
# USENIX Security '26 Artifact Appendix: Shred-to-Shine Metamorphosis of (Distributed) Polynomial Commitments

Weihan Li

## A Artifact Appendix

### A.1 Abstract

Polynomial commitment schemes (PCSs) enable verifying evaluations of committed polynomials. Multilinear (ML) PCSs from linear codes are valued for fast prover time. Distributed MLPCSs further shorten it by parallelizing commitment and proof generation across multiple provers.

We propose PIP$_{\text{FRI}}$, an FRI-based MLPCS that unites the fast proving of linear-time-encodable-code PCS with the compact proofs and fast verification of Reed–Solomon (RS) PCSs. PIP$_{\text{FRI}}$ has more efficient prover times than existing schemes such as Deepfold (Usenix Security '25), while remaining at least competitive proof size and verifier time.

We also propose DEPIP$_{\text{FRI}}$, the distributed version of PIP$_{\text{FRI}}$. It has full linear speedup: the prover time decreases linearly with the sub-prover number, while retaining the same proof size and verifier time with PIP$_{\text{FRI}}$. DEPIP$_{\text{FRI}}$ is also the first FRI-based distributed MLPCS supporting both a (large) single polynomnial and multiple independent polynomials.

### A.2 Description & Requirements

We describe how to run our non-distributed PCS PIP$_{\text{FRI}}$ and our distributed PCS DEPIP$_{\text{FRI}}$. We then compare our PCSs with state-of-the-art schemes.

#### A.2.1 Security, privacy, and ethical concerns

None.

#### A.2.2 How to access

Our code and data are available at 10.5281/zenodo.17867141.

#### A.2.3 Hardware dependencies

Our non-distributed experiments require nothing special for hardwares. Our distributed experiemnts, however, needs multiple CPU cores to simulate multiple sub-provers, where one core represents one sub-prover. Note that the number of sub-provers should be power-of-two.

Our distributed benchmarks run on an Intel Xeon Platinum 8255C CPU with 24 cores. Hence, the upper bound of sub-provers in this case is 16. It is also feasible to run our experiments on AMD CPUs or CPUs with less cores. However, we recommend to use CPUs with at least 8 cores.

#### A.2.4 Software dependencies

Please compile and run the codes on Linux machines as we do not fully test the configurations on other platforms.

We developed and tested our codes with 32 GB RAM.

#### A.2.5 Benchmarks

None.

### A.3 Set-up

Follow the Readme in the Zenodo Github repository to set up the required workspace. First install `Rust`. For post-installation, ensure everything is set up correctly with: `cargo --version` and `rustup --version`. Then on Linux, execute the following code to use the nightly toolchain of Rust: `rustup default nightly`.

One of our comparisons is Virgo, which uses a GKR subprotocol in Python. To run GKR (`/virgo/bench_gkr.py`), installing `python3` is required.

#### A.3.1 Installation

Run `cargo build`.

#### A.3.2 Basic Test

None.

### A.4 Evaluation workflow

We only validate our paper's key results and claims, and ignore minor results or results determined by the key ones.

Specifically, for non-distributed experiments, we provide the comparison of PIP$_{\text{FRI}}$ with the state-of-the-art DeepFold.

For distributed experiments, we show that DEPIP$_{\text{FRI}}$ has full linear speedup upon PIP$_{\text{FRI}}$ and DEPIP$_{\text{FRI}}$ improves DeVirgo. The state-of-the-art distributed PCS DeVirgo is not open-sourced. To compare DEPIP$_{\text{FRI}}$ with DeVirgo, we

use the implementation of Virgo and estimate DeVirgo's performance assuming that it has full linear speedup.

Some minor results are not provided in this artifact. We do not provide other PCS comparisons (*e.g.*, Orion, PolyFRIM, mZKG and Hyrax in Figure 4) or zero-knowledge PCS comparions. These can be obtained similarly from open-sourced and our implementations, or doubly verified in the DeepFold paper. We do not provide (distributed) SNARK comparisons, which can be calculated from our (distributed) PCS benchmarks and existing (distributed) PIOP ones. We do not provide the evaluations of zero-knowledge machine learning, which can be calculated from our PCS benchmarks and other non-PCS parts from existing implementations.

### A.4.1 Major Claims

**(C1):** Compared with DeepFold, PIP$_{\text{FRI}}$ is 5× faster in committing and 40× faster in opening, for a 14× faster prover time (committing + opening) overall. Its proof size and verifier time are competitive (around 7% worse). This is proven by the experiment (E1), whose results are reported in "**Performance of PIP$_{\text{FRI}}$.**", §6.1 and illustrated in Figure 4.

**(C2):** DEPIP$_{\text{FRI}}$ shows a full linear speedup over PIP$_{\text{FRI}}$. This is proven by the experiment (E2), whose results are reported in §6.2 and illustrated in Figure 6(a).

