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

Chris Cao
University of Toronto
chrisj.cao@mail.utoronto.ca

Gururaj Saileshwar
University of Toronto
gururaj@cs.toronto.edu

***Abstract*—Recent work presented at USENIX Security 2025 (SEC'25) claims that occupancy-based attacks can recover AES keys from the MIRAGE randomized cache. In this paper, we examine these claims and find that they arise from a modeling flaw in the SEC'25 paper. Most critically, the SEC'25 paper's simulation of MIRAGE uses a constant seed to initialize the random number generator used for global evictions in MIRAGE, causing every AES encryption they trace to evict the same deterministic sequence of cache lines. This artificially creates a highly repeatable timing pattern that is not representative of a realistic implementation of MIRAGE, where eviction sequences vary randomly between encryptions. When we instead randomize the eviction seed for each run, reflecting realistic operation, the correlation between AES T-table accesses and attacker runtimes disappears, and the attack fails. These findings show that the reported leakage is an artifact of incorrect modeling, and not an actual vulnerability in MIRAGE.**

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 [2] 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) [3] claimed to induce set-conflicts in MIRAGE, breaking its security guarantees. However, subsequent work [4] showed that these were the result of incorrect modeling by ARCTR, caused by a buggy cipher implementation, and that MIRAGE's security guarantees remained intact.

More recently, the SEC 2025 paper, "*Systematic Evaluation of Randomized Cache Designs against Cache Occupancy*" (RCO) [5], 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?*" [6] reiterates these claims in its Figure 17. This paper examines whether these claims hold up or whether they are artifacts of modeling flaws like the ARCTR paper.

First, we tried to reproduce the results of the RCO paper. Using the unmodified artifact released with the RCO paper[1], we attempted to reproduce their AES key recovery results, specifically the guessing entropy for an unknown victim AES key reported in Figure 10 of the RCO paper. [5]. Surprisingly, we were unable to reproduce the entropy degradation for MIRAGE as reported by them. Figure 1 shows the guessing entropy of an unknown AES key (higher is better), as reported by RCO for MIRAGE and other randomized cache designs, and our own reproduction using RCO's artifact.

As shown in Figure 1, the RCO paper reports the guessing entropy to decrease for MIRAGE as the number of AES encryptions increases. However, our reproduction[2] of their result shows that MIRAGE has high entropy (above 90%) even after thousand AES encryptions, similar to other randomized cache designs such as CEASER-S, SassCache, and ScatterCache.

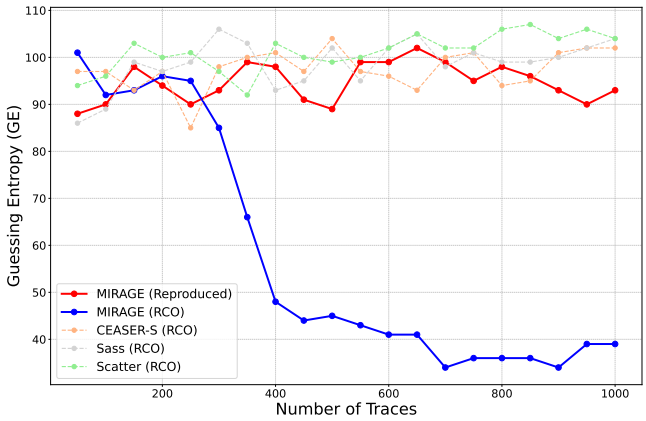


Fig. 1: Guessing Entropy (GE) for an unknown AES key, as the number of AES encryptions increases, for our reproduction of MIRAGE from RCO's artifact compared to results from the RCO paper for MIRAGE, CEASER-S, SassCache, ScatterCache. In our reproduction, MIRAGE has GE of more than 90%, showing no leakage, departing from the RCO paper.

Next, given that the entropy results in RCO could not be reproduced, we examine whether the stated root cause of the reported leakage is valid. RCO attributes the leakage to the accesses to the AES S-Box in the last round of AES encryption

[1]We use the the code artifact released with RCO's paper [5] - https://zenodo.org/records/14869981 which corresponds to the latest release on their Github repository, v2.0 (commit: fc07ea6), at the time of writing.

[2]Our reproduction code and scripts for all the figures in this paper is open-sourced at https://github.com/sith-lab/yet-another-mirage-of-breaking-mirage.

and its impact on cache occupancy, and its correlation with an attacker's access time for its own cached array. By creating access time templates for each possible key-byte value of a profiled key, the attacker matches the observed timings for an unknown victim key against the templates to recover the key. However, this fails to explain how the attack works around the noise introduced by random global evictions in MIRAGE.

