SNAIL: Verifiable Computation within 30 % of Native Speed

Ole Hylland Spjeldnæs
Repyh Labs / delta.network

Abstract. SNAIL (Succinct, Non-interactive, Alon-compressed, Instant argument for Layered circuits) turns any depth-d arithmetic circuit into a non-interactive argument whose prover runs within $\big(1+c(d,k,n)\big)$ of uninstrumented execution, where $c(d,k,n) = 3(k+n+1)/(k\,d+n+1)$; for the representative choice $k = n = 4$, $d \in [24, 32]$ this equates to just **21–28 %** added overhead.

Core idea. A constant-round *zerocheck*—derived from a *difference-based Alon decomposition* (see § 2)—reduces *all* layers to one multivariate polynomial identity; a Fiat–Shamir transform in the random-oracle model yields a stand-alone argument whose hot loop performs *only* field adds/muls (no multiscalar multiplications, FFTs, or Merkle authentication paths).

Cost summary. For arity $k = 4$, degree bound $n = 4$ and live-variable depth $24 \le d \le 32$[1]

$$T_{\text{prov}}(S,d) = \big(1 + \tfrac{6.75}{d}\big)S, \quad |\pi| = (d^2+2d)(4d+1), \quad \varepsilon_{\text{sound}} = \tfrac{4\,n\,d}{|\mathbb{F}|}.$$

A 2.6 G-gate trace at $d = 32$ is proven in < 40 ms on an RTX-4090[2].

Width scalability. Proof size depends only on depth, so widening to billions of parallel gates leaves the verifier unchanged—ideal for ultra-wide workloads such as live-audited VM traces.

Security. The six-round interactive core is *statistically* sound (ε above); the non-interactive form inherits *computational* soundness in the random-oracle model while remaining transparent and CRS-free.

Implications. SNAIL demonstrates that a prover can run *near real-time* on commodity hardware (1 TB s^{-1} bandwidth) and verified on a smartphone in < 1 ms *arithmetic time once the 1.1 MB proof is resident in RAM* while emitting proofs small enough for mobile light clients, enabling extremal constructions such as live-streamed provable gaming, pay-per-cycle cloud nodes, and always-on verifiable CPUs.

Keywords: Zero-knowledge proofs, succinct non-interactive arguments (SNARK/SNARG), interactive proofs, GKR, Combinatorial Nullstellensatz, verifiable computation, zkVM, layered arithmetic circuits

Contents

1 Introduction **2**
1.1 SNAIL in a nutshell 3
1.2 Why it matters 4
1.3 Main results 4

[1] Depths > 32 inflate the proof as $(d^2 + 2d + 2)(nd + 1) = \Theta(d^3)$ yet shave $< 3\%$ off prover time; depths < 24 raise the arithmetic multiplier above 30 %. Memory budgets on 16–32 GiB devices reinforce the same band.

[2] Model-based estimate using the throughput parameters of § 5.2.

2 Difference-based Alon Decomposition **5**
2.1 Alon's Combinatorial Nullstellensatz 6
2.2 Difference-Based Alon Decomposition 7
2.3 Alon Decomposition at a Point 9

3 A New Zerocheck Interactive Proof **10**
3.1 A Naive First Attempt (Unsound) 11
3.2 Why This First Attempt Is Unsound 11
3.3 Fixing Soundness: Alon Decomposition with Dimensional Shifts 12
3.4 An Interactive Proof for $\mathcal{R}_{\text{Zero}}$ 13

4 Verifiable Computation Based on the Zero-Check IP **16**
4.1 From Circuit Layers to One Zero-Check 16
4.2 Efficient Alon differences for layered circuits 17
4.3 An Eight-Round Interactive Protocol for Layered Circuits 20
4.4 Why classical information-theoretic lower bounds do not bite 21

5 The SNAIL Protocol **21**
5.1 Layer-by-layer cost ledger 23
5.2 Depth–width trade-offs 23
5.3 Toward a SNAILy zkVM 24

1 Introduction

Interactive–oracle proofs and their non-interactive descendants (SNARKs) have become the workhorse of verifiable computation, powering blockchains, privacy systems, and zkVMs. State-of-the-art schemes such as Plonk [GWC19], STARK [BSBHR18], and Groth16 [Gro16] achieve impressive proof sizes and verification times, but their *prover cost* is generally many orders of magnitude larger than plain execution.

This paper introduces *SNAIL*: a Succinct, Non-interactive, Alon-compressed, Instant argument for Layered circuits. SNAIL dispenses entirely with FFTs and polynomial commitments, replacing them with a single *difference-based Alon decomposition* that transforms *all layers at once* into one zerocheck identity. The result is a transparent succinct non-interactive proof whose prover performs *only* field additions and multiplications—operations that saturate commodity memory bandwidth.

Motivation. Across the spectrum of modern proof systems, the prover typically pays a *four-to-five-order-of-magnitude* slowdown compared with native execution, due to heavyweight polynomial transforms and hash-or-curve-based authentication checks. This work shows that by folding the entire circuit into a single difference-driven zerocheck and eliminating per-layer interaction, the gap can be tightened to within 20–30 % of plaintext runtime while the verifier stays sub-millisecond.

Baseline used throughout. All slowdown ratios in this paper are measured *against the same arithmetic circuit evaluated without any proof machinery.* That circuit-only run already incurs the usual encode-to-gates cost ($\approx 2\times$ for straight-line arithmetic, up to $10\times$ for branch-heavy logic). SNAIL's prover adds only a further 21–28% on top of that cost. When comparing to a native CPU or CUDA implementation, multiply our ratios by the circuit-encoding factor accordingly.

1.1 SNAIL in a nutshell

One zerocheck for the entire circuit. We revisit Alon's Combinatorial Nullstellensatz and show that a *difference-driven* variant can be computed efficiently without requiring an additional pass over the trace. Running this "difference Alon" three times yields a public-coin interactive proof (Protocol 1) that

- is *perfectly complete* and *statistically sound* over $\mathbb{F}^d$;
- uses 6 rounds and $O\big(n\, d^3\big)$ verifier work;
- touches each gate exactly once per variable.

From circuit layers to one zerocheck. Adopting the standard GKR-style [GKR08] arithmetisation, each layer $\ell \in \{1, \ldots, d\}$ is written as a degree-$\leq n$ multivariate polynomial $P_\ell(x_1, \ldots, x_d)$ over the *same* d Boolean variables.

Toy example. Consider a depth-2 circuit with three gates per layer (indices need two bits, so here $d = 2$):

$$L_2\colon\ y_1 = a + b,\ y_2 = c \cdot d,\ y_3 = e + f, \qquad L_1\colon\ z = y_1 \cdot y_2 + y_3.$$

Writing the GKR multilinear extensions in variables x_1, x_2 (two bits to address 3 gates) gives $P_2(x) = \sum_{i=1}^{3} \lambda_i(x)\, g_i$ and $P_1(x) = \sum_{i=1}^{3} \mu_i(x)\, y_i$. The difference-Alon routine symbolically decomposes each P_ℓ into $Q_{1,\ell}\, x_1(x_1 - 1) + Q_{2,\ell}\, x_2(x_2 - 1)$ *in the same pass that streams the witness*. Section 4.2 shows the coefficients can be updated with nine scalar ops per gate, so no extra memory pass arises. Concretely, for the gate $y_1 = a + b$ the pair $(a|_{x_1=0}, a|_{x_1=1})$ is already in registers; we add b's pair (2 adds), recompute $a{+}b$ twice (2 adds), subtract the cached value once (1 add) and repeat for x_2 (4 adds): $2{+}2{+}1{+}4 = 9$ scalar operations, matching the general bound.

From the IP to SNAIL. A Fiat–Shamir transform [FS87] replaces every verifier challenge in the six-round interactive protocol with a hash of the running transcript. The prover makes *seven* random-oracle queries in total:

- 1 query *before* streaming, to derive the layer-aggregation vector $\boldsymbol{\alpha}$;
- 6 queries *after* streaming, one per zerocheck round.

All cryptography therefore lies outside the memory-bound scan; the hot loop remains pure field arithmetic. The result is Protocol 3—a transparent, computationally sound, non-interactive argument whose hot loop contains only field additions and multiplications (no commitments, Merkle trees, FFTs, or MSMs).

Why the prover is only $1.2\times$ native execution (the following numbers use $k = n = 4$).

- **No polynomial commitments.** Mainstream SNARKs pay $10^2 - 10^3\times$ for cryptography— either multiscalar-multiplication (MSM) inside pairing-based PCS, or hash ladders in FRI. SNAIL's prover carries *neither*: a single Fiat–Shamir hash is computed *after* the trace scan, and no MSMs or FFTs appear at all.
- **No witness transforms.** The difference-Alon zerocheck works directly on coefficients, so the trace is streamed exactly once.
- **Single-pass memory.** Each gate is fetched, processed, and retired in one linear scan; no strided columns or random look-ups.
- **Depth folded away.** Layer-by-layer protocols reread the trace d times. Aggregating all layers into one identity removes those passes.

- **Tiny kernel.** A few hundred scalar lines saturate memory bandwidth on any modern CPU/GPU; no architecture-specific tricks required.

Net effect. With cryptography, $\log S$-scale transforms, and depth re-reads gone, only the $6.75/d$ arithmetic surcharge remains—explaining the $1.21{-}1.28\times$ gap between SNAIL and plain execution for $24 \le d \le 32$. The constant arises as 3 passes $\times$ 9 scalar ops $= 27$, divided by the baseline $4d$ scalar ops that evaluate one k=4-input gate, giving $27/(4d) = 6.75/d$. Details appear in Section 4.2.

Witness capture adds only single-digit latency. Writing the trace requires one b-byte store per gate. With sustained memory bandwidth B_{m} and compute throughput B_{f}, the overhead is

$$\frac{T_{\mathrm{store}}}{T_{\mathrm{exec}}} \;=\; \frac{b/B_{\mathrm{m}}}{\mathsf{ops}_{\mathrm{gate}}/B_{\mathrm{f}}}, \quad \mathsf{ops}_{\mathrm{gate}} = k\,d + n.$$

For an RTX-4090 ($B_{\mathrm{m}} = 1.0{\times}10^{12}\,\mathrm{B\,s^{-1}}$, $B_{\mathrm{f}} = 1.1{\times}10^{13}\,\mathrm{ops\,s^{-1}}$, $b = 8$, $k = n = 4$, $d = 32$) this ratio is

$$\frac{8/1.0{\times}10^{12}}{132/1.1{\times}10^{13}} \;\approx\; 0.074.$$

Thus the extra trace store adds about *7 %* to wall-time, keeping the full SNAIL kernel within the advertised 1.21–1.28× of native execution for $24 \le d \le 32$. On a 600 GB/s laptop GPU the ratio rises to $\lesssim 10\%$; on a 5 TB/s MI300X it drops below 5%.

