Hyperproofs: Aggregating and Maintaining Proofs in Vector Commitments

Shravan Srinivasan, *University of Maryland;* Alexander Chepurnoy, *Ergo Platform;* Charalampos Papamanthou, *Yale University;* Alin Tomescu, *VMware Research;* Yupeng Zhang, *Texas A&M University*

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

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

Open access to the Artifact Appendices to the Proceedings of the 31st USENIX Security Symposium is sponsored by USENIX.

A Artifact Appendix

A.1 Abstract

Hyperproofs artifact contains two components: (1) source code of the Hyperproofs vector commitment (VC) scheme and (2) scripts to compare the performance of Hyperproofs aggregation with SNARKs based Merkle-proof aggregation (implemented by Ozdemir *et al.* [3,4]).

We use the Golang bindings of the `mcl` library [2] to implement Hyperproofs. Hyperproofs source code contains three major components: (1) the vector commitment scheme, (2) implementation of the argument system for $\mathcal{L}^{b,\ell}_{\mathsf{BATCH}}$ using the inner-product argument (IPA) proposed by Bünz *et al.* [1], and (3) KZG commitment scheme to optimize the verifier of the IPA.

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

- **Algorithm:** We implement the Hyperproofs vector commitment scheme described in the paper.
- **Compilation:** Hyperproofs require go 1.16 or above and `mcl` requires GCC 9.3.0 and above. Baseline implementation from Ozdemir *et al.* requires rust [3].
- **Run-time environment:** Ubuntu 20.04 or similar with sudo privileges (for `mcl` installation).
- **Hardware:** Our benchmarks used a machine with Intel Core i7-4770 CPU @ 3.40 GHz with 8 cores and 32 GiB of RAM.
- **Execution:** Benchmarks are single-threaded and memory-intensive thus the benchmarking results can vary due to simultaneous usage of resources by other processes. Approximately, micro and macro benchmarks (excluding Com and OpenAll) take 1.5+ hours, and comparison with SNARKs based Merkle-proof aggregation takes 6.5+ hours.
- **Metrics:** Experiments report the execution time of VC operations.
- **Output:** The artifact returns the execution time of benchmarks reported in the paper.
- **Experiments:** At a high level, we evaluate the performance of our VC scheme through micro-benchmarks, macro-benchmarks, and baseline comparison. Instructions to set up and run the experiments are included in the readme of the corresponding project repositories (see App. A.3.1).
- **How much disk space required (approximately)?:** In total, 150 GiB of storage is required. This is because the public parameters of the vector commitment scheme requires 100 GiB and SNARKs based Merkle-proof aggregation requires 50 GiB of storage.
- **How much time is needed to prepare workflow (approximately)?:**

Step	Estimated time (hours)
Software installation	1+
Generating Hyperproof public parameters	1.5+
Generating SNARK [3] public parameters	8+
Total	10.5+

- **How much time is needed to complete experiments (approximately)?:**

Step	Estimated time (hours)
Benchmark Open, Com	6.5+
Other micro/macro-benchmarks	1.5+
Hyperproofs aggregation	4+
SNARK + Merkle aggregation	2.5+
Total	14.5+

- **Publicly available (explicitly provide evolving version reference)?:** Yes (see App. A.3.1).
- **Code licenses (if publicly available)?:** Apache License, Version 2.0
- **Archived (explicitly provide DOI or stable reference)?:** Yes (see App. A.3.1).

A.3 Description

A.3.1 How to access

The stable URL to access the artifact:

https://github.com/hyperproofs/hyperproofs/releases/tag/1.0.0

The latest version of the artifact is available at:

https://github.com/hyperproofs/hyperproofs/

A.3.2 Hardware dependencies

Requires at least 32 GiB of RAM and 150 GiB of storage.

A.3.3 Software dependencies

Requires Ubuntu 20.04 with sudo privileges, go 1.16 or above, rust nightly, GCC 9.3.0 or above, CMake, libgmp, libflint, git, python3 (pandas and matplotlib), curl, and other standard tools.

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

We include the detailed installation instructions in the project repository (see App. A.3.1).

A.5 Experiment workflow

Once the necessary software tools are installed:

- **Setup:** First, run the scripts/hyper-go.sh in hyperproofs-go. This generates the public parameters for the VC scheme, which will be located in the folders pkvk-26 and pkvk-30. Second, run the merkle-snarks-setup.sh script in bellman-bignat. This generates the public parameters for the SNARKs baseline in the folders pedersen-30 and poseidon-30.
- **Benchmarks:** First, run the scripts/hyper-bench.sh in hyperproofs-go. This generates the execution times of various VC operations that constitute micro- and macro-benchmarks reported in the evaluation section of the paper. Moreover, this script also generates the proving and verification times of Hyperproofs aggregation scheme. Second, run the merkle-snarks-bench.sh script in bellman-bignat. This script computes and reports the proving and verification times of SNARK based Merkle-aggregation.

A.6 Evaluation and expected results

The evaluation section of the paper presents: (1) micro-benchmarks, (2) macro-benchmarks, and (3) comparison with SNARK based Merkle-tree aggregation. By running the scripts/hyper-bench.sh, raw data for micro-benchmarks can be obtained. Thus, the micro-benchmarking numbers can be used to directly derive the macro-benchmarks. Additionally, micro-benchmarking script returns the performance of aggregation in Hyperproofs for varying batch sizes. The SNARKs baseline can be obtained by running merkle-snarks-bench.sh.

A.7 Experiment customization

N/A

A.8 Notes

N/A

A.9 Version

Based on the LaTeX template for Artifact Evaluation V20220119.

References

[1] Benedikt Bünz, Mary Maller, Pratyush Mishra, Nirvan Tyagi, and Psi Vesely. Proofs for Inner Pairing Products and Applications. Cryptology ePrint Archive, Report 2019/1177, 2019. https://ia.cr/2019/1177.

[2] Mitsunari Shigeo. mcl: a portable and fast pairing-based cryptography library. https://github.com/herumi/mcl/, 2015. Accessed: 2020-10-14.

[3] Alex Ozdemir. bellman-bignat, 2020. https://github.com/alex-ozdemir/bellman-bignat.

[4] Alex Ozdemir, Riad Wahby, Barry Whitehat, and Dan Boneh. Scaling Verifiable Computation Using Efficient Set Accumulators. In *29th USENIX Security Symposium (USENIX Security 20)*, 2020.