zip: Reducing Proof Sizes for Hash-Based SNARGs

Giacomo Fenzi
giacomo.fenzi@epfl.ch
EPFL

Yuwen Zhang
yuwen@succinct.xyz
Succinct

August 8, 2025

Abstract

The argument size of succinct non-interactive arguments (SNARG) is a crucial metric to minimize, especially when the SNARG is deployed within a bandwidth constrained environment.

We present a non-recursive proof compression technique to reduce the size of hash-based succinct arguments. The technique is black-box in the underlying succinct arguments, requires no trusted setup, can be instantiated from standard assumptions (and even when $\mathsf{P} = \mathsf{NP}$!) and is concretely efficient.

We implement and extensively benchmark our method on a number of concretely deployed succinct arguments, achieving compression across the board to as much as 60% of the original proof size. We further detail non-black-box analogues of our methods to further reduce the argument size.

Keywords: hash-based succinct arguments; proof compression

1 Introduction

Succinct non-interactive arguments (SNARGs) are short cryptographic proofs that admit fast verification. SNARGs are widely deployed across different applications in blockchain technology, cloud computing, and decentralized systems.

Proliferation of hash-based succinct arguments. Traditionally, SNARGs based on pairing-friendly elliptic curves such as [Gro16] have been widely deployed thanks to their extremely short proof sizes, which made them very attractive for bandwidth constrained settings such as blockchain. The drawbacks of this approach include the need to use a circuit-specific trusted setup (partially circumvented by schemes such as [KZG10; CHMMVW20; BBBPWM18]) and the need to use expensive public-key cryptography, which was particularly expensive for prover computation.

An alternative approach has thrived: succinct hash-based arguments of knowledge[1] [Kil92; Mic00]. These are succinct arguments that are secure in the *pure* random oracle model [BR93]. Typically, these protocols compile an information theoretical object (such as a probabilistically checkable proof [AS98] or an interactive oracle proof [BCS16; RRR16]) into a succinct argument by using the random oracle to securely realize the oracles and using the Fiat–Shamir [FS86] transformation to remove interactivity. Hash-based succinct arguments have proliferated both as an area

[1] Which we have been told we should not under any circumstance call SHARKs.

of research (see e.g. [AHIV17; BBHR18; ZCF24; ACFY24; CY24]) and practically (see e.g. Succinct, RISC0, Polygoin, StarkNet...). In large part, this is due to the fact that hash-based succinct arguments have a transparent setup and are not tied to public key cryptography, and so can have extremely fast proving and verification times.

Argument size? The drawback of hash-based succinct arguments is that, well, they are not *that* succinct. Proof sizes are relatively large in the order of hundreds of KiB to many MiB. Optimizing argument size for hash-based succinct arguments is a very interesting question, with both practical implications (such as efficient verification in bandwidth constrained contexts) and connections to deep theoretical problems such as the sliding scale conjecture [BGLR93]. An effort to explore this beautiful connection [ACY23] led to STIR [ACFY24] and WHIR [ACFY25], which are currently the most succinct of the hash-based succinct arguments, leading to approximately a $2\times$ argument size reduction against the FRI protocol [BBHR18]. To reduce argument size over FRI, these protocols use very beautiful[2] ideas of recursive code-switching [RR20], rate improvement[3], quotienting and the sumcheck protocol [LFKN92] to achieve this improvement.

In this note, we ask the provocative question:

What if we did none of that?

Non-recursive argument aggregation. We present zip: a black-box transformation which concretely improves the argument size of hash-based SNARGs. The technique reduces the argument size of concretely deployed arguments by between 5% and 40%. Contrary to other techniques, zip has *no recursive overhead* in that it does not require heuristically instantiating the random oracle and proving it. This is particularly important in light of recent attacks on the Fiat-Shamir transformation [KRS25; AY25]. Finally, the prover and verifier cost of our transformation is negligible, and involves *no field operations or hashing.*

2 Techniques

We define the interface of a lossless compression algorithm [Wel84].

Definition 2.1. *A* **lossless compression algorithm** *is a pair of algorithms* $(\mathsf{Compress}, \mathsf{Decompress})$ *such that* $\forall x \in \{0,1\}^*$ *it holds that* $\mathsf{Decompress}(\mathsf{Compress}(x)) = x$.

We present our transformation next.

Construction 2.2. Consider the following ingredients:
- A succinct non-interactive argument $(\mathbf{P}_0, \mathbf{V}_0)$ for a relation $\mathcal{R}$.
- A lossless compression algorithm $(\mathsf{Compress}, \mathsf{Decompress})$ [Wel84].

