Weir: Bounded-Memory Indexing and Proving with Sequential Storage

Kyeongtae Lee
Kookmin University
Seoul, Republic of Korea
sklee63kr@gmail.com

Jihye Kim
Kookmin University
Seoul, Republic of Korea
jihyek@kookmin.ac.kr

Hyunok Oh
Hanyang University
Seoul, Republic of Korea
hoh@hanyang.ac.kr

Abstract

A low-memory prover must control sumcheck tables, sparse accesses, commitments and proof assembly. We present Weir, a Shockwave-derived protocol with a complete read–write (RW) index-and-prove schedule. The schedule restores sparse request order, retains commitment state through later openings and assembles the proof externally. These steps preserve the direct RAM prover's canonical bytes across buffer choices, with a common verifier. Under an external-query-tape model, the schedule uses $O(B + \log T)$ field RAM, constant digest RAM and polylogarithmic auxiliary RAM. Algebraic work and record traffic are quasilinear; oracle-query byte work is charged separately. A classical random-oracle analysis bounds knowledge error.

Fused Split–Butterfly–Merge removes $n - 1$ materialized transform passes at length 2^n. Its ordered routing contract extends to multiplicative-coset Reed–Solomon (RS) and recursive foldable encoders; component checks match reference codewords, traffic counts and RS fold states. In a fixed 64 MiB worker-cap grid, RW completes all tested matrix and Poseidon inputs; RAM completes matrix dimension 8. Within the matrix grid, the largest RW point has 64 times as many padded constraints as that RAM point, using 3.33 MB prover RSS and 9.51 minutes index+prove. We report latency, external payload, query bytes, proof size and verification time for this offline operating range.

1 Introduction

Succinct non-interactive arguments of knowledge (SNARKs) let a verifier check a computation from a short proof [16, 24]. Producing that proof can require large intermediate tables, making prover memory an obstacle to scaling. Streaming and elastic provers address this cost through different time–space tradeoffs [9, 10, 22]. Read–write (RW) streaming additionally places intermediate state in sequential external storage [4]. A whole prover must control all of this state: streaming a fast Fourier transform (FFT) alone can still leave a witness-sized sumcheck table, a sparse gather result, a column accumulator, or a complete proof object in RAM.

The difficulty lies at the interfaces between these operations. A sparse gather can return the right multiset in the wrong request order, changing the polynomial evaluated by its consumer. Commitment state must survive until sampled openings, and transcript events must follow their own order before the proof is serialized. Weir resolves these interfaces for a Shockwave-derived profile with Reed–Solomon (RS) encoding and hash commitments, from sparse indexing to canonical proof output.

A complete streaming prover. Our first contribution is a schedule covering sparse indexing, four commitments, all proving phases, full-prefix oracle queries and canonical output (Section 6).

For the size T of Section 3, the schedule uses $O(B + \log T)$ field RAM, constant digest RAM and polylogarithmic auxiliary RAM. Algebraic work and record traffic are quasilinear; oracle-query byte traffic is charged separately. We prove exact replacement of the direct prover and establish knowledge soundness for their common verifier in the classical random-oracle model (ROM) [6].

Ordered execution and exact traffic savings. The schedule enforces these interface obligations (Table 3). Fused Split–Butterfly–Merge (FSBM) preserves each encoder coordinate while removing $n - 1$ full transform passes at $N = 2^n$, including block-local endpoint batching. Matched controls isolate the added routing from endpoint and zero-tail optimizations.

Transfer to code encoders. The paired-stage theorem makes routing independent of coefficients. We map its sequential-access contract to multiplicative-coset RS and BaseFold-form recursive generators, including a field without the required FFT roots (Section 4.1). Component experiments check ordered codewords and RS folding against independent references (Section 8.3).

Scope and evidence. We target offline workers with a prepared sparse instance and witness, a fixed RAM quota and sequential scratch storage. An operator already using the RS/hash profile can select RAM or RW execution and its buffer size per worker while retaining the same proof grammar and verifier. The choice trades latency and scratch capacity against RAM; Section 3 fixes the storage and oracle interfaces.

Our 64 MiB study measures completed workload, latency, external payload and proof-delivery costs (Section 8).

Contribution boundary.

