BinarySpartan: Spartan over binary fields

Srinath Setty

Microsoft Research

Abstract.

Spartan is a SNARK for R1CS that can be instantiated with any multilinear polynomial commitment scheme. We instantiate Spartan over a binary field, using Ligerito as the commitment scheme along with the ring-switching technique of Diamond and Posen; we refer to the instantiation as BinarySpartan. It is transparent, so it requires no trusted setup, and it provides polylogarithmic-sized proofs. Its security rests on a hash function, so it is plausibly post-quantum. We apply well-known optimizations to Spartan and sum-check: the SIMD R1CS of Phalanx; the next multilinear extension of SuperSpartan; sum-check optimizations from Gruen, from Dao and Thaler, and from Bagad, Dao, Domb, and Thaler; and Binius64's byte lookup tables, which speed up the prover's early rounds of Spartan's outer sum-check. All but the last optimization were developed in the context of prime fields and in fact over large prime fields in the elliptic curve group setting. Furthermore, they are widely used in modern Spartan implementations. Thus, BinarySpartan is a natural instantiation of Spartan over binary fields.

We implement and evaluate BinarySpartan end to end. On a MacBook Pro M4 Max (using only its 12 performance cores, and without GPU/Metal acceleration), BinarySpartan proves BLAKE3 at 547,000 hashes/second and SHA-256 at 255,000 hashes/second, including witness generation. These clear the 200,000 hashes/second rate proposed as sufficient for a post-quantum Ethereum transition, as well as the roughly 30,000–180,000 hashes/second a possible post-quantum Bitcoin transition would require. We also evaluate BinarySpartan on the Ethereum Foundation's client-side proving benchmark, where it proves a single SHA-256 of a 2 KiB message in 6.2 ms, making it the fastest scheme in the benchmark suite.

1 Introduction

Spartan [37] is a SNARK for R1CS that can be instantiated with any multilinear polynomial commitment scheme. In a nutshell, Spartan's prover commits to a purported R1CS witness using a polynomial commitment scheme. It then runs two invocations of the sum-check protocol [28] in sequence, the outer and the inner sum-check, which together reduce R1CS satisfiability to an evaluation of the witness polynomial and evaluations of the sparse polynomials that encode the R1CS matrices. The witness evaluation is resolved by the commitment scheme's evaluation argument. The verifier either computes the sparse evaluations on its own or the prover proves them using Spark, Spartan's sparse polynomial commitment scheme.[1]

[1] Spark commits to the matrix polynomials once, in a preprocessing step; only their evaluations are proven online. When the matrices are small enough for the verifier

Spartan has been instantiated over prime fields with group-based commitments [37,38,24,1,22], with hash-based commitments [13,42,35], and with lattice-based commitments [25,21], and over the integers and the rationals [6,9,11]. Its generalization to CCS, SuperSpartan [39], has also been instantiated with hash-based commitments [27]. An early-stopping version of Spartan yields a folding scheme [23], which has lattice-based instantiations [4,5,30,31].

Over binary fields, prior works [7,18] raise efficiency concerns about instantiating Spartan. Flock [7, Remark 3] observes that a straightforward implementation of Spartan over $\mathbb{F}_2$ is inefficient and remarks that "this is why the Spartan protocol is designed for R1CS over large fields". Binius64 [18] raises a separate concern, about Spartan's inner sum-check and sparse polynomial evaluation argument, and adopts a different arithmetization instead. We return to both in Section 2.

A concrete efficiency target is proving 200,000 hashes/second, the rate proposed as sufficient for a post-quantum Ethereum transition [8]. (A possible post-quantum Bitcoin transition, where a block's transaction signatures are replaced by a single proof of their validity, implies a target of the same order, roughly 30,000–180,000 compressions/second; see Appendix A.) We ask: can a binary-field instantiation of Spartan, with only widely-used techniques, reach that proving throughput on standard hash functions (e.g., SHA-256, BLAKE3)? We find that it can: Spartan over binary fields proves BLAKE3 at over 2.7× the target rate and SHA-256 above it.