We construct a succinct non-interactive argument $(\mathbf{P}, \mathbf{V})$ for $\mathcal{R}$ as follows:

$\mathbf{P}(\mathbb{x}, \mathbb{w})$:
1. Compute $\pi_0 \coloneqq \mathbf{P}_0(\mathbb{x}, \mathbb{w})$.
2. Return $\pi \coloneqq \mathsf{Compress}(\pi_0)$.

[2] If we may say so ourselves.

[3] We have been told this is actually rate worsening but this lead to worse acronyms so we will disregard this completely.

$\mathbf{V}(\varkappa, \pi)$:
1. Compute $\pi_0 \coloneqq \mathsf{Decompress}(\pi)$.
2. Return $\mathbf{V}_0(\varkappa, \pi_0)$.

We defer the proof of completeness and knowledge soundness of Construction 2.2 to Appendix A

Evaluation. To evaluate performance of Construction 2.2, we instantiate with the proof systems currently supported by [Eth] and with the schemes of [AHIV17; ACFY24; ACFY25; NA25]. As compression algorithms, we use [Zst] and the `zip` and `gzip` standard utilities. The input proofs can be found at `gfenzi.io/proofs.zip`[4] We report our findings in Table 1. We happily note

Table 1: Argument sizes. When argument size improves we color green and red otherwise

Argument	Original (KiB)	zstd (KiB)	zip (KiB)	gzip (KiB)
SP1 Hypercube	862	528 (61%)	532 (62%)	527 (61%)
Ligerito	248	248 (100%)	246 (99%)	245 (99%)
Ligero	3400	3110 (90%)	3198 (91%)	3196 (91%)
Pico	967	648 (67%)	840 (87%)	646 (67%)
Risc0	218	208 (95%)	216 (99%)	209 (96%)
SP1 Turbo	1410	973 (67%)	1206 (84%)	974 (67%)
STIR	87	85 (97%)	85 (97%)	86 (99%)
WHIR	108	90 (84%)	92 (85%)	92 (85%)
WHIR-p3	138	117 (85%)	116 (85%)	117 (85%)
Zisk	324	240 (74%)	240 (74%)	241 (74%)
ZKM	1060	731 (67%)	936 (86%)	731 (67%)

that zip improves argument sizes across the board, with improvements as large as compressing arguments to 61% of the initial size. Of the $12 \times 3 = 36$ pairs of non-interactive arguments and lossless compression algorithms that we benchmarked, only a single one did not get compressed by our technique, namely Ligerito [NA25] against `zstd`. The latency of our solutions is also minimal, as we benchmarked that `zstd` takes on the order of $6.3ms$ to encode a proof and $1.3ms$ to decode (with no parallelism enabled in either case).

Inefficiencies. For the semi serious part of this note, we will remark the obvious and state that *hash-based SNARGs proofs are supposed to be incompressible....* The fact that compression algorithms are able to actually compress these proofs successfully means that somewhere in the implementation of the proof system some suboptimal choices have been made.

We have identified a few root causes of inefficiency, which we believe account for most of the inefficiency across the proof systems that we benchmarked.

- *Merkle path pruning.* The largest contributor to argument size in hash-based proof systems is (clearly) hashes. Most of these appear within Merkle authentication paths [Mer89], and within a single path are incompressible. When more than one authentication path is requested for a single Merkle tree however some optimizations can be done, such as removing duplicate hashes occurring at the higher levels of the tree. At a high level, when opening a Merkle tree on n elements at t (uniformly random) locations, this reduces the argument size from $\lambda \cdot t \cdot \log n$ to

[4] Please appreciate the irony.

roughly $\lambda \cdot t \cdot (\log n - \log t)$. See [CY24, Chap. 29] for details. Lossless compression algorithms are pretty good at detecting duplication and so can perform versions of this optimization on the fly.

- *Wasted encoding bits.* Most elements in a hash-based proof which are not hash images are elements of some finite field $\mathbb{F}$ (and even the hashes are when the hash used is algebraic such as Poseidon [GKKRRS19]). Typically, these field elements have a bit-size that is close to (but not exactly) a power of two. As an example, BabyBear has $|\mathbb{F}| = 2^{31} - 2^{27} + 1$ which implies that, when encoded as a binary string in $\{0,1\}^{32}$ one bit of the encoding is wasted.

- *Base field elements.* Sometimes, elements in a base field $\mathbb{F}$ of a proof are for simplicity handled as elements of an extension field $\mathbb{H}$ which might have a $2\times$ or $6\times$ larger bit length.

- *Just trash?* Finally, looking at some of the downloaded proofs, one can sometimes find some human readable strings. We have no idea what is up with that.

As Table 1 clearly shows not even implementations that the authors personally wrote are optimal. Perhaps surprisingly, the Julia implementations seem to be the most efficient at encoding proof sizes, hinting at the fact that Rust serialization and deserialization might be part of the problem. See also Appendix B for some somewhat interesting observations.

