Two Shuffles Make a RAM: Improved Constant Overhead Zero Knowledge RAM

Yibin Yang, *Georgia Institute of Technology;*
David Heath, *University of Illinois Urbana-Champaign*

https://www.usenix.org/conference/usenixsecurity24/presentation/yang-yibin

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 33rd USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 33rd USENIX Security Symposium.

August 14–16, 2024 • Philadelphia, PA, USA

978-1-939133-44-1

USENIX Security '24 Artifact Appendix: Two Shuffles Make a RAM: Improved Constant Overhead Zero Knowledge RAM

Yibin Yang
Georgia Institute of Technology, USA
yyang811@gatech.edu

David Heath
University of Illinois Urbana-Champaign, USA
daheath@illinois.edu

A Artifact Appendix

A.1 Abstract

The artifact includes code for all benchmarks presented in the paper. It mainly includes ZK set/ROM/RAM schemes in our paper. It also includes baseline [FKL+21] and our implementation of [GOT+22].

This document describes how one can use our code to reproduce *all* results in Section 6 of the proceedings paper. For other benchmarks, that are not presented in our proceedings version (some of them are deferred to the full version[1]), please refer to the `README.md` in our repository.

A.2 Description & Requirements

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

GitHub link:
https://github.com/gconeice/improved-zk-ram/tree/d7f6b56e9c630484d3d93a8073fabd7e538c5f9a
Hash tag:
`d7f6b56e9c630484d3d93a8073fabd7e538c5f9a`
We will maintain new versions, and this appendix will be included and updated accordingly in our repository.

A.2.3 Hardware dependencies

Our repository can be executed on a single machine to emulate ZK Prover P and ZK Verifier V using a localhost network. However, our results were tested using two standalone machines: one for ZK Prover P, and another for ZK Verifier V.

We tested our code on two machines each having $\geq$ 32GiB memory. Namely, we used two Amazon Web Services (AWS) EC2 **m5.2xlarge** machines.

Our paper numbers were obtained by testing with x86_64 CPUs, but our repository also supports ARM CPUs (e.g., Apple M1).

A.2.4 Software dependencies

We tested our code on a clean installation of Ubuntu 22.04. Our repository includes simple scripts to install everything starting from a clean installation.

We depend on emp-toolkit[2] (in particular, the VOLE functionalities inside). Our scripts will help you set it up properly. In particular, one of our baselines is [FKL+21], which is included under emp-zk[3]. We include the version of emp-zk we used in the folder `baseline` with a simple script to compile. We made some small changes on emp-zk to simplify testing.

We also tried our code on macOS. It works well but our numbers were *not* generated over macOS.

We use Linux command `tc` to simulate the network with a certain bandwidth.

A.2.5 Benchmarks

None.

A.3 Set-Up

You can simply download our repository and type **"bash setup.sh"**. (You may need root access, e.g. sudo, in some machines.) Just hit 'return/enter' button on the keyboard whenever a question shows. Please read our `README.md` in the repository if you prefer to do it step-by-step yourself.

A.3.1 Installation

Our ZK set/ROM/RAM and implemented baseline ZK RAM in [GOT+22]. You can simply download our repository and type **"bash build.sh"**. Please read our `README.md` in the repository if you prefer to do it step-by-step yourself.

[1] https://eprint.iacr.org/2023/1115.pdf

[2] https://github.com/emp-toolkit

[3] https://github.com/emp-toolkit/emp-zk

Baseline ZK ROM/RAM in [FKL+21]. Please cd in the folder `baseline`. Similarly, you can simply type **"bash build.sh"**.

A.3.2 Basic test

Note that we have two machines P and V. For P and V, goto the folder `build` (resp. the folder `baseline/build`). Let `ip` denote the machine P's IP address. Please set environment variable 'IP=`ip`' on V's machine.

Our ZK set/ROM/RAM and implemented baseline ZK RAM in [GOT+22]. To test our implementation:

P executes: `./bin/test_arith_inset_zk_rom_block 1 44444 12 16 localhost`

V executes: `./bin/test_arith_inset_zk_rom_block 2 44444 12 16 $IP`

If everything goes through, you should see the execution results on P and V.

Baseline ZK ROM/RAM in [FKL+21]. To test our baseline [FKL+21]'s implementation:

P executes: `./bin/test_ram_ro_ram_test 1 44444 12 localhost`

V executes: `./bin/test_ram_ro_ram_test 2 44444 12 $IP`

If everything goes through, you should see the execution results on P and V.