**(C3):** Compared with DeVirgo, DEPIP$_{\text{FRI}}$ reduces the prover time by 20×. This is proven by the experiment (E3), whose results are reported in §6.2 and illustrated in Table 6.

### A.4.2 Experiments

**(E1):** *[Non-distributed PCS] [30 human-minutes + 30 compute minutes + 32GB memory]: Performance of* PIP$_{\text{FRI}}$ *and* DeepFold *as shown in (C1) when the polynomial size ranges from* $2^{20}$ *to* $2^{25}$ *(if the memory allows).*

**Preparation of PIP$_{\text{FRI}}$:** Please comment out lines 121 (`zk_commit(c, i);`), 236 (`zk_open(c, i);`), 466 (`zk_verify(c, i);`), 539-540 (`bench_single_fft, bench_multi_fft,`) in `pip_fri/benches/bench.rs`. They are not used for (C1).

**Execution of PIP$_{\text{FRI}}$:** Run `cargo bench -p pipfri`. It prints *type* (commit, open, and verify), *variable* (from 20 to 25, corresponding to polynomial size $2^{20}$ to $2^{25}$), and the estimated times (the bold one of three). It also prints the proof size when running the verify part.

**Results of PIP$_{\text{FRI}}$:** Collect the commit, open, verify times and the proof size of PIP$_{\text{FRI}}$. In our experiments, they are shown in Table 1. Our claim (C1) directly follows from the table.

Note that apart from running times, the obtained proof size may also slightly differ from the table, as the proof includes Merkle-tree multi-openings, which are random.

Also, in Table 1, the opening time of PIP$_{\text{FRI}}$ when the polynomial size is $2^{24}$ does not double over that when the size is $2^{23}$. This is because the adjustable parameter of our shred-to-shine framework ($m$ and $\ell$) changes, as shown in **Parameters**, §6.1.

Table 1: Results of (E1) in our experiments

| Scheme | Size | Commit (s) | Open (ms) | Verify (ms) | Proof (KB) |
|---|---|---|---|---|---|
| PIP$_{\text{FRI}}$ | $2^{20}$ | 0.55 | 0.23 | 1.56 | 156 |
| | $2^{21}$ | 1.13 | 0.46 | 1.69 | 176 |
| | $2^{22}$ | 2.34 | 0.94 | 1.87 | 196 |
| | $2^{23}$ | 4.80 | 1.85 | 2.08 | 228 |
| | $2^{24}$ | 9.29 | 1.89 | 2.27 | 262 |
| | $2^{25}$ | 19.11 | 3.69 | 2.38 | 288 |
| DeepFold | $2^{20}$ | 2.85 | 8.19 | 1.50 | 153 |
| | $2^{21}$ | 5.83 | 16.67 | 1.58 | 169 |
| | $2^{22}$ | 11.57 | 33.79 | 1.84 | 194 |
| | $2^{23}$ | 23.79 | 69.76 | 1.90 | 213 |

**Preparation of DeepFold:** Given a 32GB RAM, DeepFold can only run when the polynomial size is no more than $2^{23}$. Hence, modify the lines 11 from `const SIZE: usize = 25;` to `const SIZE: usize = 23;` in `deepfold/benches/bench.rs`.

**Execution of DeepFold:** Run `cargo bench -p deepfold`. It prints *type* (commit, open, and verify), *variable* (from 20 to 23, corresponding to polynomial size $2^{20}$ to $2^{23}$), and the estimated times (the bold one of three). It also prints the proof size when running the verify part.

**Results of DeepFold:** Collect the commit, open, verify times and the proof size of PIP$_{\text{FRI}}$. In our experiments, they are shown in Table 1. Note that apart from running times, the obtained proof size may also slightly differ from the table, as the proof includes Merkle-tree multi-openings, which are random.

**Results of (E1):** As shown in Table 1, compared with DeepFold, PIP$_{\text{FRI}}$ has around 5× faster committing time, 35-37× faster opening time. The verifier time and proof size are competitive. This verifies our major claim (C1).

**(E2):** *[Distributed PCS with full linear speedup] [30 human minutes + 30 compute minutes + 32 GB memory +* $\geq$*8 cores]:* DEPIP$_{\text{FRI}}$*'s full linear speedup as shown in (C2) when the sub-prover number ranges from* 2 *to* 8*. For simiplicity, we fix the polynomial size to be* $2^{20}$.

**Preparation of DEPIP$_{\text{FRI}}$:** Modify the line 46 as `let variable_num: usize = 20;` in `de_pip_fri/examples/de_pip_fri.rs`. This fixes the polynomial size to be $2^{20}$.

**Execution of DEPIP$_{\text{FRI}}$:** Run `cd de_pip_fri`. Run `./run_benchmark.sh <sub-prover number> <running times>`. Set `running time` = 10. Set `sub-prover number` to be 2, 4, or 8.

