

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SONIC: Concurrent Oblivious RAM & Data Structures for Low-Latency and High-Throughput

Nihal Talur and Ioannis Demertzis, *UC Santa Cruz*

https://www.usenix.org/conference/usenixsecurity26/presentation/talur

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 35th USENIX Security Symposium.

August 12–14, 2026 • Baltimore, MD, USA

ISBN 978-1-939133-58-8

Open access to the Artifact Appendices to the Proceedings of the 35th USENIX Security Symposium is sponsored by



جامعة الملك عبدالله للعلوم والتقنية

King Abdullah University of Science and Technology



USENIX Security '26 Artifact Appendix: SONIC: Concurrent Oblivious RAM & Data Structures for Low-Latency and High-Throughput

Nihal Talur
UC Santa Cruz

Ioannis Demertzis
UC Santa Cruz

A Artifact Appendix

A.1 Abstract

This artifact contains the SONIC implementation, GraphMap baseline, O2TH and PMCHAIN, and EnigMap/GraphOS baseline artifacts. It supports SGX builds, validation tests, and representative throughput anchors for the paper's single-node evaluation claims.

A.2 Description & Requirements

- `sonic/`: SONIC, GraphMap, O2TH, PMCHAIN.
- `baselines/sgx-baseline-container/`: SGX baseline image.
- `baselines/enigmap/`: EnigMap baseline.
- `baselines/graphos/`: GraphOS baseline.

A.2.1 Security, privacy, and ethical concerns

The artifact uses synthetic workloads and no human-subject or evaluator data. It runs benchmark programs in ephemeral Docker containers with SGX device passthrough. It does not modify host data.

A.2.2 How to access

https://doi.org/10.5281/zenodo.20646912

A.2.3 Hardware dependencies

- x86_64 Linux host with Intel SGXv2
- Azure `Standard_DC32s_v3` instance, with 32 vCPUs, 256 GiB RAM, 192 GiB EPC
- `/dev/sgx_enclave` and `/dev/sgx_provision` (SGX driver is upstreamed since Linux 5.11)

A.2.4 Software dependencies

Docker on x86_64 Ubuntu 22.04. The artifact includes Dockerfiles for the build toolchain.

Host sanity checks:

```
ls -l /dev/sgx_enclave /dev/sgx_provision
docker version
uname -a
docker run --rm \
  --device /dev/sgx_enclave \
  --device /dev/sgx_provision \
  ubuntu:22.04 true
```

Expected: SGX device files exist, Docker functions.

A.2.5 Benchmarks

No external dataset is required as ORAM/OMAP workloads are synthetic. The SONIC demo CLI uses implementation names: `pathoram` is the GraphMap baseline, `zingoram` is SONIC, and `zingoram-disjoint` is disjoint-window SONIC.

A.3 Set-up

A.3.1 Installation

```
# Host shell:
tar -xzf sonic-usenix26-artifacts-20260705.tar.gz
cd sonic-usenix26-artifacts
export HOST_ARTIFACT_ROOT=$PWD
cd "$HOST_ARTIFACT_ROOT/sonic"

docker build -t localhost/sonic2-sgx \
  -f docker/sonic2_sgx.docker .

docker run --privileged \
  --device /dev/sgx_enclave:/dev/sgx_enclave \
  --device /dev/sgx_provision:/dev/sgx_provision \
  --rm -it \
  -e ARTIFACT_ROOT=/artifact \
  -v "$HOST_ARTIFACT_ROOT":/artifact \
  -w /artifact/sonic \
  localhost/sonic2-sgx bash

# SONIC container:
CC=clang CXX=clang++ cmake \
  -G Ninja \
  -B build-sgx-clang \
  -DCMAKE_BUILD_TYPE=Release \
  -DSONIC_PLATFORM=sgx \
```

```
  -DSONIC_DEMO_SGX_MODE=HW \
  -DSONIC_BUILD_APPLICATIONS=ON \
  -DSONIC_BUILD_DISTRIBUTED=ON
cmake --build build-sgx-clang --parallel
BIN="$ARTIFACT_ROOT/sonic/build-sgx-clang/bin"
BIN="$BIN/sonic_demo_sgx"
export BIN

cd
"$HOST_ARTIFACT_ROOT/baselines/sgx-baseline-container"
docker build -t sonic-baseline-sgx .
```

A.3.2 Basic Test

```
# SONIC container:
cd "$ARTIFACT_ROOT/sonic"
# GraphMap
$BIN pathoram validate \
  -N 1024 -B 64 -Z 4 -m 8

# SONIC
$BIN zingoram validate \
  -N 4096 -B 64 -Z 16 -S 16 \
  --routing-depth 3 --evict-batch 2 \
  -T 4 -m 64

# SONIC, disjoint
$BIN zingoram-disjoint validate \
  -N 32768 -B 64 -Z 14 -S 16 \
  --routing-depth 5 --evict-batch 4 \
  -T 4 --disjoint-epoch 17408 -m 128
```

