Another Mirage of Breaking MIRAGE: Debunking Occupancy-Based Side-Channel Attacks on Fully Associative Randomized Caches

Chris Cao and Gururaj Saileshwar

***Abstract*—A recent work presented at USENIX Security 2025, *Systematic Evaluation of Randomized Cache Designs against Cache Occupancy (RCO)*, claims that cache-occupancy-based side-channel attacks can recover AES keys from the MIRAGE randomized cache. In this paper, we examine these claims and find that they arise from a flawed modeling of randomized caches in RCO. Critically, we find that the security properties of randomized caches strongly depend on the seeding methodology used to initialize random number generators (RNG) used in these caches. RCO's modeling uses a constant seed to initialize the cache RNGs for each simulated AES encryption, causing every simulated AES encryption to artificially evict the same sequence of cache lines. This departs from accurate modeling of such randomized caches, where eviction sequences vary randomly for each program execution. We observe that an accurate modeling of such randomized caches, where the RNG seed is randomized in each simulation, causes correlations between AES T-table accesses and attacker observations to disappear, and the attack to fail. These findings show that the previously claimed leakages are due to flawed modeling and that with correct modeling, MIRAGE does not leak AES keys via occupancy based side-channels.**

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

I. Introduction

MIRAGE [1] is a randomized cache design, proposed in 2021, that emulates a fully associative cache with globally random evictions, eliminating set-conflict cache side channels. It builds on theoretical foundations such as multiple randomized set indexing functions using block ciphers, and power-of-two-choices based load-balancing, guaranteeing that set-associative evictions are practically impossible in a system's lifetime. Given these strong guarantees, several works have examined whether MIRAGE's security holds in practice. In 2023, "*Are Randomized Caches Truly Random*" (ARCTR) [2] claimed to induce set-conflicts in MIRAGE, breaking its security. However, subsequent work [3] showed that these were the result of incorrect modeling by ARCTR, caused by a buggy cipher implementation, and that MIRAGE remained secure against set-conflict-based attacks.

More recently, a SEC 2025 paper, "*Systematic Evaluation of Randomized Cache Designs against Cache Occupancy*" (RCO) [4], claims that MIRAGE is vulnerable to cache-occupancy-based side-channel attacks that can leak secret AES keys. Specifically, RCO (in Section 7), claims that the AES T-Table implementation can leak the AES key on MIRAGE via the cache occupancy side-channel, and that MIRAGE's fully associative eviction policy makes it more susceptible than other randomized caches. A subsequent paper at SEC 2025, "*SoK: So, You Think You Know All About Secure Randomized Caches?*" [5] also reiterates these claims. This paper examines the validity of these claims and the underlying root-cause.

RCO claims that the root cause of the AES key leakage on MIRAGE is that the key-dependent S-Box accesses in the AES encryption are correlated with cache occupancy. The attacker observes cache occupancy of an AES encryption by measuring access times to its own cached array. By building templates for each key-byte with timings for a profiled key, and matching these templates against timings for an unknown key, an attacker leaks an unknown AES key. However, this explanation ignores the effect of MIRAGE's **random global evictions**: MIRAGE evicts a LLC line selected randomly from the entire cache on each cache insertion. Hence, repeated AES encryptions of even the same plaintext and key evict different addresses and produce different cache occupancy (unrelated to the key), and the attacker's own accesses also evict addresses randomly on each run, introducing further noise. Thus, repeated encryptions with the same plaintext and key would itself result in significant temporal variations in attacker observations (as we notice in Fig. 3b), making it unclear *how* the attack can leak the entire key byte-wise in a few hundred observations. In investigating how RCO's attack works, we uncover a flaw in RCO's modeling of MIRAGE.

> **Modeling Flaw in RCO.** RCO's simulations initialize the RNG used by MIRAGE's evictions with a **static seed** before each AES encryption, causing a **fixed** sequence of evictions on each AES run, not practical in a real attack.

