

USENIX

THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SophOMR: Improved Oblivious Message Retrieval from SIMD-Aware Homomorphic Compression

Keewoo Lee, *Ethereum Foundation;* Yongdong Yeo, *Seoul National University*

https://www.usenix.org/conference/usenixsecurity26/presentation/lee-keewoo

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: SophOMR: Improved Oblivious Message Retrieval from SIMD-Aware Homomorphic Compression

Keewoo Lee
Ethereum Foundation

Yongdong Yeo
Seoul National University

A Artifact Appendix

A.1 Abstract

We provide a C++ reference implementation of the Oblivious Message Retrieval (OMR) and Oblivious Message Detection (OMD) schemes, referred to as SophOMR and SophOMD, respectively, in the accompanying paper.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

The artifact does not pose any security, privacy, or physical risks to evaluators, nor does it raise ethical concerns. The payloads used to simulate data posted on a bulletin board are randomly generated.

A.2.2 How to access

The artifact is available at: https://doi.org/10.5281/zenodo.17958466.

A.2.3 Hardware dependencies

The artifact has no specific hardware dependencies and can be run on standard platforms. However, the HEXL backend is optimized for processors with AVX512_IFMA support (e.g., Intel Ice Lake). The benchmarks reported in our paper were run single-threaded on a Google Compute Cloud `n2-standard-8` instance (Intel Ice Lake, 2.6 GHz CPU with 32 GB RAM[1]).

A.2.4 Software dependencies

The artifact is based on the following dependencies:

- C++ build environment
- CMake build infrastructure
- NTL library
- OpenFHE library (Tested with v1.4.0[2])
- HEXL library (Optional; Skip on ARM)

The benchmarks reported in our paper were run on Ubuntu 20.04 LTS with OpenFHE v1.2.0.

A.2.5 Benchmarks

None.

A.3 Set-up

A.3.1 Installation

The commands below are based on Ubuntu 24.04 LTS with x86-64 and assume the use of the HEXL backend.

1. Install CMake, GMP, and NTL (if needed).

```
sudo apt-get update
sudo apt-get install build-essential
sudo apt-get install cmake
sudo apt-get install libgmp3-dev
sudo apt-get install libntl-dev
```

2. Install OpenFHE + HEXL. Follow the instructions at https://github.com/openfheorg/openfhe-hexl, or run:

```
git clone https://github.com/openfheorg/
    openfhe-configurator.git
cd openfhe-configurator
# To reproduce the version used in the paper, run:
    git reset --hard 9ac9090
scripts/configure.sh

# Would you like to stage an openfhe
    -development build?      [y/n] : n  (y on ARM)
# Would you like to stage an openfhe-hexl
    build?                   [y/n] : y

sudo scripts/build-openfhe-development.sh
```

[1] The installation of OpenFHE and HEXL may require more than 32 GB of RAM (we were able to install using 64 GB RAM). After installation, however, SophOMR runs comfortably with 32 GB of RAM.

[2] To implement ring-switching, we use OpenFHE in a manner not officially supported by its APIs, which may be incompatible with OpenFHE versions beyond 1.4.0.

3. Build the Library.

```
cd ..  # if still in openfhe-configurator
    directory
git clone https://github.com/keewoolee/
    SophOMR.git  # clone the repository
cd SophOMR
mkdir build
cd build
cmake .. -DCMAKE_PREFIX_PATH=~/openfhe
    -configurator/openfhe-staging/install
    # adjust the path to the location of the
    openfhe libraries
make
```

A.3.2 Basic Test

The following command runs the fastest predefined configuration (1-2 compute-minutes). On success, the final line of the output will be: `Result is Correct!`

```
./test OMD 65536 16
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): *SophOMR achieves, compared to the prior state of the art PerfOMR, reductions of* 3.4× *in runtime,* 2.2× *in digest size, and* 1.5× *in key size in a setting with 65,536 payloads (each 612 bytes), of which up to 50 are pertinent. In particular, the runtime of the homomorphic compression step is reduced by* 7.5×*. These results are reported in Tab. 4 and Fig. 2, together with the corresponding OMD variants (SophOMD and PerfOMD). Similar trends are observed for 524,288 payloads (Tab. 8 and Fig. 4). The SophOMR and SophOMD runtimes presented in the tables and figures are reproducible using Experiment E1.*[3]

(C2): *SophOMR and SophOMD exhibit trends consistent with Fig. 3 of our paper for varying values of k and t. The SophOMR and SophOMD runtimes presented in Fig. 3 are reproducible using Experiment E1.*

A.4.2 Experiments

(E1): *Full Benchmarks (6-7 compute-hours): This experiment runs all benchmarks presented in the paper and supports Claims C1 and C2. Each benchmark is repeated five times, and we report the average runtime along with the standard deviation.*
Execution: *In the build directory, run the following command:*

```
python3 -u ../benchmark.py > benchmark.txt 2>&1
```

Results: *The file* `benchmark.txt` *contains the timing results and corresponding standard deviations for all benchmarks presented in the paper. The results are organized by the figures and tables in which they appear. When run in the same environment described in Section A.2.3 and A.2.4, the output should reproduce the reported results; in other environments, it is expected to follow similar performance trends.*

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/usenixsec2026/.

References

[1] Zeyu Liu, Eran Tromer, and Yunhao Wang. PerfOMR: proof of concept C++ implementation for OMR (Oblivious Message Retrieval) with Reduced Communication and Computation. https://github.com/ObliviousMessageRetrieval/ObliviousMessageRetrieval/tree/perfomr.

[3] PerfOMR is outside the scope of this artifact. The runtime results reported in our paper can be obtained using the original PerfOMR implementation [1]. To ensure reproducibility, we provide a fork of the original repository with minor modifications necessary for our evaluation, which reproduces the results by following the provided instructions. The fork is available at: https://github.com/keewoolee/PerfOMR.