USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

V-ORAM: A Versatile and Adaptive ORAM Framework with Service Transformation for Dynamic Workloads

Bo Zhang and Helei Cui, *Northwestern Polytechnical University;* Xingliang Yuan, *The University of Melbourne;* Zhiwen Yu, *Northwestern Polytechnical University and Harbin Engineering University;* Bin Guo, *Northwestern Polytechnical University*

https://www.usenix.org/conference/usenixsecurity25/presentation/zhang-bo-voram

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 34th USENIX Security Symposium.

August 13–15, 2025 • Seattle, WA, USA

978-1-939133-52-6

Open access to the Artifact Appendices to the Proceedings of the 34th USENIX Security Symposium is sponsored by USENIX.

USENIX Security '25 Artifact Appendix: V-ORAM: A Versatile and Adaptive ORAM Framework with Service Transformation for Dynamic Workloads

Bo Zhang[†], Helei Cui[†*], Xingliang Yuan[◇], Zhiwen Yu[†‡], and Bin Guo[†]
[†]*School of Computer Science, Northwestern Polytechnical University, China*
[◇]*School of Computing and Information Systems, The University of Melbourne, Australia*
[‡]*College of Computer Science and Technology, Harbin Engineering University, China*
[*]*Corresponding author, email: chl@nwpu.edu.cn*

A Artifact Appendix

This artifact includes all implemented prototypes and the code/scripts for reproducing the major results/figures mentioned in the paper. We implement V-ORAM on-premise using Python's built-in libraries and the lightweight encryption library `pycryptodome`. Our source code, with a detailed README, is available at https://github.com/BoZhangCS/V-ORAM. The entire evaluation can be performed through simple command-line instructions. We also uploaded the artifact to Zenodo, the DOI is http://dx.doi.org/10.5281/zenodo.14732806.

A.1 Abstract

V-ORAM is a versatile and adaptive ORAM framework with service transformation for dynamic workloads. The provided artifact enables efficient transformations between Path ORAM, Ring ORAM, and ConcurORAM.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact is evaluated through publicly available anonymized datasets and local randomized read-write. It does not involve any security, privacy, or ethical concerns.

A.2.2 How to access

The source code of the artifact is available at https://github.com/BoZhangCS/V-ORAM. We also uploaded the artifact to Zenodo, the DOI is http://dx.doi.org/10.5281/zenodo.14732806.

A.2.3 Hardware dependencies

The artifact is evaluated locally without requiring hardware beyond a general-purpose CPU (no TEE required). During execution, the artifact persists ORAM data and maintains metadata (*e.g.*, PosMap). So please ensure the current directory has at least 90GB free storage and 16GB available RAM. Our test machine is a Mac mini equipped with 512GB storage and an M2 chip with 16GB RAM.

A.2.4 Software dependencies

The artifact has been adapted for Windows, macOS, and Linux. We evaluated it under a macOS Sequoia 15.3.2 system. It is implemented on `Python-3.8`, and we recommend using the same version. All required third-party libraries are listed in `requirements.txt` and can be installed directly via `pip`.

A.2.5 Benchmarks

The artifact only requires subsets of the datasets in the real-world case study, which are already included in the artifact at `data/real_workloads.zip`. The full version of three datasets are available through: MSRC: https://iotta.snia.org/traces/block-io/388, AliCloud: https://github.com/alibaba/block-traces, Twitter: https://github.com/twitter/cache-trace.

A.3 Set-up

After cloning the project, please follow the steps below to set up the artifacts (README.md#Installation).

A.3.1 Check for Storage

Please check whether the current directory has enough storage space by running the command "`df -h .`".

A.3.2 Set up the environments

Set up the virtual `Python` environment and install all third-party dependencies through the following commands:

```
pip3 install virtualenv
virtualenv .venv -python=python3.8
source .venv/bin/activate
pip3 install -r requirements.txt
```

A.3.3 Installation

Extract the stripped real-world datasets via commands:

```
cd ./data
unzip real_workloads.zip
```

A.3.4 Basic Test

Perform random read/write tests for prototypes:

```
pytest
```

A.4 Evaluation workflow

The artifact reproduces major plots, tables, and statistical data in the paper. We develop scripts for each experiment, which can be evaluated entirely through command line instructions. All experiments can be run in parallel, except Figures 7, 9, and 10, which require the results from Figure 5. Briefly, the evaluation procedures are as follows:

1. Get artifact from https://github.com/BoZhangCS/V-ORAM.

2. Set up the environment and install the necessary dependencies according to README.

3. Run the experiments through the command line instructions provided in README.

A.4.1 Major Claims

(C1): V-ORAM enables efficient ORAM service transitions. It incurs only constant-level transformation costs, which do not scale with the number of the buckets. This is proven by experiment (E1). It also introduces constant extra costs to ORAM services, as proven by experiment (E2). The storage cost of V-ORAM is $n \log n$ bits, as shown in experiment (E3).

(C2): The planner in V-ORAM assists clients in selecting appropriate ORAM parameters, based on the distribution of its datasets and request. This is proven by experiments (E4) and (E5).