Convention. Throughout the paper we *include* this single witness-capture pass in the prover time, so every latency figure—and the headline $1.21-1.28\times$ ratio—compares *full proving* (trace store + execution + difference-Alon overlay) against an uninstrumented native run of the same circuit.

Call for implementers. We are actively looking for contributors who would like to build high-performance, open-source implementations of the SNAIL prover and verifier. Contact ole@repyhlabs.com if interested.

1.2 Why it matters

Single-shot verification. The *interactive* verifier is hash-free and runs in $O\big(n\,d^3\big)$ field operations ($\le$130 k at $d = 32, n = 4$). The non-interactive verifier merely recomputes the seven Fiat–Shamir hashes ($\approx 2\mu$s on ARM) and executes the same $O\big(n\,d^3\big)$ arithmetic.

1.3 Main results

Theorem 1 (Informal)**.** *Let the layered circuit have S gates, live-variable depth $24 \le d \le 32$, gate arity k, and per-gate polynomial degree $\le n$. Define*

$$c(d,k,n) \;=\; \frac{3\,(k+n+1)}{k\,d+n+1}.$$

Then:

Prover time	$T_{\mathrm{prov}}(S,d,k,n)$	$=$	$\big(1 + c(d,k,n)\big)\,\dfrac{S}{f_{\mathrm{op}}}$ $\quad(1.28{-}1.21\times$ *when* $k = n = 4, d \in [24,32])$,
Proof size	$\lvert\pi\rvert$	$=$	$(d^2 + 2d + 2)\,\big(n\,d + 1\big) \;=\; O\big(n\,d^3\big)$,
Verifier work	$O\big(n\,d^3\big)$	$<$	1 ms *for* $d \le 32$, $n \le 4$.

For $n = 4, d = 32$ this gives 140 352 coefficients; the interactive verifier inspects $4d\,d^2 =$ 131 072 field values, a ratio below 1.1.

Here f_{op} is the sustained field-operation throughput (add or mul) of the target machine.

Real-time proofs, not post-mortems.
At 21–28 % overhead a SNAIL prover can run *next to* the application it audits—think < 20 ms per 60 fps frame or < 40 ms for a 3 B-gate shard. Proofs become a live safety belt, not an after-the-fact check.

Soundness.
The interactive protocol is statistically sound with $\varepsilon_{\text{stat}} = 4nd/|\mathbb{F}|$; after Fiat–Shamir it has computational soundness $\leq \varepsilon_{\text{stat}} + \text{negl}(\lambda)$ in the random-oracle model

Light-client friendly.
A $d = 32$ proof—140 k field words—verifies in < 1 ms on a phone's scalar core. Anyone with a web browser can validate a 2 B-gate computation. At $d = 32$ the verifier evaluates 1 088 univariates of degree $\leq 128 \approx (4d)\,d^2 = 139\,\text{k}$ field ops; a 3 GHz mobile core finishes in $139\text{k}/(3 \times 10^9) \approx 46\ \mu$s.

Unlimited fan-out.
Proof size and verifier work depend *only* on depth. Adding a billion parallel gates leaves the proof unchanged, making SNAIL ideal for ultra-wide workloads—e.g. a *live-streamed, provably fair game server* that certifies every physics tick to millions of spectators in real time.

Path to provable silicon.
Because the prover is one bandwidth-bound kernel, an ASIC or HBM-rich FPGA can drop straight in, turning every CPU cycle into a cryptographically certified one.

Paper outline

2 revisits Alon's theorem and presents the difference-based decomposition.

3 develops the interactive zero-check protocol.

4 compresses a layered circuit into a single arithmetic constraint.

5 applies the Fiat–Shamir transform to obtain SNAIL.

Scope of the exposition. We assume the reader is already comfortable with the standard definitions of interactive proofs, completeness and soundness (statistical vs. computational), arguments vs. proofs, and notions of succinctness. We also assume familiarity with the Schwartz–Zippel lemma, which largely underlies the soundness arguments.

2 Difference-based Alon Decomposition

Many algebraic proof systems reduce global circuit satisfiability to *local* divisibility checks of the form $P(\vec{x}) = \sum_{i=1}^{d} Q_i(\vec{x})\,\phi_i(x_i)$. A classical tool for producing such split forms is **Alon's Combinatorial Nullstellensatz**. This section reviews the theorem and, crucially, shows how to compute the Q_i *symbolically in* $O(n\,d)$ *time* when $P(\vec{x}) = f\big(a_1(\vec{x}), \ldots, a_k(\vec{x})\big)$ has *multilinear* inputs a_j and bounded outer degree $\deg f = n = O(1)$. The key optimisation is to operate on *finite differences* rather than expand the a_j into their 2^d monomials.

Representation model. Throughout this section a polynomial is stored as a *size-$O(d)$ DAG* whose internal nodes are multilinear gates, whose root is a degree-$\leq n$ Horner form, and whose leaves are field constants.

Substituting a single variable therefore revisits at most the $k = 4$ frontier leaves *per gate* and re-evaluates the outer degree $n = 4$ Horner form once.

Why the DAG is $O(d)$, not 2^d. Every multilinear input $a_j(\vec{x})$ depends on at most one live variable per layer, so after arithmetising a depth-d circuit the resulting symbolic expression contains $\leq d$ distinct variables and a constant number of syntactic copies of each a_j. Hence the sharing graph of P has $O(d)$ internal nodes even though the fully-expanded sum would have 2^d monomials.

All asymptotic bounds below refer to this compact DAG model and hold for the fixed constants $(k, n) = (4, 4)$ often used in the rest of the paper.

2.1 Alon's Combinatorial Nullstellensatz

Theorem 2 (Alon's Combinatorial Nullstellensatz [Alo99])**.** *Let $\mathbb{F}$ be a field and let $P(x_1, \ldots, x_d) \in \mathbb{F}[x_1, \ldots, x_d]$. For subsets $S_1, \ldots, S_d \subset \mathbb{F}$ the following are equivalent:*

1. *P vanishes on the grid $S_1 \times \cdots \times S_d$.*

2. *There exist polynomials $Q_1, \ldots, Q_d \in \mathbb{F}[x_1, \ldots, x_d]$ such that*

$$P(x_1, \ldots, x_d) \;=\; \sum_{i=1}^{d} Q_i(x_1, \ldots, x_d)\, Z_{S_i}(x_i),$$

where $Z_{S_i}(x_i) = \prod_{s \in S_i}(x_i - s)$ vanishes exactly on S_i.

The tuple $(Q_1, \ldots, Q_d)$ is called an Alon decomposition *of P over $(S_1, \ldots, S_d)$.*

For the two grids that matter in this paper—the Boolean hypercube and an arbitrary point—we obtain the familiar corollaries.

Remark 1 (Choice of reference point). For the hypercube we fix $c_i = 0$. Then $(x_i - c_i) = x_i$ and the factor $x_i(x_i - 1)$ splits cleanly in every characteristic, avoiding denominators and keeping the symbolic division integral.

Corollary 1 (Boolean cube)**.** *P vanishes on $\{0, 1\}^d$ iff there exist $Q_i(\vec{x})$ such that*

$$P(\vec{x}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\, x_i(x_i - 1).$$

Corollary 2 (Single point)**.** *For $\vec{r} = (r_1, \ldots, r_d) \in \mathbb{F}^d$ and $y \in \mathbb{F}$ we have*

$$P(\vec{r}) = y \iff \exists\, Q_i(\vec{x}) \text{ s.t. } P(\vec{x}) - y = \sum_{i=1}^{d} Q_i(\vec{x})\,(x_i - r_i).$$

Sections 2.2–2.3 that follow give an $\mathbf{O(n\,d)}$-time algorithm for producing the Q_i *without* expanding P's multilinear inputs, and prove its correctness and complexity bounds.

2.2 Difference-Based Alon Decomposition

2.2.1 General Algorithm

Let $\vec{x} = (x_1, \ldots, x_d) \in \mathbb{F}^d$, and let $\vec{c} = (c_1, \ldots, c_d) \in \mathbb{F}^d$ be a *reference point*. Suppose each univariate factor $\phi_i(x_i)$ satisfies

$$\phi_i\big(c_i\big) \;=\; 0 \quad \text{for each } i.$$

Below, we choose these factors differently for the hypercube vs. the point case:

$$\phi_i(x_i) \;=\; \begin{cases} x_i\,\big(x_i - 1\big), & \text{for the hypercube,} \\ x_i \;-\; r_i, & \text{for an arbitrary point } \vec{r}. \end{cases}$$

We then seek

$$P(\vec{x}) - P(\vec{c}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\,\phi_i\big(x_i\big).$$

The following single procedure accomplishes this by successively peeling off each factor.

Algorithm 1 AlonDecomposition$\big(P(\vec{x}), \vec{c}, \{\phi_i\}\big)$

procedure AlonDecomposition$(P, \vec{c}, \{\phi_i\}_{i=1}^{d})$
  Input:
  - A polynomial $P(\vec{x})$ in variables $x_1, \ldots, x_d$.
  - A reference point $\vec{c} \in \mathbb{F}^d$.
  - Linear factors $\phi_i(x_i)$, each satisfying $\phi_i(c_i) = 0$.

  Output: Polynomials $Q_1, \ldots, Q_d$ such that

$$P(\vec{x}) \;-\; P(\vec{c}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\,\phi_i\big(x_i\big).$$

  $R(\vec{x}) \leftarrow P(\vec{x}) \;-\; P(\vec{c})$ ▷ Initial "residual"; note $R(\vec{c}) = 0$.
  for $i = 1$ to d **do**
    ▷ **1) Interpolate R at $x_i = c_i$:**
    $\widetilde{R}_i(\vec{x}) \;\leftarrow\; R(\vec{x})\big|_{x_i = c_i}$
    ▷ **2) Difference; factor out $\phi_i(x_i)$:**
    $D_i(\vec{x}) \;\leftarrow\; R(\vec{x}) \;-\; \widetilde{R}_i(\vec{x})$
    $Q_i(\vec{x}) \;\leftarrow\; \dfrac{D_i(\vec{x})}{\phi_i\big(x_i\big)}$
    ▷ **3) Update the residual, removing the factor:**
    $R(\vec{x}) \;\leftarrow\; R(\vec{x}) - Q_i(\vec{x})\,\phi_i(x_i) \;=\; \widetilde{R}_i(\vec{x})$ ▷ (Now R no longer depends on x_i.)
  end for
  return $(Q_1, \ldots, Q_d)$
end procedure

Notation. We reuse the symbol R *in place*: after the i-th iteration R denotes the current residual $R^{(i)}$, independent of $x_1, \ldots, x_i$.

