Recent Advances in Zero-Knowledge Proofs and SNARKs

Faster Provers, Shorter Proofs, and Post-Quantum Security

Justin Thaler

Georgetown University and a16z crypto research

October 28, 2026

NIST Workshop on Privacy-Enhancing Cryptography

Updated on 2026-11-02 (typo fixes, acronym definitions, formatting)

What is a SNARK?

The trivial proof system

Claim

“I know a witness w satisfying this property.”

Proof

Send w.

Verification

Read all of w and check it directly.

A SNARK improves on this baseline for the verifier

Succinct Non-interactive ARgument of Knowledge

Send a proof π shorter than the witness w.

Ideally, checking π is faster than checking w directly.

Zero knowledge adds privacy

The verifier learns nothing about the witness,
beyond the fact that the prover knows it.

Example

Prove a signed credential says “over 18,”
without revealing the credential or birth date.

Prover and verifier costs: ballparks

SNARK verifier costs

Under 1 KB

Pairing-based proofs
With a trusted setup

~60–100 KB

Lattice-based proofs
Post-quantum

Hundreds of KB

Hash-based proofs
Post-quantum

Verification often takes milliseconds.

Prover overheads have fallen dramatically

~Millions×

3+ years ago

~3,000–10,000×

Fastest general-purpose proving systems today

~10×

Some applications today e.g., aggregating Falcon signatures

- **Overhead = total prover work ÷ native checking work.**
  - i.e., single-core proving time ÷ single-core computing time.
- Prover work parallelizes extremely well.

Two historical obstacles to SNARK deployment

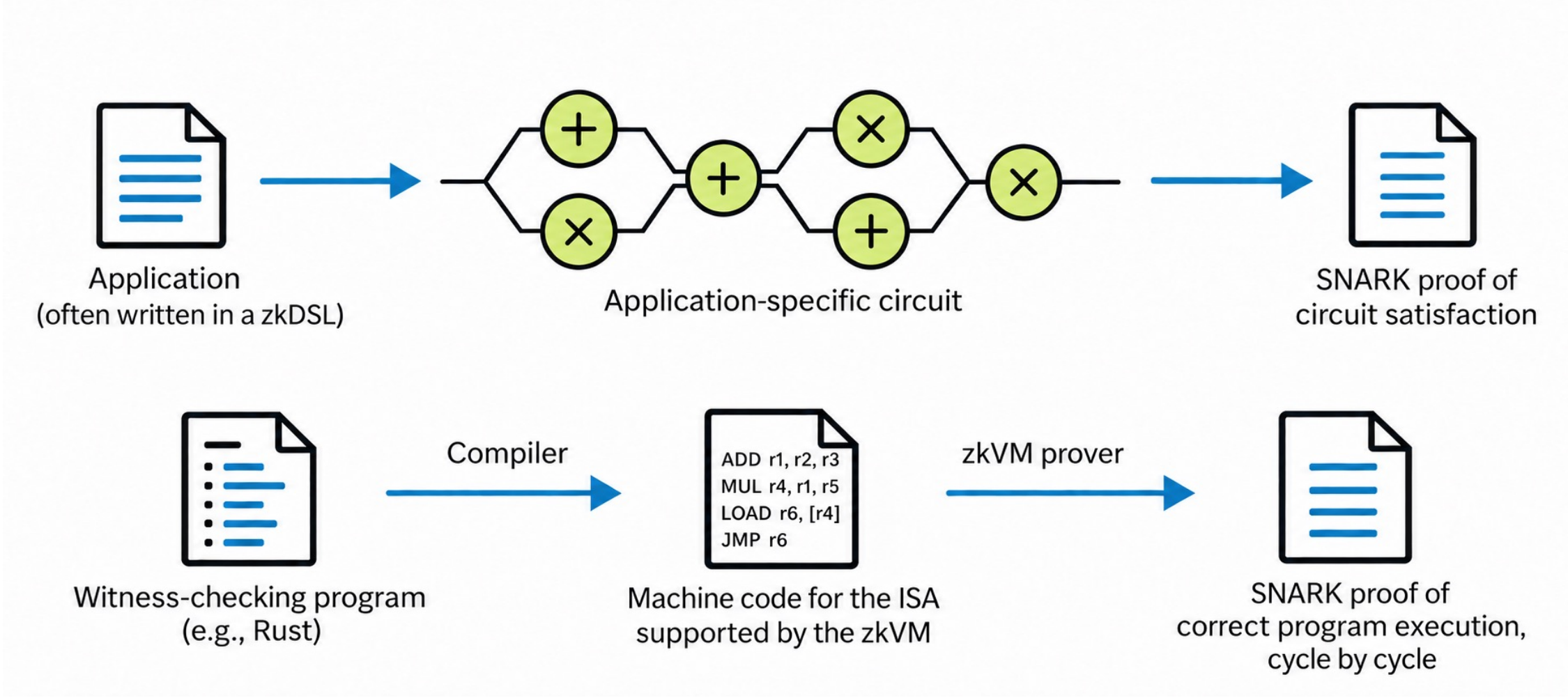
- **Prover time**
- **Bugs**

  SNARK protocols are vastly more complicated than digital signatures and encryption schemes.

