Addendum: Systematic Evaluation of Randomized Cache Designs against Cache Occupancy

Presented at USENIX Security 2025

Anirban Chakraborty[†] Nimish Mishra[‡] Sayandeep Saha[§]
Sarani Bhattacharya[‡] Debdeep Mukhopadhyay[‡]

[†]Max Planck Institute for Security and Privacy, Germany
[‡]Indian Institute of Technology Kharagpur, India
[§]Indian Institute of Technology Bombay, India

anirban.chakraborty@mpi-sp.org nimish.mishra@kgpian.iitkgp.ac.in {sarani, debdeep}@cse.iitkgp.ac.in sayandeepsaha@cse.iitb.ac.in

In the main text published at USENIX Security 2025 [3], we presented a systematic analysis of the role of cache occupancy in the design considerations for randomized caches (from the perspectives of performance and security). On the performance front, we presented a uniform benchmarking strategy that allows for a fair comparison among different randomized cache designs. Likewise, from the security perspective, we presented three threat assumptions: (1) covert channels; (2) process fingerprinting side-channel; and (3) AES key recovery. The main takeaway of our work is an open problem of designing a randomized cache of comparable efficiency with modern set-associative LLCs, while still resisting both contention-based and occupancy-based attacks.

This note is meant as an addendum to the main text in light of the observations made in [2]. To summarize, the authors in [2] argue that (1) L1d cache size plays a role in adversarial success; and that (2) a patched version of MIRAGE with randomized initial seeding of global eviction map prevents leakage of AES key. We discuss the same in this addendum.

1 Sources Used, Attack Reproducibility, and Number of Traces needed in [3]

In [3], we used the sources of MIRAGE available to us at the time of writing[1] to develop a downstream artifact for our evaluation[2]. We maintain that the AES key recovery described in [3] is reproducible on those original sources, and has also been likewise reproduced in revision 3 of [2] (in version 1, the authors reported the inability to reproduce our results out-of-the-box).

Moreover, the exact number of traces needed to observe a drop in Guessing Entropy will differ across runs of the attack. This is an expected consequence of using a statistical attack, as we do in [3]. Limited number of side-channel traces (as is the case with any real world attack) approximate the actual underlying distribution to different statistical distances (more traces imply better approximation), and thus these approximations vary across repeated runs of the attack. Therefore, while the *exact* number of traces needed to achieve the guessing entropy of 32 (or below) will vary, we maintain the *overall general trend* of reduction in guessing entropy for MIRAGE remains intact when the attack is executed on the original sources available to us at the time of writing.

2 Role of L1d cache

A point raised in [2] is the role of L1d in the attack: an implication that the original attack in [3] was successful due to its unrealistic setting of L1d cache size, and that using a larger L1d cache size (ex. 64 KB) masks the leakage. We first note the role the L1d cache plays in the attack and then provide clarifications on the chosen L1d size. Since the L1 instruction cache is separate from L1d, it bears no consequence to the attack; we thus omit it in our discussion here. We begin by first recalling the attack overview from Sec.7.1 in [3]:

1. The attacker first lets the AES victim setup its secret key, and precompute T-Tables.
2. Before the experiment begins, the attacker fills the LLC with spurious occupancy. These memory accesses are *never* re-accessed again during the course of the actual attack. This step also ensures that the AES victim T-Tables are reliably flushed from the LLC.
3. Given a fixed occupancy $X\%$, the attacker `malloc`s about $\frac{16\times X}{100}$ MB of memory, and repeatedly accesses it to ensure occupancy of complete L1d and $X\%$ of LLC, since

[1] https://github.com/gururaj-s/mirage; commit: 2c763da
[2] https://github.com/SEAL-IIT-KGP/randomized_caches

LLC is inclusive cache[3].

4. Given a randomly generated plaintext P, the attacker then lets AES victim run a *single* encryption of P, and obtains the ciphertext C.
5. Finally, the attacker uses `rdtsc` to time access to its previously allocated $\frac{16\times X}{100}$ MB of memory. Call it T.

Note here that in both step (2), the adversary needs to cover the entirety of L1d cache. This is intentional: in step (1), when the T-tables are initialized, they reside in the L1d cache. If the attack is carried out without flushing the T-table (i.e. execute step (3) directly after step (1), skipping step (2)), the **attack would still work**, but the leakage source would then be contention in the L1d cache, and not the LLC. Conclusions from such an attack would be of no consequence to the LLC, which is where the randomized mappings studied in [3] function.

