Duoram: A Bandwidth-Efficient Distributed ORAM for 2- and 3-Party Computation

Adithya Vadapalli, *University of Waterloo;* Ryan Henry, *University of Calgary;* Ian Goldberg, *University of Waterloo*

https://www.usenix.org/conference/usenixsecurity23/presentation/vadapalli

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 32nd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 32nd USENIX Security Symposium.

August 9–11, 2023 • Anaheim, CA, USA

978-1-939133-37-3

USENIX'23 Artifact Appendix: Duoram: A Bandwidth-Efficient Distributed ORAM for 2- and 3-Party Computation

Adithya Vadapalli
avadapal@uwaterloo.ca
University of Waterloo

Ryan Henry
ryan.henry@ucalgary.ca
University of Calgary

Ian Goldberg
iang@uwaterloo.ca
University of Waterloo

A Artifact Appendix

A.1 Abstract

This artifact contains DUORAM's source code and the necessary scripts to run the experiments and reproduce the major claims of our paper, DUORAM: A Bandwidth-Efficient Distributed ORAM for 2- and 3-Party Computation. Our major claims are reflected in Figures 7, 8, 9, and 10 in Section 6.2 of our paper. Along with the source code of DUORAM and the DUORAM replication scripts, the artifact also provides the scripts to replicate the experiments for Doerner and shelat's FLORAM and Jarackei and Wei's 3-party Circuit ORAM, the two implementations the paper compares DUORAM to. The experiments are run in Docker containers under different simulated network conditions. We simulate these conditions by using `tc qdisc add dev eth0 root netem delay Xms rate Ymbit`, to set the latency to X ms, and restrict the bandwidth capacity to Y Mbit/s. In our experiments, standard network conditions refer to a latency of 30 ms and a bandwidth capacity of 100 Mbit/second. Similarly, colocated network conditions refer to 30 μs of Internet latency and 100 Gbit/s of bandwidth capacity.

A.2 Description & Requirements

A.2.1 How to access

The artifact can be accessed at https://git-crysp.uwaterloo.ca/avadapal/duoram/src/usenixsec23_artifact.

To download the artifact:

- `git clone https://git-crysp.uwaterloo.ca/avadapal/duoram`
- `cd duoram`
- `git checkout usenixsec23_artifact`

A.2.2 Hardware dependencies

The hardware dependencies to run our artifact are as follows:

- A system with an x86 processor that supports `AVX2` instructions. This instruction set was released in 2013, so most recent processors should be fine. We have tested it on both Intel and AMD processors. On Linux, `grep avx2 /proc/cpuinfo` to see if your CPU can be used (if the output shows you CPU flags, then it can be; if the output is empty, it cannot).
- At least 16 GB of available RAM. To run the optional "large" tests, you will require at least 660 GB of available RAM (an atypical machine, to be sure, which is why the large tests are optional and not essential to our major claims).

A.2.3 Software dependencies

The Software dependencies to run our artifact are a basic Linux installation, with `git` and `docker` installed. We have tested it on Ubuntu 20.04 and Ubuntu 22.04, with `apt install git docker.io`.

A.3 Setup

Detailed setup instructions are outlined in the `README.md` file in the artifact. We briefly summarize it here.

A.3.1 Installation

The following will download and build the dockers for the DUORAM, FLORAM, and Circuit ORAM systems (approximate compute time: 15 minutes).

```
 cd repro && ./build-all-dockers
```

A.3.2 Basic test

A simple "kick the tires" test can be run using `./repro-all-dockers test` from the `repro` directory (approximate compute time: 1 minute). The expected output looks as follows:

```
2PDuoramOnln readwrite 16 1us 100gbit 2
0.86099545 s
2PDuoramOnln readwrite 16 1us 100gbit 2
```

```
22.046875 KiB
2PDuoramTotl readwrite 16 1us 100gbit 2
1.48480395 s
2PDuoramTotl readwrite 16 1us 100gbit 2
177.7138671875 KiB
3PDuoramOnln readwrite 16 1us 100gbit 2
0.012897 s
3PDuoramOnln readwrite 16 1us 100gbit 2
0.104166666666667 KiB
3PDuoramTotl readwrite 16 1us 100gbit 2
0.225875 s
3PDuoramTotl readwrite 16 1us 100gbit 2
12.7916666666667 KiB

Floram read 16 1us 100gbit 2 0.879635 s
Floram read 16 1us 100gbit 2 3837.724609375 KiB

CircuitORAMOnln read 16 1us 100gbit 2 0.313 s
CircuitORAMOnln read 16 1us 100gbit 2 710.625
KiB
CircuitORAMTotl read 16 1us 100gbit 2 0.753 s
CircuitORAMTotl read 16 1us 100gbit 2 4957 KiB
```

