Experimenting with Collaborative zk-SNARKs: Zero-Knowledge Proofs for Distributed Secrets

Alex Ozdemir and Dan Boneh, *Stanford University*

https://www.usenix.org/conference/usenixsecurity22/presentation/ozdemir

This artifact appendix is included in the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium and appends to the paper of the same name that appears in the Proceedings of the 31st USENIX Security Symposium.

August 10–12, 2022 • Boston, MA, USA

978-1-939133-31-1

A Artifact Appendix

A.1 Abstract

This artifact reproduces distributed systems experiments that benchmark the *collaborative zkSNARKs* that we evaluate in our paper.

Some experiments run on Google Cloud Platform, so we will give the evaluators SSH access to one of our machines which has appropriate credentials to launch the experiments.

The artifact includes scripts to re-run a limited version of our experiments, and to re-render our plots.

A.2 Artifact check-list (meta-information)

- **Algorithm:** GSZ20 and SPDZ MCP protocols. Groth16, Marlin, and PLONK zkSNARKs.
- **Compilation:** Rust compiler, nightly.
- **Run-time environment:** Linux, some experiments on GCP
- **Execution:** A distributed protocol
- **How much time is needed to prepare workflow (approximately)?:** 30 minutes
- **How much time is needed to complete experiments (approximately)?:** 30 minutes
- **Publicly available (explicitly provide evolving version reference)?:** Yes, at `https://github.com/alex-ozdemir/multiprover-snark`.
- **Archived (explicitly provide DOI or stable reference)?:** `https://github.com/alex-ozdemir/multiprover-snark/tree/98cc63c7b885ade04989a5505050504ae7f2aac0`

A.3 Description

A.3.1 How to access

The source is available at `https://github.com/alex-ozdemir/multiprover-snark/tree/98cc63c7b885ade04989a5505050504ae7f2aac0`.

We will provide access to a machine that can run the experiments.

A.3.2 Hardware dependencies

At least 8GB of RAM.

A.3.3 Software dependencies

Ubuntu packages: zsh libgmp-dev neovim autoconf pkg-config libtool apache2-dev apache2 dnsmasq-base protobuf-compiler libprotobuf-dev libssl-dev libxcb-present-dev libcairo2-dev libpango1.0-dev tmux units r-base

Rust compiler: nightly after 2022-01-31.

Ripgrep

Mahimahi network emulator, patched as described in the source distribution at `/mpc-snarks/artifact_eval.md`

R libraries: ggplot2, dplyr, readr, scales

A.3.4 Data sets

N/A

A.3.5 Models

N/A

A.3.6 Security, privacy, and ethical concerns

N/A

A.4 Installation

(You can skip this. We'll give you access to a machine with the software set up, and alternatively to a VM that is already set up.)

1. New machine, at least 8GB RAM, 10GB disk
   - Ubuntu 20.04 server
2. Do Ubuntu installation
   - username: user password: user
   - updating took a while
3. `apt install zsh libgmp-dev neovim autoconf pkg-config libtool apache2-dev apache2 dnsmasq-base protobuf-compiler libprotobuf-dev libssl-dev libxcb-present-dev libcairo2-dev libpango1.0-dev tmux units r-base virutalbox-guest-utils`
4. `curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`
   - nightly
5. `cargo install ripgrep`
6. Install the mahimahi shell network emulator
   - clone it
   - apply patches
     - empty PICKY_CXXFLAGS in configure.ac (compiler is pickier now)
     - add mm-rate-to-events to install list in scripts/Makefile.am (need this)
   - `./autogen.sh && ./configure && make -j 8`
   - `sudo sysctl -w net.ipv4.ip_forward=1`
7. Install R libraries: ggplot2, dplyr, readr, scales
8. Set up folder sharing `sudo adduser user vboxsf && sudo systemctl enable virutalbox-guest-utils.service`

A.5 Experiment workflow

1. Give us your public key using HotCRP.
2. Wait for us to confirm that we have granted that key access.
3. `ssh aeval@128.12.176.8`
4. `cd ~/multiprover-snark/mpc-snarks`
5. Run `git clean -fd` to clear any existing data.
6. Check that `git rev-parse HEAD` outputs 98cc63c7b885ade04989a5505050504ae7f2aac0.
7. `cargo build --release`

- You can `cargo clean` first to force a clean build.

5. Optional: run the test suite `./test.zsh`

- If it exits with a zero return code, it was successful.

Now, run the experiments (next section)

A.6 Evaluation and expected results

1. Run all experiments with `time ./analysis/collect/artifact_eval.zsh`

- This should take approximately 24 minutes.
- Alternatively: you can run the experiments one-by-one:

    1. `time ./analysis/collect/bad_net.zsh | tee ./analysis/data/bad_net.csv`

    – This runs locally and should take approximately 6 minutes

    2. `time ./analysis/collect/weak_machines.zsh`

    – This runs on GCP and should take approximately 10 minutes

    3. `time ./analysis/collect/Npc.zsh`

    – This runs on GCP should take approximately 8 minutes

2. Generate all plots: `./analysis/plotting/artifact_eval.zsh`
3. Copy plots to your machine: `scp 'aeval@128.12.176.8:multiprover-snark/mpc-snarks/analysis/plots/*.pdf' .`
4. Analyze:

    1. Varying constraint counts: mpc.pdf should be comparable to Figure 8
        - At large constraint counts, 3PC GSZ should have runtime similar to "Single Prover". The SPDZ MPCs should have approximately twice the runtime.
    2. Varying prover count: Npc.pdf should be comparable to Figure 9
        - Both SPDZ and GSZ should be parabolas. Slowdown should be ~2x and ~1x respectively for 2 parties.
    3. Varying link capacity: bad_net.pdf should be comparable to Figure 10
        - Slowdown should be going to ~2x as bandwidth increases. Plonk should be slower than the others.

A.7 Experiment customization

If you want to reproduce the single-machine experiments (those that vary link capacity using a network emulator) on your machine, follow the directions below.

This is optional. You have already produced this graph on our machine.

A.7.1 Build the collaborative proofs

Download the VM here: https://doi.org/10.5281/zenodo.5889564. User: user. Password: user.

1. `cd ~`
2. `git clone -b artifact-eval https://github.com/alex-ozdemir/multiprover-snark`
3. `cd multiprover-snark/mpc-snarks`
4. `cargo build --release`
5. Optional: run the test suite `./test.zsh`

- If it exits with a zero return code, it was successful.

A.7.2 Collect the data

1. `time ./analysis/collect/bad_net.zsh | tee ./analysis/data/bad_net.csv`

- This should take approximately 6 minutes

A.7.3 Make & inspect the plots

1. Varying numbers of provers

- Run: `Rscript ./analysis/plotting/bad_net.R`
- Output plot: ./analysis/plots/bad_net.pdf
- It should be comparable to Figure 10

A.8 Notes

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.