To this end, we instantiate Spartan with Ligerito [32] as the multilinear commitment scheme along with the ring-switching technique of Diamond and Posen [12] to efficiently commit to binary witnesses. The resulting instantiation, which we call BinarySpartan, is transparent, so it requires no trusted setup, and it provides polylogarithmic-sized proofs. Its security rests on a hash function, so it is plausibly post-quantum. We apply several prior optimizations to Spartan and its underlying sum-check protocol:

- the SIMD R1CS of Phalanx [41] to efficiently prove many R1CS instances defined over the same R1CS matrices;
- the next multilinear extension of SuperSpartan [39] to check input and output consistency across neighboring SIMD instances;
- Gruen's univariate skip and factorization of the equality polynomial eq [14];
- optimizations to the sum-check round work: the sublinear-sized decomposed eq tables of Dao and Thaler [10] and the round-computation improvements of Bagad, Dao, Domb, and Thaler [2]; and
- position-specific byte lookup tables that speed up the early rounds of Spartan's outer sum-check, following Binius64 [19,20]: for the bit-valued linear map there, at each byte position we precompute the 256 subset sums of its basis images and XOR one selected entry per byte.

All but the last optimization were developed in the context of prime fields and in fact over large prime fields in the elliptic curve group setting. Furthermore, they

to evaluate on its own, no such preprocessing is needed and the witness is the only committed polynomial.

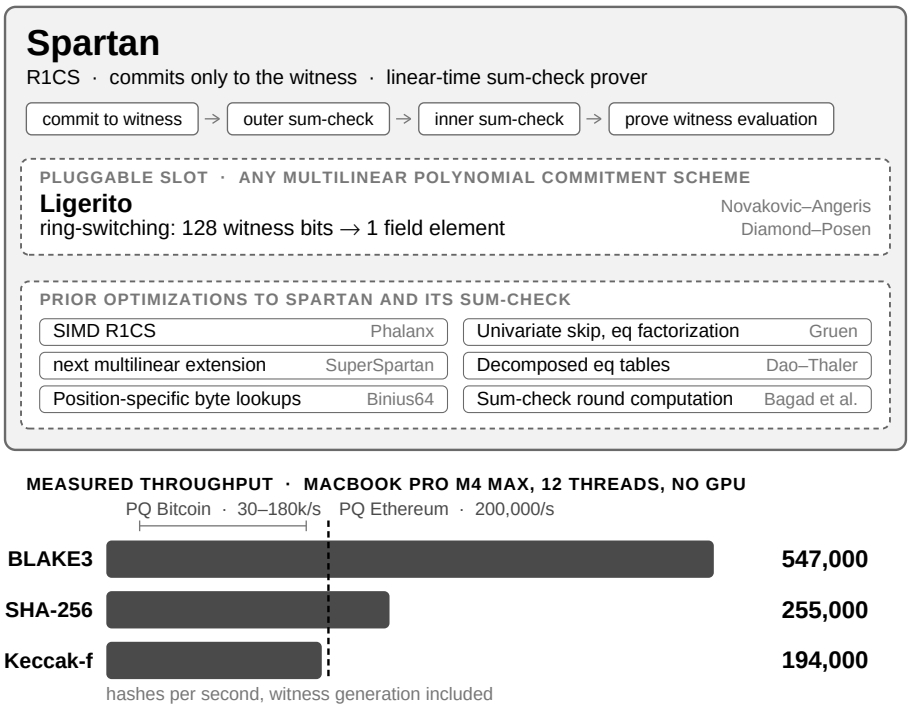

Fig. 1. BinarySpartan at a glance: Spartan [37] over a binary field, instantiated with the Ligerito [32] polynomial commitment scheme and ring-switching [12], along with well-known optimizations to Spartan and to its underlying sum-check [41,39,14,10,2,19,20]. The row below the header is the Spartan protocol. Throughputs are the 12-thread measurements of Table 1 and include witness generation. The 200,000/s line is the rate proposed as sufficient for a post-quantum Ethereum transition [8]; the post-quantum Bitcoin range is derived in Appendix A.

are widely used in modern Spartan implementations [21]. Thus, BinarySpartan is a natural binary-field instantiation of Spartan. We implement and evaluate it end to end. Figure 1 summarizes these ingredients and the throughput they achieve.