Thanks for all the fish. The final message of the note is not that one should start piping their shiny hash-based proofs across a compression algorithm to magically shrink their proof sizes. Instead, there are *easy gains* in proof size that one can achieve by carefully engineering their proof serialization. From an engineering standpoint, these seem to be easier to achieve than changing algorithms or parameters (and further will compound with these changes!). We should further develop standards and libraries to help developers of these systems avoid wasting bandwidth and having suboptimal proof sizes.

Acknowledgements

We would like to thank Guillermo Angeris for providing Ligerito proofs and the red in the table, and Jérémi Do Dinh for providing a Ligero proof. Further, we would like to thank Ron Rothblum, Nick Spooner, Gal Arnon, Grzegorz Gluch, Eylon Yogev, Arantxa Zapico and many others for invaluable suggestions on this note.
Work done in part while the first author was on leave at Succinct and at the Simons Institute.

References

[ACFY24] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. "STIR: Reed–Solomon Proximity Testing with Fewer Queries". In: *Proceedings of the 44th Annual International Cryptology Conference*. CRYPTO '24. 2024, pp. 380–413.

[ACFY25] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. "WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification". In: *Proceedings of the 44th Annual International Conference on Theory and Application of Cryptographic Techniques*. EUROCRYPT '25. 2025.

[ACY23] Gal Arnon, Alessandro Chiesa, and Eylon Yogev. "IOPs with Inverse Polynomial Soundness Error". In: *64th IEEE Annual Symposium on Foundations of Computer Science , FOCS 2023, Santa Cruz, CA, USA, November 6-9, 2023*. IEEE, 2023, pp. 752–761.

[AHIV17] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. "Ligero: Lightweight Sublinear Arguments Without a Trusted Setup". In: *Proceedings of the 24th ACM Conference on Computer and Communications Security*. CCS '17. 2017, pp. 2087–2104.

[AS98] Sanjeev Arora and Shmuel Safra. "Probabilistic checking of proofs: a new characterization of NP". In: *Journal of the ACM* 45.1 (1998). Preliminary version in FOCS '92., pp. 70–122.

[AY25] Gal Arnon and Eylon Yogev. "Towards a White-Box Secure Fiat-Shamir Transformation". In: *Proceedings of the 45th Annual International Cryptology Conference*. CRYPTO '25. 2025.

[BBBPWM18] Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, and Greg Maxwell. "Bulletproofs: Short Proofs for Confidential Transactions and More". In: *Proceedings of the 39th IEEE Symposium on Security and Privacy*. S&P '18. 2018, pp. 315–334.

[BBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. "Fast Reed–Solomon Interactive Oracle Proofs of Proximity". In: *Proceedings of the 45th International Colloquium on Automata, Languages and Programming*. ICALP '18. 2018, 14:1–14:17.

[BCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. "Interactive Oracle Proofs". In: *Proceedings of the 14th Theory of Cryptography Conference*. TCC '16-B. 2016, pp. 31–60.

[BGLR93] M. Bellare, S. Goldwasser, C. Lund, and A. Russell. "Efficient Probabilistically Checkable Proofs and Applications to Approximations". In: *Proceedings of the 25th Annual ACM Symposium on Theory of Computing*. STOC '93. 1993, pp. 294–304.

[BR93] Mihir Bellare and Phillip Rogaway. "Random Oracles are Practical: A Paradigm for Designing Efficient Protocols". In: *Proceedings of the 1st ACM Conference on Computer and Communications Security*. CCS '93. 1993, pp. 62–73.

[CHMMVW20] Alessandro Chiesa, Yuncong Hu, Mary Maller, Pratyush Mishra, Noah Vesely, and Nicholas Ward. "Marlin: Preprocessing zkSNARKs with Universal and Updatable SRS". In: *Proceedings of the 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques*. EUROCRYPT '20. 2020.

[CY24] Alessandro Chiesa and Eylon Yogev. *Building Cryptographic Proofs from Hash Functions*. 2024. URL: https://github.com/hash-based-snargs-book.

[Eth] *Ethproofs*. https://www.ethproofs.org.

[FS86] Amos Fiat and Adi Shamir. "How to prove yourself: practical solutions to identification and signature problems". In: *Proceedings of the 6th Annual International Cryptology Conference*. CRYPTO '86. 1986, pp. 186–194.

[GKKRRS19] Lorenzo Grassi, Daniel Kales, Dmitry Khovratovich, Arnab Roy, Christian Rechberger, and Markus Schofnegger. *Starkad and Poseidon: New Hash Functions for Zero Knowledge Proof Systems*. IACR Cryptology ePrint Archive, Report 2019/458. 2019.