Effect of Random Global Evictions. MIRAGE randomly evicts an existing LLC line upon every new insertion. This means that even repeated AES encryptions of the same plaintext and key would produce different cache occupancy patterns, due to the different eviction patterns, unrelated to the key. Moreover, an attacker accessing its own array to estimate the cache occupancy of the victim, would see further noise due to its own addresses getting randomly evicted differently on each run. Thus, it seems difficult for such an attacker to even fingerprint a particular key, let alone recover the entire key byte by byte under random global evictions. In investigating this, we uncover a modeling bug in RCO's evaluation.

Modeling Flaw in RCO. RCO's simulations initialize MIRAGE's global eviction RNG with a **static seed** before each AES encryption, causing a **fixed** sequence of evictions on each AES encryption, unlike real hardware.

RCO runs each AES encryption in a separate simulation starting from a clean cache state. In this setup, the final cache occupancy is determined by both the victim's accesses and the global eviction pattern. However, with a fixed RNG seed, the eviction pattern never changes, making occupancy entirely a function of the victim's access pattern and spuriously creating correlations with the key. If we randomize the seed for each run (e.g., using `std::random_device`[3] to seed the RNG), the eviction pattern changes on each run, and the correlation between the key and attacker timings disappears, as expected in MIRAGE. This models a realistic setting, where the attacker cannot reset or guess the state of the RNG for the evictions.

Results with Correct Modeling. In evaluations of an unknown victim key's guessing entropy (GE), after randomizing the RNG seed for global evictions in MIRAGE, the GE remains high (>90%), highlighting that AES key leakage on MIRAGE via occupancy attacks is infeasible.

To summarize, we make the following contributions:

1) We show that the AES key guessing entropy remains high in MIRAGE, in a reproduction of RCO's artifact, contradicting the paper's claims.
2) We identify that RCO's fixed RNG seeds for global evictions in MIRAGE create artificial correlations, and randomizing the seed, as in real hardware, eliminates any observed correlation and leakage.

[3]Note that on some systems without a hardware-based entropy source, `std::random_device` can also result in a deterministic output [7]. Please validate on your system if the source of randomness you use is truly random.

II. Background

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. To support this, MIRAGE over-provisions invalid tags in each set and uses load-balancing via the Power-of-2-Choices to maintain available space, avoiding 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.

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 [5] paper claims a stronger side-channel: recovering AES keys through modulation of cache occupancy.

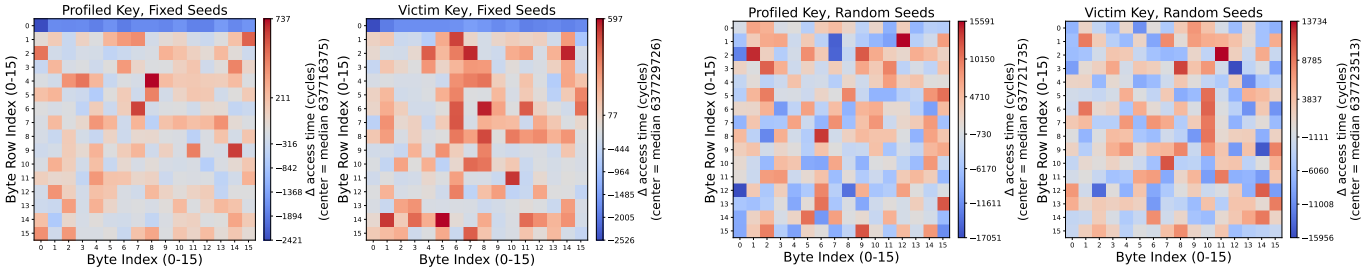
The RCO paper claims that a victim using an AES T-Table implementation, can be forced to leak the key in MIRAGE, by a spy first priming the LLC to a chosen occupancy, letting the victim run one encryption, then timing accesses to the attacker's own cache lines. By correlating these timings with last-round T-table accesses for guessed keys, they report low guessing entropy for AES keys (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

Using the authors' publicly released artifact, we attempted to reproduce the results in Figure 10 of the RCO paper. However, contrary to their claims, we found that the guessing entropy for an unknown AES key remained high and did not drop even after thousands of encryptions in our reproduction, as shown in Figure 1. Therefore, we analyze the claimed root cause of the attack: that the execution time for an attacker accessing its own array depends on the cache occupancy of the AES encryption, which is a function of the AES key.

A. Pitfall-1: Fixed Sequence of Evictions Modeled by RCO

RCO Attack Root Cause. The RCO [5] paper claims that the T-Table implementation of AES running on MIRAGE [1]



(a) Fixed Seed for Global Evictions (RCO [5])

(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 [5], 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.