Performance. On a MacBook Pro M4 Max, using only its 12 performance cores and with no GPU acceleration, BinarySpartan proves BLAKE3 at 547,000 hashes/second and SHA-256 at 255,000 hashes/second, including witness generation. These clear the 200,000 hashes/second target. BinarySpartan also proves Keccak-f at 194,000 permutations/second.

BinarySpartan's throughput is 10.2–25.9× higher than Plonky3 [33], 4.2–10.6× higher than Binius64 [17], and 2.7× higher than Hashcaster [40]. Of these,

only Binius64 implements all three hash functions: Plonky3 implements BLAKE3 and Keccak, and Hashcaster implements Keccak.

Flock achieves about 1.5× BinarySpartan's throughput [7] (1.47× on BLAKE3, 1.51× on SHA-256, and 1.49× on Keccak-f; Section 3.1). Flock is likewise Spartan over binary fields with well-known optimizations, plus further optimizations of its own; BinarySpartan includes only optimizations that pre-date Flock (Section 2).

BinarySpartan is also fast on a single hash. On the Ethereum Foundation's client-side proving benchmark, it proves one SHA-256 of a 2 KiB message in 6.2 ms and of a 128 B message in 3.5 ms. On the same benchmark, it is 4.2–7.0× faster than Flock, the next fastest scheme, and at least 6.5× faster than Binius64 [17] and Vega [22] (Section 3.2). Flock is optimized for throughput, not necessarily for latency.

GPU/Metal proving. All performance reported in this paper is CPU-only: no prover offloads work to a GPU (e.g., Metal on Apple silicon). Doing so can produce a significant increase in throughput on top of what we report. For instance, an optimization contest has produced a port of Flock's BLAKE3 prover to Metal on Apple silicon [26], which achieves a speedup over the CPU prover. We expect similar speedups in BinarySpartan when offloading work to the GPU.

2 Related work

Spartan's efficiency over binary fields. Prior work raises two efficiency concerns about Spartan for R1CS over $\mathbb{F}_2$: Flock points at Spartan's outer sum-check and notes that a straightforward prover implementation must perform excessive extension field operations [7, Remark 3]. Binius64 [18] points at Spartan's inner sum-check and sparse polynomial evaluation argument, and notes that with one matrix entry per witness bit, the prover pays at least $O(\lambda)$ bit operations per bit, where λ is the security parameter.

The concern about Spartan's outer sum-check was addressed before both Flock and Binius64, by techniques that are not specific to binary fields: Spartan implementations over large prime fields use them too [21]. In more detail, soundness requires challenges from a large extension field, but it does not force the prover to spend an extension field multiplication per witness bit. Gruen's univariate skip [14] binds the first rounds as a single univariate message whose construction weights `eq` by bits, a selection rather than a multiplication, so extension arithmetic begins only on a domain smaller by the skip factor; and the sublinear-sized `eq` tables [10] avoid materializing the linear-sized one.

The concern about Spartan's inner sum-check and sparse polynomial evaluation argument was likewise addressed before both Flock and Binius64, by the batch structure. Phalanx's SIMD R1CS [41] shares one set of matrices across every instance in a batch, so the entry count scales with the block and not the batch. Sparse polynomial evaluation proofs do not arise at all: the shared matrices are small enough for the verifier to evaluate directly, so BinarySpartan never invokes Spark, Spartan's sparse polynomial commitment. Phalanx makes the same observation: evaluating the matrices locally costs time linear in the block and independent of the batch size. Even if a large enough block made Spark

worthwhile, it would apply only to the shared matrices, and so would scale with the block and not the batch.