Crucially, for probabilistic designs like MIRAGE which consult a random number generator (RNG) to select cache lines to be evicted, accurately modeling its security requires accurately modeling its RNG behavior. In RCO's modeling of the attack on MIRAGE, it runs successive AES encryptions in separate simulations, with MIRAGE's RNG initialized with a **constant seed (42)** at the start of each simulation, as shown in Fig. 1(a). As a result, on each AES encryption, the same sequence of cache line indices are evicted from the cache. In contrast, on a real system, back to back AES encryptions would each have an unpredictably different initial RNG state, and different cache lines would be evicted on each AES run.

To see the problem, consider the final cache occupancy (**O**) after each AES encryption. Under a realistic model of MIRAGE, **O** should be determined by both the victim's accesses (**V**) and the

Received 22 September 2025; revised 31 October 2025; accepted 14 November 2025. Date of publication 27 November 2025; date of current version 21 January 2026. This work was supported in part by NSERC Discovery under Grant RGPIN-2023-04796 and in part by NSERC-CSE Research Communities under Grant ALLRP-588144-23. *(Corresponding author: Gururaj Saileshwar.)*

The authors are with the University of Toronto, Toronto, ON M5S 2E4, Canada (e-mail: chrisj.cao@mail.utoronto.ca; gururaj@cs.toronto.edu).

Digital Object Identifier 10.1109/LCA.2025.3638260

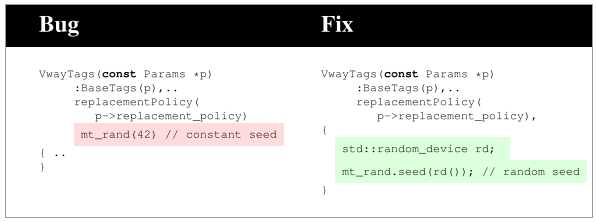


Fig. 1. Bug in RCO's modeling of MIRAGE's Evictions. **(a) Bug.** RCO initializes the RNG used for evictions with a constant seed (42), causing each AES encryption to have a fixed sequence of evictions. **(b) Fix.** We randomize the seed, ensuring each AES encryption has a different sequence of evictions, modeling a realistic implementation of MIRAGE.

global random eviction sequence during the victim's execution ($\mathbf{R}$), $O \approx f(R, V)$. However, as RCO uses a constant RNG seed, producing a fixed eviction sequence in each run, the cache occupancy becomes a function of just victim accesses, $O \approx f(V)$. This artificially creates observable correlations between the cache occupancy and the key, departing from a realistic model of MIRAGE. As a result, RCO's attack artificially observes the AES key guessing entropy (GE) to drop below 30 as number of encryptions increase, suggesting the key can be leaked by brute-force.

To ensure correct modeling of the RNG attack, one approach is to mimick a real system, and launch the simulation once and run successive AES encryptions without restarting the simulation. This ensures RNG diversity across runs, but prevents parallel simulations limiting simulation speed. To support parallel simulations and correctly model MIRAGE's RNG, we run each AES encryption in a separate simulation, like RCO, but **randomize the RNG seed for each AES run**, using `std::random_device`[1] to seed the RNG, as shown in Fig. 1(b). This models a realistic setting, where the attacker cannot reset the RNG to a constant state on each AES run. After this fix, we observe the correlations between attacker timings and keys disappear, and the guessing entropy (GE) for an unknown AES key is high (above 90), as shown in Fig. 4. This shows that MIRAGE, when correctly modeled, is is not vulnerable to occupancy attacks leaking AES keys.

Results with Correct Modeling. After randomizing the RNG seed for MIRAGE in each AES run, the victim key's guessing entropy (GE) remains high (>90), see Fig. 4. Thus, AES key leakage on MIRAGE is infeasible.

To summarize, we make the following contributions:

1) We observe that for probabilistic defenses relying on RNGs, the RNG seeding methodology during simulation strongly influences the measured security properties.
2) We identify that RCO's incorrect modeling of MIRAGE, using a fixed RNG seed for each AES simulation, creates artificial correlations between keys and cache occupancy, resulting in a false claim of successful occupancy attack.
3) With correct modeling of MIRAGE, by randomizing the initial RNG seed for each simulation, AES key guessing entropy remains high, showing that MIRAGE is secure against AES key leakage via occupancy-based attacks.