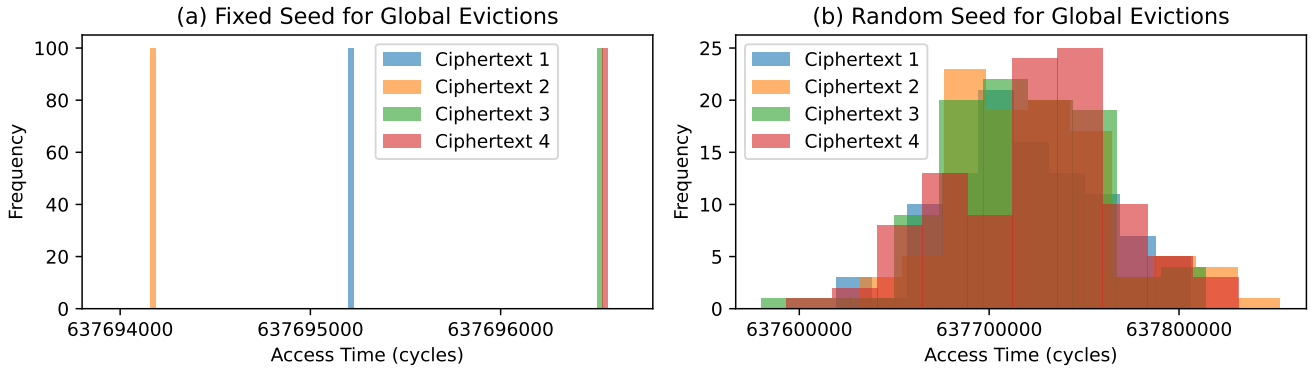


Fig. 3: Access Times for the attacker with (a) original RCO implementation and (b) our bug fix for four sample plaintext-ciphertext pairs with the same key. (a) The original implementation uses a Fixed Seed (42) for Global Evictions in each AES encryption. (b) Our bug fix initializes the RNG used for Global Evictions, with a random seed (e.g., using `time()`), for each AES encryption, mimicking a real system where the seed cannot be reset to a static value each time. Once we address this issue, the different encryptions are indistinguishable based on access times to the attacker's array.

can leak the secret key through a cache occupancy attack. The root cause of this leakage as mentioned 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 execution time for an attacker to access a large 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 [5] 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.

Our Reproduction of RCO's Root Cause. Since we are unable to reproduce the exact Guessing Entropy results from RCO (see Figure 1) , we try to reproduce their root cause, the template with correlation between profiled and victim keys. Figure 2a 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 matches the heatmap with the guessed victim-key, the guess is likely to be the correct key. 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 just a lookup table, and visualize a single heatmap, averaging the heatmaps of key bytes 0 to 15.

As shown in Figure 2(a), when we generate the heatmaps by using the RCO artifact [5] there is a clear correlation between the templates of the profiled key and the guessed victim key. This correlation can allow the attacker to leak the victim key since the templates will be correlated only when the guessed key is actually the correct victim key.

RCO's Bug: Fixed Sequence of Global Evictions. To understand why this correlation exists, we measured the attacker access times, after victim AES encryptions, for four sample plaintext-ciphertexts pairs chosen at random and repeated 100 times. Each of these encryptions has a distinct AES T-Table access sequence in the last round. Figure 3(a) shows the histogram of the access-times measured for these four encryptions: each of these (CT 1, 2, 3) has a different access time, although CT 3 and 4 overlap with the same access time. This explains the correlations of access-times with AES T-table access sequences. Surprisingly, all 100 repetitions yield the exact same measurement for the attacker access time, although global evictions during the attacker accesses are supposed to be random. This allowed us to identify the bug in the RCO implementation, that the Global Evictions used a random-number generator (RNG) seeded with a static seed (42). Hence, in each AES execution, the same sequence of cache indices is selected for global eviction, making the operation of MIRAGE quite deterministic in each AES encryption, which is quite unrealistic in real implementations, where the attacker has no way to reset the RNG to a fixed state each time.

Our Fix: Accurate Modeling with Random Seed. The cache occupancy (O) is a function of both the Victim accesses (V) and the Global Eviction decisions (GE), i.e., $O \approx f(V, GE)$. RCO incorrectly assumes a fixed sequence of GE for each encryption, making the fingerprinting of V using O measurements possible. Fixing the bug in RCO, by changing the fixed seed for global evictions to a randomly chosen seed for each AES encryption, representative of real systems, where the Global Eviction decisions would be performed by a RNG whose state cannot be reset by the attacker, we see that the CO is now strongly impacted by the GE in addition to the V. As shown in Figure 3(b), with a random seed for each encryption, the attacker's access times show random variations of the order of 100,000 cycles, due to the global evictions unpredictably evicting the attacker's own lines during its measurement phase, that overwhelm minor occupancy differences causing by the victim which varied timings by a few 1000 cycles in Figure 3(a).

After our fix, when we replace the fixed seed with a random seed for each AES encryption in Figure 2(b), the correlation between profiled and victim keys disappears, making the heatmaps virtually unrelated and eliminating the signal required for key recovery. This realistic modeling of MIRAGE's global evictions removes the attack's timing signal, preventing AES key leakage on MIRAGE via occupancy attacks.