A.3.3 Expected executable files

You should generate the following executable files located in `build/bin/`:

1. `test_arith_inset_zk_rom_block`: Our ZK set data structure.

   Usage: `test_arith_inset_zk_rom_block PARTY PORT logN blockSize IP` where `PARTY`=1,2, `logN` denotes the bit-length of each entry in (set) ROM, and `blockSize` denotes the block size parameter of the product optimization.

2. `test_arith_zk_rom_block`: Our ZK ROM.

   Usage: `test_arith_zk_rom_block PARTY PORT logN blockSize IP` where `PARTY`=1,2, `logN` denotes the bit-length of each entry in ROM, and `blockSize` denotes the block size parameter of the product optimization.

3. `test_arith_zk_ram_block`: Our ZK RAM.

   Usage: `test_arith_zk_ram_block PARTY PORT logN blockSize IP` where `PARTY`=1,2, `logN` denotes the bit-length of each entry in RAM, and `blockSize` denotes the block size parameter of the product optimization.

4. `test_arith_GOT_block`: (Optimized) baseline [GOT+22]'s ZK RAM.

   Usage: `test_arith_GOT_block PARTY PORT logN blockSize IP` where `PARTY`=1,2, `logN` denotes the bit-length of each entry in RAM, and `blockSize` denotes the block size parameter of the product optimization.

You should generate the following executable files located in `baseline/build/bin/`:

1. `test_ram_ro_ram_test`: Baseline [FKL+21]'s ZK ROM.

   Usage: `test_ram_ro_ram_test PARTY PORT logN IP` where `PARTY`=1,2 and `logN` denotes the bit-length of each entry in ROM.

2. `test_ram_ram_test`: Baseline [FKL+21]'s ZK RAM.

   Usage: `test_ram_ram_test PARTY PORT logN IP` where `PARTY`=1,2 and `logN` denotes the bit-length of each entry in RAM.

A.4 Evaluation Workflow

Please set environment variable 'IP=`ip`' on V's machine, where `ip` is the machine P's IP address.

A.4.1 Major Claims

(C1): Parameter ε for accelerating permutation proofs affect the execution time for our ZK set/ROM/RAM as in Figure 10. This is proven by the experiment (E1) described in Section 6 paragraph "Parameter ε for accelerating permutation proofs".

(C2): Our ZK RAM outperforms the baseline [FKL+21]'s ZK RAM. This is proven by the experiment (E2) described in Section 6 paragraph "Our RAM/ROM vs. [26]'s RAM/ROM" whose results are illustrated/reported in Figures 12 and 14.

(C3): Our ZK ROM outperforms the baseline [FKL+21]'s ZK ROM. This is proven by the experiment (E3) described in Section 6 paragraph "Our RAM/ROM vs. [26]'s RAM/ROM" whose results are illustrated/reported in Figures 13 and 14.

(C4): Our ZK RAM outperforms the (optimized) baseline [GOT+22]'s ZK RAM. This is proven by the experiment (E4) described in Section 6 paragraph "Our RAM vs. [20]'s RAM" whose results are illustrated/reported in Figure 15.

(C5): The bottleneck of our ZK ROM/RAM is the Access phase. This is proven by the experiment (E5) described in Section 6 paragraph "Microbenchmarks" whose results are illustrated/reported in Figure 16.

A.4.2 Experiments

(E1): [Figure 10] [10 human-minutes + 0.5 compute-hours of two machines + 32GB memory each machine/party]: Please `cd` to the folder `build`.
Machines we used: AWS EC2 m5.2xlarge.
Preparation: For both machines, let the name of network card be `ens5`, please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 25Mbit burst 100000 limit 10000` (resp. `100Mbit, 500Mbit, 1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is 25Mbps, 100Mbps, 500Mbps, 1Gbps. You can use `iperf` to check it. For each network setting, the following execution needs to be executed repeatedly (i.e., 4 times).
Execution: There are three related executable files: our ZK set, ROM, and RAM. Execute them as:

- Our ZK set:
  For each $\varepsilon = 2,4,8,16,32,64,128$:
  P machine:
  `./bin/test_arith_inset_zk_rom_block 1 44444 20` ε `localhost`
  V machine:
  `./bin/test_arith_inset_zk_rom_block 2 44444 20` ε `$IP`
- Our ZK ROM:
  For each $\varepsilon = 2,4,8,16,32,64,128$:
  P machine:
  `./bin/test_arith_zk_rom_block 1 44444 20` ε `localhost`
  V machine:
  `./bin/test_arith_zk_rom_block 2 44444 20` ε `$IP`
- Our ZK RAM:
  For each $\varepsilon = 2,4,8,16,32,64,128$:
  P machine:
  `./bin/test_arith_zk_ram_block 1 44444 20` ε `localhost`
  V machine:
  `./bin/test_arith_zk_ram_block 2 44444 20` ε `$IP`

Results: The time outputted on P or V's terminal reflects the time (per access) in Figure 10. The total communication (per access) is calculated as the sum of the comm. outputted P's terminal and V's terminal. In the following experiments, we fix $\varepsilon = 16$.