  Bugs are discovered regularly, even in production systems today.

Circuits vs. zkVMs

Circuits and zkVMs: two SNARK design paradigms



zkVM: zero-knowledge virtual machine. zkDSL: domain-specific language for writing circuits. ISA: instruction set architecture.

Why I tend to favor zkVMs

- SNARKs for custom circuits tend to have 10–100× faster provers than zkVMs.

Why I tend to favor zkVMs

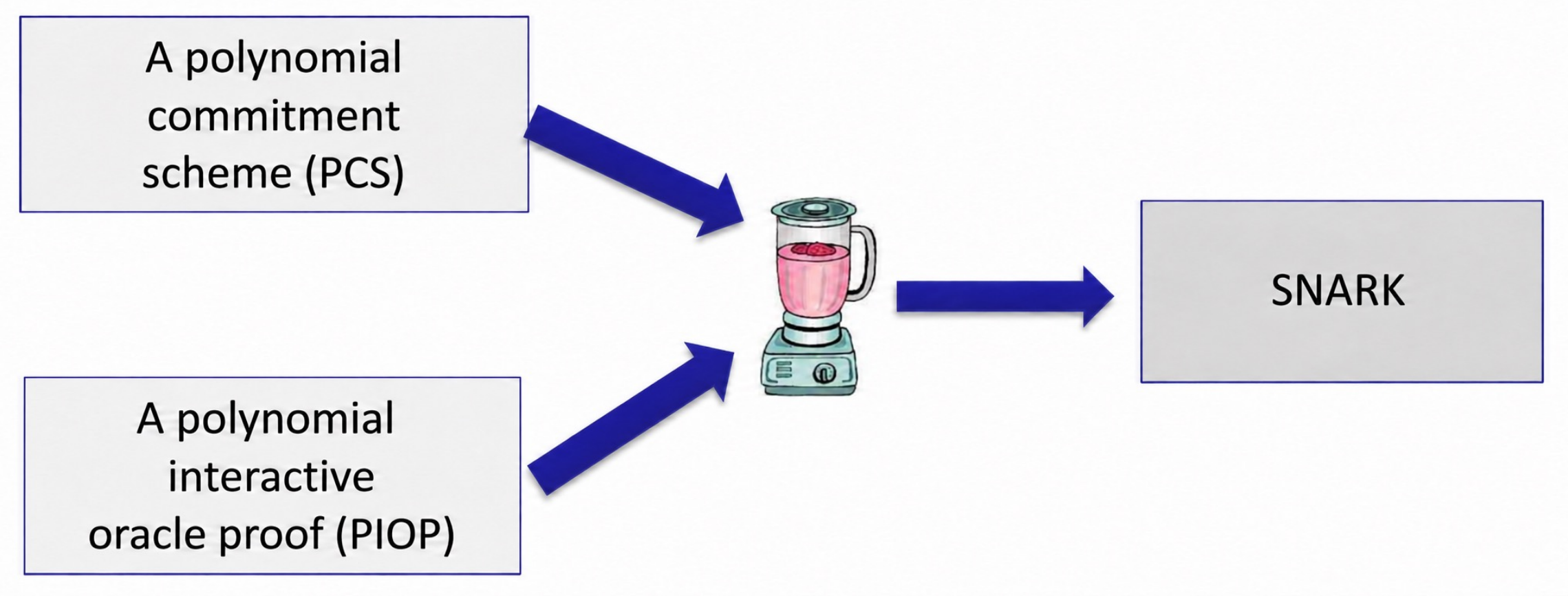
- SNARKs for custom circuits tend to have 10–100× faster provers than zkVMs.
  - But building and maintaining these circuits has historically been a major obstacle.
- zkVMs reuse ordinary languages, compilers, and libraries.
  - No SNARK or circuit expertise needed to develop applications.
- Sometimes proving a VM's execution is the task itself.
  - E.g., blockchains built on the Ethereum Virtual Machine (EVM).

Why I tend to favor zkVMs

- SNARKs for custom circuits tend to have 10–100× faster provers than zkVMs.
  - But building and maintaining these circuits has historically been a major obstacle.
- zkVMs reuse ordinary languages, compilers, and libraries.
  - No SNARK or circuit expertise needed to develop applications.
- Sometimes proving a VM's execution is the task itself.
  - E.g., blockchains built on the Ethereum Virtual Machine (EVM).
- But AI makes circuits easier to build and debug.
- I expect a lasting role for both zkVMs and circuits.
  - Especially combinations of the two.

How SNARKs work
and how to make them faster

Two ingredients for a SNARK



Polynomial interactive oracle proofs (PIOPs)

The prover sends large polynomials to the verifier.

