USENIX Security '25 Artifact Appendix: TEEcorrelate: An Information-Preserving Defense against Performance-Counter Attacks on TEEs

Hannes Weissteiner[1], Fabian Rauscher[1], Robin Leander Schröder[3,4], Jonas Juffinger[1]
Stefan Gast[1], Jan Wichelmann[2], Thomas Eisenbarth[2], Daniel Gruss[1]
[1]Graz University of Technology, Austria, [2]University Luebeck
[3]Fraunhofer SIT, Darmstadt, Germany, [4]Fraunhofer Austria, Vienna

A Artifact Appendix

A.1 Abstract

Trusted-execution environments (TEEs) offer confidentiality in shared environments. While Intel restricts performance counter access, limiting load-balancing and anomaly detection on TEEs, AMD exposes performance counters to the host, leaving the TEE vulnerable to side-channel leakage. In this paper, we propose TEEcorrelate, a lightweight information-preserving defense against performance counter attacks on TEEs. TEEcorrelate reconciles monitoring capabilities of the host and confidentiality requirements of the TEE, by statistically decorrelating performance counters. TEEcorrelate combines two components, temporal decorrelation using counter aggregation windows, and value decorrelation using fuzzy performance counter increases.

The artifact for this paper models the behavior of TEEcorrelate to analyze sample sizes, probabilities, and trend-following performance. Releasing the artifact allows for simulation and tweaking of the defense.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our artifact does not perform any destructive or privacy-invasive actions.

A.2.2 How to access

Our artifact is available on Zenodo at `https://doi.org/10.5281/zenodo.15699920`.

A.2.3 Hardware dependencies

As our artifact only simulates the behavior of TEEcorrelate, it does not require any specific hardware. To generate sample data, we require a system that supports performance counters, which are available on most modern CPUs.

A.2.4 Software dependencies

Our artifact requires Python 3 (tested on Python 3.12.10), and the following Python packages:

- numpy
- matplotlib
- scipy

To collect sample data, we use the Linux `perf` tool, which is available on most Linux distributions.

A.2.5 Benchmarks

None.

A.3 Set-up

Set-up requires installing Python 3 along with the required packages. On Ubuntu, this can be done with the following command:

```
sudo apt install python3 python3-scipy \
python3-numpy python3-matplotlib
```

If the Nix package manager is available, it is possible to create a shell with all required dependencies by running:

```
nix-shell -p python3 python3.pkgs.matplotlib \
python3.pkgs.scipy python3.pkgs.numpy
```

A.3.1 Installation

Download the artifact from Zenodo at `https://doi.org/10.5281/zenodo.15699920` and extract the archive.

A.3.2 Basic Test

The artifact contains multiple directories, each containing different experiment scripts. Each script can be run independently without any additional parameters, except for two: `base_simulation.py` is a base class for the other scripts

which does not run on its own, and `simulate_defense.py` requires a parameter to specify the path to a trace file. We have included a sample trace file `sample_trace.txt` in the `simulate_defense` directory. Run the different scripts (`python <script_name>`) to perform the experiments.

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): TEEcorrelate decorrelates the reported signal from the real signal, while still following coarse-grained trends. This is demonstrated by the experiments (E1) and (E2).

(C2): TEEcorrelate's fast normal distribution approximates a low-resolution binomial distribution. This is demonstrated by experiment (E3).

(C3): The number of required samples to break TEEcorrelate's defense is high, and not significantly affected by using a fast normal distribution instead of a binomial distribution. This is demonstrated by experiment (E4) and (E5).

A.4.2 Experiments

(E1): Minimal Simulation [3 human-minutes]:
How to: Run the minimal defense simulation script `defense_simulation/simulate_small_trace.py`.
Results: A figure gets rendered, showing the original values (blue), and the decorrelated values (orange) for a small, simulated trace. The green area signifies the maximum deviation. Cf. Figure 4 in the paper.

(E2): Full Simulation [10 human-minutes]:
How to: Run the full defense simulation script `defense_simulation/simulate_defense.py` with a trace file (e.g., the included `sample_trace.txt`).
Optional: [20 compute-minutes] A custom trace file can be generated using the Linux `perf` tool, e.g., by running the following command:

```
perf stat -e ex_ret_instr -I 1 \
-e l2_pf_miss_l2_l3 > trace.txt
```

This command runs until interrupted, and collects performance counters every millisecond. It is important that one of the counters counts the number of retired instructions, and that the name of the counter is edited in line 46 when not using the AMD `ex_ret_instr` counter. Target performance counter names need to be added to the `perfcounts` list in the script.
Results: The left figure shows the original values (blue), and the decorrelated values (orange) for a real performance counter trace. On the right, we display the distribution of the offset between real and decorrelated values (cf. Figure 5 in the paper). Mean and standard deviation of the offset are printed to the console.

(E3): Distribution Simulation [5 human-minutes + 10 compute-minutes]:
How to: Run the distribution simulation script `distribution_samplesizes/generate_distribution_shape.py`. The script runs until interrupted, and updates the figure every 200000 samples.
Results: The figure shows a binomial-like distribution. Thus, as claimed in (C2), the fast normal distribution approximates a low-resolution (64 samples) binomial distribution, while having a much larger output parameter space.

(E4): Sample Size Calculation [15 human-minutes]:
How to: Run the sample size calculation script `distribution_samplesizes/distribution_with_32bit.py`. The script generates a figure by calculating the probability gradient (i.e., the difference in probability between each possible offset and the next) for a 32-bit fast normal distribution, a ideal gaussian distribution, and 64-bit distributions with multiple offset values.
Optional: The default parameter of this script is set to a deviation window size of 2048. To compute other window sizes, the parameter `DEVIATION_RANGE` in the script can be changed to the desired value.
Results: Figure 1 displays the CDF for the 3 different functions (normal, 64-bit fast normal, and 32-bit fast normal). Figure 2 displays the probability gradient for the same functions. Figure 3 displays the number of required samples to distinguish two probabilities, at each point of the curves. Figures 1, 2, and 3 generated by this script correspond to figures 8, 6, and 10 in the paper, respectively. We also print the minimum number of required samples for each parameter set, which can be compared to table 3 in the paper.

(E5): HQC Attack Runtime Computation [5 human-minutes, 5 compute-minutes]:
How to: Run the HQC runtime estimation script `distribution_samplesizes/hqc_runtime.py`. The script generates approximate attack runtimes for an attack on HQC with TEEcorrelate enabled, with for the deviation window sizes of 64, 2048, and 32768.
Results: For each window size and desired oracle accuracy, we print the estimated runtime (in seconds) for a single oracle query, the total runtime, and the number of required *fast divisions* to the console. the minimum calculated runtime can be compared with our claims at the end of section 6.4 in the paper.

A.5 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/.