USENIX Security '25 Artifact Appendix: SoK: So, You Think You Know All About Secure Randomized Caches?

Anubhav Bhatla*, Hari Rohit Bhavsar*, Sayandeep Saha, Biswabandan Panda
Indian Institute of Technology Bombay

A Artifact Appendix

A.1 Abstract

The artifact is used to evaluate the security of various cache designs discussed in the paper. We provide the source files and scripts required to reproduce all security results (Figures 3–19 and Table 2). A comprehensive README file is also included to facilitate easy reproduction of the results.

A.2 Description & Requirements

This section lists all the hardware and software dependencies required for the experimental setup to run all security simulations. Given below is a description of the important directories and scripts provided in the artifact:

- **cache-model:** An extended version of the behavioral cache simulation model. This model is used to generate results for Figures 3–14, 18–19, and Table 2.
- **cachefx:** An extended version of the CacheFX simulator used to generate results for Figures 15–16.
- **low-occupancy:** An extended simulation model for our low-occupancy-based attack simulations. This model is used to generate results for Figure 17.
- **`buildAll.sh`:** Script used to build all source files for the cache model, CacheFX, and low-occupancy simulators.
- **`genAllFigs.sh`:** Script used to run all experiments and generate PDF files for Figures 3–19.
- **`genTable.sh`:** Script used to run experiments and generate Table 2.
- **`requirements.txt`:** A list of the Python libraries needed for the smooth functioning of all scripts.
- **`README.md`:** Provides a description of the directories and detailed steps to build projects and run experiments to reproduce results from this work.

*These authors contributed equally to this work

A.2.1 Security, privacy, and ethical concerns

Our study does not introduce any novel or previously unknown attacks. Instead, it focuses exclusively on attacks that have already been disclosed and documented in prior work. All the attacks discussed in our evaluation are well-established and widely recognized within the community. The experiments were carried out using software-based, open-source simulators to replicate the required conditions and analyze system behaviors. No real-world systems were accessed, and no vulnerabilities were discovered, exploited, or tested on live or operational systems. Therefore, there are no ethical concerns in this work.

A.2.2 How to access

Our artifact is available through Zenodo. The artifact can be accessed at `https://doi.org/10.5281/zenodo.15529618`.

A.2.3 Hardware dependencies

We implemented and evaluated our artifact on a server with the Intel Xeon Gold 6342 CPU, 128GB of RAM, and 4TB of disk storage. To effectively parallelize simulations and ensure timely completion, we recommend utilizing a large number of CPU cores, ideally distributed across multiple servers.

A.2.4 Software dependencies

We test our artifact on a system running Ubuntu 20.04.6. However, any operating system which supports C++ and Python is sufficient to run our experiments. The Python packages required have been listed in the `requirements.txt` file. We also require docker and the C++ boost library for our simulations. The instructions to install this library and the Python libraries can be found in the `README.md` file.

A.2.5 Benchmarks

Our experiments evaluate the eviction rate of cache designs, number of LLC evictions required for eviction-set-generation, and the guessing entropy for key recovery. These experiments are self-sufficient and do not require any benchmarks for evaluation.

A.3 Set-up

A.3.1 Installation

We require that python, pip3, and docker are already installed on the system. For installing required Python libraries, run:

```
$ pip3 install -r requirements.txt
```

We also require the C++ boost library, installed using:

```
$ sudo apt install libboost-all-dev
```

The source files can be compiled using the following:

```
$ bash buildAll.sh build
```

A.3.2 Basic Test

An easy way to check if the simulation set-up is complete is to generate all figures and the table using our existing results. These can be generated using:

```
$ bash genAllFigs.sh 1
$ bash genTable.sh 1
```

If the set-up is complete, 17 PDF files will appear, named `figure`*x*`.pdf`, where *x* ranges from 3 to 19. Table 2 will also be printed on the terminal.
We recommend removing these figures before moving forward, especially if one aims to reproduce all the security results from scratch. Use the following command:

```
$ rm *.pdf
```

A.4 Evaluation workflow

We provide flexibility in how one can reproduce our security results. A single script can be used to run experiments for all security figures (Figures 3–19) at once. We also provide the option to use our existing simulation results to generate the figures without a fresh run:

```
$ bash genAllFigs.sh 1
```

In order to generate all figures using a fresh simulations, run:

```
$ bash genAllFigs.sh 0
```

Similarly, to generate Table-2 using existing results, run:

```
$ bash genTable.sh 1
```

A fresh set of simulations for Table-2 can be run using:

```
$ bash genTable.sh 0
```

All scripts that will be discussed later have the same option of using either our existing results (set option to `1`) or running a fresh set of simulations (set option to `0`). We assume that the user wants a fresh set of simulations, and set this option to `0` by default in Section A.4.2.

A.4.1 Major Claims

(C1): Increasing the number of skews helps reduce the eviction rate. Additionally, load-aware skew selection improves eviction rate. These are validated by **E1** described in Section 3.2, with results in Figures 3 and 4. We later talk about how this depends on the warm-up state in **Ex**.

(C2): Only the combination of skews with load-aware skew selection, invalid tags and global LRU eviction provides a security benefit. Additionally, increasing the invalid tags improves the security of such cache designs. These are validated by **E2** described in Section 3.3, with results in Figures 5, 6 and 7.

(C3): High associativity provides significant security benefits. This is validated by **E3** described in Section 3.4, with results in Figures 8 and 9.

(C4): Random replacement policy performs worse than LRU and RRIP for conflict-based attacks, as it results in higher eviction rates. This is validated by **E4** described in Section 3.5, with results in Figures 10 and 11.