Lemma 1 (Degree bound)**.** *If* $\deg P \le n\,d$*, then after the* i*-th loop*

$$\deg R^{(i)} \;\le\; n\,d - i \quad \textit{and} \quad \deg Q_i \;\le\; n\,d - 1.$$

In particular, every Q_i *has total degree* $\le n\,d - 1$.

Proof. Induct on i. Base case $i = 0$: $\deg R^{(0)} = \deg P \le n\,d$.

Inductive step: write $R^{(i-1)} = \widetilde{R}_i + (x_i - c_i)\,E_i$ with $\deg E_i \le \deg R^{(i-1)} - 1$, because extracting the factor $(x_i - c_i)$ lowers the exponent of x_i by 1 while leaving all other exponents unchanged.

Setting $Q_i := E_i$ subtracts the term $(x_i - c_i)E_i$, so $\deg R^{(i)} = \deg \widetilde{R}_i \le \deg R^{(i-1)} - 1 \le n\,d - i$. Moreover $\deg Q_i \le n\,d - 1$. □

Theorem 3 (Correctness and Complexity of Alon Decomposition)**.** *Let $P(\vec{x}) = f\big(a_1(\vec{x}), \ldots, a_k(\vec{x})\big)$, where each a_j is multilinear in $\vec{x} \in \mathbb{F}^d$ and $\deg(f) = n$. Suppose $\vec{c} \in \mathbb{F}^d$ and $\phi_i(x_i)$ are linear factors satisfying $\phi_i(c_i) = 0$. Then*

$$P(\vec{x}) \;-\; P(\vec{c}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\,\phi_i(x_i)$$

can be computed by AlonDecomposition$\big(P, \vec{c}, \{\phi_i\}\big)$ *in $\mathrm{O}(n\,d)$ time, without expanding P into exponentially many monomials.*

Proof. **Correctness.** At each iteration i, the algorithm constructs

$$\widetilde{R}_i(\vec{x}) \;=\; R_{i-1}(\vec{x})\big|_{x_i = c_i}, \quad D_i(\vec{x}) \;=\; R_{i-1}(\vec{x}) \;-\; \widetilde{R}_i(\vec{x}).$$

Since $D_i(\vec{x})$ vanishes whenever $x_i = c_i$, we have $D_i(\vec{x}) \equiv (x_i - c_i)\,E_i(\vec{x})$ in the unique-factorisation domain $\mathbb{F}[x_1, \ldots, x_d]$; for the hypercube case the additional factor $(x_i - 1)$ also divides D_i. Hence

$$Q_i(\vec{x}) \;:=\; E_i(\vec{x}) \in \mathbb{F}[x_1, \ldots, x_d].$$

Hence $R_i(\vec{x})$ no longer depends on x_i. After processing all d variables, the final residual $R_d(\vec{x})$ is independent of every x_i. However, since $R_d(\vec{c}) = 0$, it follows that R_d is the zero polynomial. Thus

$$P(\vec{x}) \;-\; P(\vec{c}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\,\phi_i(x_i).$$

Complexity. With $(k, n) = (4, 4)$ fixed constants, one substitution touches at most k multilinear leaves and re-evaluates the outer degree-n Horner form once, costing $O(k{+}n) = O(1)$ scalar operations. Division by a linear factor is syntactic: we strip the known factor from the top monomial list without coefficient arithmetic. Hence each loop costs $O(1)$, and the total over d variables is $O(n\,d)$ as claimed.

□

2.2.2 Hypercube Decomposition

For polynomials P that vanish on the Boolean hypercube $\{0, 1\}^d$, we can take:

$$\phi_i(x_i) \;=\; x_i\,\big(x_i - 1\big), \quad \vec{c} = \vec{0}.$$

Since P vanishes at $\vec{0} \in \{0, 1\}^d$, it follows that $P(\vec{0}) = 0$. Hence

$$P(\vec{x}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\,x_i\,(x_i - 1).$$

Algorithm 1 specializes to the code below, which we refer to as AlonHypercubeDecompositionSymb:

Algorithm 2 AlonHypercubeDecompositionSymb$\big(P(\vec{x})\big)$

procedure AlonHypercubeDecompositionSymb(P)
  Input: $P(\vec{x}) = f\big(a_1(\vec{x}), \ldots, a_k(\vec{x})\big)$, each a_j multilinear in $\vec{x} \in \mathbb{F}^d$, and $\deg(f) = n$.
  Output: $\{Q_i(\vec{x})\}_{i=1}^{d}$ such that

$$P(\vec{x}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\, x_i\,(x_i - 1).$$

  $R(\vec{x}) \leftarrow P(\vec{x})$
    ▷ *R is updated in place: after iteration i it no longer depends on $x_1, \ldots, x_i$.*
  for $i = 1$ to d **do**
    $\widetilde{R}_i(\vec{x}) \leftarrow R(\vec{x})|_{x_i=0} \;+\; x_i\big(R(\vec{x})|_{x_i=1} - R(\vec{x})|_{x_i=0}\big)$
      ▷ *Both substitutions are cached once; later uses are $\mathcal{O}(1)$.*
    $Q_i(\vec{x}) \leftarrow \dfrac{R(\vec{x}) - \widetilde{R}_i(\vec{x})}{x_i(x_i - 1)}$
    $R(\vec{x}) \leftarrow R(\vec{x}) - Q_i(\vec{x})\,\big[x_i^2 - x_i\big]$
  end for
  return $(Q_1, \ldots, Q_d)$
end procedure

Division by $x_i(x_i - 1)$ is syntactic: the common factor is stripped from the monomial list, so no field inverse is computed.

Theorem 4. *Let $P(\vec{x}) = f\big(a_1(\vec{x}), \ldots, a_k(\vec{x})\big)$ where $\deg(f) = n$ and each $a_j(\vec{x})$ is multilinear in dimension d. If P vanishes on $\{0,1\}^d$, then* AlonHypercubeDecompositionSymb *returns*

$$P(\vec{x}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\,\big[x_i^2 - x_i\big]$$

in $\mathrm{O}(n\,d)$ time.

Proof. Correctness. Because

$$D_i(\vec{x}) \;=\; R^{(i-1)}(\vec{x}) - \widetilde{R}_i(\vec{x})$$

vanishes at both Boolean points $x_i = 0$ and $x_i = 1$, the product $x_i(x_i - 1)$ must divide D_i in $\mathbb{F}[x_1, \ldots, x_d]$. Set

$$Q_i(\vec{x}) \;:=\; \frac{D_i(\vec{x})}{x_i(x_i - 1)}.$$

By Lemma 1 we then have $\deg Q_i \le \deg D_i - 2 \le nd - 1$.

Complexity. Each loop performs exactly the nine scalar operations (two cached substitutions, four adds, one subtraction, one syntactic divide, one update). With fixed $(k, n) = (4, 4)$ this is $O(1)$ work, totalling $O(d)$ (written $O(n\,d)$ to match later sections) over the d variables. □

2.3 Alon Decomposition at a Point

We now illustrate how the same *difference-based* procedure applies to

$$\phi_i(x_i) \;=\; x_i \;-\; r_i,$$

producing the decomposition

$$P(\vec{x}) \;-\; P(\vec{r}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\,\big(x_i - r_i\big).$$

Specifically, we choose the reference point $\vec{c} = \vec{r}$ and at each iteration extract the factor $(x_i - r_i)$ from the current residual.

Algorithm 3 AlonPointDecompositionByDifference$\big(P, \vec{r}\big)$

procedure AlonPointDecompositionByDifference$(P, \vec{r})$
  Input: $P(\vec{x}) = f\big(a_1(\vec{x}), \ldots, a_k(\vec{x})\big)$ with each a_j multilinear in $\mathbb{F}^d$, $\deg(f) = n$, and point $\vec{r} \in \mathbb{F}^d$.
  Output: Polynomials $Q_1, \ldots, Q_d$ satisfying

$$P(\vec{x}) - P(\vec{r}) \;=\; \sum_{i=1}^{d} Q_i(\vec{x})\,\big(x_i - r_i\big).$$

  $R(\vec{x}) \leftarrow P(\vec{x}) \;-\; P(\vec{r})$
  for $i = 1$ to d **do** ▷ *Eliminate x_i by subtracting its constant part at $x_i = r_i$:*
    $\widetilde{R}_i(\vec{x}) \;\leftarrow\; R(\vec{x})\big|_{x_i = r_i}$
    $D_i(\vec{x}) \;\leftarrow\; R(\vec{x}) - \widetilde{R}_i(\vec{x})$ ▷ D_i vanishes at $x_i = r_i$.
    $Q_i(\vec{x}) \;\leftarrow\; \dfrac{D_i(\vec{x})}{x_i \;-\; r_i}$
    $R(\vec{x}) \;\leftarrow\; R(\vec{x}) - Q_i(\vec{x})\,(x_i - r_i) \;=\; \widetilde{R}_i(\vec{x})$
  end for
  return $\big(Q_1, \ldots, Q_d\big)$
end procedure

Correctness and Complexity. Because $R(\vec{r}) = 0$, each $(x_i - r_i)$ indeed divides the current residual. Removing these factors one at a time eventually eliminates all variables, so the last residual is the zero polynomial. As with the hypercube case, each iteration can be done in $\mathrm{O}(n)$ time on the symbolic circuit (no monomial expansion), totaling $\mathrm{O}(n\,d)$ and therefore mirroring the analysis of Theorem 3.

In Section 3 the routine is invoked *three* times—once for the layer compilation and twice for the random $\boldsymbol{s}$- and $\boldsymbol{w}$-decompositions— yielding the 6-round zerocheck protocol.