It is therefore imperative that the tables are flushed from the cache hierarchy post initialization to allow us to study effects of LLC cache line installs on different randomized cache designs. Below is detailed a lifecycle of a single T-table entry (abbr. **t**) throughout the different stages of our attack (extension to the entire T-table is straightforward):

- Entry **t** is initialized by the AES victim. Since this initialization uses `load` and `store` instructions, the entry **t** occupies a cache line in the L1d cache.
- Since we do not assume adversarial capability to use ISA (like `clflush` on `x86` ISA) to flush **t**, we flush the *entire* L1d and the *entire* LLC to ensure **t** is reliably flushed from the LLC.
- When the victim makes the first T-table access **t** during AES encryption, because of Step 2, we are guaranteed to have a *cache line install* in the LLC. For the specific case of MIRAGE, we are guaranteed to have a global eviction with all but negligible probability. Post the cache line install in LLC, **t** is installed in the L1d, and utilized by the AES victim process.

In order to study contention in the presence of adversarial cache occupancy in the LLC, the event of interest (and subsequent leakage source exploited in [3]) is the event of cache line install in the LLC. **There is no role of L1d in the attack, since the exploited leakage source across different randomized cache designs is the manner of cache line installs in the LLC, and effect of such LLC line installs on adversarial occupancy of the LLC**. The fact that such LLC line installs transtively have L1d line installs is a consequence of an inclusive cache hierarchy, and was not the source of the leakage exploited in [3].

[3]Most cache designs (including the ones we evaluate here) consider an inclusive cache-hierarchy.

2.1 Choice of L1d size in [3]

The choice of L1d size in [3] was thereby driven by the need to *minimize the gem5 simulation time*, in order to not spend simulation cycles on a micro-architectural element (i.e. L1d cache) not being targeted by the attack. Tab. 1 gives a comparative of the gem5 simulation ticks; observe the additional work *per* AES execution being performed by the simulator, which scales linearly with the attack complexity *per* cache design. We stress that such a consideration on simulation time is solely for the platform available to us for testing (gem5) and does not apply to a (future) implementation of randomized caches on real hardware, as also acknowledged in the artifact appendix accompanying [3]:

We encourage the user to try out data collection with different keys to get a trend (and GE) closer to what is reported in the paper. However, as also noted in the paper (footnote 21*), the rate of data collection is at best* 500 *observations per hour. We were able to thereby deploy about* 350 *cores per Intel Xeon server, across three such servers. The overall data collection for all designs and multiple keys took over 2 weeks of compute hours. [...]*

Note that such an inhibitory rate is not a problem of attack design, but rather is the consequence of the gem5 simulations (which is the go-to simulation strategy for state-of-the-art randomized cache literature). In a realistic setting (when these randomized caches are deployed on real hardware), our attack will be much faster.

In Fig. 1, we compare the Kernel Density Estimators (KDEs) of the adversarial measurements of its own occupancy (Step 5 in the attack overview detailed earlier this section) for the considered L1d cache size in [3] vs the size used in [2]. Notice both distributions have the same gaussian envelope with different means, with the larger L1d cache having an expected lower mean. In other words, the statistical attack in [3] exploits the difference in the (gaussian) distribution approximation between two attack runs (first with a profiled key; second with the victim key). Changing the L1d cache relatively displaces *both* distributions in the X-axis (wrt. adversarial timing measurement) but does not distort the gaussian envelope itself, as clear from Fig. 1.

Finally, on the original MIRAGE sources considered in [3] with the L1d cache size considered in [2], *we were able to observe AES key leakage*. **We maintain our position thereby: on the sources available to us at the time of writing, increasing L1d cache size does not prevent the leakage introduced by global evictions during LLC line installs.**

3 Role of seeding global eviction mapping

Another point raised in [2] was the release of a new patch which randomizes the seeding of the global eviction mapping, hereafter abbreviated as MIRAGE+ to indicate (1) the original

MIRAGE Configuration	L1d size	Simulation ticks
Original sources	Original Paper	837793997703
Original sources	Same as [2]	881671966812

Table 1: Number of recorded gem5 ticks for a single AES execution.

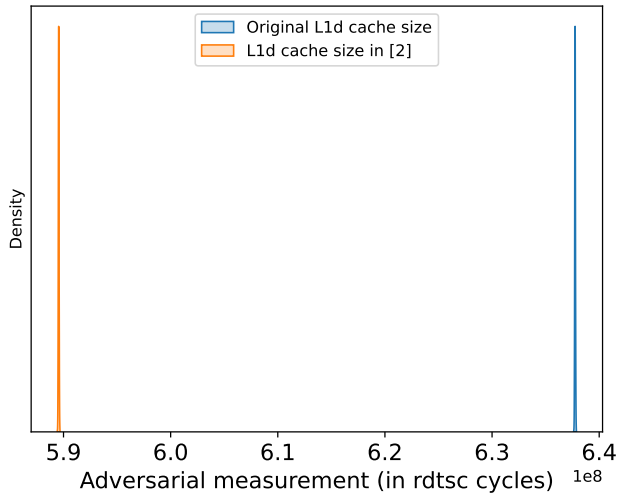

Figure 1: Kernel Density Estimators for (1) the original L1d cache size considered in [3]; vs (2) the L1d cache size in [2]. The shape assumed by both distribution is gaussian.

sources available to us at the time of writing with (2) L1d cache size in [2], plus the (3) additional patch of randomizing the global eviction mapping. The observation in [2] was that MIRAGE+ prevents the AES leakage exploited in [3].