Comparison with Flock. Flock [7] is the closest related work.

- BinarySpartan is Spartan over binary fields with well-known optimizations.
- Flock is Spartan over binary fields with well-known optimizations and further optimizations of its own.

BinarySpartan includes only optimizations that pre-date Flock. Several of the well-known optimizations are shared: BinarySpartan and Flock both use Phalanx's SIMD R1CS, and both commit with Ligerito [32] and the ring-switching protocol of Diamond and Posen [12]. Both target 100 bits of security, but configure Ligerito differently: BinarySpartan stays in the unique-decoding regime and runs no proof of work, whereas Flock extends Ligerito to the list-decoding regime, taking distance and proximity gaps up to the Johnson bound with proximity-gap grinding.

For batch workloads, Flock achieves 1.47× BinarySpartan's throughput on BLAKE3, 1.51× on SHA-256, and 1.49× on Keccak-f (Section 3.1). Both systems start from Spartan over binary fields with well-known optimizations, so we read this gap as a measure of what Flock's further optimizations add to that baseline. Those optimizations are additive rather than structural, so applying them to BinarySpartan would eliminate the gap.

On a single hash, BinarySpartan is 4.2–7.0× faster (Section 3.2); Flock is optimized for throughput, not necessarily for latency.

Antecedents. The SIMD R1CS formulation and the next extension that BinarySpartan uses both appear earlier in works on delegating computation. Ben-Sasson et al. [3] give a similar R1CS structure for a univariate polynomial IOP. Holmgren and Rothblum [15] give the same next extension, together with an efficient algorithm to evaluate it, where it links adjacent steps of a computation transcript.

3 Implementation and evaluation

We implement BinarySpartan as a single Rust crate. Arithmetic is in $\mathbb{F}_{2^{128}}$ modulo $x^{128} + x^7 + x^2 + x + 1$. The witness is a vector of bits: ring-switching packs $2^7 = 128$ of them into one field element, so committing to 128 witness bits is equivalent to committing to a single field element. We apply Gruen's univariate skip over the first 4 rounds of Spartan's outer sum-check.

Security. We target 100 bits of security. We configure Ligerito to recurse until at most 8 variables remain, then send the terminal polynomial in the clear. Our soundness accounting assumes only the unique-decoding bound, not the wider Johnson radius, and the query count alone provides the 100 bits: we run no proof of work, so no grinding is counted. Summing the query-consistency and proximity-gap terms over every Ligerito level leaves a soundness error of at most $2^{-102.6}$ at the sizes evaluated here, and the bound stays below 2^{-100} up to 2^{28} committed coefficients.

Circuits. We write each hash function as a SIMD R1CS circuit over bits: SHA-256 in 30,376 constraints per compression, BLAKE3 in 14,000, and Keccak-f in 38,400 per permutation. Following Hashcaster, our Keccak encoder packs three permutations into one instance. A single permutation fills only 61% of its padded witness block and three fill 92%, which raises Keccak throughput by 1.2× and makes the Keccak comparison like-for-like.

For the single-hash latency experiments, which use SHA-256, one instance holds one compression, and SuperSpartan's `next` multilinear extension [39] links each instance's output to the following instance's input. The chain's first input is pinned at instance 0, but nothing reads its last output, so we add a tail matrix: linear rows that pin chosen bits of one chosen instance to public values. We use it to bind the final digest as a public output and to check the padding bytes against the claimed message length. Which instance is last depends on the message length, so the index is public and travels with the instance rather than the shape, and one circuit serves every message length. The rows pin existing variables, taking SHA-256 to 31,656 constraints at no additional commitment cost.

Measurement method. All measurements are on a MacBook Pro M4 Max with 12 performance cores, 4 efficiency cores, and 128 GB of memory, without GPU acceleration, on an otherwise idle machine. We run every prover on the 12 performance cores, and leave the efficiency cores idle. Using all 16 cores, including the 4 efficiency cores, increases BinarySpartan's throughput by only 2%, but reduces Flock's throughput by 3–10% and Binius64's by 15.7–17.2%; the performance cores alone are also where Flock is designed to run [36]. Scheduling BinarySpartan's prover well across heterogeneous cores is future work.