3 A New Zerocheck Interactive Proof

Definition 1 (Diagonalisation)**.** For $f \in \mathbb{F}[x_1, \ldots, x_d]$ define

$$\varphi(f) \;:=\; f(x, \ldots, x).$$

(a) *Ring homomorphism.* φ preserves $+$ and $\cdot$.

(b) *Total degree.* $\deg\big(\varphi(f)\big) \le \deg f$ because substitution never *increases* the highest monomial exponent.

(c) *Individual degree.* If every variable in f has degree $\le n$ (i.e. $\deg_{x_i} f \le n$) then $\deg\big(\varphi(f)\big) \le d\,n$. This is the bound we use when f is known to be *multilinear-in-each-variable.*

Definition 2 (Zerocheck relation)**.** Fix public parameters $\mathsf{pp} = (d, n)$ over a field $\mathbb{F}$. The relation $\mathcal{R}_{\mathrm{Zero}}$ contains those pairs

$$(\mathbb{x}, \omega) = \big((\varphi(Q_1), \ldots, \varphi(Q_d)); P\big),$$

where the following hold.

1. $P(\vec{x}) = f\big(a_1(\vec{x}), \ldots, a_k(\vec{x})\big)$ with $\deg(f) = n$ and each a_j multilinear in $\vec{x} \in \mathbb{F}^d$. Consequently $\deg P \leq n\,d$ (total degree), because each a_j is multilinear in the d variables while f has outer degree n.
2. $Q_i(\vec{x}) \in \mathbb{F}[x_1, \ldots, x_d]_{\leq nd-2}$ for $i = 1, \ldots, d$.
3. The following polynomial identity holds in $\mathbb{F}[x_1, \ldots, x_d]$:

$$P(\vec{x}) = \sum_{i=1}^{d} Q_i(\vec{x}) x_i \big(x_i - 1\big). \tag{1}$$

3.1 A Naive First Attempt (Unsound)

We first present an unsound but directionally correct interactive proof for the relation $\mathcal{R}_{\text{Zero}}$: Here we outline an *incorrect* attempt at a interactive proof for the zero-check relation. We describe it in a protocol-like style, but **emphasize that it is actually not sound**.

Naïve Zerocheck — unsound sketch (Invalid)

1. **Public input.** The verifier knows the univariate polynomials $\varphi(Q_1), \ldots, \varphi(Q_d) \in \mathbb{F}[x]_{\leq nd-2}$ and the claimed d-variate $P(\vec{x})$.
2. **Random point.** $\mathcal{V}$ samples $\vec{r} \leftarrow \mathbb{F}^d$ and sends it to $\mathcal{P}$.
3. **First decomposition.** $\mathcal{P}$ computes polynomials $s_1, \ldots, s_d$ with $P(\vec{x}) - P(\vec{r}) = \sum_i s_i(\vec{x})\,(x_i - r_i)$ and sends the *univariate* polynomials $\varphi(s_i)(x)$ to $\mathcal{V}$.
4. **Random weights.** $\mathcal{V}$ samples $t_1, \ldots, t_d \leftarrow \mathbb{F}$, sends them.
5. **Second decomposition.** $\mathcal{P}$ sets $y_{Q_i} := Q_i(\vec{r})$ and finds $w_1, \ldots, w_d$ such that $\sum_i t_i \big(Q_i(\vec{x}) - y_{Q_i}\big) = \sum_j w_j(\vec{x})\,(x_j - r_j)$. He transmits the univariates $\varphi(w_j)(x)$.
6. **Verifier test.** $\mathcal{V}$ chooses a fresh $z \leftarrow \mathbb{F}$, evaluates every univariate sent so far at the point z, and checks

$$0 \stackrel{?}{=} \sum_{i=1}^{d} \varphi(Q_i)(z)\big(z^2 - z\big) - \sum_{i=1}^{d} y_{Q_i}\big(r_i^2 - r_i\big) - \sum_{i=1}^{d} \varphi(s_i)(z)\big(z - r_i\big) \tag{2}$$

$$0 \stackrel{?}{=} \sum_{i=1}^{d} t_i\big(\varphi(Q_i)(z) - y_{Q_i}\big) - \sum_{i=1}^{d} \varphi(w_i)(z)\big(z - r_i\big) \tag{3}$$

Remark 2. The polynomials w_j exist for the following reason: Each term $Q_i - y_{Q_i}$ has an Alon decomposition $\{w_j^i\}_j$, whereby we get the set $\{w_j^i\}_{i,j}$. We can consequently define

$$w_j(\vec{x}) := \sum_{i=1}^{d} t_i\, w_j^i(\vec{x}). \tag{4}$$

In practice, it will be faster to first compute the linear combination and then decompose.

3.2 Why This First Attempt Is Unsound

The problem is that a malicious $\mathcal{P}$ can exploit the univariate polynomials $\varphi(s_i)$, $\varphi(w_j)$ to "factor out" $(x - r_i)$ easily *even if* the original $P(\vec{r})$ or $Q_i(\vec{r})$ is incorrect. As a univariate

polynomial, any non-constant $u(x)$ can be decomposed as $\sum_i v_i(x)\,(x - r_i)$ by choosing $v_1(x)$ suitably, then factoring out $(x - r_2)$, etc. (cf. *Division by* $(x - r)$ argument). Thus the protocol fails to constrain the full d-variate form of P properly and yields no real soundness guarantee.

Key Reason for Failure: The verifier only checks a *single* diagonal polynomial identity, so any bad polynomial can appear consistent in univariate form at randomly chosen points. The approach never forces the s_i and w_j polynomials to match the actual multivariate structure.

3.3 Fixing Soundness: Alon Decomposition with Dimensional Shifts

The core issue in our naïve approach (Section 3.2) was that the $\mathcal{P}$'s univariate commitments for s_i, w_j could be faked, since dividing a univariate polynomial by $(x - r)$ is easy even if the true multivariate structure does not match.

Idea. We force $\mathcal{P}$ to exhibit *two* Alon decompositions for *slightly shifted* points, ensuring that these two expansions must coincide in their shared variables. In particular, we take two points

$$\vec{r} \;=\; (r_1, \ldots, r_{d-1}, r_d), \quad \tilde{\vec{r}} \;=\; (r_1, \ldots, r_{d-2}, \tilde{r}_{d-1}, \tilde{r}_d)$$

that coincide everywhere but in the last two coordinates. Then

$$P(\vec{x}) \;-\; P(\vec{r}) \;=\; \sum_{i=1}^{d} s_i(\vec{x})\,\big(x_i - r_i\big), \qquad P(\vec{x}) \;-\; P(\tilde{\vec{r}}) \;=\; \sum_{i=1}^{d} \tilde{s}_i(\vec{x})\,\big(x_i - \tilde{r}_i\big).$$

Since the Alon decomposition algorithm peels off one variable at a time in order, we must have

$$s_i(\vec{x}) \;=\; \tilde{s}_i(\vec{x}) \quad \text{for } i = 1, \ldots, d-2.$$

Subtract the equations and cancel the common terms:

$$\boxed{s_{d-1}(\vec{x})\,(x_{d-1} - r_{d-1}) - \tilde{s}_{d-1}(\vec{x})\,(x_{d-1} - \tilde{r}_{d-1}) + s_d(\vec{x})\,(x_d - r_d) - \tilde{s}_d(\vec{x})\,(x_d - \tilde{r}_d) = P(\tilde{\vec{r}}) - P(\vec{r})} \tag{5}$$

We get an analogous equation for the w_j:

$$\boxed{w_{d-1}(\vec{x})\,(x_{d-1} - r_{d-1}) - \tilde{w}_{d-1}(\vec{x})\,(x_{d-1} - \tilde{r}_{d-1}) + w_d(\vec{x})\,(x_d - r_d) - \tilde{w}_d(\vec{x})\,(x_d - \tilde{r}_d) = \sum_i t_i\,(\tilde{y}_{Q_i} - y_{Q_i})} \tag{6}$$

Note also that

$$P(\vec{r}) = \sum_i y_{Q_i} r_i (r_i - 1),$$

and

$$P(\tilde{\vec{r}}) = \sum_i \tilde{y}_{Q_i} \tilde{r}_i (\tilde{r}_i - 1).$$

The verifier will use these equations to force the prover to adhere to the difference-based Alon algorithm, whereby he cannot arbitrarily manipulate the univariate polynomials. In the protocol we are about to introduce, the verifier performs the same tests as in the invalid protocol, but using the variables:

$$s_i^\star \;:=\; \begin{cases} \varphi(s_i)(z) & i \le d-2, \\ \varphi\big(\tilde{s}_i\big)(z) & i = d-1, d, \end{cases} \tag{7}$$

$$w_i^\star \;:=\; \begin{cases} \varphi(w_i)(z) & i \le d-2, \\ \varphi\big(\tilde{w}_i\big)(z) & i = d-1, d. \end{cases} \tag{8}$$

3.4 An Interactive Proof for $\mathcal{R}_{\mathsf{Zero}}$

Zerocheck

Public input. The verifier receives the univariate polynomials $\varphi(Q_1), \ldots, \varphi(Q_d) \in \mathbb{F}[x]_{\leq nd-2}$ and the d-variate $P(\vec{x})$.