**Results of DePip$_{\text{FRI}}$:** Collect the commit, open, verify times and the proof size of DePip$_{\text{FRI}}$ from `Process 0`, *i.e.*, the master prover. In our experiments, they are shown in Table 2. Note that apart from running times, the obtained proof size may also slightly differ from the table, as the proof includes Merkle-tree multi-openings, which are random. Also note that our paper run non-distributed and distributed experiments on two machines (as shown in §6), so the performances of Pip$_{\text{FRI}}$ in the two tables are different.

As shown, with the increase of the sub-prover number, the committing and opening times decreaes linearly. Further, the verifier time and proof size do not change greatly. This verifies our major claim (C2).

Table 2: Results of (E2) and (E3) in our experiments

| Scheme | #Prover | Commit (s) | Open (s) | Verify (ms) | Proof (KB) |
|---|---|---|---|---|---|
| Pip$_{\text{FRI}}$ | 1 | 0.88 | 0.33 | 2.18 | 156 |
| | 2 | 0.44 | 0.17 | 2.32 | 156 |
| DePip$_{\text{FRI}}$ | 4 | 0.22 | 0.087 | 2.31 | 154 |
| | 8 | 0.11 | 0.047 | 2.32 | 154 |
| Virgo | 1 | 2.95 | 18.6 + 10.8 | 1.47 + 0.27 | 169 + 51 |
| | 2 | 1.47 | 14.7 | 1.74 | 220 |
| DeVirgo | 4 | 0.74 | 7.4 | 1.74 | 220 |
| | 8 | 0.37 | 3.7 | 1.74 | 220 |

**(E3):** *[Distributed PCS comparison] [30 human minutes + 30 compute minutes + 32 GB memory + ≥8 cores]: Performance of* DePip$_{\text{FRI}}$ *and* DeVirgo *as shown in (C3) when the sub-prover number ranges from 2 to 8 and the polynomial size is* $2^{20}$.

**Preparation, execution and results of Virgo:** We estimate the performance of non-open-sourced DeVirgo from its non-distributed scheme Virgo.

First, modify lines 21-22 as `const SMALL: usize = 20; const SIZE: usize = 20;` in `virgo/benches/bench.rs`. This fixes the polynomial size to be $2^{20}$.

Second, comment out lines 96 (`zk_commit(c, i);`), 204 (`zk_open(c, i);`), and 336 (`zk_verify(c, i);`) in `virgo/benches/bench.rs`. This is because we do not need the zk version here.

Third, run `cargo bench -p virgo` to obtain the **the first-part** performance of the Virgo PCS. It prints *type* (commit, open, and verify) and the estimated times (the bold one of three). It also prints the proof size when running the verify part.

Fourth, obtain **the second-part** performance of the Virgo PCS. Modify line 9 as `for i in range(20, 21):` in `virgo/bench_gkr.py`. This fixes the polynomial size to be $2^{20}$. Run `python3 bench_gkr.py`. It would print the prover time (working during the open phase), verifier time, and the proof size.

Finally, compute the performance of Virgo from the two parts above. The results of our experiments is shown in Table 2. For columns `Open, Verifiy, Proof`, the first term comes from the first part, and the second term comes from the second part.

**Results of** DeVirgo**:** Estimate the performance of DeVirgo assuming that it has full linear speedup. The results is shown in Table 2. As shown, fixing the sub-prover number, the prover time of DePip$_{\text{FRI}}$ is $> 20\times$ faster than DeVirgo. This verifies our major claim (C3).

**Notes on communication:** Our experiments of DePip$_{\text{FRI}}$ prints the communication for each prover. To compare with the unopen-sourced DeVirgo, we estimate its communication using our implementations. Specifically, using lines 104-105 `Net::stats().bytes_recv` and `Net::stats().bytes_sent` helps to record the bytes the current sub-prover recieves or sends. Hence, it is feasible to record the concrete communication overhead of the `de_prover.de_commit_polynomial()` in line 164. Our distributed commitment algorithm is basically the same as DeVirgo for both using the committing algorithm of distributed FRI. Thus we can estimate DeVirgo's communication overhead of distributed commitment using our algorithm. Besides, the communication of DeVirgo's opening algorithm is basically the same as its commitment algorithm. Hence we estimate DeVirgo's total communication by multiplying its commitment overhead by two.

## A.5 Notes on Reusability

We implement Pip$_{\text{FRI}}$ using our shred-to-shine framework Pip and an existing PCS PolyFRIM. The parameters $m$ (*i.e.*, the size of shredded polynomials) are adjustable for the trade-offs between the running times and the proof size. Our framework is also promising to align with other code-based PCSs like WHIR and Basefold. This is left for the future work.

## 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/usenixsec2026/.