Too large for the verifier to read.
Too large for the PIOP alone to be succinct.

But the verifier only queries their evaluations at a single point.

A PCS turns a PIOP into a succinct argument

Polynomial commitment scheme (PCS)

Replace the PIOP's idealized access with commitments and proofs.

Commit Send a short commitment that binds the prover to W.

Open At a requested point r, prove that $W(r) = v$.

Verify Check using only the commitment and a short evaluation proof.

Fiat–Shamir removes interaction to obtain a SNARK.

Large speedups require faster proving in both layers

PCS commit+proving

Commit to a polynomial extension of the witness (and prove one evaluation query answer)

PIOP proving

Prove validity of committed witness

Large speedups require faster proving in both layers

PCS commit+proving

Commit to a polynomial extension of the witness (and prove one evaluation query answer)

Key techniques for faster proving

Minimize amount of committed data.

Only commit to small values, not random ones.

Small values are faster to commit to.

PIOP proving

Prove validity of committed witness

Sum-check protocol instead of quotienting.

Speed up sum-check proving when its input values are small.

Quotienting PIOPs vs. Sum-Check

The “old” way to build a PIOP: quotienting

Committed values $a = (a_i), \qquad b = (b_i), \qquad c = (c_i)$

Claim: $a_i \cdot b_i = c_i$ for every entry $i \in H$

Example: given a circuit, the prover claims an assignment of wire values satisfies it. For multiplication gate i, a_i and b_i are its assigned input values, and c_i is its output.

(A full circuit proof must also prove correct “wiring”, addition gates, etc.)

The “old” way to build a PIOP: quotienting

Committed values $a = (a_i), \qquad b = (b_i), \qquad c = (c_i)$

Claim: $a_i \cdot b_i = c_i$ for every entry $i \in H$

P commits to polynomials that match the entries

$$A(i) = a_i, \qquad B(i) = b_i, \qquad C(i) = c_i \qquad (i \in H)$$

The claim is equivalent to divisibility by the vanishing polynomial Z_H.

$a_i \cdot b_i = c_i$ for all $i \in H$ if and only if

$$\underbrace{\prod_{i \in H} (X - i)}_{Z_H(X)} \quad \text{divides} \quad A(X) \cdot B(X) - C(X)$$

Prove divisibility by committing to the quotient

1. P commits to the quotient polynomial $Q(X)$

$$A(X)B(X) - C(X) = Q(X)Z_H(X)$$

2. V checks at a random point using PCS openings

$$A(r)B(r) - C(r) = Q(r)Z_H(r)$$

The prover runtime bottleneck

Compute Q, then commit to it.

Even if A, B, C take small values a_i, b_i, c_i on H, Q does not.

The “new” way to build a PIOP: sum-check protocol

Sum-check: LFKN (1990). Efficient implementations: CTY (2011), CMT (2012), T (2013).

For N entries, let $\ell = \log_2 N$ and index entries by ℓ-bit strings x.
Use polynomials A, B, C in ℓ variables, degree at most 1 in each.
Contrast with quotienting: used one-variable polynomials of degree $N - 1$.

V chooses a random point r.

$$\sum_{x \in \{0,1\}^\ell} \mathrm{eq}(r, x) \left(A(x)B(x) - C(x) \right) = 0$$

Even with “structured” $\mathrm{eq}(r, x)$ weights, a violated constraint makes the sum nonzero with high probability—just as with a random linear combination.

No quotient polynomial anywhere.

Costs of the sum-check protocol

Prover time

- P's time: a few field multiplications per term of the sum.
  - Each round, the prover's work roughly halves.

Proof size and rounds

- P sends 2–3 field elements per round.
- One round per variable; typically, $\log_2(N)$ for a sum of N terms.

Why small values make proving faster

Why small values are key to fast proving

What counts as small?

A value that needs far fewer bits to represent than a full 128-bit field element. Examples: a bit (0 or 1), a byte {0, 1, …, 255}, or a short integer {0, 1, …, 2^{32} - 1}.

Faster commitments

Small values cost less to commit to.

Faster sum-check proving

Small input values can make the prover's arithmetic cheaper.

Why small values speed up sum-check proving

Computing the sum is cheaper

Adding and multiplying 32-bit integers is cheaper than adding and multiplying 128-bit field elements.

The savings extend to proving the sum

Recent work gets **P** through most sum-check rounds without general field arithmetic.

Goal: (much) less than one full-field multiplication per term of the original sum.

Example works in this vein: [G24, DDBDT26]

Why committing to small values is fast

Curve-based succinct commitments

$$C = \prod_{i=1}^{N} g_i^{x_i}$$

Commit to the entire vector x via one group element.
If x has small entries, fewer group operations suffice to compute C.

Example: every entry x_i is either 0 or 1.

$$\prod_i g_i^{x_i} = \prod_{i:\, x_i = 1} g_i \qquad (x_i \in \{0, 1\})$$

Why committing to small values is fast

Curve-based succinct commitments

$$C = \prod_{i=1}^{N} g_i^{x_i}$$

Commit to the entire vector x via one group element.
If x has small entries, fewer group operations suffice to compute C.