1. (**Random shift I**) $\mathcal{V}$ samples $\vec{r} \leftarrow \mathbb{F}^d$ and sends it.
2. (**First Alon decomposition**) $\mathcal{P}$ computes $s_1, \ldots, s_d$ satisfying $P(\vec{x}) - P(\vec{r}) = \sum_i s_i(\vec{x})\,(x_i - r_i)$ and sends the univariates $\varphi(s_i)(x)$ to $\mathcal{V}$.
3. (**Random weights**) $\mathcal{V}$ samples $t_1, \ldots, t_d \leftarrow \mathbb{F}$ and sends them.
4. (**Second Alon decomposition**) $\mathcal{P}$ computes $y_{Q_i} := Q_i(\vec{r})$ and polynomials $w_1, \ldots, w_d$, then sends the univariates $\varphi(w_i)(x)$ along with the y_{Qi}.
5. (**Random shift II**) $\mathcal{V}$ samples fresh $\tilde{r}_{d-1}, \tilde{r}_d \leftarrow \mathbb{F}$ and sends the two scalars.
6. (**6. Shifted values**) $\mathcal{P}$ sends
   a) the scalars $\tilde{y}_{Q_i} := Q_i(\tilde{\vec{r}})$ for every $i \in [d]$, and
   b) the two univariates $\varphi(\tilde{s}_{d-1}), \varphi(\tilde{w}_{d-1})$
7. (**Final check**) $\mathcal{V}$ chooses $z \leftarrow \mathbb{F}$, evaluates every received univariate at z, locally derives $\varphi(\tilde{s}_d)(z)$ and $\varphi(\tilde{w}_d)(z)$ via Eqs. (5)–(6), defines $s_i^\star, w_i^\star$ via Eqs. (7)–(8), and verifies

$$0 \overset{?}{=} \sum_{i=1}^{d} \varphi(Q_i)(z)\big(z^2 - z\big) - \sum_{i=1}^{d} \tilde{y}_{Q_i}\big(\tilde{r}_i^2 - \tilde{r}_i\big) - \sum_{i=1}^{d} s_i^\star\big(z - \tilde{r}_i\big), \tag{9}$$

$$0 \overset{?}{=} \sum_{i=1}^{d} t_i\big(\varphi(Q_i)(z) - \tilde{y}_{Q_i}\big) - \sum_{i=1}^{d} w_i^\star\big(z - \tilde{r}_i\big). \tag{10}$$

Protocol 1: Interactive Protocol for $\mathcal{R}_{\text{Zero}}$

Remark 3 (One–shot check). Sampling $\alpha \leftarrow \mathbb{F}$ and adding α times Eq.(9) to Eq.(10) yields a single identity to test, saving one evaluation.

Remark 4. If $\tilde{r}_{d-1} = r_{d-1}$ or $\tilde{r}_d = r_d$, the verifier resamples. Similarly in the case $z = \tilde{r}_d$ (this would lead to division by 0 for the verifier as she computes the values $s_d^\star, w_d^\star$).

Lemma 2 (Residual–polynomial lemma)**.** *Fix the verifier's randomness up to (but not including) the point $z \in \mathbb{F}$, and let the prover's fourth and sixth messages be*

$$\Big(\{\widehat{s}_i\}_{i=1}^{d},\ \{\widehat{\tilde{s}}_i\}_{i=d-1}^{d},\ \{\widehat{w}_i\}_{i=1}^{d},\ \{\widehat{\tilde{w}}_i\}_{i=d-1}^{d},\ \{\hat{y}_{Q_i}\}_{i=1}^{d},\ \{\widehat{\tilde{y}}_{Q_i}\}_{i=1}^{d}\Big).$$

Define the two univariates the verifier will later evaluate:

$$F(z) := \sum_{i=1}^{d} \varphi(Q_i)(z)\,(z^2 - z) - \sum_{i=1}^{d} \widehat{\tilde{y}}_{Q_i}\,(\tilde{r}_i^2 - \tilde{r}_i) - \sum_{i=1}^{d} s_i^\star(z)\,(z - \tilde{r}_i),$$

$$G(z) := \sum_{i=1}^{d} t_i\big(\varphi(Q_i)(z) - \widehat{\tilde{y}}_{Q_i}\big) - \sum_{i=1}^{d} w_i^\star(z)\,(z - \tilde{r}_i),$$

with $s_i^\star, w_i^\star$ taken from (7)–(8).

(a) Biconditional. *$F \equiv 0$ and $G \equiv 0$ if and only if every transmitted value is the honest one, i.e.*

$$(\widehat{s}_i, \widehat{\tilde{s}}_i, \widehat{w}_i, \widehat{\tilde{w}}_i, \hat{y}_{Q_i}, \widehat{\tilde{y}}_{Q_i}) = (s_i, \tilde{s}_i, w_i, \tilde{w}_i, y_{Q_i}, \tilde{y}_{Q_i}) \quad \forall i \in [d].$$

(b) Soundness for a single random evaluation. *If the prover deviates in* any *of the $2d+2$ univariate polynomials or the $2d$ scalars, then at least one of F, G is a non-zero univariate of degree $\le nd$. Consequently, for a uniformly random $z \xleftarrow{\$} \mathbb{F}$,*

$$\Pr_z\big[F(z) = 0 \wedge G(z) = 0\big] \;\le\; \frac{n\,d}{|\mathbb{F}|}.$$

Proof. (a) Biconditional. The "if" direction is immediate from the construction of F, G. For "only if", assume $F \equiv G \equiv 0$ and proceed in three steps:

(i) **Recover the scalars.** Substitute $z = \tilde{r}_j$ in $F \equiv 0$. All factors $(z - \tilde{r}_i)$ vanish except possibly when $i = j$; because $\tilde{r}_j \notin \{0, 1\}$, $(z^2 - z)$ stays non-zero. One isolates $\widehat{\tilde{y}}_{Q_j} = \tilde{y}_{Q_j}$. A similar substitution in G gives $\hat{y}_{Q_j} = y_{Q_j}$ for every j.

(ii) **Recover the s– and w–polynomials.** Plugging the now-honest scalars into the two identities around $\vec{r}$ yields $\sum_i (\widehat{s}_i - s_i)(x_i - r_i) \equiv 0$ and $\sum_i (\widehat{w}_i - w_i)(x_i - r_i) \equiv 0$. Setting each variable x_j free one at a time forces $\widehat{s}_j = s_j$ and $\widehat{w}_j = w_j$ for all j.

(iii) **Recover the "tilde" polynomials.** Equality of the first $d - 2$ quotients together with $F \equiv 0$ at $z = r_{d-1}$ and $z = r_d$ determines $\widehat{\tilde{s}}_{d-1} = \tilde{s}_{d-1}$, $\widehat{\tilde{s}}_d = \tilde{s}_d$, and analogously for $\tilde{w}$.

All transmitted data are therefore honest.

(b) Single-point soundness. If the prover deviates anywhere, at least one coefficient in F or G differs from the honest value, so the affected polynomial is non-zero. Its degree is $\le nd$ because $\deg \varphi(Q_i) \le nd - 2$ and the linear factors add at most 2. The univariate Schwartz–Zippel bound gives $\Pr[F(z) = 0] \le nd/|\mathbb{F}|$; the same holds for G, but the events are nested, so the same bound applies to their conjunction. □

Let S denote the size (number of gates) of an arithmetic circuit that simultaneously evaluates P and all Q_i at one point; this is the natural unit for prover work throughout the paper.

Theorem 5 (Properties of Protocol 1)**.** *Let $P, Q_1, \ldots, Q_d$ and the public parameters be as in Section 3, and let $d \ge 3$. The interactive protocol of Section 3.4 (Protocol 1) enjoys:*

(a) ***Perfect completeness:*** *if the identity $P(\vec{x}) = \sum_{i=1}^{d} Q_i(\vec{x})\, x_i(x_i - 1)$ holds as a polynomial, an honest prover is always accepted.*

(b) ***Soundness error:*** *every (computationally unbounded) cheating prover is accepted with probability at most $\dfrac{3\,nd}{|\mathbb{F}|}$.*

(c) ***Prover time:*** *$O\big(S + nd\big)$ field operations, dominated by the two difference-based Alon decompositions.*

(d) ***Verifier time:*** *$O(n\,d^2)$ field operations*

(two diagonal tests, each evaluating d univariates of degree $\le nd$, plus $O(d)$ scalar arithmetic).

(e) ***Communication:***

1) $2d{+}2$ *univariate polynomials sent by the prover* $\big(\varphi(s_i), \varphi(w_i), \varphi(\tilde{s}_{d-1}), \varphi(\tilde{w}_{d-1})\big)$ — *the public* $\varphi(Q_i)$ *are fixed input and incur no bandwidth,*
2) $(d+2)$ *field elements sent* from the verifier *for the two shift messages* $(\vec{r})$ *and* $(\tilde{r}_{d-1}, \tilde{r}_d)$,
3) d *scalar weights* (t_i),
4) $2d$ *scalar evaluations* $\big(y_{Q_i}, \tilde{y}_{Q_i}\big)$.

Total communication:

$$(2d+2)\,(nd+1) \;+\; 4d+2 \;=\; O(n\,d^2).$$

(f) ***Rounds:*** 6 *message exchanges.*

Proof. **(a) Completeness.** An honest prover computes the canonical Alon decompositions

$$P(\vec{x}) - P(\vec{r}) = \sum_{i=1}^{d} s_i(\vec{x})\,(x_i - r_i), \qquad \sum_{i=1}^{d} t_i\big(Q_i(\vec{x}) - y_{Q_i}\big) = \sum_{j=1}^{d} w_j(\vec{x})\,(x_j - r_j),$$

and the shifted pair $(\tilde{s}_{d-1}, \tilde{w}_{d-1})$ at $\tilde{\vec{r}}$ via Eqs. (5)–(6). Substituting the verifier's test point z into these identities gives the two equalities (9)–(10), so the verifier always accepts.

(b) Soundness. We prove that every (computationally unbounded) cheating prover convinces the verifier with probability at most

$$\Pr[\text{accept}] \;\le\; \frac{3\,n\,d}{|\mathbb{F}|}$$

over the verifier's coins.

Notation. Let

$$R(\vec{x}) \;:=\; P(\vec{x}) \;-\; \sum_{i=1}^{d} Q_i(\vec{x})\,x_i(x_i-1), \qquad 0 < \deg_{\text{tot}} R \le nd.$$

Bad event $\mathcal{E}_1$ ($R(\vec{r}) = 0$). For the first shift $\vec{r} \xleftarrow{\$} \mathbb{F}^d$, Schwartz–Zippel gives

$$\Pr_{\vec{r}}[R(\vec{r}) = 0] \;\le\; \frac{\deg_{\text{tot}} R}{|\mathbb{F}|} \;\le\; \frac{n\,d}{|\mathbb{F}|}.$$

Bad event $\mathcal{E}_2$ (forged univariates). Condition on $\vec{r}, \tilde{r}_{d-1}, \tilde{r}_d$. By Lemma 2, if the prover deviates in *any* of the $2d+2$ univariates or the $2d$ scalars $y_{Q_i}, \tilde{y}_{Q_i}$, at least one of F, G is a non-zero degree-$\le nd$ polynomial. The lemma then gives us

$$\Pr_{z}[F(z) = 0 \wedge G(z) = 0] \;\le\; \frac{nd}{|\mathbb{F}|}.$$

Thus $\Pr[\mathcal{E}_2] \le nd/|\mathbb{F}|$.