(E2): [Our RAM vs. [FKL+21]'s RAM] [20 human-minutes + 1 compute-hours of two machines + 32GB memory each machine/party]: Please `cd` to the folder `build` for our ZK RAM; and `cd` to the folder `baseline/build` for baseline [FKL+21]'s ZK RAM.
To simplify the re-experiment, we **highly recommend** the reader to use Figures 22 and 23 in our full version[4] to check Figure 12. Figures 22 and 23 include the precise data used to plot Figure 12. Meanwhile, Figure 14 includes *identical* (partial) data from Figures 22 and 23.
Machines we used: AWS EC2 m5.2xlarge.
Preparation: For both machines, let the name of network card be `ens5`, please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 25Mbit burst 100000 limit 10000` (resp. `100Mbit, 500Mbit, 1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is 25Mbps, 100Mbps, 500Mbps, 1Gbps. You can use `iperf` to check it. For each network setting, the following execution needs to be executed repeatedly (i.e., 4 times).
Execution: There are two related executable files: our ZK RAM and baseline [FKL+21]'s ZK RAM. Execute them as:

- Our ZK RAM (`cd` to the folder `build`):
  For each $\ell = 11,12,\cdots,20$:
  P machine:
  `./bin/test_arith_zk_ram_block 1 44444` ℓ `16 localhost`
  V machine:
  `./bin/test_arith_zk_ram_block 2 44444` ℓ `16 $IP`
- Baseline [FKL+21]'s ZK RAM (`cd` to the folder `baseline/build`):
  For each $\ell = 11,12,\cdots,20$:
  P machine:
  `./bin/test_ram_ram_test 1 44444` ℓ `localhost`
  V machine:
  `./bin/test_ram_ram_test 2 44444` ℓ `$IP`

Results: The time printed on V's terminal reflects the number in Figures 22 and 23 (row RAM) of the full version, which is used to plot Figure 12. Note that each point in Figure 12 is defined by $t_{\mathsf{baseline}}/t_{\mathsf{ourRAM}}$.
The comm. outputted on P's terminal is the communication (per access) from P to V. Similarly, the comm. outputted on V's terminal is the communication (per access) from V to P.

(E3): [Our ROM vs. [FKL+21]'s ROM] [20 human-minutes + 0.5 compute-hours of two machines + 32GB memory

[4] https://eprint.iacr.org/2023/1115.pdf

each machine/party]: Please `cd` to the folder `build` for our ZK ROM; and `cd` to the folder `baseline/build` for baseline [FKL+21]'s ZK ROM.
To simplify the re-experiment, we **highly recommend** the reader to use Figures 22 and 23 in our full version[5] to check Figure 13. Figures 22 and 23 include the precise data used to plot Figure 13. Meanwhile, Figure 14 includes *identical* (partial) data from Figures 22 and 23.
Machines we used: AWS EC2 m5.2xlarge.
Preparation: For both machines, let the name of network card be `ens5`, please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 25Mbit burst 100000 limit 10000` (resp. `100Mbit, 500Mbit, 1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is 25Mbps, 100Mbps, 500Mbps, 1Gbps. You can use `iperf` to check it. For each network setting, the following execution needs to be executed repeatedly (i.e., 4 times).
Execution: There are two related executable files: our ZK ROM and baseline [FKL+21]'s ZK ROM. Execute them as:

- Our ZK ROM (`cd` to the folder `build`):
  For each $\ell = 11, 12, \cdots, 20$:
  P machine:
  `./bin/test_arith_zk_rom_block 1 44444 ℓ 16 localhost`
  V machine:
  `./bin/test_arith_zk_rom_block 2 44444 ℓ 16 $IP`
- Baseline [FKL+21]'s ZK ROM (`cd` to the folder `baseline/build`):
  For each $\ell = 11, 12, \cdots, 20$:
  P machine:
  `./bin/test_ram_ro_ram_test 1 44444 ℓ localhost`
  V machine:
  `./bin/test_ram_ro_ram_test 2 44444 ℓ $IP`

Results: The time printed on V's terminal reflects the number in Figures 22 and 23 (row ROM) of the full version, which is used to plot Figure 12. Note that each point in Figure 12 is defined by $t_{\mathsf{baseline}}/t_{\mathsf{ourROM}}$.
The comm. outputted on P's terminal is the communication (per access) from P to V. Similarly, the comm. outputted on V's terminal is the communication (per access) from V to P.

