EUROCRYPT '25 Artifact Appendix: Snake-eye Resistant PKE from LWE for Oblivious Message Retrieval and Robust Encryption

Zeyu Liu
Yale University

Katerina Sotiraki
Yale University

Eran Tromer
Boston University

Yunhao Wang
Yale University

A Artifact Evaluation

A.1 Abstract

Our artifact is a C++ library implementing the constructions DoS-PerfOMR in [1]. The source code is public on github under ObliviousMesageRetrievel repo. Our main claims produced from this artifact are the detector runtime of the construction in Table 1 and Table 2.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

Our artifact should not incur any risk to the evaluators regard of the machines security, data privacy or ethical concerns. The payload we use to simulate the public data published on the bulletin board is drawn from random distributions. Note that we are using an old version of OpenSSL, but it does not require the root privilege and should only be built in the directory specified by the evaluator. Despite all these, we still recommend the evaluators open a **fresh** GCP instance without testing our code using their own machines.

A.2.2 Hardware dependencies

Our main benchmarks should be able to be reproduced on a normal Google Compute Cloud n4-standard-8 instance type (4 hyperthreads of an Intel Xeon 3.10 GHz CPU with 32GB RAM),

A.2.3 Software dependencies

On a Google Compute Cloud n4-standard-8 instance, we run the benchmarks with boot disk configured with Ubuntu 20.04 LTS operating system and a 128GB disk. Notice that the disk memory is used to store the large database our experiments run against.

We also rely on the following softwares and libraries:

- C++ build environment and cmake build infrastructure
- SEAL library 4.1 and all its dependencies Notice that we made some manual change on SEAL interfaces to facilitate our implementation and thus a built-in dependency of SEAL is directly included under 'build' directory.
- PALISADE library release v1.11.2 and all its dependencies, as v1.11.2 is not publicly available anymore when this repository is made public, we use v1.11.3 in the instructions instead.
- NTL library 11.4.3 and all its dependencies
- OpenSSL library on branch OpenSSL_1_1_1-stable We use an old version of OpenSSL library for plain AES function without the complex EVP abstraction.

A detailed installation script is provided in the README.md file in our artifact. the datasets we use are simulated directly when running the experiements and thus no third-party models/datasets are used.

A.2.4 Benchmarks

We benchmark the main scheme DoS-PerfOMR (Section 7.5). We choose the number of messages to be $N = 2^{19}, 2^{21}, 2^{23}$, and let the total number of pertinent messages (and the bound estimated by the recipient) be $\gamma = \bar{\gamma} = 50, 100, 150$.

A.3 Set-up

The instructions below are also documented in the README file of our source code submitted.

A.3.1 Installation

```
# If permission required, please add sudo
# before the commands as needed

sudo apt-get update && sudo apt-get install
build-essential # if needed
sudo apt-get install autoconf # if no autoconf
sudo apt-get install cmake # if no cmake
sudo apt-get install libgmp3-dev # if no gmp
sudo apt-get install libntl-dev # if no ntl
sudo apt-get install unzip # if no unzip

# If you have the snake_eye_code.zip directly,
# put it under ~/OMR and unzip it into
# ObliviousMessageRetrieval dir, otherwise
clone the tag:
```

	Detector Runtime (ms/msg)	Clue Key Size (KB)	Clue Size (bytes)	Detector Key Size (MB)	Digest Size (bytes/msg)	Recipient Runtime (ms)	DoS-resistance
DoS-PerfOMR [1, Section 7.5]	1-thread: 4.9 2-thread: 2.6	4.73	1996	183	2.71	20	Yes ([1, Thm 5.3])

Table 1: Our result $N = 2^{19}, \gamma = \bar{\gamma} = 50$ (as shown in our full version [1]).

		$\gamma = \bar{\gamma} = 50$			
	N	Amortized runtime (ms/msg)	Total runtime (s)	Amortized digest size (bytes/msg)	Total digest size (MB)
DoS-PerfOMR	2^{19}	4.95	2597.53	2.71	1.35
	2^{21}		10663.81	0.81	1.70
	2^{23}		42467.92	0.20	1.70

		$N = 2^{19}$			
	$\gamma = \bar{\gamma}$	Amortized runtime (ms/msg)	Total runtime (s)	Amortized digest size (bytes/msg)	Total digest size (MB)
DoS-PerfOMR	50	4.95	2597.53	2.71	1.35
	100	5.44	2850.74	4.87	2.43
	150	5.83	3067.73	7.04	3.52