[1]Note that when running simulations on a system without a hardware-based entropy source, `std::random_device` can result in a deterministic output. It is best to validate that the source of randomness used is truly random.

II. Background

II.-A. The MIRAGE Cache

MIRAGE [1] is a randomized cache that prevents conflict-based side channels by emulating a fully-associative randomized cache: every eviction is global and selected uniformly at random from the entire cache. MIRAGE over-provisions invalid tags and uses Power-of-2-Choices load balancing to avoid set-associative evictions. On a miss, the data-store victim is chosen randomly from all cache lines, its tag is located via a Reverse Pointer (RPTR) and removed, and the new tag is inserted via a Forward Pointer (FPTR). With an 8-way cache augmented with 6 extra ways (75% extra) in the tag-store, MIRAGE guarantees the probability of a set-associative eviction is once in 10^{34} cache installs, an event that takes 10^{17} years to occur, making set-conflicts practically impossible in a system's lifetime and eliminating conflict-based attacks.

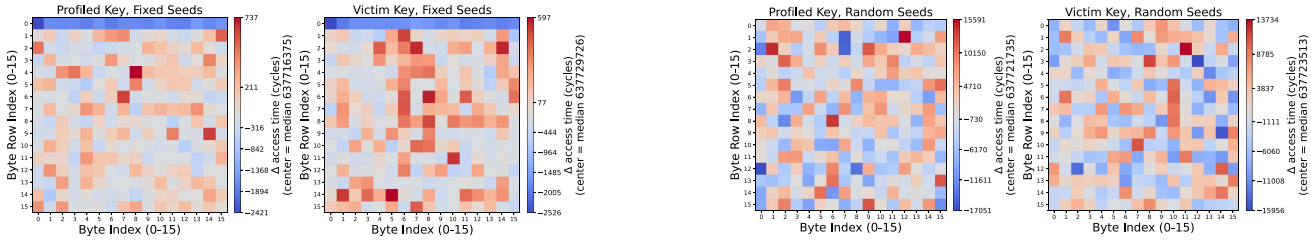
II.-B. Cache Occupancy Attacks on Mirage

Cache occupancy attacks measure changes in the overall occupancy of a shared cache, rather than targeting specific sets as in set-conflict attacks like Prime+Probe. Because they exploit aggregate cache usage, all randomized caches without explicit cache partitioning, such as CEASER, ScatterCache, and Mirage in principle, leak some information via cache occupancy. In fact, MIRAGE's threat model explicitly claims to not protect against such occupancy-based channels. While covert channels between two colluding processes can be naively constructed by modulating the cache occupancy, the RCO [4] paper claims the existence of a stronger side-channel on MIRAGE: leaking AES keys via cache occupancy effects.

The RCO paper claims that a victim using an AES T-Table implementation, can be forced to leak the key in MIRAGE, by a spy first priming the LLC to a chosen occupancy, letting the victim run one encryption, then timing accesses to the attacker's own cache lines. By correlating these timings for profiled keys and unknown victim keys, they report low guessing entropy for a victim AES key (lower than 30), and claim full 128-bit AES key recovery on MIRAGE within a few hours. This paper examines these claims of RCO.

III. Analyzing Claims of Occupancy-Based Side-Channel Attacks on MIRAGE

We investigate the claims made in RCO [4], using their publicly released artifact. To examine the root cause of the reported occupancy side-channel attack on MIRAGE, we first study the correlation of AES encryption keys and attacker-measured timings to access its cached array. We do so using a (1) constant RNG seed in MIRAGE, the seeding strategy in RCO, and (2) random RNG seed for each simulation, a more accurate model. We then study an end-to-end attack leaking an unknown AES key, with both RNG seeding strategies.