On our end, we built MIRAGE+ from source[4], and applied the provided patches as recommended in [2]. As a first level of analysis, we chose to perform the following experiment:

- **Run 1**: Fix plaintext input `0xa7d960e3eac4b884fdcde51438edb007` borrowed from [2] and AES key `0x7766554433221100ffeeddccbbaa9988` (originally considered in [3]). Collect traces using the script `run_mirage_fixedCT.sh` provided in [2]. Call these traces T_1.
- **Run 2:** Similar as run 1, except the victim key is now changed to `0xffeeddccbbaa99887766554433221100` (originally considered in [3]). Call such traces T_2.
- **Test:** Apply the standard minimum-maximum scaler[5] on the data, and perform Welch's T-test [1] on T_1 and T_2 to determine whether the distributions assumed by the two runs are significantly different.

This experiment serves as a first level analysis of whether MIRAGE+ leaks, with such testing having been a literature standard for side-channel leakage evaluations. In case the test fails to provide evidence of a leakage, further evaluations (like the statistical attack in [3]) is fruitless effort.

Our test results gave a **T-statistic = 7.1269; pvalue = 1.42e-12**. A negligibly small p-value indicates that, if the null hypothesis were true, the likelihood of observing such a difference is negligible low; and an above-threshold T-statistic establishes leakage. In simpler terms, the interpretation of the test is that T_1 and T_2 are statistically different. Recall that the *only* difference between T_1 and T_2 is the secret AES key, further solidifying that MIRAGE+ leaks statistically by virtue of using different AES keys. A full key recovery attack will follow similar lifecycle as in [3], with the additional recommendation of including (1) *noise averaging*[6]; and (2) gaussian tail-cutting, to offset the `rdtsc` cycle variations observed in [2].

Therefore, **we maintain our position that, as noted in an observation in [2], (1) increased L1d cache size; and (2) randomizing global eviction mapping, are *not* self-sufficient to *prevent* AES leakage** (see next section on our takeaway from this). At the same time, we note that an investigation of the attack parameters of full key recovery on a design unavailable to us at the time of writing [3] is beyond the scope of this addendum, and is left for a follow-up work.

4 Discussion and Takeaways

From a broader perspective, the initial exploit in [3] and the leakage in MIRAGE+ is the consequence of a fundamental facet of (randomized) cache design: allowing two processes to *contend* for the same hardware. As such, while the exact attack parameters differ, we find it unsurprising that *contention* leads to *observable leakage* dependent on secret cryptographic material.

We reassert our conclusions from [3]: the way forward to prevent a side-channel attack as ours is to avoid the fundamental leakage source itself- contention in LLC. As noted in the main paper too, SassCache [4] does this through its domain isolation, and we observed it to exhibit the highest resilience to all our attacks. Although compartmentalization nevertheless incurs performance overheads, we believe it is the correct strategy from a security perspective since it aims to eliminate the root cause of LLC contention itself, which other designs included in [3] (as well as MIRAGE+ as discussed in this addendum) fundamentally do not allow.

[4] github.com/sith-lab/yet-another-mirage-of-breaking-mirage

[5] https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html

[6] See [5] for examples where noise averaging improves Signal-to-Noise ratio (SNR) and reduces side-channel attack complexity in terms of number of traces needed.

References

[1] G Becker, J Cooper, E De Mulder, G Goodwill, J Jaffe, G Kenworthy, et al. Test vector leakage assessment (tvla) derived test requirements (dtr) with aes. In *International Cryptographic Module Conference*, 2013.

[2] Chris Cao and Gururaj Saileshwar. Yet another mirage of breaking mirage: Debunking occupancy-based side-channel attacks on fully associative randomized caches. *arXiv preprint arXiv:2508.10431*, 2025.

[3] Anirban Chakraborty, Nimish Mishra, Sayandeep Saha, Sarani Bhattacharya, and Debdeep Mukhopadhyay. Systematic evaluation of randomized cache designs against cache occupancy. In *35th USENIX Security Symposium (USENIX Security 25)*, 2025.

[4] Lukas Giner, Stefan Steinegger, Antoon Purnal, Maria Eichlseder, Thomas Unterluggauer, Stefan Mangard, and Daniel Gruss. Scatter and split securely: Defeating cache contention and occupancy attacks. In *2023 IEEE Symposium on Security and Privacy (S&P)*, pages 1101–1115. IEEE Computer Society, 2022.

[5] François-Xavier Standaert. How (not) to use welch's t-test in side-channel security evaluations. In *International conference on smart card research and advanced applications*, pages 65–79. Springer, 2018.