B. Pitfall-2: Unrealistic L1 Cache Configuration

The RCO paper also claims to use a 512 **kilobyte** L1 cache, as per Section 7.1 of their paper [5]. However, their code artifact[4] uses a 512 **byte** L1 cache by default which is not representative of real systems which use at least a 64KB L1 Cache. Modeling a smaller L1 cache, such as 512 byte L1 cache, can inflate the L1 cache misses and LLC accesses, compared to a 64KB L1 Cache which can have hits for *all* T-table accesses after the first round. Thus, modeling a 512 byte L1 cache can overestimate the attack. We confirmed that after updating the initialized seed to be random, regardless of the L1 cache size being 512-byte like in the RCO artifact or a more realistic 64KB, there is a lack of correlation between the heatmaps of the profiled and victim key templates, similar to Figure 2 (b), indicating that AES key leakage is impractical.

[4]We refer to the code artifact version 2.0 released by the authors on Zenodo - https://zenodo.org/records/14869981

C. Guessing AES Key after Fixing RCO's Modeling Issues

We evaluate the Guessing Entropy (GE) for a victim's AES key, using the template for a profiled AES key similar to RCO, using the formula: $GE = \sum_{i=0}^{15} log_2(R_i)$, where R_i is the rank of the correct guess for key byte i. Figure 4 shows the guessing entropy (GE) for Mirage from the RCO paper, our reproduction based on their artifact, and after our fixes of RCO's modeling issues (after using a random seed for initializing the RNG for global evictions). In our reproductions (both before and after our fixes), we continue to see Mirage with high GE, demonstrating that it is resilient to brute-force key guessing attacks on AES.

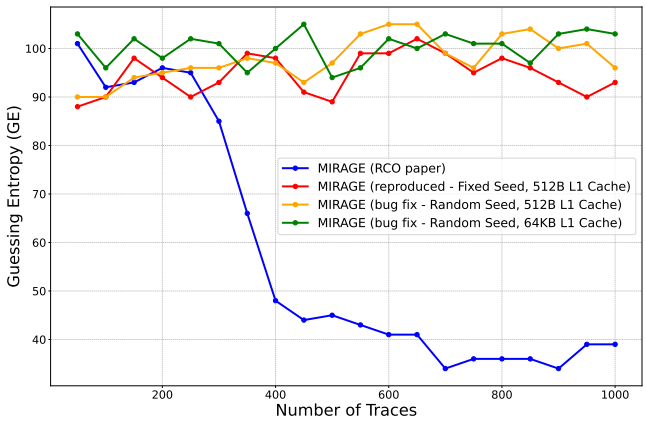


Fig. 4: Guessing Entropy (GE) for an unknown AES key, after our fixes, as the number of AES encryptions increases. MIRAGE continues to have a high GE of more than 90%, showing no leakage, with Random Seed used to initialize the RNG for Global Evictions, with both an unrealistic 512B L1 Cache (like RCO's artifact) and a realistic 64KB L1 Cache.

IV. Concluding Remarks

Our analysis shows that, when modeled faithfully, MIRAGE remains resilient to occupancy-based side-channel attacks aiming to recover AES keys. The results reported in the RCO paper [5] arise from unrealistic modeling, most notably the use of a deterministic sequence of global evictions, across multiple AES encryptions, which do not reflect MIRAGE's design. We encourage the authors of the RCO paper [5] and the SoK paper on randomized caches [6] to revisit their conclusions in light of these findings. Our code is open-sourced at https://github.com/sith-lab/yet-another-mirage-of-breaking-mirage.

V. Acknowledgment

We thank Moinuddin Qureshi and Tom Ristenpart for their helpful feedback on an earlier draft of this work.

REFERENCES

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

[2] M. Mitzenmacher, "The power of two choices in randomized load balancing," *IEEE Trans. on Parallel and Distributed Systems*, 2001.

[3] 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

[4] G. Saileshwar and M. Qureshi, "The mirage of breaking mirage: Analyzing the modeling pitfalls in emerging "attacks" on mirage," *IEEE Computer Architecture Letters*, vol. 22, no. 2, pp. 121–124, 2023.

[5] A. Chakraborty, N. Mishra, S. Saha, S. Bhattacharya, and D. Mukhopadhyay, "Systematic evaluation of randomized cache designs against cache occupancy," in *35th USENIX Security Symposium (USENIX Security 25)*, 2025.

[6] A. Bhatla, H. R. Bhavsar, S. Saha, and B. Panda, "Sok: So, you think you know all about secure randomized caches?" in *35th USENIX Security Symposium (USENIX Security 25)*, 2025.

[7] cppreference.com, "Random device," https://en.cppreference.com/w/cpp/numeric/random/random_device.html, 2025.