Expected:

```
pathoram validate ok
zingoram validate ok
zingoram-disjoint validate ok
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Artifact builds and validates SONIC and GraphMap. See E1.

(C2): GraphMap, EnigMap, and GraphOS reproduce the Figure 6 baseline results. See E2.

(C3): SONIC reproduces the Figures 7–8 standard ORAM throughput/latency results. See E3.

(C4): SONIC reproduces the Figure 9 thread-scaling trend. See E4.

(C5): SONIC reproduces the Figure 10 deferred-eviction results. See E5.

(C6): O2TH and PMCHAIN benchmarks support the Figure 11 subORAM evaluation story. See E6.

A.4.2 Experiments

For a standard and representative SGX hardware environment, we use the Azure `Standard_DC32s_v3` instance type. The full benchmark matrix in the paper takes $\approx$ 4 VM-days to run for all single-server experiments. The multi-server experiment additionally requires significant effort for provisioning on multiple machines and high ($\approx$ \$800) costs.

Therefore, for artifact evaluation, we recommend running **representative anchors** to corroborate all claims: exact plotted parameter configurations with bounded access counts. This keeps evaluation to a few hours while checking claim-bearing endpoints and scaling trends. The package contains `experiment.py`; run the anchor with `-anchor`. The full run can be started with `-full`.

Expected values are paper-figure op/s. We have observed minor ($\approx$ 5-10%) performance variations across instances on Azure, which falls within expected tolerances; the relative performance story is consistently reproducible.

(E1): Functional validation. [20m human; 1h compute.] Run Basic Test; expect three `validate ok` lines.

(E2): Fig. 6 baselines at $N = 2^{20}$. [15m human; 45m compute.]

```
# Host shell:
cd "$HOST_ARTIFACT_ROOT"
python3 experiment.py baselines --anchor
python3 experiment.py baselines --full
--commands
```

Expected op/s (B=64/256): GraphMap 21639.463/12367.066; EnigMap 13802.878/5098.318; GraphOS 2250.324/1615.670.

(E3): Figures 7–8 SONIC throughput/latency anchors. [5m human; 1h compute.]

```
# Host shell:
cd "$HOST_ARTIFACT_ROOT"
python3 experiment.py scale-n --anchor
python3 experiment.py scale-n --full --commands
```

Expected op/s: Ser/Par8/Par16 = 54209.480/193409.508/254681.666.

(E4): Figure 9 thread-scaling anchors. [5m human; 1h compute.]

```
# Host shell:
cd "$HOST_ARTIFACT_ROOT"
python3 experiment.py scale-t --anchor
python3 experiment.py scale-t --full --commands
```

Expected op/s for T1/T8/T16 = 108184.811/361251.568/450624.877 at $N = 2^{20}$ and 69093.680/240167.842/340705.848 at $N = 2^{24}$.

(E5): Figure 10 deferred-eviction anchors. [5m human; 1h compute.]

```
# Host shell:
cd "$HOST_ARTIFACT_ROOT"
python3 experiment.py disjoint-evict --anchor
python3 experiment.py disjoint-evict --full
--commands
```

Expected op/s: Window/Online = 474060.014/901892.813, with OnlineT16 $>$ WindowT16 $>$ Par16.

(E6): Single-node subORAM benchmarks. [15m human; 1h compute.]

```
# Host shell:
cd "$HOST_ARTIFACT_ROOT"
python3 experiment.py suboram --anchor
python3 experiment.py suboram --full --commands
```

At 512 MiB, E6 tests $K = 90432$ for PMCHAIN and $K = 65536$ for O2TH. Both should have batch latency at most 1s, validating support for at least 90432 and 65536 requests/s, respectively. E6 does not itself directly produce Figure 12; Figure 12 is the separate multi-server experiment requiring nine servers, with one load balancer and eight subORAMs.

A.4.3 Comparing CSVs to the paper

Figure	CSV comparison
6, baseline throughput	`baselines`: compare `throughput_ops_per_sec` by `impl`, `B`, and `N_exp`
7, SONIC throughput	`scale-n`: compare `throughput_ops_per_sec` by `impl`, `B`, and `N_exp`
8, access latency	`scale-n` + `disjoint-evict` (`WindowT16`): compare `mean_access_latency_us`
9, thread scaling	`scale-t`: `T` in `impl` is x; `throughput_ops_per_sec` is y
10, deferred eviction	`disjoint-evict`: compare `throughput_ops_per_sec` by `impl`, `B`, and `N_exp`
11, isolated subORAM	`suboram`: x is `N*B`; y is `K` when `meets_1s_target=true`
12, 1+8-server deployment	Multi-server 1 LB + 8 SO deployment
13, modeled AVL-OMAP	`baselines` + `scale-n`: use ORAM measurements, dividing throughput by AVL-OMAP ORAM access count from `2*h(N)`, where `h(N)` is worst case AVL tree height

A.5 Notes on Reusability

The artifact was carefully designed to be modular and suitable for use as a composable set of libraries for oblivious computation in TEEs. Dependent applications can link the layers and abstractions that they need. The code is also portable and easily retargetable across platforms and architectures.

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/usenixsec2026/.