified the following bugs in both the bucket and balls simulation and the cache simulator models of ARCTR [2], as shown in Figure 2:

1) No implementation of Global Evictions in both the bucket and balls model and the python cache simulator.
2) A broken cipher implementation in the address to set mapping function in the cache simulator, that does not provide uniform randomization.
3) A simulation bug in the cache simulator where the cache state is initialized with fully occupied sets which are improbable in MIRAGE.

After fixing these bugs, we observed NO set-associative evictions, highlighting that MIRAGE is indeed not broken. Our code with bug-fixes is open-sourced at this link[2]. Next, we provide an analysis of the bugs in the ARCTR work [2].

[1] https://github.com/SEAL-IIT-KGP/randCache
[2] https://github.com/gururaj-s/refuting_HPCA23_randCache

Authorized licensed use limited to: Nvidia Corp. Downloaded on August 11,2023 at 23:24:00 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Computer Architecture Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LCA.2023.3297875

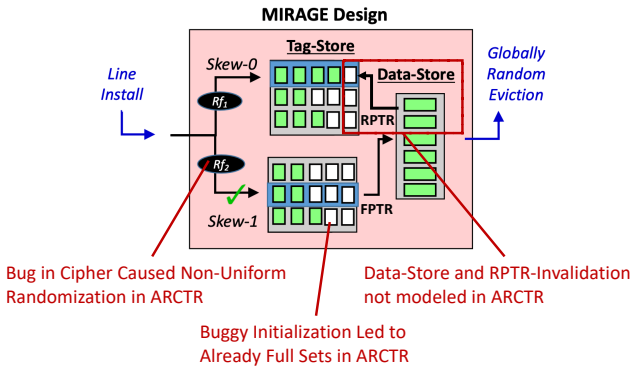


Fig. 2: Components of MIRAGE which were incorrectly modeled in the ARCTR [2] work are highlighted in red text.

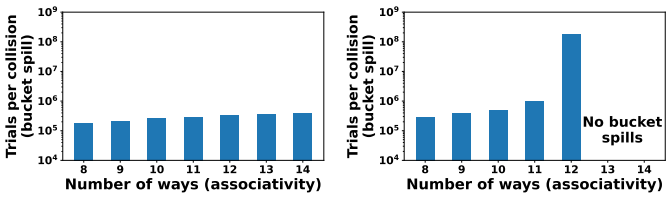
A. Bugs in the Bucket and Balls Simulation

The bucket and balls simulation of MIRAGE simulates a large number of ball throws (cache line installs) into randomly selected buckets (cache sets) to simulate cache lines being installed into random sets in MIRAGE.

A key component of MIRAGE is global eviction which randomly evict entries from the cache and crucially reduces the high occupancy sets. To model this, with each ball insertion, a randomly selected ball needs to be evicted from among all the balls in buckets, to signify a cache replacement occurring on each cache line install.

The ARCTR bucket and balls simulation crucially does not model global evictions, as shown in Figure 2. This causes the simulator to insert more balls in buckets than even the cache capacity leading to distributions of balls in buckets that are incorrect and not representative of MIRAGE [1]. We detected this by adding an assert statement checking that the number of balls in buckets is no more than the cache capacity, and we detect an assert-failure before the first bucket spill (set associative eviction).

In Figure 3, we reproduce results from Figure 6 of the ARCTR paper [2]. In the original result (which had assert failures) in Figure 3a, the number of ball throws (trials) before a collision (bucket-spill or set-conflict) grows linearly. By adding a $remove\_ball()$ implementation to model global evictions as a bug fix, we obtain the results in Figure 3b showing a super-exponential trend with increasing associativity. With the bug fix, we observe no bucket spills (set conflicts) with associativity of 13 or 14 ways even after 1 billion ball throws, matching the results from MIRAGE [1].



(a) Original [2] (assert failure) (b) After Bug-Fix

Fig. 3: Number of trials before a collision (bucket spill) – Figure-6 in ARCTR paper. (a) In the original code [2], the trials before a set-conflict grows linearly. (b) After the bug-fix, it is super-exponential: we see no bucket spills at associativity of 14 (8 + 6 extra ways), the default in MIRAGE [1].

B. Bugs in the Python Cache Simulator

The ARCTR paper [2] also presents a python cache simulator model of MIRAGE showing set-conflicts are observable under random accesses. We observed the following bugs causing these incorrect results.

Bug-1. The python cache simulator code also does not implement Global Evictions in MIRAGE. As a result, it also suffers from similar assert failures as the buckets and balls model (lines in cache exceeding the cache capacity), once we added in an assert checking whether the lines in the cache exceed the cache capacity. In personal correspondence, the authors of the ARCTR work shared an updated code implementing global evictions, but we identified two additional bugs that led to incorrect results.

