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

USENIX Security '24 Artifact Appendix: Fast RS-IOP Multivariate Polynomial Commitments and Verifiable Secret Sharing

Weihan Li[*,†], Yanpei Guo[*,‡], Kexin Shi[*,§]

A Artifact Appendix

A.1 Abstract

In this paper, We propose PolyFRIM, a multivariate polynomial commitment from *fast Reed-Solomon interactive oracle proof of proximity* with optimal linear prover complexity, 5-25× faster than prior arts while ensuring competent proof size and verification. We also propose PolyFRIM with *one-to-many proof* for proving multiple evaluations to multiple verifiers. PolyFRIM surpasses Zhang et al.'s (Usenix Sec. '22) one-to-many univariate PCS, accelerating proving by 4-7× and verification by 2-4× with 25% shorter proof. Leveraging PolyFRIM, we propose an *asynchronous verifiable secret sharing* (AVSS) scheme FRISS with a better efficiency trade-off than prior arts from multivariate PCS, including Bingo (Crypto '23) and Haven (FC '21).

In the artifact evaluation, we conduct performance evaluations for PolyFRIM, one-to-many PolyFRIM, FRISS, and some baselines for comparison.

A.2 Description & Requirements

We implement PolyFRIM, one-to-many PolyFRIM, and FRISS using approximately 3.5k code in Rust.

As shown in Figure 2 of the paper, we compare PolyFRIM with transparent multivariate PCSs: HyperPlonk (implemented by ourselves), Virgo (mainly implemented by ourselves), and Bulletproofs (modified from open-source code).

As in Figure 3, we compare one-to-many PolyFRIM (reduced to univariate for fair comparison) with univariate one-to-many PCSs: ZXH$^+$22, KZG (trivially), and AMT.

As in Figure 4, we compare the performance of PCSs in different AVSS schemes (eAVSS, HAVEN, and Bingo) for varying numbers of parties $n = 3t+1$ and size-t polynomials. Note that we omit PCS-irrelevant parts (*e.g.*, broadcast/encryption) and only focus on how PCS improves AVSS.

[*]Beihang University, China.
[†]E-mail: weihan.lee.wickham@gmail.com
[‡]E-mail: gyp2847399255@gmail.com
[§]E-mail: SY2139209@buaa.edu.cn

A.2.1 Security, privacy, and ethical concerns

None.

A.2.2 How to access

The artifact is open-sourced in Git repository `https://github.com/gyp2847399255/PolyFRIM` with the stable URL `https://github.com/gyp2847399255/PolyFRIM/tree/9beb37f643e9c43beea81771d77f4f6235ff7f7c` submitted in the Artifact Evaluation.

A.2.3 Hardware dependencies

We strongly recommend readers compile and run the program on Linux machines since we do not fully test the configurations on other platforms. We developed and tested this artifact with AMD processors and 80 GB RAM.

A.2.4 Software dependencies

As for Linux distributions, we recommend Ubuntu 22.04 LTS as a platform, on which we ran all experiments.

A.2.5 Benchmarks

None.

A.3 Set-up

Follow the "Requirement" of Readme in the Git repository, which will guide you through setting up the required workspace. Specifically, first install `Rust`. For post-installation, ensure everything is set up correctly with: `cargo --version` and `rustup --version`. Then on Linux, you can execute the following code to use the nightly toolchain of Rust: `rustup default nightly`.

We implement Virgo ourselves except for the GKR subprotocol (in Python). To run GKR (`/virgo/bench_gkr.py`, installing `python3` is required.

A.3.1 Installation

Run `cargo bench`.

A.4 Evaluation workflow

A.4.1 Major Claims

Our repository supports tests for PolyFRIM, Virgo, HyperPlonk, one-to-many PolyFRIM, and PCS costs for FRISS.

(C1): PolyFRIM is 6-25$\times$ faster in prover time than HyperPlonk, Virgo, and Bulletproofs. PolyFRIM has a 4-10$\times$ faster verifier and a 2-3$\times$ smaller proof size than HyperPlonk. The verifier time and proof size of PolyFRIM are competitive with Virgo (less than 50% and 10% worse, respectively). As for Bulletproofs, PolyFRIM has a 100-200$\times$ larger proof size but can be 1000$\times$ faster for verification. This is proven by experiment E1 described in Section 5.1 with results reported in Figure 2.