Inherited	New in this paper
Scribe's blocks and composition contract	Ordered adapters and simultaneous live-state bound for complete index-and-prove
Shockwave's RS path	Fixed Weir coordinates, events, canonical bytes and security analysis
Sluice's SBM and fusion observation	Exact ordered-stage routing, endpoint batching and code-family mappings

2 Related Work

Prover memory. Block et al. [9] study space-efficient arguments with streaming access to proving data. Gemini [10] offers prover configurations with different time–space costs and a common output proof. Hobbit [22] combines streaming sumcheck with a space-efficient multilinear commitment to obtain optimal prover time. Table 1 compares closely related streaming provers and their input and storage boundaries.

Our closest scheduling predecessors are Sluice and Scribe. Sluice [19] applies SBM to a read–write Groth16 prover [16]. Its transform uses decimation in time (DIT) with initial bit reversal; our RS encoder uses natural-input decimation in frequency (DIF). Sluice identifies order-compatible fusion and possible FFT/RS reuse [19, §3.6, §5.2]. We make the ordered execution contract explicit and map it to coset RS and a recursive foldable generator; Section 4.1 states the coefficient-access condition. Exact DIF savings separate boundary and endpoint fusion (Section B.2). Scribe [4, 5]

supplies the RW model, composition contract, reusable blocks and a complete HyperPlonk prover. Its streaming PCS paths use group operations [5, Table 2, §2.8]. Weir closes a different path: field-based RS encoding, hash-based column commitments and sampled-column openings. Sparse indexing, column relayout, retained Merkle trees and event/wire order all remain within its resource bound.

Work	Protocol and preserved interface	Given inputs / preprocessing	Storage and technique
Sluice [19]	Groth16 proof and verifier	Proving worker after CRS, QAP/R1CS and witness streams	Sequential external streams; SBM and chunked MSM
Scribe [4, 5]	HyperPlonk PIOP with RW PCS	PIOP indexer supplies selector and wiring polynomials	Logarithmic RAM, linear external storage; reusable RW blocks
Hobbit [22]	Modified HyperPlonk; buffer-dependent proof size	Circuit evaluation generates streams; wiring checks preprocessed	Buffer plus evaluation state; streaming sumcheck and tensor codes
Weir	Common verifier and fixed canonical proof across B	Sparse instance/witness streams; sparse index and proof assembly included	Sequential records/query bytes; FSBM and ordered composition

Table 1: Input, preprocessing and storage contracts of related provers.

Polynomial commitments and proof systems. Spartan [24] supplies the R1CS and sparse-evaluation setting. Brakedown and its Shockwave variant [14, 15] use code-based commitments; Shockwave selects RS encoding. Ligero [2] also builds arguments from interleaved RS codes. Weir retains Shockwave's four commitment sites and makes their ordered encoding, opening and transcript rules explicit. Section 8.6 compares the pinned historical implementation's memory-cap outcomes under its concrete transcript.

Other commitment constructions change the encoding or opening protocol. FRI [7] tests RS proximity by recursive folding; BaseFold [28] extends this approach to foldable codes over more general fields. PolyFRIM [30] and DeepFold [17] develop efficient multilinear commitments based on RS proximity proofs. HyperPlonk [11] uses multilinear representations to avoid the FFT bottleneck in its arithmetization. Section 4.1 maps multiplicative-coset RS encoding and the fixed-basis BaseFold recurrence to our paired-stage contract. Our extraction proof uses the RS proximity bound of Ben-Sasson et al. [8], with the parameter mapping given in Lemma E.7.

Data movement. External-memory FFTs [3], the red–blue pebble model [18], block-transfer algorithms [1, 26] and cache-oblivious algorithms [13] study computation under different storage interfaces. Our interface charges individual sequential record reads and appends. Within it, FSBM removes materialization passes from a specified SBM schedule, and its $2N$ routing lower bound applies to fresh materialized outputs.

3 Problem and Reused Building Blocks

Weir proves a prepared rank-one constraint system (R1CS): each constraint equates the product of two linear forms to a third. The sparse instance, public input and witness arrive as ordered streams. Our RW bound includes indexing and proof assembly; the security experiment gives the adversary an honestly generated public index. The guarantee is knowledge in the classical ROM. Witness generation, zero knowledge and extraction-memory bounds are outside this construction.