Every prover runs through its own upstream benchmark code, including its within-point repetition policy. We sweep each configuration over a range of batch sizes, let each prover use its own encoder, and report the peak over batch sizes. For BinarySpartan, each point is the best of five repetitions. To check BinarySpartan's repeatability, we additionally repeat the full sweep in five independently shuffled rounds and average those rounds rather than select the fastest one. The relative standard deviations at the reported peaks are 0.6% for SHA-256, 0.7% for BLAKE3, and 1.9% for Keccak. Reported times include witness generation. BinarySpartan's proof sizes are uncompressed `bincode` serializations.

We follow Flock's experimental setup: the baselines, their pinned commits, and their configurations are Flock's, and we run all of them through Flock's own benchmark harness at `8790722`, which builds each baseline from its pinned commit and runs it, and which is also where we measure Flock [7] itself. That harness pins Binius64 [17] at `8f21b348`, Plonky3 [33] at `109e95c1`, and Hashcaster [40] through the `bench-hash-in-snark` harness at `1af6fc55`; it also patches Binius64 to generate its witness in parallel, since its stock witness generator is single-threaded, a change in Binius64's favor.

The baselines do not all target the same security level. Plonky3's default configuration targets 113 bits under proximity-gap conjectures and proves only 65; Flock reconfigures it to 206 queries with 16-bit query grinding to reach roughly 100 proven bits, and that reconfigured Plonky3 is the one we measure. The

Prover	BLAKE3	SHA-256	Keccak-f
Plonky3	53,683	—	7,514
Binius64	51,717	44,401	46,011
Hashcaster	—	—	72,670
Flock	805,556	384,714	290,068
BinarySpartan	546,742	255,105	194,300

Table 1. End-to-end throughput in hashes/second at 12 threads, including witness generation. A hash here is one SHA-256 or BLAKE3 compression function or one Keccak-f permutation. Keccak rows pack three permutations per instance except Binius64 and Plonky3, which do not pack.

stronger setting is the fair one to compare against, but it is slower than Plonky3's default, so our margin over Plonky3 reflects it rather than stock Plonky3. Binius64 targets 96 bits and Hashcaster 100, against our 100, so our margin over Binius64 is if anything understated.

3.1 Throughput

We compare against four provers: Flock [7], Binius64 [17], Plonky3 [33], and Hashcaster [40]. Flock, Binius64, and BinarySpartan prove all three hash functions; Plonky3 covers two and Hashcaster only Keccak.

Table 1 gives the results. BinarySpartan's throughput is 10.2× higher than Plonky3's on BLAKE3 and 25.9× higher on Keccak, 4.2–10.6× higher than Binius64's, the only other prover that covers all three hash functions, and 2.7× higher than Hashcaster's, which comes closest from below but implements only Keccak. Flock achieves 1.47× BinarySpartan's throughput on BLAKE3, 1.51× on SHA-256, and 1.49× on Keccak-f. Of Flock's 6.3–15.6× advantage over Binius64, BinarySpartan already attains 4.2–10.6×. As Section 2 discusses, the gap between BinarySpartan and Flock is not structural: Flock's further optimizations are additive, so applying them to BinarySpartan would close it.

These peaks are attained at the following batch sizes: BinarySpartan at 2^{15} BLAKE3, 2^{14} SHA-256, and 24,576 Keccak permutations; Flock at 2^{18}, 2^{16}, and 2^{16}; Binius64 at 2^{14} throughout; Plonky3 at 2^{14} BLAKE3 and 2^{12} Keccak; and Hashcaster at 2^{18}. Throughput declines past each peak.

