USENIX Security '25 Artifact Appendix: "Achilles: A Formal Framework of Leaking Secrets from Signature Schemes via Rowhammer"

Junkai Liang[1,*], Zhi Zhang[2,*], Xin Zhang[1,*], Qingni Shen[1,†], Yansong Gao[2], Xingliang Yuan[3], Haiyang Xue[4], Pengfei Wu[4], Zhonghai Wu[1,†]
[1]Peking University, [2]The University of Western Australia,
[3]The University of Melbourne, [4]Singapore Management University
{ljknjupku, zzhangphd}@gmail.com, zhangxin00@stu.pku.edu.cn,
qingnishen@pku.edu.cn, garrison.gao@uwa.edu.au, xingliang.yuan@unimelb.edu.au,
haiyangxc@gmail.com, pfwu@smu.edu.sg, wuzh@pku.edu.cn

A Artifact Appendix

A.1 Abstract

To demonstrate our main results in the paper, we prepare our artifact which contains 3 main components: 1) The attacker's code related to fault injection (in the directory 1.Fault-injection/). This code implements the specific fault injection techniques used in the research, including memory profiling, memory massage and online row refresh. This code needs to be run on a personal computer with vulnerable DIMMs. We have promised to share this component for availability. 2) The attacker's code that analyses the faulty signature (in the directory 2.Post-analysis/). This code analyses the faulty signatures and outputs the leakage of the secret bits. We have promised to share this component for availability and functionality. 3) The automation tool for Achilles which analyses the signature schemes (in the directory 3.Automation-tool/). This code automates Achilles analysis for signature schemes. We have promised to share this component for availability and functionality.

A.2 Description & Requirements

The fault injection part in our artifact has to be run on a machine with vulnerable DIMMs and other parts can be run on any general-purpose personal computer.

A.2.1 Security, privacy, and ethical concerns

Our artifact may be destructive. When running with vulnerable DIMMs, the display screen may stop working and the system may abort. We recommend doing a data or system backup before starting the experiment.

A.2.2 How to access

Stable URL pointing to the latest version of our artifact: https://doi.org/10.5281/zenodo.14735639.

Stable URL for the feedback and dynamic changes: https://github.com/liang-junkai/Achilles.

A.2.3 Hardware dependencies

The fault injection part in the artifact needs to be run on linux kernel in a machine with vulnerable DIMMs. If the underlying hardware is not vulnerable, the program will work but will not publish bit-flip information. We perform the experiments using a vulnerable Samsung DDR3-1300 4G DIMM (part number: M473B5273DH0-YK0) deployed in a Lenovo T420 equipped with Intel Core i5-2430M CPU (Sandy Bridge) in Ubuntu 20.04.

A.2.4 Software dependencies

To run the post analysis part in our artifact, the corresponding library, e.g., wolfssl, relic and liboqs should be installed first. In component 2, we evaluated 3 libraries which are `relic-toolkit-0.6.0`, `wolfssl-5.6.6`, and `liboqs-0.10.0` on ubuntu 20.04.

A.2.5 Benchmarks

None.

*: The authors contribute equally to this paper.

†: Corresponding author. This work was supported by the National Key R&D Program of China under Grant No. 2022YFB2703301, PKU-OCTA Laboratory for Blockchain and Privacy Computing and the Singapore Ministry of Education (MOE) Academic Research Fund (AcRF) Tier 1 grant.

A.3 Set-up

A.3.1 Installation

We recommend installing 3 libraries: WolfSSl-5.6.6, Relic-0.6.0, Liboqs-0.10.0.

A.3.2 Basic Test

After installation, test the executable file in each directory (using the ReadMe.md), to see if the library is correctly linked.

A.4 Evaluation workflow

A.4.1 Major Claims

The major claims related to our artifact are about the feasibility of the fault injection and the post-Rowhammer analysis, and the automation tool.

(C1): The fault injection component in our artifact contains all the code related to Rowhammer preprocessing and online injection, such as memory profiling, memory waylaying and double-sided hammering

(C2): The post-analysis component in our artifact can successfully recover the secret bits using faulty signatures.

(C3): The automation tool analyses the vulnerability of several signature schemes with correct input.

A.4.2 Experiments

Here are the instructions that we run each component.

(E1): *[Fault injection] [1 human-hour + 10+ compute-hour + 5GB disk]:*

How to: First run memory profiling code and identify vulnerable positions in the DIMMs. The perform the end-to-end Rowhammer attack and starts online injection.

Execution: In directory ./1.Fault-injection/memory-profiling, run the following codes for about 10 or more hours:

```
sudo ./rowhammer-sandy -d 1 -p 0.6
```

Then in directory ./1.Fault-injection/Online-injection, run the code in each subdirectory

```
sudo ./attack
```

If some offset page information is output, it means the test is successful. If the DIMMs are vulnerable and the result of the memory profiling is correct, faulty signatures will be output.

(E2): *[Post-analysis] [1 human-hour + 20 compute-minute]:*

How to: This component analyses the faulty signatures in libraries relic, wolfssl and liboqs.

Execution: In each subdirectory, first install each library following the commands in ReadMe.md, then run

```
./[scheme name]
```

Results: If the leakage information is output (position and value of the secrets), the test is successful.

(E3): *[Automation tool] [10 human-minutes + 5 compute-minutes]:*

How to: This code automatically analyses several signatures and output if they are vulnerable to our 2 attack algorithms, DFA and SCA.

Execution: In the directory, after installing the packages in the requirements.txt, run

```
python3 auto.py
```

Results: If the vulnerabilities of the schemes are output, the test is successfully.

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/.