Statement and size. The relation below uses the field $\mathbb{F}$ specified in Section D. Stored duplicates contribute additively; explicit zero entries remain in the ordered lists.

Definition 3.1 (Stored instance, witness, and relation)**.** An instance is $\mathsf{inst} = (C_{\mathrm{r1cs}}, V, I, \mathcal{A}, \mathcal{B}, \mathcal{C})$, where $C_{\mathrm{r1cs}} = 2^c$, $V = 2^v$, $c, v \geq 0$ are integers, $0 \leq I < V$, and each $\mathcal{X}$ is a finite ordered list of triples $(a_{X,k}, b_{X,k}, \nu_{X,k})$ with $0 \leq a_{X,k} < C_{\mathrm{r1cs}}$, $0 \leq b_{X,k} < 2V$, and $\nu_{X,k} \in \mathbb{F}$. Let $q_X = |\mathcal{X}|$, $O = 2^o = \mathrm{np2}(\max_X q_X)$ with $\mathrm{np2}(0) = 1$, and $H = 2^h = \max(C_{\mathrm{r1cs}}, 2V)$. The selected simultaneous shape predicate is $7 \leq v \leq 43$, $4 \leq o \leq 41$, $c \leq 44$. For $x \in \mathbb{F}^I$ and $w \in \mathbb{F}^W$, $0 \leq W \leq V$, define

$$w^+ = w \| 0^{V-W}, \qquad z^{\mathrm{assn}} = w^+ \| (1) \| x \| 0^{V-I-1} \in \mathbb{F}^{2V}. \tag{1}$$

For $X \in \{A, B, C\}$ and $0 \leq i < C_{\mathrm{r1cs}}$ set

$$(Xz^{\mathrm{assn}})_i := \sum_{k : a_{X,k} = i} \nu_{X,k} z^{\mathrm{assn}}_{b_{X,k}}.$$

The relation $\mathcal{R}_{\mathrm{R1CS}}$ contains $((\mathsf{inst}, x), w)$ exactly when these types hold and

$$(Az^{\mathrm{assn}})_i (Bz^{\mathrm{assn}})_i = (Cz^{\mathrm{assn}})_i \quad \text{for every } 0 \leq i < C_{\mathrm{r1cs}}. \tag{2}$$

Empty sums are zero; the relation is independent of prover challenges and buffer size.

Notation guide. The evaluation abbreviates C_{r1cs} as C. Context tuples $C(x)$, sparse matrix $\mathcal{C}$, and codeword matrix $\mathsf{E}[i, q]$ have separate notation. The sparse matrix $\mathcal{B}$ is distinct from the local buffer B.

Symbol	Size or role
C, V	Padded constraints; witness-half capacity
O, H	Padded entries per matrix; $\max(C, 2V)$
I, W	Supplied public/private scalar counts
N, B	Transform length; local field buffer
T	Whole resource size defined below

The padding to O retains stored duplicates and explicit zeros.

For an application-context byte string a, our resource size is

$$T = 2 + C_{\mathrm{r1cs}} + 2V + O + H + |a|.$$

Thus T includes the dimensions of the generated dense tables, not only the bit length of their dimension headers.

Arithmetic and coordinates. Write $[L] = \{0, \dots, L-1\}$. All table indices are most-significant-bit (MSB)-first Boolean strings. For $z \in \mathbb{F}^k$ define

$$E_z[i] = \prod_{j=0}^{k-1} z_j^{i_j}(1-z_j)^{1-i_j}, \qquad \widetilde{U}(z) = \sum_{i<2^k} E_z[i]U[i]. \tag{3}$$

Here the factors mean z_j or $1-z_j$, including at Boolean challenges; no division is implicit. The empty tensor is (1). For $N = 2^n$ and a root ω of exact order N, $\mathsf{NTT}_{N,\omega}(u)_q = \sum_{t<N} u_t \omega^{tq}$. Decimation in frequency (DIF) takes natural input and returns bit-reversed frequencies.

We use the BN254 scalar field $\mathbb{F}$ and a fixed family of roots of unity of exact power-of-two order. The modulus, roots and canonical scalar encoding are specified in Section D.

Storage and units.