BinarySpartan proves BLAKE3 at 547,000 hashes/second and SHA-256 at 255,000 hashes/second, clearing the 200,000 hashes/second target proposed as sufficient for a post-quantum Ethereum transition [8]. It also clears the comparable target implied by a possible post-quantum Bitcoin transition (Appendix A).

At a common batch of 2^{14} BLAKE3 hashes, BinarySpartan's proof is 519 KiB and Flock's is 357 KiB. BinarySpartan sends an independent Merkle path for every query instead of deduplicating the nodes those paths share, so it repeats them. Compressing both serialized proofs with `gzip -9` recovers most of that redundancy and narrows the gap to 422 KiB against 355 KiB; we report this as a proxy for what explicit path deduplication would achieve.

Message	Binius64	Vega$_{\mathsf{SC}}$	Vega$_{\mathsf{MC}}$	Flock	BinarySpartan
128 B	49.31	24.65	25.35	24.26	3.47
256 B	49.92	25.14	28.23	24.53	3.88
512 B	53.80	36.70	32.90	25.04	4.22
1 KiB	56.65	59.89	37.98	25.66	4.75
2 KiB	62.02	104.19	44.23	26.01	6.24

Table 2. Time in milliseconds to prove one SHA-256, at 12 threads. BinarySpartan, Flock, and Binius64 include witness generation; the Vega columns are the online phase only. This table isolates hashing. Vega targets statements over signed data, which combine hashing with signature verification and parsing; its native 256-bit field arithmetic is an advantage on that workload that the binary-field provers here lack. This comparison does not reflect it.

Verification. Deserializing and verifying a proof takes 2.3–4.6 ms at the configurations of Table 1: 2.6 ms at 2^{15} BLAKE3 hashes, 2.3 ms at 2^{14} SHA-256 hashes, and 4.6 ms at 24,576 Keccak permutations. Verifier time grows slowly in the batch size: increasing the SHA-256 batch eightfold from its peak configuration costs only 1.2×.

3.2 Latency

We also measure the time to prove a *single* SHA-256, using the Ethereum Foundation's client-side proving benchmark and its harness [34]. Here we compare against Flock [7], Binius64 [17], and the two configurations of Vega [22], single-circuit (Vega$_{\mathsf{SC}}$) and multi-circuit (Vega$_{\mathsf{MC}}$). For BinarySpartan, we measure all five message sizes in 20 independently shuffled rounds and report the mean. We omit the other systems in the client-side benchmark because their published SHA-256 proving times exceed those in Table 2. Both Vega provers split into a reusable preprocessing phase and an online phase, and we quote the online phase alone. Binius64 and both Vega configurations are zero-knowledge, which BinarySpartan and Flock are not.

Table 2 gives the results. BinarySpartan is fastest at every message size. It is 4.2–7.0× faster than Flock, at least 6.5× faster than either Vega configuration, and 9.9–14.2× faster than Binius64. Flock proves the same relation as BinarySpartan here: it also binds the digest as a public output, checks the padding against the claimed message length, and chains across blocks.

Acknowledgments

We thank Ron Rothblum and Justin Thaler for helpful conversations and comments on a prior version of this paper.