(C2): One-to-many PolyFRIM is 4-7$\times$, 25-100$\times$, and 10^3-$10^6\times$ faster in prover time than ZXH$^+$22, AMT, and KZG, respectively. One-to-many PolyFRIM is 3-9$\times$ (resp., 2-4$\times$) faster in verifier time than AMT (resp., ZXH$^+$22). Our verifier time is concretely faster than KZG when party number $n <\sim 2^{14}$. This is proven by experiment E2 described in Section 5.2 with results reported in Figure 3.

(C3): The dealer time of FRISS caused by the *PCS* is 4-800$\times$ faster than AVSS using a similar method (eAVSS and HAVEN-1/2). Our party time is 0.5-1.2$\times$ of eAVSS and HAVEN-1. Compared with Bingo, FRISS is 10$\times$ slower for the dealer but 10$\times$ faster for parties. This is proven by experiment E3 described in Section 5.3 with results reported in Figure 4.

A.4.2 Experiments

(E1): *[Multivariate PCS]: Test of transparent multivariate PCS as shown in (C1), A.4.1*

PolyFRIM: Run `cargo bench -p pcs` for the prover and verifier time. It prints *type* (commit, open, *i.e.*, prover, and verifier), *variable* (from 5 to 20, corresponding to polynomial size 2^5 to 2^{20}), and the estimated time (the bold one of three). Run `cargo test -p pcs -- --nocapture` for the proof size. It prints the variable number and corresponding proof size (in bytes).

HyperPlonk: Run `cargo bench -p gemini-fri` for the prover and verifier time. Run `cargo test -p gemini-fri -- --nocapture` for the proof size.

Virgo: Virgo includes two parts: Virgo main protocol provided in our repository, and the Virgo GKR. The final performance is the sum of the two parts. For the main protocol, run `cargo bench -p virgo` for the prover and verifier time. Run `cargo test -p virgo -- --nocapture` for the proof size. For Virgo GKR, in `/virgo`, run `python3 bench_gkr.py` to see the prover time, verifier time, and proof size. To explain as an example, the printed "size 5" means polynomial size 2^5.

Bulletproofs: We modify the open-source code[1], available at `github.com/frolling-paper/bulletproofs`. The modification mainly adds a PCS based on the existing inner product argument. The newest version of nightly Rust led to errors in package `simd`. To solve this, install an older version of nightly Rust. We installed `rust toolchain install nightly-2023-07-16`. Then set `rustup default nightly-2023-07-16`. Run `cargo -bench` for the prover (printed `prover/open point`) time, verifier time, and proof size. The commit time (printed `commit`) is also outputted, and used in AVSS experiment.

We note that except Bulletproofs, the proof sizes are not fixed as the verification size of multiple entries on a Merkle tree is not fixed.

(E2): *[One-to-many univariate PCS]: Test of one-to-many univariate PCS as shown in (C2), A.4.1.*

For all one-to-many univariate PCS, the x-axis is party number n. The relation between n and polynomial size t is $n = 2t$. Here, the verifier time is per time. The prover time is total time. The proof size is per size.

One-to-many PolyFRIM: Run `cargo bench -p vss` for the prover and verifier time of party number 2^{10}-2^{20}. To explain for example, "vss prove 2ˆ10 party" means the prover time of party number 2^{10}. Run `cargo test -p vss -- --nocapture` for the proof size. Here, for example, the printed "vss proof size of 2ˆ10 parties" means party number $n = 2^{10}$.

ZXH$^+$22: We modify the open-source code[2], available at `github.com/frolling-paper/eVSS`. The main modification is changing the code rate and query number of FRI to be the same as ours, as shown in the commit history. First, in the `/Virgo`, run `cmake .`, then run `make -j4` for building. Second, in the `/eVSS`, run `bash transparent_version.sh` to obtain the results. The printed "N" is the party number, "Prove Time (s)" is the prover time, and "Verification Time (s)" is the verifier time. The code does not output the proof size. We contacted the author Jiaheng Zhang for the proof size. For $N = 2^k$, $k \in \{12, 13, \ldots, 20\}$, the proof size (KB) is 97, 115, 134, 154, 175, 202, 224, 252, and 288.