Table 2: Scalability of our construction DoS-PerfOMR.

```
mkdir -p ~/OMR
cd ~/OMR
git clone --branch ec25_artifact
--single-branch https://github.com/Oblivious
MessageRetrieval/ObliviousMessageRetrieval.git

unzip snake_eye_code.zip

# change build_path to where you want the
# dependency libraries installed
OMRDIR=~/OMR
BUILDDIR=$OMRDIR/ObliviousMessageRetrieval/build

cd $OMRDIR && git clone -b v1.11.3
https://gitlab.com/palisade/palisade-release
cd palisade-release
mkdir build
cd build
cmake .. -DCMAKE_INSTALL_PREFIX=$BUILDDIR
make
make install

# Old OpenSSL used for plain AES function
# without EVP abstraction
cd $OMRDIR && git clone -b OpenSSL_1_1_1w
https://github.com/openssl/openssl
cd openssl
./config --prefix=$BUILDDIR
make
make install

# a separate fork of SEAL library that
# overwrite some private functions, used
# in prior works
cd $OMRDIR && git clone
https://github.com/wyunhao/SEAL
cd SEAL
cmake -S . -B build -DCMAKE_INSTALL_PREFIX=$BUILDDIR
-DSEAL_USE_INTEL_HEXL=ON
cmake --build build
cmake --install build

# Optional
# Notice that although we 'enable' hexl via
# command line, it does not take much real
# effect on GCP instances and thus does not
# have much impact on our runtime
cd $OMRDIR && git clone --branch v1.2.3
https://github.com/intel/hexl
cd hexl
cmake -S . -B build -DCMAKE_INSTALL_PREFIX=$BUILDDIR
cmake --build build
cmake --install build

cd $OMRDIR/ObliviousMessageRetrieval/build
mkdir ../data
mkdir ../data/payloads
mkdir ../data/clues
```

```
cmake .. -DCMAKE_PREFIX_PATH=$BUILDDIR
make
```

A.3.2 Basic Test

The instruction to run a test on our construction is as follows:

```
./OMRdemos dos <number_of_cores>
<number_of_messages_in_bundle>
<number_of_bundles> <number_of_pert_msgs>
```

For example, a valid sanity test could be:

```
cd $BUILDDIR
./OMRdemos dos 1 2 32768 50
```

The expected output should look like this:

```
Preparing database and paramaters...
Pertient message indices: [ 3558 ... 32215 ]
/
| Encryption parameters :
|   scheme: BFV
|   poly_modulus_degree: 32768
|   coeff_modulus size: ... bits
|   plain_modulus: 65537
\
Database and parameters prepared.

Prepare switching key time: 254576131
ClueToPackedPV time: 175768918 us.
PVUnpack time: 148435548 us.
ExpandedPVToDigest time: 60941476 us.

Detector running time: 475043830us.
Digest size: 568087 bytes

Result is correct!
```

A.4 Evaluation workflow

A.4.1 Major Claims

Our benchmark claims are all in Table 1 and Table 2. The major one to be reproduced is the detector runtime (note that clue key, clue sizes and digest sizes can be calculated with the parameters we wrote in our paper; the recipient runtime is not optimized for), up to some testing variation. In particular, the "detector runtime" column in Table 1 and the "amortized runtime" in Table 2.

A.4.2 Experiments

Before executing any experimental scripts in this section, we assume that one has finished the installation steps in Appendix A.3.1. The expected outcomes should be similar to the one given under Appendix A.3.2.

Notice that the runtime of our experiments are quite long (the main scheme takes most of the time, while the preparation of the dataset also takes one to dozens of hours, depending on how long the dataset is), **we highly recommend one to use** screen **command** to detach all running scripts from the current session and put it on the back-end, so that one is still able to re-attach it after the current session times out (which does happen a lot when running our experiments). We also recommend one to initialize several fresh instances and run those experiments in parallel.

(E1): Runtime of DoS-PerfOMR as in Table 1, which is also our main claim in [1].

Execution:

```
./OMRdemos dos 1 8 65536 50
./OMRdemos dos 2 8 65536 50
```

Results: After seeing the detector running time (in us) in the log, divide it by the number of transactions (notice that the total number of transactions equals to number_of_messages_in_bundle multiplied with number_of_bundles). For example, if by running script `./OMRdemos dos 1 8 65536 50` with log: `Detector running time: 2601415612 us`, the amortized runtime should be $2601415612/(65536 * 8) = 4961\text{us} = 4.96\text{ms}$.

(E2): Runtime scaling with the number of transactions N: the following run scripts aim to reproduce the detector runtime stated in the top half of Table 2.

Execution:

```
./OMRdemos dos 1 8 65536 50
./OMRdemos dos 1 8 262144 50
./OMRdemos dos 1 8 1048576 50
```

Results: The total runtime of column 2 in Table 2 is shown directly as the detector runtime in the log and the amortized runtime (ms/msg) could be calculated similarly as above.

(E3): Runtime scaling with the number of pertinent messages $\bar{\gamma}$: the following run scripts aim to reproduce the detector runtime stated in the bottom half of Table 2.

Execution:

```
./OMRdemos dos 1 8 65536 50
./OMRdemos dos 1 8 65536 100
./OMRdemos dos 1 8 65536 150
```

Results: Same as above.

References

[1] Z. Liu, K. Sotiraki, E. Tromer, and Y. Wang. Snake-eye resistant PKE from LWE for oblivious message retrieval and robust encryption. Cryptology ePrint Archive, Paper 2024/510, 2024. Full version of this paper. Version 2025/02/07.