(a) Fixed Seed for Global Evictions (RCO [4])

(b) Random Seed for Global Evictions (our fix)

Fig. 2. Heatmap of access times for the attacker to iterate through its array. We bin the access times, based on the 256 possible T-table entries accessed in the last round. The 256 bins are represented in the 16 x 16 matrix. (a) With Fixed Seed for Global Eviction, as used in RCO [4], there is strong correlation between the heatmaps for profiled key and victim key. (b) After our fix, with Random Seeds for Global Eviction, the correlations disappear, showing it is infeasible to guess victim AES keys.

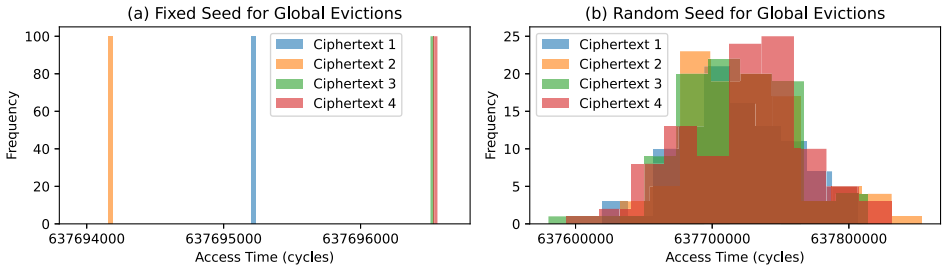


Fig. 3. Access Times for the attacker with (a) RCO's implementation and (b) our bug fix, for four plaintext-ciphertext pairs with the same key. (a) RCO's implementation uses a Fixed Seed (42) for Global Evictions in each AES encryption. (b) Our bug fix initializes the RNG for Global Evictions with a random seed, for each AES encryption, mimicking a real system where the seed is not a fixed value each time. After our fix, different encryptions are indistinguishable based on attacker access times.

III.-A. Analyzing RCO Attack's Root Cause

The RCO [4] paper claims that the T-Table implementation of AES running on MIRAGE [1] can leak the secret key through a cache occupancy attack. The root cause of the leakage claimed by RCO is that the sequence of T-Table accesses in the last round of AES encryption, which depends on the secret key, can impact the cache occupancy in MIRAGE. Therefore, the time for an attacker to access a cached array, parts of which may have been evicted by the AES encryption, can leak the cache occupancy, and therefore the secret key.

RCO Attack Mechanism: To perform this attack, RCO creates a template for the execution times with all possible T-table entries accessed in the last round (T_1 to T_{255}), by using a known key (*profiled key*). The attacker measures the execution time to access its own cached array that occupies 50% of the MIRAGE cache, using randomly generated plaintext-ciphertext pairs, and creates the template (T_1 to T_{255}) by averaging the access times for T_n= SBOX-Inv($K \oplus CT$), where *K* and *CT* are bytes of the last-round key and ciphertext. Such templates can be created for each of the bytes of the round-key (0 to 15). Later, for an unknown victim key, by creating a similar template using a guessed key and random plaintext-ciphertext pairs, the attacker identifies likely key values having the highest correlation with the profiled template.

Reproducing RCO's Root-Cause: To validate RCO's leakage, we try to reproduce its root cause, the templates with correlation between profiled and victim keys. Fig. 2(a) shows a heatmap visualizing the template built by the attacker, where each entry represents one of the 256 possible T-table entries (arranged as a 16×16 matrix) and the cell color encodes the attacker's average access time, when that entry is accessed in the last round of AES by the victim. If the profiled-key heatmap (attacker's template) closely match the heatmap with guessed victim-key, the guess is likely correct. For simplicity, instead of T_n = SBOX-INV($K \oplus CT$), we use $T_n = K \oplus CT$ for our bins, since SBOX-INV is a lookup table, and visualize a single heatmap, averaging heatmaps of key bytes 0 to 15.