References

1. Arun, A., Setty, S., Thaler, J.: Jolt: SNARKs for VMs via lookups. In: EUROCRYPT (2024)

2. Bagad, S., Dao, Q., Domb, Y., Thaler, J.: Speeding up sum-check proving. Cryptology ePrint Archive, Paper 2025/1117 (2025)
3. Ben-Sasson, E., Chiesa, A., Goldberg, L., Gur, T., Riabzev, M., Spooner, N.: Linear-size constant-query IOPs for delegating computation. In: TCC (2019)
4. Boneh, D., Chen, B.: LatticeFold: A lattice-based folding scheme and its applications to succinct proof systems. Cryptology ePrint Archive, Paper 2024/257 (2024)
5. Boneh, D., Chen, B.: LatticeFold+: Faster, simpler, shorter lattice-based folding for succinct proof systems. Cryptology ePrint Archive, Paper 2025/247 (2025)
6. Bünz, B., Fisch, B., Szepieniec, A.: Transparent SNARKs from DARK compilers. In: EUROCRYPT (2020)
7. Bünz, B., Rothblum, R.D., Wang, W.: Flock: Fast proving for batch Boolean computations. Cryptology ePrint Archive, Paper 2026/1329 (2026)
8. Buterin, V.: Possible futures of the Ethereum protocol, part 4: The verge. `https://vitalik.eth.limo/general/2024/10/23/futures4.html` (2024)
9. Campanelli, M., Hall-Andersen, M.: Fully succinct arguments over the integers from first principles. Cryptology ePrint Archive (2024)
10. Dao, Q., Thaler, J.: More optimizations to sum-check proving. Cryptology ePrint Archive, Paper 2024/1210 (2024)
11. DeStefano, Z., Golub, N., Huang, Z., Zhang, J., Frank, S., Walfish, M.: Spain: Succinct proofs for numerical computations. Cryptology ePrint Archive (2026)
12. Diamond, B.E., Posen, J.: Polylogarithmic proofs for multilinears over binary towers. Cryptology ePrint Archive, Paper 2024/504 (2024)
13. Golovnev, A., Lee, J., Setty, S., Thaler, J., Wahby, R.: Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In: CRYPTO (2023)
14. Gruen, A.: Some improvements for the PIOP for ZeroCheck. Cryptology ePrint Archive, Paper 2024/108 (2024)
15. Holmgren, J., Rothblum, R.D.: Delegating computations with (almost) minimal time and space overhead. In: FOCS (2018)
16. Hülsing, A., Butin, D., Gazdag, S.L., Rijneveld, J., Mohaisen, A.: XMSS: eXtended Merkle Signature Scheme. IRTF RFC 8391 (2018), `https://www.rfc-editor.org/rfc/rfc8391`
17. Irreducible Team: Binius64. `https://github.com/binius-zk/binius64` (2025)
18. Irreducible Team: Binius64 blueprint: Overview. `https://www.binius.xyz/blueprint/overview/` (2025)
19. Irreducible Team: Module for folding words with four russians. `https://github.com/binius-zk/binius64/commit/bd41ba7a916ed403e5fa2e5aecdb82561c6a2f16` (2025)
20. Irreducible Team: NTT lookup for univariate AND reduction. `https://github.com/binius-zk/binius64/commit/9aa33699f3a0c126810df55fcfbd8ef6a5b2d7d0` (2025)
21. Jolt Team: Jolt codebase. `https://github.com/a16z/jolt` (2026)
22. Kaviani, D., Setty, S.: Vega: Low-latency zero-knowledge proofs over existing credentials. In: S&P (2026)
23. Kothapalli, A., Setty, S.: HyperNova: recursive arguments for customizable constraint systems. In: CRYPTO. pp. 345–379 (2024)
24. Kothapalli, A., Setty, S., Tzialla, I.: Nova: Recursive Zero-Knowledge Arguments from Folding Schemes. In: CRYPTO (2022)
25. LayerZero Labs: Akita. `https://github.com/LayerZero-Labs/akita` (2026)
26. Layr Labs: Flock BLAKE3 benchmark challenge. `https://github.com/Layr-Labs/flock-challenge` (2026)