Example: every entry x_i is either 0 or 1.

$$\prod_i g_i^{x_i} = \prod_{i:\, x_i = 1} g_i \qquad (x_i \in \{0, 1\})$$

Lattice-based

$$A\mathbf{x} = \sum_i x_i A_i$$

For bits $x_i \in \{0,1\}$, just add the selected columns.

Hash-based polynomial commitments

Suppose we have N bits to be committed.

How these commitments work

1. Apply an error-correcting code to the field elements to be committed.
2. Hash the encoded data in a Merkle tree. Its root is the commitment.

Why use a large field?

128-bit security needs a field of at least 128 bits, or costly repetition.

Naïve approach: each of the N bits to be committed is assigned a whole 128-bit field element.

Binius: pack before committing [DP23, DP24]

Specific to binary fields (fields of characteristic two)

Each element of GF(2^{128}) can hold 128 bits.

Pack the N bits to be committed

Pack 128 bits per field element: now there are $N/128$ field elements instead of N.
Then apply the error-correcting code and Merkle-hash the encoded data.

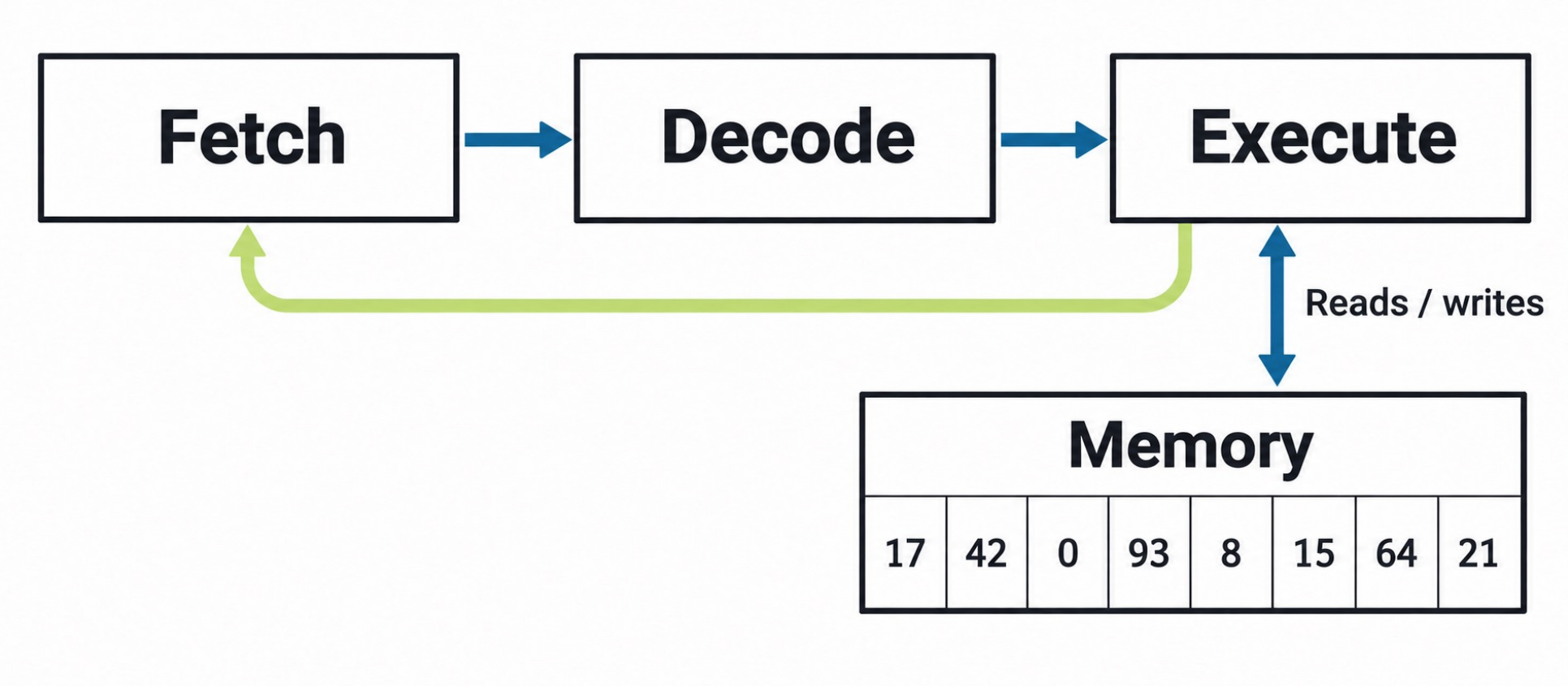
Sum-check connects packed data to the original bits

Sum-check links evaluations of the original and packed polynomials.
Only the packed polynomial needs a commitment.