(C3): V-ORAM is efficient and effectively reduces monetary costs in real-world scenarios. The planner helps the client decide whether to switch ORAM services. This is proven by experiments (E6) and (E7).

A.4.2 Experiments

(E1): [Figure 4] [1 human-minutes + 1 compute-minutes]: (E1) compares V-ORAM with the other two baselines (D and M) mentioned in §6.1 (README.md#Figure-4).
How to: All the preparations are stated in Artifact Appendix A.3. Execute the following command to reproduce our results of (E1):

```
python3 ./artifacts/Fig4_transformation.py -pdc
```

Results: The communication, computation, and RTT costs of three schemes under different ORAM sizes. Figures plotted from the results of the artifact. Comparison of the plotted figures and the figure in the paper.

(E2): [Figure 5] [1 human-minutes + 5 compute-minutes]: (E2) runs V-ORAM and records the costs of EvictRecord during the process. The three types of access cost in the figures are described in §6.2#"EvictRecord costs" (README.md#Figure-5).
How to: All the preparations are stated in Artifact Appendix A.3. Execute the following commands to reproduce our results of (E2):

```
python3 ./artifacts/Fig5_EvictRecord_prfm.py -pdc
```

Results: The costs of three types of V-ORAM access when running Ring ORAM or ConcurORAM services, under different ORAM sizes. Figures plotted from the results of the artifact. Comparison of the plotted figures and the figures in the paper.

(E3): [Figure 6] [1 human-minutes + 5 compute-minutes]: (E3) runs three ORAMs and record their stash size, and compares it with the sizes of PosMap and RecMap (README.md#Figure-6).
How to: All the preparations are stated in Artifact Appendix A.3. Execute the following commands to reproduce our results of (E3):

```
python3 ./artifacts/Fig6_storage_cost.py -pdc
```

Results: The storage costs and proportion of three ORAMs under different ORAM sizes. Figures plotted from the results of the artifact. Comparison of the plotted figures and the figures in the paper.

(E4): [Figure 7] [1 human-minutes + 1 compute-minutes]: (E4) runs three ORAMs under various system performance requirements and estimates their monetary costs under different bucket sizes (README.md#Figure-7).
How to: All the preparations are stated in Artifact Appendix A.3. Execute the following commands to reproduce our results of (E4):

```
python3 ./artifacts/Fig7_decision_choose_ORAM.py
-pdc
```

Results: The monetary costs of three ORAMs under different system performance requirements. Figures plotted from the results of the artifact. Comparison of the plotted figures and the figures in the paper.

(E5): [Figure 8] [1 human-minutes + 1 compute-minutes]: (E5) compares the communication blowup of three ORAMs under different parameters and block sizes (README.md#Figure-8).

How to: All the preparations are stated in Artifact Appendix A.3. Execute the following commands to reproduce our results of (E5):

```
python3 ./artifacts/Fig8_opt_block_size.py -pdc
```

Results: The communication blowup of three ORAMs under different parameter settings. Figures plotted from the results of the artifact. Comparison of the plotted figures and the figures in the paper.

(E6): [Figure 9] [1 human-minutes + 10 compute-minutes]: (E6) evaluates V-ORAM performance under two realistic settings (different RTT and block sizes). The RTT experiments are derived from the results of (E2), while the block sizes experiments require re-running V-ORAM (README.md#Figure-9).

How to: All the preparations are stated in Artifact Appendix A.3. Execute the following commands to reproduce our results of (E6):

```
python3 ./artifacts/Fig9_realistic_settings.py
-pdc
```

Results: The response time under different RTT and the processing time under different block size. Figures plotted from the results of the artifact. Comparison of the plotted figures and the figures in the paper.

(E7): [Figure 10] [1 human-minutes + 1 compute-minutes]: (E7) reads three real-world datasets and simulates V-ORAM according to its requests (README.md#Figure-10).

How to: All the preparations are stated in Artifact Appendix A.3. Execute the following commands to reproduce our results of (E7):

```
python3 ./artifacts/Fig10_real_world_case_studies.py
-d
```

Results: Established V-ORAM workloads for three datasets. Plotted figures from three datasets. Since the results are stable, there is no comparison.

(E8): [Figure 11] [1 human-minutes + 1 compute-minutes]: (E8) runs V-ORAM under our two simulated workloads and records performance during execution. These two workloads represent dynamic throughput and latency (README.md#Figure-11).

How to: All the preparations are stated in Artifact Appendix A.3. Execute the following commands to reproduce our results of (E8):

```
python3 ./artifacts/Fig11_simulated_workloads.py
-pdc
```

Results: The throughput of V-ORAM under two simulated workloads. Plotted figures from three datasets. Since the results are stable, there is no comparison.

(E9): [Table 3] [1 human-minutes + 1 compute-minutes]: (E9) reads the datasets, calculated statistics, and provides the estimate tree heights, recommended block sizes, and communication blowups under Ring ORAM (README.md#Table-3).

How to: All the preparations are stated in Artifact Appendix A.3. Execute the following commands to reproduce our results of (E9):

```
python3 ./artifacts/Tab3_real_case_study_para.py
-p
```

Results: The statistics and calculated results. Plotted figures from three datasets. Since the results are stable, there is no comparison.

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/.