Bug-2. We identified a bug in the cipher used for randomizing the cache set indexing. The security of MIRAGE relies on a uniformly random set-index function. However, we discovered that the cache set indices generated in the ARCTR paper's code were not uniformly random, but in fact biased towards certain sets. The root-cause was a buggy implementation of the PRESENT cipher used for the cache set indexing in the author's code. We tested it with known test vectors from the PRESENT paper [10] and the ARCTR implementation fails those tests, generating incorrect ciphertexts for known plaintext and keys. Replacing this buggy cipher with a standard implementation of AES-128 or PRINCE-64 (used in the MIRAGE paper [1]), with random keys, addressed this.

Figure 4 shows the distribution in the set index for 1 million random addresses with the original code and after the bug fix. Figure 4(a) shows much skew in distribution of indices across possible values in the original code. Replacing this cipher with AES-128 or PRINCE-64, which is used in the MIRAGE paper, with random keys, we were able to produce uniform set-indices across addresses, as shown in Figure 4(b,c).

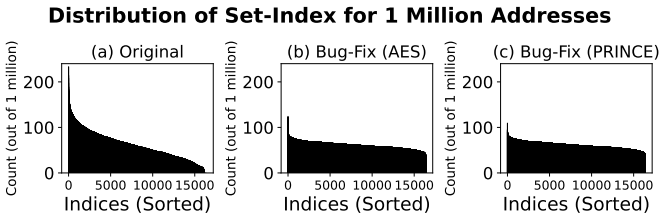


Fig. 4: Bug in the Set-Index function causes non-uniform indices. MIRAGE relies on a uniformly random cipher. Across 1 million random addresses, (a) the cipher in the original code has a skewed distribution with std-dev of 31 that is not uniformly random. (b,c) Using a standard AES or PRINCE cipher results in a uniform distribution, with std-dev of 7.9.

Sanity Check with Analytical Model: We can corroborate the results from Fig 4 using a simple analytical model. If 1 million balls (lines) are thrown in 16,364 buckets (sets), then the average (μ) is equal to 61. Given this is a Poisson process, we can estimate the standard deviation (σ) across buckets (sets) to equal the square root of the mean, thus $\sigma = \sqrt{61} = 7.8$. We observe that with AES and PRINCE indeed we get a σ of 7.9, which is close to the expected value. For a well-behaved process, we expect values that

Authorized licensed use limited to: Nvidia Corp. Downloaded on August 11,2023 at 23:24:00 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.

This article has been accepted for publication in IEEE Computer Architecture Letters. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/LCA.2023.3297875

are 6 standard deviations away from the mean to be highly improbable (about 1 in a billion), so we should not expect to see values above $61 + 6 \cdot 7.9 = 108$, and that is indeed the case with AES and PRINCE.

However, with the author's implementation of PRESENT, we get $\sigma = 31$, denoting a non-Poisson process with very high variation across buckets (sets). Indeed, we see values exceeding 200 with the PRESENT implementation, indicating significant non-uniformity across sets. This non-uniformity breaks the assumption of MIRAGE that the cache index is decided by a uniformly random mapping of addresses to sets.

Bug-3. We discovered an additional bug in the simulator that the tag-store is initialized by assigning each tag valid with a probability of 50%. However, this results in high occupancy levels in a few sets, which is virtually impossible under load balancing in MIRAGE. Figure 5(a) shows the set-occupancy levels for different sets at initialization in the ARCTR code, and we observe that ARCTR simulation starts from a state where more than one set is full at initialization time. In essence, to break MIRAGE, the paper assumes a broken state.

The correct initialization methodology for the tag-store is to start with an invalid cache and insert K random addresses (without loss of generality, we used 1 million addresses) through the cache interface including load-balancing and global evictions. Figure 5(b) shows the initial set-occupancy levels in the simulations after this bug fix, where the high occupancy buckets are prevented by Mirage.

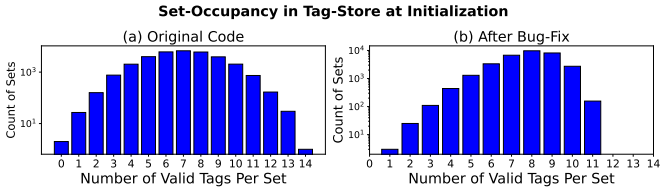


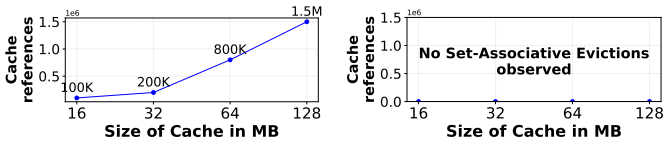
Fig. 5: Bug in Initialization of Tags. (a) Original code assumes each tag can be valid with probability=0.5, which results in already full sets at initialization. (b) Bug fix: Cache starts from an invalid state and has K random addresses inserted and resulting in a representative set occupancy at initialization.