As shown in Fig. 2(a), when we generate the heatmaps by using the RCO artifact [4] there is a correlation between templates of the profiled key and the guessed victim key. This allows the attacker to leak the victim key since templates are highly correlated only when the guessed key is correct.

III.-B. RCO's Bug: Fixed Sequence of Global Evictions.

To investigate the observed correlation, we measure attacker access times after victim AES encryptions for four randomly chosen plaintext-ciphertext pairs, each repeated 100 times. Each encryption has a distinct last-round AES T-Table access sequence. Fig. 3(a) shows the histogram of attacker access times. CT1, CT2 and CT3 differ in access times (although CT3 and CT4 overlap), highlighting that access times are correlated with T-Table accesses. However, notably, all 100 repetitions of each encryption produce identical access-time measurements, despite global evictions being intended as random. This reveals a bug in RCO: the global eviction RNG is seeded with a static value (42). As a result, each AES execution follows the same fixed eviction sequence, making MIRAGE's behavior deterministic. In a real implementation, the attacker cannot reset the RNG between runs, so the deterministic behavior is unrealistic and artificially creates correlations between access times and AES T-Table sequences.

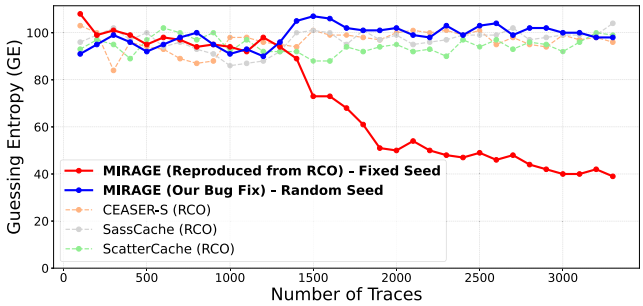


Fig. 4. Guessing Entropy (GE) for an unknown AES key, after our fix, as number of AES encryptions increases. RCO's implementation of MIRAGE (RNG with Fixed Seed), has GE below 40 after 1300 encryptions. MIRAGE modeled correctly (RNG with Random Seed), has GE above 90, with no leakage.

III.-C. Our Fix: Accurate Modeling With Random Seed.

The cache occupancy (O) in MIRAGE is a function of both the Victim accesses (V) and the global eviction decisions (E), i.e., $O \approx f(V, E)$. RCO incorrectly assumes a fixed sequence of E for each encryption, making the fingerprinting of V using O measurements possible. We fix this modeling bug in RCO, by instead initializing the global eviction RNG with a randomly chosen seed for each simulation, as shown in Fig. 1. This models a real system, where the attacker cannot reset the RNG to a fixed state before each AES encryption.

After this bug fix, we see O is strongly impacted by the changing E in each run, due to the varying RNG seed, in addition to V. As shown in Fig. 3(b), with a random RNG seed for each encryption, the attacker's access times show random variations of 100,000+ cycles, due to the global evictions unpredictably evicting the attacker's own lines during its measurements. These overwhelm victim-dependent occupancy changes causing 1000 s of cycles of variations in Fig. 3(a). As shown in Fig. 2(b) shows that after using a random RNG seed, correlations between profiled and victim keys disappear, making heatmaps virtually unrelated, eliminating the signal needed for key leakage. Thus, realistic modeling of MIRAGE's global evictions prevents AES key leakage via occupancy attacks.

III.-D. Results: Guessing AES Key After Correct Modeling

We evaluate the success of RCO's cache occupancy based attack leaking AES keys, with and without our bug fix. The figure of merit for the attack is the Guessing Entropy (GE) for a victim AES key using templates for a profiled key (higher GE is more secure). Like RCO, we use $GE = \sum_{i=0}^{15} log_2(R_i)$, where R_i is the rank of the correct guess for key byte i.