Bad event $\mathcal{E}_3$ (all messages honest but final check fails). Assume $\neg\mathcal{E}_1 \wedge \neg\mathcal{E}_2$. Then $F(z) = \varphi(R)(z)$. Since $R(\vec{r}) \neq 0$, F is non-zero of degree $\le nd$, whence

$$\Pr_{z}[F(z) = 0] \;\le\; \frac{nd}{|\mathbb{F}|}.$$

Conclusion. The events $\mathcal{E}_1, \mathcal{E}_2, \mathcal{E}_3$ are mutually exclusive and cover every accepting transcript, so

$$\Pr[\text{accept}] \;\le\; \frac{nd}{|\mathbb{F}|} + \frac{nd}{|\mathbb{F}|} + \frac{nd}{|\mathbb{F}|} \;=\; \boxed{\frac{3\,n\,d}{|\mathbb{F}|}}. \qquad \square$$

(c)-(f) Efficiency and communication. *Prover time* is governed by two runs of Algorithm 1, costing $O(n)$ per variable, hence $O(S + nd)$ in total. *Verifier time* is one evaluation of each of $3d$ degree-$\le nd-1$ univariates plus $O(d)$ scalar arithmetic, i.e. $O(n\,d^2)$ field operations. *Communication* counts the coefficients of the sent polynomials and the auxiliary scalars, as detailed in the statement. Finally, the *Rounds* consist of the following:

Message flow (6 rounds).

1. $\mathcal{V} \to \mathcal{P}$: the first shift vector $\vec{r} \in \mathbb{F}^d$ (d field elements).
2. $\mathcal{P} \to \mathcal{V}$:
   - the univariates $\varphi(s_1), \ldots, \varphi(s_d)$,
   - the d scalar evaluations $\big(y_{Q_1}, \ldots, y_{Q_d}\big)$.
3. $\mathcal{V} \to \mathcal{P}$: the weight tuple $(t_1, \ldots, t_d) \in \mathbb{F}^d$.
4. $\mathcal{P} \to \mathcal{V}$: the d univariates $\varphi(w_1), \ldots, \varphi(w_d)$.
5. $\mathcal{V} \to \mathcal{P}$: the two fresh coordinates $(\tilde{r}_{d-1}, \tilde{r}_d) \in \mathbb{F}^2$.
6. $\mathcal{P} \to \mathcal{V}$:
   - the univariates $\varphi(\tilde{s}_{d-1})$, $\varphi(\tilde{w}_{d-1})$,
   - the d scalar evaluations $\big(\tilde{y}_{Q_1}, \ldots, \tilde{y}_{Q_d}\big)$.

   (The verifier derives $\varphi(\tilde{s}_d)(z), \varphi(\tilde{w}_d)(z)$ herself at the test point z using Eqs. (5)–(6).)

All claimed properties are established.

4 Verifiable Computation Based on the Zero-Check IP

In this section we extend our zerocheck (Protocol 1, the six-round interactive IP of Section 3.4) to the full interactive proof for *layered arithmetic circuits*. As in GKR [GKR08], we represent a depth-ordered circuit layer by layer and check the consistency of all layers with *one* algebraic protocol. Our variant replaces every per-layer sum-check with a *single* invocation of the zerocheck IP.

Let d denote *both* the circuit depth and the arithmetisation dimension; every layer polynomial is written in the same d variables $x_1, \ldots, x_d$.

4.1 From Circuit Layers to One Zero-Check

A *layered* (a.k.a. leveled) circuit is an acyclic arithmetic circuit whose gates are partitioned into layers $L_0, \ldots, L_d$ such that edges point strictly from layer L_ℓ to $L_{\ell-1}$. We use d to denote the number of layers (the depth).

Layer polynomials. For each layer $\ell \in \{1, \dots, d\}$ the standard GKR arithmetisation gives a polynomial

$$P_\ell(\vec{x}) \;=\; f_\ell\big(a_{1,\ell}(\vec{x}), \dots, a_{k,\ell}(\vec{x})\big), \quad \deg(f_\ell) \le n,$$

where every $a_{j,\ell}$ is multilinear. Running AlonHypercubeDecompositionSymb (Sec. 2.2.2) with the fixed peeling order $x_1, \dots, x_d$ yields the canonical decomposition

$$P_\ell(\vec{x}) \;=\; \sum_{i=1}^{d} Q_{i,\ell}(\vec{x})\, x_i(x_i - 1), \qquad \deg Q_{i,\ell} \le n - 2.$$

Diagonal polynomials. The prover sends the univariate polynomials

$$\big\{\varphi(Q_{i,\ell}) : 1 \le i, \ell \le d\big\}, \qquad \deg \varphi(Q_{i,\ell}) \le d\,(n-2) \;<\; nd - 2.$$

Random aggregation. The verifier samples $\alpha_1, \dots, \alpha_d \xleftarrow{\$} \mathbb{F}$ and the prover sets

$$Q_i(\vec{x}) = \sum_{\ell=1}^{d} \alpha_\ell Q_{i,\ell}(\vec{x}), \qquad P(\vec{x}) = \sum_{i=1}^{d} Q_i(\vec{x})\, x_i(x_i - 1).$$

If any layer is wrong, then with probability at least $1 - \frac{nd}{|\mathbb{F}|}$ (over the α_ℓ) the aggregated identity fails on $\{0,1\}^d$.

Single zerocheck. The parties invoke Protocol 1 once on the input $\big(\{\varphi(Q_i)\}_{i=1}^d,\, P\big)$. Hence the total soundness error is

$$\frac{nd}{|\mathbb{F}|} \;+\; \frac{3\,nd}{|\mathbb{F}|} \;=\; \frac{4\,nd}{|\mathbb{F}|}.$$

Definition 3 (Layered-circuit relation)**.** Fix public parameters $\mathsf{pp} = (d, k, n)$ with $d > 2, n, k > 0$, and a field $\mathbb{F}$. The relation $\mathcal{R}_{\text{Layer}}$ contains those pairs

$$(\mathbb{x}, \omega) \;=\; \Big((\varphi(Q_{i,\ell})_{1 \le i,\ell \le d});\, (P_\ell)_{\ell=1}^{d}\Big)$$

that satisfy, *for every layer* $\ell \in \{1, \dots, d\}$,

(i) **Gate polynomial.** $P_\ell(\vec{x}) = f_\ell\big(a_{1,\ell}(\vec{x}), \dots, a_{k,\ell}(\vec{x})\big)$ with $\deg(f_\ell) = n$ and each $a_{j,\ell}$ multilinear in $\vec{x} = (x_1, \dots, x_d) \in \mathbb{F}^d$ (so $\deg P_\ell \le n$).

(ii) **Finite–difference factors.** $Q_{i,\ell}(\vec{x}) \in \mathbb{F}[x_1, \dots, x_d]_{\le n-2}$ for all $i \in [d]$.

(iii) **Layer identity on the hypercube.**

$$P_\ell(\vec{x}) \;=\; \sum_{i=1}^{d} Q_{i,\ell}(\vec{x})\; x_i(x_i - 1) \quad \text{in } \mathbb{F}[x_1, \dots, x_d]. \qquad \text{Zerocheck}_\ell$$

4.2 Efficient Alon differences for layered circuits

This section derives the *only* arithmetic surcharge that SNAIL introduces beyond a plain forward evaluation of a $w \times d$ layered circuit (total $S = wd$ gates). The key fact is that **all** finite–difference work is fused into the same streaming pass that produces the trace, so the prover's work remains $\Theta(S)$. In other words, even though the SNAIL protocol (Protocol 3) will rely on our zerocheck (Protocol 1), the latter can be performed in time linear in S, using the layered arithmetization of the former.

Gate template. Each gate is

$$g(\mathbf{x}) \;=\; f\big(a_1(\mathbf{x}),\dots,a_k(\mathbf{x})\big), \qquad \deg f \le n,$$

a_j multilinear in $x_1,\dots,x_d$.

Macro-operation. A *macro op* is one gate evaluation; in a forward sweep it bundles $k\,d+(n+1)$ scalar field operations (k affine evaluations, each using d adds, plus n multiplies and one final add for Horner).

1. Baseline cost

Let each gate have k input wires $a_j(\mathbf{x}) \;=\; \sum_{b\in\{0,1\}^d} \lambda_{j,b} \prod_{t=1}^{d} x_t^{b_t}$, i.e. a multilinear polynomial of degree ≤ 1 in every variable (live-variable depth is d).

Evaluating one multilinear wire. Write the gate and parent indices with d bits each. The selector $\widetilde{\mathsf{left}}$ (and its siblings) is the product of d linear terms, one per bit: $\prod_{t=1}^{d}(u_t v_t + (1-u_t)(1-v_t))$. At a Boolean point $(u,v)\in\{0,1\}^{2d}$ this costs d multiplications (no additions).

Cost model. Following the common GKR convention, we count *field multiplications* as the dominating cost once data are resident in registers and treat additions as free.[3] Under this model a Horner evaluation of a degree-n univariate costs exactly n scalar operations.

Macro-operation (one gate). A forward sweep therefore performs (fixed fan-in k) $k\,d$ multiplications for the k parent selectors and n additional multiplications (plus one add) for the Horner evaluation of f. Hence a macro-op bundles $k\,d+n+1$ scalar operations, of which exactly $k\,d+n$ are multiplies charged by our cost model. A circuit with S gates costs $\boxed{S}$ such macro-ops.

2. Three Difference–Alon updates, fused in one pass

Conceptually the prover performs:

(i) **Layer compaction** – build $Q_{i,\ell} = x_i(x_i-1)F_\ell(\mathbf{x})$ for each layer ℓ;

(ii) s_i–**decomposition** of $P(\mathbf{x}) - P(\mathbf{r})$ where $P = \sum_i Q_i$;

(iii) w_j–**decomposition** on a random field combination of the Q_i's.

It executes all three while the gate struct is still hot in cache, so no new DRAM traffic arises.

3. Surcharge for one Difference–Alon update

While the gate's selectors are still in registers we reuse the two "frozen" evaluations $a_j|_{x_i=0},\ a_j|_{x_i=1}$ already computed for the forward step:

- k subtractions/additions to toggle the chosen bit,
- $n+1$ scalar ops to re-evaluate f.

Total per update: $k+n+1$ scalar operations.

[3] On modern SIMD/SIMT hardware, one addition and one multiplication issue per ALU cycle and are equally memory-bound, so charging only multiplies gives the standard tight bound.