V. Results: No Set-Conflicts after Bug-Fixes

After fixing these bugs – ensuring a correct global eviction implementation, correct initialization of tag-store, and using a functionally correct cipher (AES/PRINCE) – in the author's python cache simulator, we observe no set-associative evictions (valid tag evictions) for varying cache sizes. This further invalidates the claims made in the paper that Mirage is broken. Figure 6 shows the results with the original ARCTR code and the results after the bug fixes.

VI. Vulnerability to Cache Occupancy Attacks

Given that MIRAGE is now shown to be secure against set-conflict based attacks so far, a natural question would be to study its vulnerability against other cache attacks, like cache occupancy attacks. Note that cache occupancy attacks are explicitly highlighted to be outside the threat model of MIRAGE in the original paper [1]. And yet, a recent work [11] claims to



(a) Original ARCTR Code [2] (b) After Bug-Fix

Fig. 6: Number of cache references before a set-associative eviction (valid tag eviction) – Figure-7 in the ARCTR paper. (a) In the original code [2], the cache references before a set-conflict grows linearly. (b) After the bug-fix, there are no set-associative evictions in MIRAGE with 6 extra ways.

demonstrate occupancy attacks on MIRAGE. Given that MIRAGE (under global evictions) is functionally equivalent to a fully associative cache with random replacement, an attack on MIRAGE is equivalent to attacking a fully associative cache, which is not designed to prevent occupancy attacks. Preventing occupancy attacks requires alternative design paradigms such as cache partitioning which restrict sharing of cache space, and these are beyond of the purview of MIRAGE (or in general, any scheme that randomizes the line-to-set mapping), which is designed to prevent set-conflict based attacks.

VII. Conclusion

During the last decade, the importance of security has greatly increased in the architecture community. With a new paradigm, comes a new way of thinking. For ensuring security, it is important to think in terms of the worst-case, and pay attention to the details, otherwise, approximations and minor changes can greatly affect the security properties. We show this phenomenon with MIRAGE, where correct modeling provides strong security. However, incorrect modeling, poor approximations, and buggy ciphers, as done in the ARCTR paper, can lead to incorrect conclusions about the security properties of MIRAGE. We thank the authors of the ARCTR work for sharing their code publicly which enabled our analysis.

References

[1] G. Saileshwar and M. Qureshi, "MIRAGE: Mitigating Conflict-Based Cache Attacks with a Practical Fully-Associative Design," in *USENIX Security*, 2021.

[2] A. Chakraborty, S. Bhattacharya, S. Saha, and D. Mukhopadhyay, "Are Randomized Caches Truly Random? Formal Analysis of Randomized-Partitioned Caches," in *HPCA*, 2023. [Online]. Available: https://github.com/SEAL-IIT-KGP/randCache

[3] M. K. Qureshi, "CEASER: Mitigating conflict-based cache attacks via dynamically encrypted address," in *MICRO*, 2018.

[4] ——, "New attacks and defense for encrypted-address cache," in *ISCA*, 2019.

[5] M. Werner *et al.*, "Scattercache: Thwarting cache attacks via cache set randomization," in *USENIX Security*, 2019.

[6] F. Liu *et al.*, "Last-level cache side-channel attacks are practical," in *S&P (Oakland)*, 2015.

[7] A. Purnal *et al.*, "Systematic analysis of randomization-based protected cache architectures," in *S&P (Oakland)*, 2020.

[8] W. Song *et al.*, "Randomized last-level caches are still vulnerable to cache side-channel attacks! but we can fix it," in *S&P (Oakland)*, 2021.

[9] A. W. Richa *et al.*, "The power of two random choices: A survey of techniques and results," *Combinatorial Optimization*, 2001.

[10] A. Bogdanov *et al.*, "PRESENT: An ultra-lightweight block cipher," in *CHES*, 2007.

[11] A. Chakraborty, S. Bhattacharya, S. Saha, and D. Mukhopadhyay, "A short note on the paper 'are randomized caches really random?'," *arXiv preprint arXiv:2304.00955*, 2023.

Authorized licensed use limited to: Nvidia Corp. Downloaded on August 11,2023 at 23:24:00 UTC from IEEE Xplore. Restrictions apply.
© 2023 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission. See https://www.ieee.org/publications/rights/index.html for more information.