Eos: Efficient Private Delegation of zkSNARK Provers

Alessandro Chiesa, *UC Berkeley and EPFL;* Ryan Lehmkuhl, *MIT;*
Pratyush Mishra, *Aleo and University of Pennsylvania;* Yinuo Zhang, *UC Berkeley*

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

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

ARTIFACT EVALUATED usenix ASSOCIATION AVAILABLE
ARTIFACT EVALUATED usenix ASSOCIATION FUNCTIONAL
ARTIFACT EVALUATED usenix ASSOCIATION REPRODUCED

USENIX'23 Artifact Appendix: "Eos: Efficient Private Delegation of zkSNARK Provers"

Alessandro Chiesa
UC Berkeley & EPFL

Ryan Lehmkuhl
MIT[*]

Pratyush Mishra
Aleo & University of Pennsylvania[†]

Yinuo Zhang
UC Berkeley

A Artifact Appendix

A.1 Abstract

We provide Eos, a Rust library that realizes our delegation protocols for zkSNARKs, and includes components of independent interest. EOS relies on, and contributes to, the state-of-the-art arkworks libraries. We generalize the existing arkworks implementations of PIOP and PC schemes of the Marlin zkSNARK to support our new abstractions for secret-shared field elements and polynomials.

A.2 Description & Requirements

Eos can be run on any system with access to a C and Rust compiler. All the experiments we describe in subsequent sections can be run on a single machine, but recreating the experimental setup used in the paper requires the following AWS instances running Ubuntu >18.04:

- *Delegator*: A `r4.xlarge` instance in the `us-west-2` region
- *Worker 1*: A `c5.24xlarge` instance in the `us-west-1` region
- *Worker 2*: A `c5.24xlarge` instance in the `us-west-2` region

This setup emulates the LaptopHB delegator setup described in the paper. To emulate the LaptopLB delegator setup, see Appendix A.2.4. We don't provide instructions for emulating the Mobile delegator setup since this assumes access to specific hardware.

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

A tarball of Eos can be found at this link.

A.2.3 Hardware dependencies

None.

A.2.4 Software dependencies

Eos can be run on any system with access to a C and Rust compiler. The Rust compiler can be installed using this link and the C compiler can be installed from here. Parsing the execution traces from the experiments required Python3 which can be installed from here.

To emulate the LaptopLB delegator setup described in the paper, the bandwidth of the delegator must be throttled to 350 Mbps downlink and 13 Mbps uplink. On Linux platforms, this can be accomplished via the wondershaper package as follows:

```
sudo wondershaper {interface} 350000 13000
```

This can be reset by running:

```
sudo wondershaper clear {interface}
```

A.2.5 Benchmarks

None.

A.3 Set-up

All machines used should accept TCP traffic on ports 8000-10000.

A.3.1 Installation

See the README contained in the artifact.

A.3.2 Basic Test

To run a simple functionality test, navigate to the `experiments/artifact_evaluation` directory and run the `bench_snark_delegator.sh`, `bench_snark_w1.sh`, and `bench_snark_w2.sh` locally within three separate shells. These scripts should finish within 5 minutes (assuming you've already built Eos) with the following output:

```
Running snark delegator with 2^15 constraints
Running snark delegator with 2^16 constraints
...
```

[*] Work partially done while at UC Berkeley.

[†] Work partially done while at UC Berkeley.

After it completes, run `python3 parse.py` from the delegator machine and ensure that no errors occured, i.e., you should get text output that looks something like:

```
SNARK (mode 0, constraints 2^15):
Online latency: 3.7720s
Delegator uptime: 1.7731s
----
Online comm.: 11.5403MB
```

and not:

```
Failure when reading trace for SNARK (mode
0, constraints 2^15) -- try rerunning
```

A.4 Evaluation workflow

A.4.1 Major Claims

(C1): Eos reduces end-to-end latency of proof generation by up to 26× and lowers the delegator's active computation time by up to 1447× with minimal communication overhead. This is proven by the experiment (E1) described in Section 8.3 of our paper whose results are shown in Table 1 and Figures 5, 6, 7, and 8.

(C2): Eos enables proving instances up to 256× larger instances within a memory budget of 3 GB. This is proven by the experiment (E2) described in Section 8.2 of our paper whose results are shown in Table 1.

A.4.2 Experiments

(E1): [10 human-minutes + 10 compute-minutes]: This experiment will confirm the numbers given in Table 1 + Figures 5, 6, 7, and 8 for the LaptopHB and LaptopLB setups. By default, it will run for instance sizes of $2^{15} - 2^{20}$, but can be easily modified to run up to instances of size 2^{25} at the cost of more compute-time.

Note that, if desired, the latency baselines can be recreated by running benchmarks for the Marlin zkSNARK over the BLS12-381 curve locally on the worker and delegator machines.

How to: See the README contained in the artifact for information on how to configure and run the experiments
Results: The latency + communication results should match the upperbounds given in Table 1 and the numbers given in Figures 5, 6, 7, and 8 for the LaptopHB and LaptopLB setups.

(E2): [10 human-minutes + 10 compute-minutes]: This experiment follows a similar workflow as above, but memory-usage is measured for each protocol execution.

How to: Run the same experiment above, except modify the relevant benchmarks/scripts to also output the memory usage. For example, on Linux we can simply prepend `/usr/bin/time -v` to the relevant command.

Results: After running, inspect the execution traces on the delegator (in `results/delegator`) to retrieve the memory usage. Using the `/usr/bin/time` command described above, this can be obtained by looking for the "Maximum resident set size" field. Ensure that this value is consistent with the upperbounds given in Table 1.

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/.