4. All three updates

Each gate contributes to the zerocheck three times:

(i) layer–compaction update,

(ii) s_i–decomposition update,

(iii) random w_j–decomposition update.

Because the three reductions are *independent* and are executed while the gate data are still in registers/cache, their arithmetic costs *add but do not multiply.* A single update needs $(k + n + 1)$ scalar operations (§3), so the gate's total contribution is

$$3 \times (k + n + 1) \;=\; 3\big(k + n + 1\big) \quad \text{scalar ops.}$$

There is **no additional DRAM traffic**: all three updates are fused into the streaming kernel that already performs the forward evaluation.

5. Macro-level surcharge factor

To compare the extra work with the baseline we divide the gate–surcharge by the cost of a *macro-operation* (one forward evaluation):

$$c(d, k, n) \;=\; \frac{\text{extra scalar ops per gate}}{\text{baseline scalar ops per gate}} \;=\; \frac{3\,(k + n + 1)}{k\,d + n + 1}.$$

- *Numerator* $3(k + n + 1)$. Three Difference–Alon updates, each reusing the k cached wire values and re-evaluating the degree-$\le n$ gate polynomial.
- *Denominator* $k\,d + n + 1$. A forward evaluation must compute k multilinear selectors over d bits ($k\,d$ multiplies) and evaluate f ($n + 1$ scalar ops).

Since the numerator is constant once (k, n) are fixed while the denominator grows linearly with depth d, the surcharge scales like $c(d, k, n) = \Theta(1/d)$. Consequently the prover's total arithmetic

$$\big(1 + c(d, k, n)\big)\, S$$

is $\Theta(S)$ for any fixed depth and repetition count, and the multiplicative overhead becomes smaller as circuits get deeper.

Verifier recap. The verifier inspects only the accumulator vectors and therefore pays $O(n_{\text{rep}}\, d^2)$ scalar work, independent of S.

Take-away. For any fixed gate arity k and degree n, the numerator of $c(d, k, n)$ is a constant, while the denominator grows linearly with the circuit depth d. Hence

$$c(d, k, n) \;=\; \Theta\big(1/d\big) \quad\Longrightarrow\quad T_{\text{prov}} = \big(1 + n_{\text{rep}}\, c(d, k, n)\big)\, S = \Theta(S).$$

In words: the SNAIL prover remains *linear in the total gate count* S, its multiplicative overhead shrinks as depth increases, and it still requires only a single sequential pass over memory. Additional zerocheck repetitions scale runtime by the same constant factor but do not change the overall $\Theta(S)$ behaviour or the one-pass I/O profile.

4.3 An Eight-Round Interactive Protocol for Layered Circuits

Interactive Proof for $\mathcal{R}_{\mathbf{Layer}}(d, k, n)$

Public parameters. Depth d; gate arity k; degree bound n; circuit size S; field $\mathbb{F}$ with $|\mathbb{F}| > 4nd$.

1. **Layer commitments (Prover → Verifier).**
   For every layer $\ell \in [d]$ compute $\big(Q_{1,\ell}, \ldots, Q_{d,\ell}\big)$ and send the d^2 univariate polynomials
   $$\big\{\varphi(Q_{i,\ell}) : 1 \le i, \ell \le d\big\}.$$
2. **Aggregation challenge (Verifier → Prover).**
   The verifier samples and sends $\alpha_1, \ldots, \alpha_d \leftarrow \mathbb{F}$.

   The prover now computes
   $$Q_i := \sum_{\ell=1}^{d} \alpha_\ell Q_{i,\ell}, \qquad P(\vec{x}) := \sum_{i=1}^{d} Q_i(\vec{x})\, x_i(x_i - 1),$$
   and the verifier forms the d univariates $\varphi(Q_1), \ldots, \varphi(Q_d)$ by the same linear combination of her stored $\varphi(Q_{i,\ell})$.
3. **Zero-check (6 rounds).**
   Run Protocol 1 on the aggregated public input
   $$\big(\,\varphi(Q_1), \ldots, \varphi(Q_d)\,;\; P\big).$$
   The verifier accepts iff the zerocheck accepts.

Protocol 2: All layers are correct iff every $(P_\ell, Q_{1,\ell}, \ldots, Q_{d,\ell})$ satisfies Zerocheck_ℓ.

Theorem 6 (Protocol 2: correctness, soundness, efficiency)**.** *Fix gate arity k and degree bound n. Let $c(d, k, n) = \frac{3(k+n+1)}{k\,d+n+1}$ as derived in Section 4.2. Protocol 2 achieves:*

(a) ***Perfect completeness.***

(b) ***Soundness error*** $\le \dfrac{4\,n\,d}{|\mathbb{F}|}$

(c) ***Prover time*** $\big(1 + c(d, k, n)\big)\, S$.

(d) ***Verifier time*** $O\big(nd^3\big)$ *(evaluation of $d^2 + 2d + 2$ univariates of degree $\le nd$).*

(e) ***Communication*** $(d^2 + 2d + 2)$ *univariate polynomials, each of degree $\le nd$ (thus $(d^2 + 2d + 2)(nd + 1)$ coefficients)*
$+ 4d$ *field elements* $= O\big(nd^3\big)$ *coefficients.*

(f) ***Rounds*** 8.

Proof. **(a) Perfect completeness.** Unchanged from the original argument.

(b) Soundness. As before, aggregation fails with probability $\le nd/|\mathbb{F}|$ and the zero-check adds $\le 3nd/|\mathbb{F}|$.

(c) Efficiency.

- *Prover.* A forward sweep costs S macro operations (§4.2, Part 1). The three fused Difference–Alon updates add the multiplicative factor $1+c(d,k,n)$; hence $\big(1+c(d,k,n)\big)S$.
- *Verifier.* Same as before: d^2+2d+2 univariates of degree $\le nd$ cost $O(nd^3)$ scalar operations in total.
- *Communication.* Counts exactly the data written to the transcript and is unchanged.

All bounds follow. □

4.4 Why classical information-theoretic lower bounds do not bite

A natural concern is that Protocol 2 appears to beat celebrated impossibility results that rule out very low-cost proofs for *arbitrary* languages. Here we recall the two main theorems and explain, in one paragraph each, why their assumptions fail for our algebraic setting.

Goldreich–Håstad (black-box) bounds. The theorem of Goldreich and Håstad [GH98] shows that, for NP-complete languages equipped with a *black-box* witness oracle, any interactive proof that makes $q \ll m$ queries (sub-linear in the witness length m) can be sound only with inverse-exponential probability. Protocol 2 circumvents this barrier because the verifier is *not* restricted to black-box access: the prover reveals the full polynomial structure of the witness evaluation circuit. Multilinearity and low individual degree give the verifier far more powerful algebraic tests than the bit-probe oracle assumed in the lower-bound model. Hence the Goldreich–Håstad argument does not apply.

Babai's linear-communication lower bound. Babai and Moran [BM88] proved that an Arthur–Merlin protocol for must incur $\Omega(m)$ total communication when the verifier treats the witness as an arbitrary m-bit string. Again, Protocol 2 escapes because its witness is *structured*: an assignment to a depth- and fan-in-bounded arithmetic circuit. The prover sends *polynomials*, not raw witness bits, and the diagonalisation trick compresses an exponentially large truth table into d^2 univariate low-degree polynomials. Such algebraic compressions are explicitly disallowed in the communication-complexity framework of [BM88].

Take-away. Both lower bounds hinge on a *black-box* view of the witness. Protocol 2 leverages the rich internal structure of arithmetic circuits (multilinearity, low per-variable degree, and the ability to evaluate polynomials at chosen points). Diagonalisation does not leak information about an arbitrary string, but it *does* let the verifier check a concise algebraic description of the same computation. Consequently, Protocol 2 is fully consistent with existing information-theoretic impossibility results.

5 The SNAIL Protocol

We obtain the SNAIL (Succinct, Non-interactive, Alon-compressed, Instant proof for Layered circuits) protocol by applying the Fiat–Shamir transform to the public-coin interactive proof of Protocol 2 in the random-oracle model (ROM).

Random oracle. Let $\mathcal{H} : \{0,1\}^* \to \mathbb{F}$ be modelled as a random oracle. For each query tag we treat $\mathcal{H}(\text{tag})$ as a uniformly random element of $\mathbb{F}$.

SNAIL proof system

Common input. Depth d; gate arity k; degree bound n; circuit size S; field $\mathbb{F}$ with $|\mathbb{F}| > 4nd$, and the circuit's output claim.

Prover algorithm.

1. Compute the d^2 layer univariates $\varphi(Q_{i,\ell})$ and append them to the transcript T.
2. Derive $\alpha_1, \ldots, \alpha_d := \mathcal{H}(T \,\|\, \text{“”})$.
3. Aggregate $Q_i = \sum_\ell \alpha_\ell Q_{i,\ell}$ and form $P(\vec{x}) = \sum_i Q_i(\vec{x})\, x_i(x_i - 1)$.
4. Emulate the six rounds of the zero-check: after every prover message append it to T; every verifier challenge is taken as $\mathcal{H}(T \,\|\, \mathsf{tag})$.

Output $\pi := T$.

Verifier algorithm. Recompute the challenges via $\mathcal{H}$, evaluate all received univariates at the derived points, and run the zerocheck tests exactly as in Protocol 1. Accept iff every test passes.

Protocol 3: SNAIL: A non-interactive argument for $\mathcal{R}_{\text{Layer}}(d, k, n)$

Theorem 7 (SNAIL metrics)**.** *Assume the random-oracle model. Then Protocol 3 is a public-parameter–free non-interactive argument with:*

- ***Perfect completeness.***
- ***Computational soundness error*** $\leq \dfrac{4nd}{|\mathbb{F}|}$.
- ***Prover time*** $\big(1 + c(d,k,n)\big)S \;+\; O(kd + n)$. (hash, transcript I/O)
- ***Verifier time*** $O\big(nd^3\big)$.
- ***Proof size*** $(d^2 + 2d + 2)(nd + 1) + 4d = O\big(nd^3\big)$ *field elements.*

Proof. Completeness is inherited verbatim from the interactive protocol. For *soundness*, any adversary $\mathcal{A}$ producing an accepting proof π can be rewound at each invocation of $\mathcal{H}$ to program the oracle with fresh randomness, thereby yielding an adversary that breaks the 8-round interactive proof with the same success probability; hence the bound $4nd/|\mathbb{F}|$ from Theorem 6 carries over. Efficiency and proof size follow by counting exactly the data written into the transcript: $d^2 + 2d + 2$ univariate polynomials of degree $\leq nd$ plus $4d$ auxiliary scalars. Fiat–Shamir removes all interaction, yielding a single-message argument. □

Remark 5 (Why the $O(kd + n)$ term is dropped later). The additive overhead $O(kd + n)$ in the prover bound stems only from (i) the single Fiat–Shamir hash over $\leq 2\,(d^2 + 2d + 2) + 4d = O(d^2)$ field words and (ii) serialising those words into the transcript. For every circuit of interest we have $S \gg kd$, so

$$\frac{O(kd + n)}{\big(1 + c(d,k,n)\big)S} \;\leq\; \frac{O(kd)}{S} \;=\; O\big(1/w\big) \;\ll\; 1.$$

Thus the lower-order term vanishes in both asymptotic analysis and empirical timings, and we omit it in all subsequent big-O and constant-factor discussions without affecting any headline numbers.