Definition 3.2 (Sequential stream model)**.** A materialized stream is a finite ordered sequence with an immutable published identity. A reader advances from its beginning; another traversal reopens it. A builder appends to a fresh, disjoint stream and publishes only after completion. There is no random access to records, no simultaneous read/write of the same mutable stream, and no identity rebinding while live. Caller-owned inputs and still-needed outputs remain charged and cannot be reclaimed as scratch. Storage is faithful and executions are crash-free, with no retries; a failed operation aborts proving. External storage can reveal witness-dependent contents and access patterns; storage confidentiality and malicious-storage protection are outside this model.

A field-record read or append counts once. An extent-N full field pass means exactly N reads and N writes; generating a zero is not an external read. Digest transfers and ordinary bytes are counted separately. RAM is typed as $M_{\mathbb{F}}$ field elements, M_{D} 256-bit digests, and M_{aux} other bits. All resident descriptors, cursors and routing keys count in the last coordinate. The local bound is $M_{\text{aux}} \le M_{\text{meta}} + O(\log N)$, where M_{meta} counts all resident stream-interface metadata.

For the constructive whole schedule we specify a narrower, bounded-descriptor realization. Input streams are designated by fixed ordinals, newly created streams by increasing integer ordinals, and a descriptor consists of an ordinal, type, layout and bounded counters. An ordinal or counter is stored in binary, with no additional resident catalogue. The schedule must bound both creations and live handles; Theorem 6.2 proves polynomially many creations and $O(\log T)$ simultaneous descriptors of $O(\log T)$ bits each.

Oracle-input interface. Weir uses the entire transcript event sequence in every challenge key. Our whole-prover resource statement is for an *external-query-tape oracle interface*: the prover writes the exact query bytes sequentially, and the ideal oracle consumes that stream and returns its fixed-size response. Query preparation and traversal are charged in $\mathcal{L}_{\text{oracle}}$, the sum of all query lengths. The oracle's internal function table is not prover memory. A buffering interface instead adds its largest query to RAM. The separate byte term matters: for $C = O = V, I = V-1$, even var row-combination keys copy $32I$ context bytes r_{var} times, growing as $V^{3/2}$ over admitted scales (Section F).

What we import from Scribe. Scribe supplies the RW model and sequential blocks [4]. The July 29, 2026 full version [5] has composition Lemma 4.3, streaming helpers in §4.2, sumcheck Lemmas 5.3 and 5.6, and equality-tensor generation in §2.6.1 (used by §5.2.2). These are the imported contracts, with all caller-held data charged.

Direct reuse.
Map/Zip/Reduce and SplitLR/JoinLR, SplitEO/JoinEO traverse ordered streams. Equality-tensor generation uses $O(\log L)$ field state, handles zero/one coordinates and restarts at the same point.

Reuse through adapters.
Sumcheck uses constant-degree geometric folding; here degree is at most three and batches have at most 12 product and 6 dot terms. Our SplitLR adapter eliminates the MSB rather than pairing adjacent least-significant-bit entries, and emits coefficient messages and compressed proofs (Lemma 6.1). Its VMProd alternating accumulator realizes both rectangular PCS collapses.

Protocol-specific composition.
Our product equations use the stream operations with Weir's message grammar. Sparse gather and timestamps use the sorting adapter of Section 6.

Scribe charges caller-held state and permits piping for single-pass consumers. We materialize replayed tables and retain borrowed data. Sluice also uses sparse sort/join/reduce and caller accounting [19, §2,§5]. We specialize these contracts to padded timestamps, MSB messages, replayed openings and canonical output.

3.1 The Weir Protocol

Four committed coefficient vectors connect the R1CS checks to sparse accesses. $Z_{\text{var}} = w^+$ (length V) fixes the private assignment. The index fixes sparse addresses, values and timestamps in Z_{ops} ($16O$), and final access counts in Z_{mem} ($2H$). After the R1CS challenges, Z_{deref} ($8O$) fixes the claimed equality-tensor reads. The memory checks test these reads against the indexed addresses and access history. Commit reshapes each vector into rows, half-rate RS encodes them, and hashes ordered columns into a Merkle tree. Table 2 connects these commitments to the checks; exact blocks and padding are in Section C.