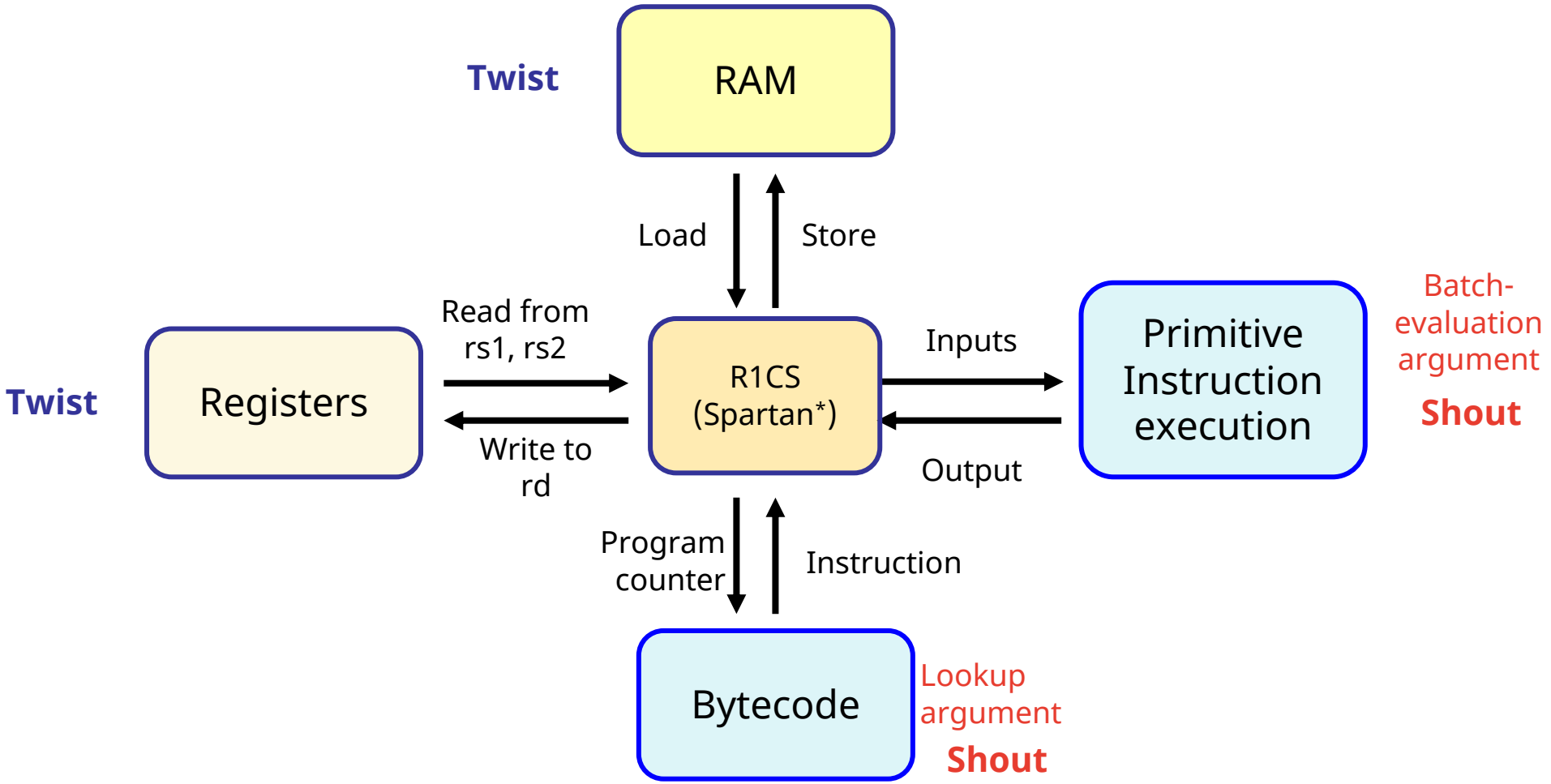
Same N committed bits. 128× fewer field elements for P to encode and hash.

Repeated structure in computation also leads to faster provers

What a CPU does



How the Jolt zkVM proves a CPU execution



R1CS: rank-1 constraint system. *A version of Spartan optimized for the same constraints repeating every CPU cycle (next slide).

Repeated structure in Jolt

Twist and Shout: accesses to the same memory

Batch many reads and writes to the same memory or lookup table.
Commit only small values, with information-theoretically minimal data.

Spartan: the same 19 constraints per CPU cycle

These constraints glue the memory and instruction checks together.

Faster sum-check proving in both

Exploit both small values and the repeated constraint pattern.
This reduces prover work compared with unstructured constraints (details omitted).

Post-quantum SNARKs

Three main families of PCSes

PCS choice largely determines proof size and cryptographic assumptions, including quantum resistance and whether a trusted setup is needed.

Curves: KZG, Bulletproofs/IPA, Dory, etc.

Not post-quantum. KZG needs a trusted setup.
IPA and Dory are transparent (no trusted setup).