What you see here are the four systems (2-party DUORAM, 3-party DUORAM, 2-party FLORAM, 3-party Circuit ORAM), with all except FLORAM showing both the online phase and the total of the preprocessing and the online phase. (FLORAM does not have a separate preprocessing phase.) Each of those seven system/phase combinations shows the time taken for the test run, as well as the average bandwidth used per party.

The output fields are:

- system and phase
- mode (reads, writes, or interleaved reads and writes)
- $\log_2$ of the number of 64-bit words in the ORAM
- one-way latency between the parties (specifying "1us" really means not to add artificial latency, so you end up with the natural latency between dockers, which turns out to be $30\,\mu$s)
- bandwidth between the parties
- number of operations (number of reads or number of writes; interleaved reads and writes do this many reads interleaved with the same number of writes)
- the time in seconds or the bandwidth used in KiB, as indicated

You should see the same set of 14 lines as shown above, though the exact times of course will vary according to your hardware. The bandwidths you see should match the above, however.

If you run the test more than once, you will see means and stddevs of all of your runs.

A.4 Evaluation workflow

A.4.1 Major Claims

Our primary claim is this:

(C1): Under realistic Internet networking conditions, DUORAM outperforms FLORAM (which itself outperforms Circuit ORAM) over a range of ORAM sizes, because it is primarily CPU-bound, while the other schemes are primarily network-bound. Our observation is that it is easier to deploy machines with more local computational power and memory than it is to increase bandwidth or reduce latency between the multiple independent parties running the protocol.

We support this primary claim with the following major claims:

(C2): DUORAM's wall-clock performance changes much less as network conditions change than does FLORAM's.

(C3): DUORAM uses much less bandwidth than FLORAM or Circuit ORAM, and 3P-DUORAM's online bandwidth usage is in fact independent of the ORAM size.

(C4): Even in the less realistic setting where the independent parties running the protocols are colocated, DUORAM's wall-clock performance is better than FLORAM's and Circuit ORAM's, but for a smaller range of ORAM sizes.

(C5): 2P-DUORAM's online performance improves noticeably with increased CPU core availability, unlike FLORAM. (Under our standard network conditions, 3P-DUORAM's online wall-clock time is much smaller than that of FLORAM, regardless of the number of cores.)

A.4.2 Experiments

We provide three sets of experiments: the "small", the "large", and the "scaling" experiments.

The "small" experiments. These experiments support most of our major claims.

(E1): Compare the wall-clock time of 2P-DUORAM, 3P-DUORAM, and FLORAM to do 128 interleaved operations under standard network conditions while varying the ORAM size from 2^{16} to 2^{26} (64-bit items); the results of this experiment appear in Figure 7(a). Supports claim (C1).

(E2): Compare the wall-clock time of 2P-DUORAM, 3P-DUORAM, and FLORAM to do 128 interleaved operations for a 2^{20}-sized ORAM and 30 ms latency, while varying the bandwidth from 10 to 110 Mbit/s; the results of this experiment appear in Figure 7(b). Supports claim (C2).

(E3): Compare the wall-clock time of 2P-DUORAM, 3P-DUORAM, and FLORAM to do 128 interleaved operations for a 2^{20}-sized ORAM and 100 Mbit/s bandwidth, while varying the latency from 10 to 70 ms; the results of this experiment appear in Figure 7(c). Supports claim (C2).

(E4): Compare the bandwidth used by 2P-DUORAM, 3P-DUORAM, and FLORAM to do 128 operations (read, write, or interleaved reads and writes) under standard network conditions while varying the ORAM size from

2^{16} to 2^{26}; the results of this experiment appear in Figures 8(a), 8(b), and 8(c). Supports claim (C3).

(E5): Compare the wall-clock time of 3P-DUORAM, FLORAM, and Circuit ORAM to do 128 read operations under standard network conditions while varying the ORAM size from 2^{16} to 2^{26}; the results of this experiment appear in Figure 9(a). Supports claim (C1).

(E6): Compare the wall-clock time of 3P-DUORAM, FLORAM, and Circuit ORAM to do 128 read operations under colocated network conditions while varying the ORAM size from 2^{16} to 2^{26}; the results of this experiment appear in Figure 9(b). Supports claim (C4).