(C5): High associativity designs require significantly more LLC evictions to create eviction sets that achieve a 30% eviction rate. Adding the knobs of load-aware skew selection, invalid tags, and global eviction further improves the security. These are validated by **E5** described in Section 3.6, with results in Figure 12.

(C6): The conflict testing algorithm is much faster than Prime, Prune and Probe in terms of number of LLC evictions needed for same-sized eviction sets. This is validated by **E6** described in Section 3.6, with results in Table 2.

(C7): Eviction rate is independent of the cache size (assuming same associativity). However, the number of LLC evictions required to create eviction sets that achieve a 30% eviction rate increase with cache size. These are validated by **E7** described in Section 3.7, with results in Figures 13 and 14.

(C8): SassCache and way-based partitioned cache designs perform significantly better than other randomized cache designs against occupancy-based attacks. Also, a random replacement policy is better suited against occupancy-based attacks compared to a deterministic replacement policy. These are validated by **E8** described in Section 4, with results in Figures 15 and 16.

(C9): Mirage is vulnerable to low-occupancy-based attacks compared to other randomized cache designs discussed in this work. This is validated by **E9** described in Section 4, with results in Figure 17.

(C10): The security benefit of designs using load-aware skew selection has a strong dependence on the cache warm-up state. This is validated by **E10** described in Appendix B, with results in Figures 18 and 19.

A.4.2 Experiments

(E1): [Impact of skews and skew selection policy] [48 compute-hour]: This experiment analyzes the impact of skews and the skew-selection policy on cache security.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 3
$ python3 get-figure.py 0 4
```

Results: `figure3.pdf` and `figure4.pdf` files are generated.

(E2): [Combinations of skew selection policies, invalid tags, and eviction policy] [72 compute-hour]: This experiment analyzes the security of various knob combinations including skews with random and load-aware skew selection, invalid tags, and local and global eviction.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 5
$ python3 get-figure.py 0 6
$ python3 get-figure.py 0 7
```

Results: `figure5.pdf`, `figure6.pdf` and `figure7.pdf` files are generated.

(E3): [Impact of high associativity] [48 compute-hour]: This experiment analyzes the security impact of high associativity on designs with just two skews, and also invalid-way-based designs.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 8
$ python3 get-figure.py 0 9
```

Results: `figure8.pdf` and `figure9.pdf` files are generated.

(E4): [Impact of replacement policy] [48 compute-hour]: This experiment analyzes the security impact of the replacement policy on designs with just two skews and high associativity, and also invalid-way-based designs.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 10
$ python3 get-figure.py 0 11
```

Results: `figure10.pdf` and `figure11.pdf` files are generated.

(E5): [Number of LLC evictions required for eviction-set-creation] [96 compute-hour]: This experiment evaluates the number of LLC evictions, required to create an eviction set achieving 30% eviction rate, on various designs.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 12
```

Results: `figure12.pdf` file is generated.

(E6): [Comparison of eviction-set-creation policies] [96 compute-hour]: This experiment compares the conflict testing and prime, prune and probe algorithms based on the number of LLC evictions they require to create same-sized eviction sets.
Execution: To run the experiment:

```
$ bash genTable.sh 0
```

Results: Table-2 is printed on the terminal.

(E7): [Sensitivity to cache size] [96 compute-hour]: This experiment analyzes the impact of cache size on the eviction rate as well as on the number of LLC evictions required for eviction-set-creation.
Execution: To run the experiment:

```
$ cd cache-model/
$ python3 get-figure.py 0 13
$ python3 get-figure.py 0 14
```

Results: `figure13.pdf` and `figure14.pdf` files are generated.

(E8): [Security against occupancy-based attacks] [96 compute-hour]: This experiment analyzes the security of various cache designs against occupancy-based attacks.
Execution: To run the experiment:

```
$ cd cachefx/
$ python3 get-figure.py 0 15
$ python3 get-figure.py 0 16
```

Results: `figure15.pdf` and `figure16.pdf` files are generated.

(E9): [Security against low-occupancy-based attacks] [30 human-minute + 128 compute-hour]: This experiment analyzes the security of various cache designs against low-occupancy-based attacks.
Execution: To run the experiment:

```
$ cd low-occupancy/
$ sudo docker run -it -v $(pwd)/randomized
_caches:/home/randomized_caches
randomized-caches
$ cd randomized_cache_hello_world/
$ bash setup.sh
$ cd ../ ; bash buildAES.sh
$ bash genNumbers.sh
$ bash getGE.sh
$ exit
$ sudo mv randomized_caches/results results
$ python3 get-figure.py 0
```

Results: `figure17.pdf` file is generated.

(E10): [Impact of warm-up states] [48 compute-hour]: This experiment analyzes the impact of warm-up state on the eviction rate, and how it causes deviation in results from the original eviction-rate experiment.

Execution: To run the experiment:

```
cd cache-model/
python3 get-figure.py 0 18
python3 get-figure.py 0 19
```

Results: `figure18.pdf` and `figure19.pdf` files are generated.

A.5 Notes on Reusability

We highly encourage others to use this work in order to analyze other cache designs and configurations. For `cache-model` simulations, one can change the cache configuration in `cache-model/config/cache.json`. If needed, the source code inside `cache-model/cache` can be modified to implement additional cache designs. New experiments can be implemented by modifying the source code in `cache-model/test`. For `cachefx` simulations, new configurations can be added in `cachefx/configs` and new cache designs can be implemented using the files in `cachefx/Cache`.

A.6 Version

Based on the LaTeX template for Artifact Evaluation V20231005. Submission, reviewing and badging methodology followed for the evaluation of this artifact can be found at `https://secartifacts.github.io/usenixsec2025/`.