(E4): [Our RAM vs. [GOT+22]'s RAM] [20 human-minutes + 0.5 compute-hours of two machines + 32GB memory each machine/party]: Please `cd` to the folder `build`.
To simplify the re-experiment, we **highly recommend** the reader to use Figures 22 and 24 in our full version[6] to check Figure 15. Figures 22 and 24 include the precise data used to plot Figure 15.
Machines we used: AWS EC2 m5.2xlarge.
Preparation: For both machines, let the name of network card be `ens5`, please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 25Mbit burst 100000 limit 10000` (resp. `100Mbit, 500Mbit, 1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is 25Mbps, 100Mbps, 500Mbps, 1Gbps. You can use `iperf` to check it. For each network setting, the following execution needs to be executed repeatedly (i.e., 4 times).
Execution: There are two related executable files: our ZK RAM and (our implemented and optimized) baseline [GOT+22]'s ZK RAM. Execute them as:

- Our ZK RAM (Note that these experiments overlap with some experiments in E2):
  For each $\ell = 11, 12, \cdots, 20$:
  P machine:
  `./bin/test_arith_zk_ram_block 1 44444 ℓ 16 localhost`
  V machine:
  `./bin/test_arith_zk_ram_block 2 44444 ℓ 16 $IP`
- Baseline [GOT+22]'s ZK RAM:
  For each $\ell = 11, 12, \cdots, 20$:
  P machine:
  `./bin/test_arith_GOT_block 1 44444 ℓ 16 localhost`
  V machine:
  `./bin/test_arith_GOT_block 2 44444 ℓ 16 $IP`

Results: The time printed on V's terminal reflects the number in Figures 22 and 24 (row RAM) of the full version, which is used to plot Figure 15. Note that each point in Figure 15 is defined by $t_{\mathsf{baseline}}/t_{\mathsf{ourRAM}}$.

(E5): [Fine-grained analysis on our ZK RAM/ROM, Figure 16] [20 human-minutes + 0.5 compute-hours of two machines + 32GB memory each machine/party]: Please `cd` to the folder `build`.
Machines we used: AWS EC2 m5.2xlarge.

[5] https://eprint.iacr.org/2023/1115.pdf

[6] https://eprint.iacr.org/2023/1115.pdf

Preparation: For both machines, let the name of network card be `ens5`, please set up the network as follows:

1. `DEV=ens5` (change `ens5` accordingly)
2. If there exists a previous old setting, initialize it: `sudo tc qdisc del dev $DEV root`
3. `sudo tc qdisc add dev $DEV root handle 1: tbf rate 25Mbit burst 100000 limit 10000` (resp. `100Mbit, 500Mbit, 1Gbit`)
4. `sudo tc qdisc add dev $DEV parent 1:1 handle 10: netem`

Recall that the intended network is 25Mbps, 100Mbps, 500Mbps, 1Gbps. You can use `iperf` to check it. For each network setting, the following execution needs to be executed repeatedly (i.e., 4 times).

Execution: There are two related executable files: our ZK RAM and our ZK ROM. Execute them as:

- Our ZK RAM (Note that these experiments overlap with some experiments in E2/E4):
  For each $\ell = 11, 12, \cdots, 20$:
  P machine:
  `./bin/test_arith_zk_ram_block 1 44444` ℓ `16 localhost`
  V machine:
  `./bin/test_arith_zk_ram_block 2 44444` ℓ `16 $IP`
- Our ZK ROM (Note that these experiments overlap with some experiments in E3):
  For each $\ell = 11, 12, \cdots, 20$:
  P machine:
  `./bin/test_arith_zk_rom_block 1 44444` ℓ `16 localhost`
  V machine:
  `./bin/test_arith_zk_rom_block 2 44444` ℓ `16 $IP`

Results: The (decomposed) time printed on V's terminal reflects the number in Figure 16.

A.5 Notes on Reusability

We note that more executable files are also generated from our repository, including those used to perform more experiments in our full version. See `README.md` for a complete list.

Finally, we remark that our ZK set/ROM/RAM can be used as a standalone C++ class, which can be easily re-used in future developments.

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/usenixsec2024/.