27. Lean Ethereum: leanVM. https://github.com/leanEthereum/leanVM (2026)
28. Lund, C., Fortnow, L., Karloff, H., Nisan, N.: Algebraic methods for interactive proof systems. In: FOCS (Oct 1990)
29. National Institute of Standards and Technology: Stateless hash-based digital signature standard. FIPS 205 (2024), https://doi.org/10.6028/NIST.FIPS.205
30. Nguyen, W., Setty, S.: Neo: Lattice-based folding scheme for CCS over small fields and pay-per-bit commitments. Cryptology ePrint Archive, Paper 2025/294 (2025)
31. Nguyen, W., Setty, S.: Neo and SuperNeo: Post-quantum folding with pay-per-bit costs over small fields. Cryptology ePrint Archive, Paper 2026/242 (2026)
32. Novakovic, A., Angeris, G.: Ligerito: A small and concretely fast polynomial commitment scheme. Cryptology ePrint Archive, Paper 2025/1187 (2025)
33. Plonky3 Team: Plonky3. https://github.com/Plonky3/Plonky3 (2024)
34. Privacy Stewards of Ethereum: Client-side proving benchmarks. https://github.com/privacy-ethereum/csp-benchmarks (2025)
35. PSE: spartan-whir. https://github.com/privacy-ethereum/spartan-whir (2026)
36. Rothblum, R.D.: Personal communication (2026)
37. Setty, S.: Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In: CRYPTO (2020)
38. Setty, S., Lee, J.: Quarks: Quadruple-efficient transparent zkSNARKs. Cryptology ePrint Archive, Report 2020/1275 (2020)
39. Setty, S., Thaler, J., Wahby, R.: Customizable constraint systems for succinct arguments. Cryptology ePrint Archive (2023)
40. Soukhanov, L.: Hashcaster. https://github.com/morgana-proofs/hashcaster (2024)
41. Tzialla, I., Kothapalli, A., Parno, B., Setty, S.: Transparency dictionaries with succinct proofs of correct operation. In: NDSS (2022)
42. World Foundation: ProveKit. https://github.com/worldfnd/ProveKit (2025)

A A rough post-quantum Bitcoin target

The 200,000 hashes/second figure of Section 1 is a target for a post-quantum Ethereum transition [8]. Here we give the analogous target for Bitcoin. The estimate is deliberately very rough: it is meant only to fix the order of magnitude, not to be a precise requirement.

Scenario. Suppose transactions carry hash-based (hence post-quantum) signatures. Rather than embedding the signatures, a block embeds everything except the signatures, together with a single proof attesting to knowledge of valid signatures for its transactions. One proof thus replaces thousands of signatures.

Estimate. A Bitcoin block holds roughly 2,000–3,500 transactions, each carrying a hash-based signature. Bitcoin proposals exist for both stateful and stateless schemes; we take the stateless SLH-DSA (SPHINCS+) standard [29], whose freedom from per-signature state makes it the more robust default. What must be proved is the circuit cost of verification: being data-independent, the circuit pays the worst-case Winternitz chains, so verifying one SLH-DSA-128s signature is about 4×10^3 hash compressions, dominated by the one-time-signature chains of its hypertree (over a FORS few-time signature, plus Merkle paths). A representative block is therefore 8–14 million compressions; we take ≈ 11 million. Blocks arrive about every ten minutes (600 s); leaving margin for propagation and reorganizations, a prover targets 10%–50% of the interval, i.e., 60–300 s. The required rate is then $(11 \times 10^6$ compressions$)/(60$–$300\,\text{s}) \approx 30{,}000$–$180{,}000$ compressions/second—the same order as the Ethereum target, and reaching it for a full block at the tightest budget. BinarySpartan's BLAKE3 (547,000/s) and SHA-256 (255,000/s) throughput clears this range (Section 3.1). Stateful XMSS [16] is about four times cheaper—roughly 10^3 compressions per signature, or 10,000–50,000 compressions/second—whereas the faster SLH-DSA-128f set or an unusually full block pushes the aggressive corner past the Ethereum target.

Caveats. The estimate omits second-order costs. Our measured throughput is for *independent* compressions, whereas a verification circuit links the compressions within a signature—the chains and Merkle paths—through the `next` extension and does auxiliary address and checksum work, so its effective rate is somewhat lower, which tightens this comparison. Keeping the on-chain proof small may also call for recursion: rather than posting a single monolithic proof over an entire block, the per-batch proofs would be recursively composed into one succinct proof. Finally, the proof is computed concurrently with mining: a block's signatures are known once its transactions are fixed, so proving runs alongside the proof-of-work search and finishes well within the block interval.