[Gro16] Jens Groth. "On the Size of Pairing-Based Non-interactive Arguments". In: *Proceedings of the 35th Annual International Conference on Theory and Applications of Cryptographic Techniques*. EUROCRYPT '16. 2016, pp. 305–326.

[KRS25] Dmitry Khovratovich, Ron D. Rothblum, and Lev Soukhanov. *How to Prove False Statements: Practical Attacks on Fiat-Shamir*. Cryptology ePrint Archive, Report 2025/118. 2025.

[KZG10] Aniket Kate, Gregory M. Zaverucha, and Ian Goldberg. "Constant-Size Commitments to Polynomials and Their Applications". In: *Proceedings of the 16th International Conference on the Theory and Application of Cryptology and Information Security*. ASIACRYPT '10. 2010, pp. 177–194.

[Kil92] Joe Kilian. "A note on efficient zero-knowledge proofs and arguments". In: *Proceedings of the 24th Annual ACM Symposium on Theory of Computing*. STOC '92. 1992, pp. 723–732.

[LFKN92] Carsten Lund, Lance Fortnow, Howard J. Karloff, and Noam Nisan. "Algebraic Methods for Interactive Proof Systems". In: *Journal of the ACM* 39.4 (1992), pp. 859–868.

[Mer89] Ralph C. Merkle. "A certified digital signature". In: *Proceedings of the 9th Annual International Cryptology Conference*. CRYPTO '89. 1989, pp. 218–238.

[Mic00] Silvio Micali. "Computationally Sound Proofs". In: *SIAM Journal on Computing* 30.4 (2000). Preliminary version appeared in FOCS '94., pp. 1253–1298.

[NA25] Andrija Novakovic and Guillermo Angeris. *Ligerito: A Small and Concretely Fast Polynomial Commitment Scheme*. Cryptology ePrint Archive, Report 2025/1187. 2025.

[RR20] Noga Ron-Zewi and Ron Rothblum. "Local Proofs Approaching the Witness Length". In: *Proceedings of the 61st Annual IEEE Symposium on Foundations of Computer Science*. FOCS '20. 2020, pp. 846–857.

[RRR16] Omer Reingold, Ron Rothblum, and Guy Rothblum. "Constant-Round Interactive Proofs for Delegating Computation". In: *Proceedings of the 48th ACM Symposium on the Theory of Computing*. STOC '16. 2016, pp. 49–62.

[Wel84] Terry A. Welch. "A Technique for High-Performance Data Compression". In: *Computer* 17.6 (1984), pp. 8–19. DOI: 10.1109/MC.1984.1659158. URL: https://doi.org/10.1109/MC.1984.1659158.

[ZCF24] Hadas Zeilberger, Binyi Chen, and Ben Fisch. "BaseFold: Efficient Field-Agnostic Polynomial Commitment Schemes from Foldable Codes". In: *Proceedings of the 44th Annual International Cryptology Conference*. Vol. 14929. CRYPTO '24. 2024, pp. 138–169.

[Zst] *zstd*. https://github.com/facebook/zstd.

A Deferred proofs

If you got this far you probably missed the joke.

B Some actually somewhat interesting observations

While exploring the effect of compression and debugging the unexpected effect, we found some interesting things that might be actually worthwhile jotting down.

- *For low degree tests, entropy of the polynomial matters.* Testing the WHIR implementation with a random polynomial against a polynomial that is structured $(1 + x + 2 \cdot x^2 + 3 \cdot x^3 + \dots)$, the compression is significantly better in the structured case (goes from a 5% reduction to a 16% reduction). This is somewhat surprising because the only place where information about this polynomial appears verbatim is in queries for the first round, and even then the data is the Reed–Solomon encoding of the polynomial. We are not sure how the compression algorithm is able to compress in this case, and if this generalizes to other structures. A possible direction is to take a NP arising from a "standard" computation, compress it and check how redundant it ends being.

- *Compressing multiple proofs for the same statement compresses more than compressing each proof individually.* Compressing many proofs for the same Ethereum block, and compressing their concatenation results in a smaller proof than compressing each of them individually. This is somewhat expected, as compression algorithms work better on larger data streams, but the effect is more dramatic than what one would predict (in fact in some cases the compression rate is better than the best individual compression rate). An intuition is that many hash-based proof systems will commit to relatively similar data when run on the same computation, so there will be some duplication when opening the corresponding Merkle trees. This might be somewhat relevant for systems where multiple proofs are requested, as the compression will scale better than expected (and this might apply even to optimally serialized protocols).

- *Compressing multiple proofs (of the same proof system) for different statement does not compress more than compressing each proof individually.* Perhaps surprisingly instead, compressing many proofs from the same system does not seem to give better results compared to compressinging individually.