Hashing: FRI, STIR, WHIR, etc.

Post-quantum candidates with transparent setup.

Lattices: LaBRADOR, Greyhound, now Akita

Post-quantum candidates with transparent setup.

KZG: Kate–Zaverucha–Goldberg. IPA: inner-product argument. FRI: Fast Reed–Solomon Interactive Oracle Proof of Proximity.

Akita: compact lattice proofs and fast verification

Just Module-SIS + Fiat–Shamir

Similar lattice / Fiat–Shamir foundation to ML-DSA.

60–70 KB

Opening proof

8–16 ms

Verification time

Cheap small-value commitments. Little extra prover memory.

Module-SIS: Module Short Integer Solution problem. ML-DSA: Module-Lattice-Based Digital Signature Algorithm (FIPS 204).

How Akita improves on earlier lattice arguments

PCS	Verifier cost
LaBRADOR / LaZer	Linear in witness size
Greyhound / Hachi	Roughly square-root
RoKoko	Polylogarithmic verifier costs **Nonstandard vSIS (vanishing SIS) assumption**
Akita	Cube- or fourth-root depending on parameters ~8–16 ms in tests

10–94× faster verification than Greyhound
Similar proof sizes (~60–70 KB).

Lattice Jolt: about 10,000× prover overhead

~10,000×

Prover work relative to running the program natively

Proofs are about 100 KB.

Magnetar: binary-field, hash-based zkVM [S26]

A few thousand×

My rough overhead estimate
Without integer multiply/divide

~600 KiB

Reported proof size

Issues still to be addressed: no zero-knowledge, integer multiply/divide, recursion, etc.

What about zero knowledge?

In principle, any non-ZK SNARK can be made ZK with minimal overhead.
In practice, making this fast and secure can be tricky.

Curve-based Jolt: BlindFold [KS25]

Hide the proof messages with commitments, then prove
that the hidden messages pass verification.

Lattice Jolt: ZK coming soon

Using a lattice analogue of BlindFold that we are calling Whiteout.

Hash-based systems: can use transformations like VEIL [DHRR26]

Lightweight techniques exist in principle, but most zkVM base proofs still are not ZK.

ZK: zero-knowledge.

Applications

1. Credential verification
2. Post-quantum signature aggregation: Falcon, then ML-DSA
3. VOPRFs and key transparency

1. Credential verification

mDOC verification in Jolt: about one second

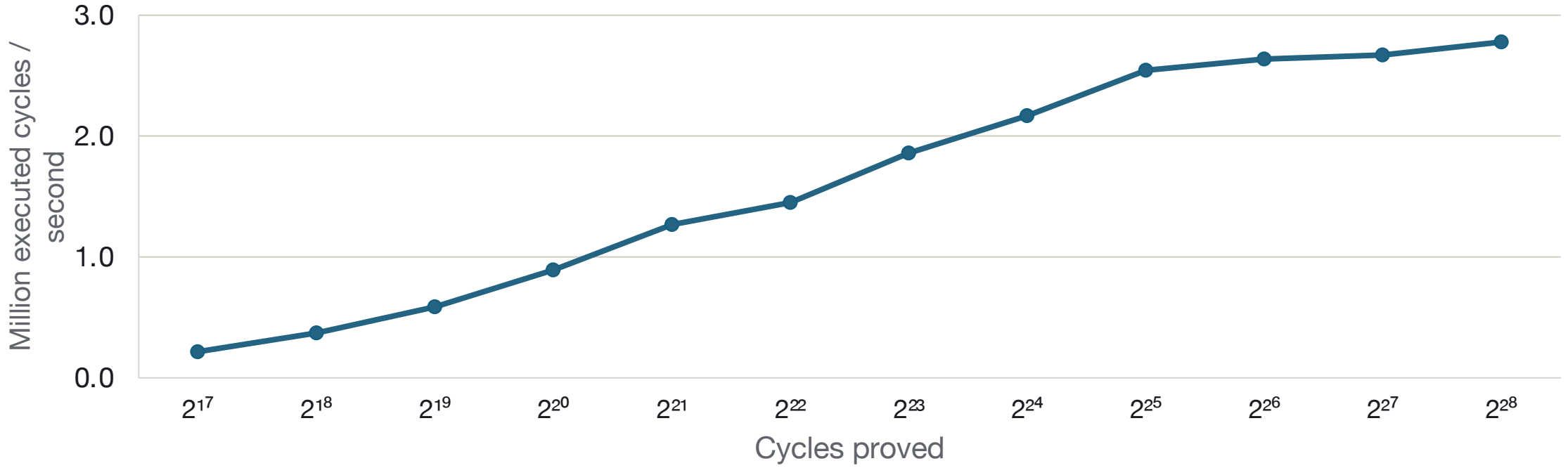
An mDOC is a digitally signed credential stored on a mobile device,
such as a mobile driver's license.