5.1 Layer-by-layer cost ledger

The previous sections showed that the prover's arithmetic cost is

$$T_{\text{prov}}(S,d,k,n) \;=\; \big(1+\,c(d,k,n)\big)S \;=\; \Theta(S), \quad c(d,k,n)=\frac{3(k+n+1)}{k\,d+n+1},$$

while its I/O consists of a *single* sequential read & write of the S-gate trace. This reminder of the paper will explore how those facts translate into *practical design rules* for a verifiable virtual machine (zkVM) and gives one concrete hardware snapshot (RTX-4090) purely as a sanity check.

Table 1: Everything the SNAIL prover ever does.

Step	A / M	Comment
(0) Trace write	M	one 8-byte store per gate
(1) Forward eval	A	S macro ops ($k\,d+n+1$ scalars ea.)
(2) 3 Diff–Alon	A	$c(d,k,n)\,S$ extra macro ops; no new M
(3) Hash & package	–	$O(kd+n)$ scalar ops; $\leq$1 kB comm.

Byte note. The "8-byte store" assumes a 64-bit field element (e.g. Goldilocks). For 256-bit prime fields the per-gate footprint is 32 B; the arithmetic accounting is unchanged.

Why no further terms appear.

- Step (0) is already part of (1)'s DRAM pass.
- Steps (2) and (1) share registers, hence add *only* arithmetic.
- Step (3) is sub-linear in S and therefore negligible once $S \gg kd$.

Result: **one** DRAM pass of length S and the single multiplicative factor $(1+\,c(d,k,n))$.

5.2 Depth–width trade-offs

Let a layered circuit have depth d and width $w = S/d$. With $(k,n)=(4,4)$ we have

$$T_{\text{prov}}(S,d) = \Big(1+\tfrac{6.75}{d}\Big)\,\frac{S}{f_{\text{op}}}, \qquad |\pi| = (d^2+2d+2)(4d+1) = O(d^3),$$

(The factor $4d+1$ comes from substituting the concrete gate degree $n=4$ into the general $(nd+1)$ term of Section 4.3).
where f_{op} is the sustained throughput (field ops $\cdot$ s^{-1}) for the *slower* of addition and multiplication on the target device.

a) **Arithmetic term.** The overhead $(1+6.75/d)$ falls as $1/d$; doubling d nearly halves the gap to native execution.

b) **Memory cap.** Each gate stores $k\,d$ field elements. A 24 GiB card fits a 2.6×10^9-gate shard up to $d\approx32$; beyond that the trace spills off-card.

c) **Proof size & verifier.** Both grow cubically with depth: 60 k coeffs at $d=24$, 140 k at $d=32$, 270 k at $d=40$.

d) **Recommended band.**

$$\boxed{24 \;\leq\; d \;\leq\; 32}$$

keeps the prover within 1.28–1.21$\times$ of the information-theoretic floor and the proof below $\sim$140 k coefficients.

Why $k = n = 4$ is a sweet spot. Four multilinear inputs already encode a full RISC-V style instruction (`rd`, `rs1`, `rs2`, `funct`), cover common crypto primitives with degree $n = 4$, and minimise the surcharge numerator $3(k + n + 1)$. Shrinking k forces deeper circuits; enlarging k yields no further saving once $k \geq n$.

Table 2: Prover ratio and proof length for two representative depths.

Depth d	Slow-down $1 + \frac{6.75}{d}$	Proof coeffs
24	1.28	6.0×10^4
32	1.21	1.4×10^5

Concrete snapshot (RTX-4090). For a shard of $S = 2.6 \times 10^9$ gates and depth $d = 32$ we have

$$T_{\text{prov}} = 1.21 \, \frac{S}{f_{\text{op}}} = 1.21 \times \frac{2.6{\times}10^9}{1.1{\times}10^{13}} \approx 38 \text{ ms},$$

versus 31 ms for plain execution on the same RTX-4090. The **1.21** ratio is hardware-independent; faster GPUs bring both numbers down proportionally.

5.3 Toward a SNAILy zkVM

Section 5.2 showed that, for any machine,

$$\text{Overhead}(d) = 1 + \frac{6.75}{d}, \qquad |\pi| = O(d^3), \qquad T_{\text{prov}}(S, d) = \left(1 + \tfrac{6.75}{d}\right) \frac{S}{f_{\text{op}}},$$

where f_{op} is the sustained field-operation throughput (add or mul) of the target. The *ratio* $1 + \frac{6.75}{d}$ is universal; only the scale factor $1/f_{\text{op}}$ depends on hardware.

Gadgets for Crypto, RAM

Heavy gadgets add only a handful of layers per macro-step:

Pre-compile	extra depth d_{g}	fan-out	note
Poseidon2-128	10	1×	5 S-boxes + 5 MDS (deg. 2)
Ed25519 scalar mul	8	3×	31 window mults in 8 layers
RAM sorted-table	4	1×	2 range + 2 perm layers

Even SHA-256 (64 rounds $\approx$ 11 layers) fits comfortably below the $d \leq 32$ budget once amortised over several instruction steps.

Memory-consistency check (width K). Loads and stores of a superscalar step with width K[4] are verified with a *sorted-table-RAM* argument:

(i) collect the K accesses of a macro-step,

(ii) bitonic-sort by address (depth $\log_2 K$),

[4]**Notation.** Capital K always denotes the *superscalar width*—the number of parallel instructions processed in a macro-step. Lower-case k is reserved for the gate arity (the number of multilinear inputs inside a single circuit gate). Choosing $K = 256$ balances two factors: (i) the log-depth sorter stays ≤ 8 layers; (ii) SIMD/SIMT engines on modern CPUs/GPUs natively handle 256-lane batches, maximising utilisation without register spill.

(iii) check last-write-wins in one linear pass.

With a typical superscalar width $K = 256$ the RAM gadget costs $d_{\mathrm{RAM}} = \log_2 K = 8$ layers—well inside the 32-layer envelope.

5.3.1 Why width matters

Depth now fixed, the only way to improve throughput is widening the layers.

- *Linear scaling.* $T_{\mathrm{prov}} \propto S = d\,w$ for fixed d.
- *No extra cryptography from width.* Hashes, signatures, RAM permutations, etc. enter only as *additional layers*. Packing more gates into an existing layer (increasing w) does **not** create new hashes or any FFT[5], so the cryptographic cost is already accounted for in the depth term.
- *Proof size untouched.* Because $|\pi| = O(d^3)$, widening layers has no effect on verifier time or bandwidth.
- *Parallel hardware happy.* SIMD/SIMT units evaluate the w gates of one layer in lock-step with one memory pass.
- *Practical guidance.* Once a depth that fits device memory is chosen, maximise width: throughput rises linearly, while the slow-down ratio and proof length remain constant.

Therefore, after fixing d by the rule above, simply **pack layers as wide as your application allows**: you obtain more throughput at the same 21–28 % overhead and without enlarging the proof.

Key take-aways

i) **Universal overhead curve.** For any hardware the prover slow-down is $1 + \frac{6.75}{d}$; device speed only scales milliseconds.

ii) **Sweet-spot depth is machine-size driven.** Today's commodity 16–32 GiB devices make the band $d \in [24, 32]$ optimal: sub-30 % overhead, proof ≤ 140 k coefficients, trace fits in RAM.

iii) **Width is free.** After depth is fixed, widening layers increases latency linearly but adds *zero* hashes, FFTs, or proof bytes.

iv) **Four-input, degree-4 gates hit the Pareto front.** They encode a full RISC-V instruction set, realise common crypto gadgets in ≤ 10 layers, and minimise the surcharge constant $c(d, k, n)$.

v) **One-pass I/O.** The entire trace (no matter how wide) is written once and read once; all Difference–Alon algebra happens while the data sit in registers.

vi) **Verifiable VM at 1.2× native.** Within the depth band above, a superscalar SNAIL-based zkVM can deliver end-to-end proofs in real-time (hundreds of milliseconds for multi-billion-gate shards) on off-the-shelf hardware.

These points let circuit designers treat SNAIL almost like a hardware cycle counter: pick a depth that fits, crank up width for throughput, and obtain a sub-30 % cryptographic audit "for free."

[5]The SNAIL prover contains *no* FFT/FRI or MSM kernels; all cryptography is confined to a single Fiat–Shamir hash after the streaming pass.

References

[Alo99] Noga Alon. Combinatorial Nullstellensatz. 8(1–2):7–29, 1999.

[BM88] László Babai and Shlomo Moran. Arthur-Merlin games: A randomized proof system, and a hierarchy of complexity classes. 36(2):254–276, 1988.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity, 2018.

[FS87] Amos Fiat and Adi Shamir. How To Prove Yourself: Practical Solutions to Identification and Signature Problems. In Andrew M. Odlyzko, editor, *Advances in Cryptology — CRYPTO' 86*, pages 186–194. Springer, 1987.

[GH98] Oded Goldreich and Johan Håstad. On the complexity of interactive proofs with bounded communication. 67(4):205–214, 1998.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *Proceedings of the Fortieth Annual ACM Symposium on Theory of Computing*, STOC '08, pages 113–122. Association for Computing Machinery, 2008.

[Gro16] Jens Groth. On the Size of Pairing-based Non-interactive Arguments, 2016.

[GWC19] Ariel Gabizon, Zachary J. Williamson, and Oana Ciobotaru. PLONK: Permutations over Lagrange-bases for Oecumenical Noninteractive arguments of Knowledge, 2019.