KZG: We follow the open-source code[3]. For setup, make sure the gcc version is 11.4.0 and install the NTL package following the Readme. Then, run `TestDKGandVSS` to see the performance of (trivial) one-to-many KZG for party number $n = 2t$ where polynomial size t varies from 2^5 to 2^{20}. The proof size is always 96 bytes. The prover time is "proof time (s)". The verifier time is "verify time (s)". The commit time is "commit time (s)" (used in AVSS experiment).

[1] `github.com/frolling-paper/bulletproofs`
[2] `github.com/sunblaze-ucb/eVSS`
[3] `github.com/frolling-paper/libpolycrypto`

AMT: In `libpolycrypto/libpolycrypto/test/TestDKGandVSS.cpp`, modify line 159 from "KatePlayer" to "MultipointPlayer". Then, run `make.sh` and `TestDKGandVSS` again to see the performance of AMT for party number n where polynomial size t varies from 2^5 to 2^{20}. The prover time is "proof time (s)". The verifier time is "verify time (s)". The proof size is $96 \cdot \log n$ bytes.

(E3): *[PCS costs for AVSS]: Test of PCS-relevant costs for AVSS (C3), as shown in A.4.1.*

Preparation. For all AVSS schemes, the relation between party number n and polynomial size t is $n = 3t+1$. The performances of most compared schemes are computed, as we only consider PCS cost in the experiment. For preparation, first compute the KZG polynomial commitment performance using the KZG results in experiment (E2). Denote the prover time in (E2) when party number $n = 2t$ is $P_{total,n}$. Then, the prover time P_t of a *single* proof for a size-t polynomial satisfies

$$P_t = P_{total,n}/n.$$

The verifier time V_t of a *single* proof for polynomial size t is the verifier time in (E2). The commit time C_t of a polynomial with size t is also the commit time in (E2).

FRISS: In `Polyfrim/`, run `cargo bench -p avss` for prover and verifier time from party number of $3 \cdot 2^5 + 1$ to $3 \cdot 2^9 + 1$. For example, the printed "avss prove 32 degree and 97 parties" means party number $3 \cdot 2^5 + 1$. "avss prove" means the dealer time. "avss verify" means the verifier time of a *single* proof. As one verifier verifies n proofs, the party time is the multiplication of the single-proof verifier time and party number n.

eAVSS: In eAVSS, the dealer commits one bivariate (t,t)-size polynomial, which equals committing n t-size polynomial. The dealer also generates n proofs on each polynomial, which occupies the majority. Each party verifies n proofs. So the dealer time $D_{eVSS,n} = C_t \cdot n + P_t \cdot n^2$. The party time $PA_{eVSS,n} = V_t \cdot n$.

Bingo: In Bingo, the dealer only needs to commit $t+1$ size-$2t$ polynomials. So the dealer time $D_{Bingo,n} = C_{2t} \cdot (t+1)$. Each party needs to commit a size-$2t$ polynomial and verify n proofs on the polynomial. Hence, the per-party time is $PA_{Bingo,n} = C_{2t} + V_{2t} \cdot n$.

HAVEN-1: In HAVEN-1, the dealer needs to commit one size-$2t$ polynomial, and n size-t polynomials. The dealer also generates n^2 proofs on a size-t polynomial, and n proofs on a size-$2t$ polynomial. So the dealer time is

$$D_{\text{HAVEN}-1,n} = C_{2t} + n \cdot C_t + n^2 \cdot P_t + n \cdot P_{2t}.$$

Each party needs to verify n proofs on the size-t polynomial and n proofs on the size-$2t$ polynomial. So the per-party time is $PA_{\text{HAVEN}-1,n} = n \cdot (V_t + V_{2t})$.

HAVEN-2: The computation of dealer and per-party time is the same as HAVEN-1, except using Bulletproofs. Note that Bulletproofs in the experiment (E1) (commit, prover, and verifier time) results are all in polynomial size instead of party number.

A.5 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/`.