1.03 s	**~13 ms**	**98.1 KiB**
Tracing + proving	Proof verification	Proof size

CPU-only proving on a laptop (Apple M5 Max).
1.04M proved cycles; 87% in two ECDSA P-256 signature checks.
Non-ZK mode. ZK mode for lattice Jolt is coming soon.

ECDSA: Elliptic Curve Digital Signature Algorithm.

Larger computations can be cheaper per cycle proved



SHA-256 chain · M5 Max, 18 CPU threads · proof-only · non-ZK

Larger traces amortize fixed costs and can use faster parameter regimes.

2. Post-quantum signature aggregation

Why post-quantum signatures need aggregation

BLS (Boneh–Lynn–Shacham) signatures support native aggregation

Combine many validator signatures into one group element.
Useful for proof-of-stake blockchain consensus, but not post-quantum.

Post-quantum signatures are much larger

Typically 10–100× larger than curve-based signatures.
No known post-quantum scheme offers BLS-style native aggregation.

Aerie: one succinct proof for many Falcon signatures

Uses Akita and Jolt's PIOP techniques, without the CPU abstraction.
Proves Falcon signatures valid for public messages and public keys.

Aerie aggregates over 50,000 signatures per second

0.65 s

32,768 Falcon-512 signatures

54 ms

Single-core verification

~42 B

Per signature, including salts

Apple M4 Max, CPU only
About 8× single-core proving overhead versus native verification

Next target: aggregate ML-DSA signatures

10,000 signatures/s

Target for proving ML-DSA verification on a MacBook

ML-DSA verification hashes data derived from the signature.
Those hash computations must be proved too.

3. VOPRFs and key transparency

Key transparency: is this really Bob’s public key?

Encryption needs an authentic public key

Alice asks a messaging service for Bob’s key.
If the service substitutes its own key, it can read her messages.

Make key substitutions detectable

Keep an auditable history of account-to-key mappings.
Clients and monitors check for unauthorized changes and inconsistent views.

Transparency must preserve privacy

Publishing phone numbers or emails would expose the user directory.
Use verifiable pseudorandom labels instead.

Where a VOPRF fits in key transparency

VOPRF: a private, verifiable label for an identifier

Client learns $y = F_k(x)$ for a phone number or email x.
Server proves use of the agreed key k without learning x in this step.

Key transparency adds an auditable key directory

Map labels to public keys, and log key changes.
Monitoring and consistency checks make key substitutions detectable.

Beasley [LWZ26]: a post-quantum VOPRF

Uses sum-check and Akita for this cryptographic building block.
Many designs use a VRF. A VOPRF adds privacy for the label computation.

VOPRF: verifiable oblivious pseudorandom function. VRF: verifiable random function.

SNARK issues not discussed today

Prover memory usage

- Peak prover RAM needs can limit the computations we can prove.
- zkVMs other than Jolt address this with SNARK recursion (break CPU executions into small pieces, prove each piece separately, recursively aggregate the proofs).

Security foundations of SNARK recursion

Which assumptions justify the security of proving you know SNARK proofs?

Formal verification

And related efforts to get the bugs out of SNARK implementations

Conclusions

Prover overheads have fallen by orders of magnitude

- Both the PIOP and the PCS must be fast.
- Key techniques:
  - Use sum-check-based PIOPs to minimize the number and size of committed values.
  - Make sure the PCS can quickly commit to small values.
  - Leverage repeated structure in computation for “economies of scale.”

The benefits reach both design paradigms, zkVMs and circuits

Jolt for correct CPU execution of ordinary programs. Aerie for Falcon signature constraints.

Post-quantum options are improving

On the lattice front, Akita combines compact proofs with fast verification.

References: Background

Shafi Goldwasser, Silvio Micali, Charles Rackoff (1985)
The Knowledge Complexity of Interactive Proof-Systems

Amos Fiat, Adi Shamir (1986)
How to Prove Yourself: Practical Solutions to Identification and Signature Problems

Dan Boneh, Craig Gentry, Ben Lynn, Hovav Shacham (2003)
Aggregate and Verifiably Encrypted Signatures from Bilinear Maps

References: Sum-check foundations

Carsten Lund, Lance Fortnow, Howard Karloff, Noam Nisan (1990)
Algebraic Methods for Interactive Proof Systems

Graham Cormode, Justin Thaler, Ke Yi (2011)
Verifying Computations with Streaming Interactive Proofs

Graham Cormode, Michael Mitzenmacher, Justin Thaler (2012)
Practical Verified Computation with Streaming Interactive Proofs

Justin Thaler (2013)
Time-Optimal Interactive Proofs for Circuit Evaluation

References: Faster sum-check proving

Angus Gruen (2024)
Some Improvements for the PIOP for ZeroCheck

Quang Dao, Zachary DeStefano, Suyash Bagad, Yuval Domb, Justin Thaler (2026)
Speeding Up Sum-Check Proving (Extended Version)