Fig. 4 shows the guessing entropy (GE) for Mirage based on RCO's artifact (reproduced from RCO), and with our bug fixes where we use a random seed to initialize the RNG for global evictions (our bug fix). After our bug fixes (random RNG seed), we see that **Mirage has high GE above 90**, even after thousands of AES encryptions, demonstrating that it is **resilient to brute-force key guessing attacks on AES** when modeled correctly. With RCO's implementation of MIRAGE (fixed RNG seed), the GE drops below 50 after 1900 AES encryptions. MIRAGE, when correctly modeled, has GE similar to other randomized caches like CEASER-S, SassCache, ScatterCache, which all have high GE (based on the data from RCO's artifact), demonstrating no key leakage.

IV. Concluding Remarks

Our analysis shows that the insecurity of MIRAGE reported in the RCO paper [4] is due to incorrect modeling: the global evictions are deterministic across multiple AES encryptions, which do not reflect MIRAGE's design. When modeled correctly, MIRAGE is secure against AES key leakage via occupancy attacks. Our code is open-sourced at https://github.com/sith-lab/yet-another-mirage-of-breaking-mirage.

Learnings: This study suggests that future works studying attacks on randomized caches in simulators should validate that their simulated models faithfully model the randomness properties relied on by the randomization-defenses for security.

Response: After analyzing our open-sourced code, the authors of the SoK paper [5] provided an *Errata* [6] with an updated Guessing Entropy (GE) analysis (Fig. 17). The result in their errata matches with our result in Fig. 4. The authors of RCO [4] provided an *Addendum* [7] showing that once the attack is correctly modeled (using a random RNG seed as we suggested), they cannot leak the AES key byte-by-byte through occupancy-based attacks. Their capability is limited to fingerprinting different keys, similar to the results shown by CacheFx [8] for several prior shared randomized caches (e.g., CEASER, CEASER-S, Scatter-Cache). This supports our claim in Section II-B that the security of MIRAGE against occupancy attacks is similar to other shared randomized caches and that MIRAGE does not worsen occupancy attacks.

Acknowledgment

The authors would like to thank Moinuddin Qureshi and Tom Ristenpart for their helpful feedback. All research, opinions, or positions expressed in this work are solely those of the authors and do not represent the official views of the NSERC, CSE or, the Government of Canada.

References

[1] G. Saileshwar and M. Qureshi, "MIRAGE: Mitigating conflict-based cache attacks with a practical fully-associative design," in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 1379–1396.

[2] A. Chakraborty, S. Bhattacharya, S. Saha, and D. Mukhopadhyay, "Are randomized caches truly random? Formal analysis of randomized-partitioned caches," in *Proc. IEEE Int. Symp. High-Perform. Comput. Architecture*, Montreal, QC, Canada, 2023, pp. 233–246.

[3] G. Saileshwar and M. Qureshi, "The mirage of breaking mirage: Analyzing the modeling pitfalls in emerging 'attacks' on mirage," *IEEE Comput. Archit. Lett.*, vol. 22, no. 2, pp. 121–124, Jul.–Dec. 2023 .

[4] A. Chakraborty, N. Mishra, S. Saha, S. Bhattacharya, and D. Mukhopadhyay, "Systematic evaluation of randomized cache designs against cache occupancy," in *Proc. 35th USENIX Secur. Symp.*, 2025, pp. 2499–2518.

[5] A. Bhatla, H. R. Bhavsar, S. Saha, and B. Panda, "Sok: So, you think you know all about secure randomized caches?," in *Proc. 34th USENIX Secur. Symp.*, 2025, pp. 2461–2480.

[6] B. Panda, "Errata to SoK: So, you think you know all about secure randomized caches?," 2025. Accessed: Oct. 31 2025. [Online]. Available: https://www.cse.iitb.ac.in/biswa/Errata-USEC25.pdf

[7] A. Chakraborty, N. Mishra, S. Saha, S. Bhattacharya, and D. Mukhopadhyay, "Addendum: Systematic evaluation of randomized cache designs against cache occupancy," 2025, *arXiv:2510.16871*.

[8] D. Genkin, W. Kosasih, F. Liu, A. Trikalinou, T. Unterluggauer, and Y. Yarom, "CacheFX: A framework for evaluating cache security," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, 2023, pp. 163–176.