(E7): Compare the bandwidth used by 3P-DUORAM, FLORAM, and Circuit ORAM to do 128 read operations while varying the ORAM size from 2^{16} to 2^{26}; the results of this experiment appear in Figure 9(c). Supports claim (C3).

To run all seven small experiments:

Preparation: `cd repro`

Execution: `./repro-all-dockers small` *`numops`*

Here, *`numops`* is the number of read, write, or interleaved operations to run in each experiment; the default of 128 is what we used in the paper. Using 128 will require about 10 hours of compute time.

Results: The above command will output the data (up to ORAM sizes of 2^{26}) for Figures 7(a), 7(b), 7(c), 8(a), 8(b), 8(c), 9(a), 9(b), and 9(c), clearly labeled and separated into the data for each line in each subfigure. Running `repro-all-dockers` multiple times will accumulate data, and means and standard deviations will be output for all data points when more than one run has been completed. From this data, one should be able to verify our major claims (though depending on your hardware, the exact numbers will surely vary).

The optional "large" experiments. These experiments do not directly support our major claims, but may optionally be run in case you are curious to see what happens at larger ORAM sizes. These experiments require at least 660 GB of available RAM, which is why they are optional.

(E8): Compare the wall-clock time of 3P-DUORAM, FLORAM, and Circuit ORAM to do 128 read operations under standard network conditions while varying the ORAM size from 2^{28} to 2^{32}; the results of this experiment appear in the rightmost three data points of Figure 9(a).

(E9): Compare the wall-clock time of 3P-DUORAM, FLORAM, and Circuit ORAM to do 128 read operations under colocated network conditions while varying the ORAM size from 2^{28} to 2^{32}; the results of this experiment appear in the rightmost three data points of Figure 9(b).

(E10): Compare the bandwidth used by 3P-DUORAM, FLORAM, and Circuit ORAM to do 128 read operations while varying the ORAM size from 2^{28} to 2^{32}; the results of this experiment appear in the rightmost three data points of Figure 9(c).

To run all three large experiments:

Preparation: `cd repro`

Execution: `./repro-all-dockers large` *`numops`*

Again, *`numops`* is the number of read operations to run in each experiment; the default of 128 is what we used in the paper. Using 128 will require about 40 hours of compute time. Lowering *`numops`* will reduce the time, but not the requirement for 660 GB of available RAM.

Results: The above command will output the data (for ORAM sizes from 2^{28} to 2^{32}) for Figures 9(a), 9(b), and 9(c), similar to the small experiments above.

The "scaling" experiment. This experiment varies the number of cores:

(E11): Compare the online wall-clock time of 2P-DUORAM, 3P-DUORAM, and FLORAM to do 128 read operations under standard network conditions while varying the number of available cores for each party from 4 to 32. The results of this experiment for ORAM sizes of 2^{16}, 2^{20}, and 2^{26} appear in Figures 10(a), 10(b), and 10(c) respectively. Supports claim (C5).

To run the scaling experiment:

Preparation: Reproducing Figure 10 (the effect of scaling the number of cores) is slightly more work because it depends more on your hardware configuration. First, `cd repro`. The top of the script `repro-scaling` in that directory has instructions. Set the variables (in the script, not environment variables) `BASE_DUORAM_NUMA_P0` and `BASE_DUORAM_NUMA_P1` to numactl commands (examples are given in the comments) to divide your system into two partitions as separate as possible: separate NUMA nodes if you have them, otherwise separate CPUs if you have them, otherwise separate cores. If each of your two partitions has n cores, ensure that the elements of `CORESLIST` do not exceed n (of course, you cannot replicate those data points in that case, but the trend should still be apparent). The paper uses values of n up to 32 cores in each partition, so 64 cores in total (P2 can reuse the cores of P0 since P2's primary work is done after P0 and P1's main work has finished).

Execution: `./repro-scaling` *`numops`*, where *`numops`* as before defaults to 128. Using 128 will require about 4 to 5 hours of compute time.

Results: The output will be similar to that described above with clearly labelled data for Figures 10(a), 10(b), and 10(c) (with an additional column for core count).

A.5 Version

Based on the LaTeX template for Artifact Evaluation V20220926. Submission, reviewing and badging methodology followed for the evaluation of this Artifact can be found at https://secartifacts.github.io/usenixsec2023/.