References: Curve-based commitments

Aniket Kate, Gregory M. Zaverucha, Ian Goldberg (2010)
Constant-Size Commitments to Polynomials and Their Applications

Benedikt Bünz, Jonathan Bootle, Dan Boneh, Andrew Poelstra, Pieter Wuille, Greg Maxwell (2018)
Bulletproofs: Short Proofs for Confidential Transactions and More

Jonathan Lee (2020)
Dory: Efficient, Transparent Arguments for Generalised Inner Products and Polynomial Commitments

References: Hash-based commitments

Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, Michael Riabzev (2018)
Fast Reed-Solomon Interactive Oracle Proofs of Proximity

Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, Eylon Yogev (2024)
STIR: Reed–Solomon Proximity Testing with Fewer Queries

Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, Eylon Yogev (2024)
WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification

Benjamin E. Diamond, Jim Posen (2023)
Succinct Arguments over Towers of Binary Fields

Benjamin E. Diamond, Jim Posen (2024)
Polylogarithmic Proofs for Multilinears over Binary Towers

References: Lattice arguments

Ward Beullens, Gregor Seiler (2023)
LaBRADOR: Compact Proofs for R1CS from Module-SIS

Vadim Lyubashevsky, Gregor Seiler, Patrick Steuer (2024)
The LaZer Library: Lattice-Based Zero Knowledge and Succinct Proofs for Quantum-Safe Privacy

Beatrice Biasioli, Madalina Bolboceanu, Vadim Lyubashevsky, Antonio Merino-Gallardo, Michał Osadnik, Gregor Seiler, Patrick Steuer (2026)
A Toolkit for Succinct Lattice-Based Zero Knowledge Proofs

Ngoc Khanh Nguyen, Gregor Seiler (2024)
Greyhound: Fast Polynomial Commitments from Lattices

References: Lattice arguments, continued

Ngoc Khanh Nguyen, George O'Rourke, Jiapeng Zhang (2026)
Hachi: Efficient Lattice-Based Multilinear Polynomial Commitments over Extension Fields

Michael Klooss, Russell W. F. Lai, Ngoc Khanh Nguyen, Michał Osadnik, Lorenzo Tucci (2026)
RoKoko: Lattice-based Succinct Arguments, a Committed Refinement

Quang Dao, Omid Bodaghi, Amirhossein Khajehpour, Giuseppe Vitto, Mohammadtaghi Badakhshan, Markos Georghiades, Fengrun Liu, Jiapeng Zhang, Justin Thaler (2026)
Akita: A High-Performance Lattice-Based Polynomial Commitment Scheme

References: Jolt and related proof systems

Arasu Arun, Srinath Setty, Justin Thaler (2023)
Jolt: SNARKs for Virtual Machines via Lookups

Srinath Setty, Justin Thaler, Michael Zhu (2025)
Twist and Shout: Faster Memory Checking Arguments via One-Hot Addressing and Increments

Srinath Setty (2020)
Spartan: Efficient and General-Purpose zkSNARKs without Trusted Setup

Srinath Setty (2026)
BinarySpartan: Spartan over Binary Fields

Srinath Setty (2026)
Magnetar: High-speed Post-quantum RV64 zkVM from Standard Hashes

References: Zero knowledge

Darya Kaviani, Srinath Setty (2025)
Vega: Low-Latency Zero-Knowledge Proofs over Existing Credentials

Abhiram Kothapalli, Srinath Setty, Ioanna Tzialla (2022)
Nova: Recursive Zero-Knowledge Arguments from Folding Schemes

Abhiram Kothapalli, Srinath Setty (2024)
HyperNova: Recursive Arguments for Customizable Constraint Systems

Rahul Dalal, Tamir Hemo, Eugene Rabinovich, Ron Rothblum (2026)
VEIL: Lightweight Zero-Knowledge for Hash-Based Multilinear Proof Systems

References: Applications

Matteo Frigo, abhi shelat (2024)
Anonymous Credentials from ECDSA

Alberto Centelles, Quang Dao, Mohammadtaghi Badakhshan, Justin Thaler (2026)
Aerie: Efficient Aggregation of Falcon Signatures

Baiyu Li, Ting-Yuan Wang, Jiapeng Zhang (2026)
Beasley: Efficient Zero-Knowledge Proofs for Lattice-based Round-Optimal Oblivious Pseudorandom Functions

References: Implementation and application sources

Markos Georghiades, Justin Thaler, Andrew Tretyakov, Michael Zhu (2026)
Entering the Era of Lattice SNARKs – with a Faster, Post-Quantum Jolt

Sagar Dhawan, Markos Georghiades, Justin Thaler, Andrew Tretyakov, Michael Zhu (2026)
Putting the zk in zkVM: Jolt Now Supports Zero Knowledge

Alberto Centelles, Quang Dao, Justin Thaler, Markos Georghiades, Michael Zhu, Andrew Tretyakov (2026)
Fast Post-Quantum Signature Aggregation with Lattice SNARKs