Stage	Commitment and verification obligation
Index	Commit ops/mem; bind sparse records and access counts in the public context before challenges.
R1CS	Commit var. Two sumchecks test constraint satisfaction and reduce it to assignment and sparse-matrix evaluations. Open var.
Memory	Commit deref before compression challenges. Check the claimed reads via initialization $\times$ writes = reads $\times$ audit, for rows and columns.
Product / caller	Reduce memory identities and sparse dot products to block evaluations; combine them into deref/ops/mem opening claims.
Accept	Authenticate columns, test both RS collapses and claimed evaluations at all four sites; check algebraic endpoints and canonical parsing.

Table 2: Protocol order and the verification obligations served by the four commitments.

Sumcheck [20] eliminates one variable per round. The first phase tests the randomly weighted R1CS residual and gives $\widetilde{Xz^{\text{assn}}}(r_x)$, $X = A, B, C$. Fresh coefficients combine these claims for the second phase, which terminates at a column point r_y. Opening var checks the private assignment

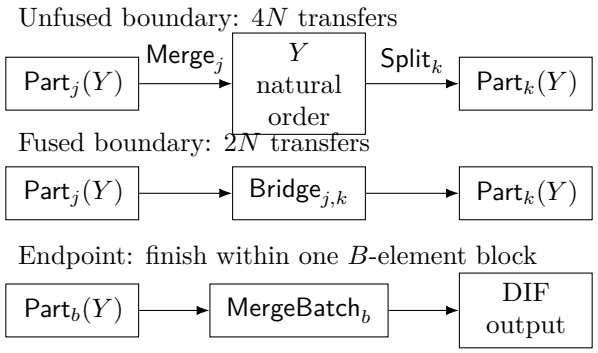


Figure 1: $\mathsf{Part}_j(Y)$ comprises two streams containing positions whose bit j is zero or one, each in original order. FSBM routes directly to the next partition, $k = j - 1$, and combines the final merge with the low butterflies. The endpoint shown is for $b < n$; $b = n$ uses one full block.

part; the public part is known. The sparse claims are

$$\widetilde{X}(r_x, r_y) = \sum_k \nu_{X,k} E_{r_x}[a_{X,k}] E_{r_y}[b_{X,k}].$$

Fresh β, γ compress each address/value/timestamp record as $a+\beta v+\beta^2 t-\gamma$. Per-address timestamps link a read to initialization or the preceding write. This *algebraic memory compression* is the hash layer's task; Merkle hashing instead authenticates committed columns. Caller reductions at deref, ops and mem check that the product network's terminal values are evaluations of the corresponding committed blocks.

The context binds application bytes, parameters, honest index roots, generators and public input. Challenge keys contain the complete framed event prefix, counter, site/role, label and output domain; responses extend that prefix. Each opening binds its site/root/point/claim, requests row-combination coefficients, absorbs both collapsed vectors, then samples 309 columns with replacement. The header binds the context; proof fields follow declaration order even when produced in another order. Framing and grammar are in Section D.

4 Fusing the SBM Transform

Sluice's SBM exposes each large radix-2 FFT stage [12] as a split, paired butterfly and merge [19, §3.3, Theorem 3.5]; its Algorithm 1 uses one materializing pass per small block-local stage. We fuse this stage schedule. Sluice's full transform is DIT with an initial bit reversal; here we use natural-input DIF with bit-reversed output for Shockwave's ordered RS code. The pass comparison below holds the DIF coordinate convention fixed and excludes a global permutation on either side. Assume $N = 2^n$, $n \geq 1$, $B = 2^b$, $0 \leq b \leq n$, and an exact-order root ω.

FSBM builds the next stage's representation directly, eliminating the intervening natural-order materialization. Figure 1 shows the two savings: route directly between stage partitions, then finish the low stages before writing each block. The formulas below specify the exact coordinates that this picture preserves.

Ordered stage layouts. For $0 \leq j < n$, $q < N/2$, and $c \in \{0, 1\}$, put

$$\mathsf{ins}_j(q, c) = \left\lfloor q/2^j \right\rfloor 2^{j+1} + c2^j + (q \bmod 2^j),$$
$$\mathsf{Part}_j(X)^c[q] = X_{\mathsf{ins}_j(q,c)}.$$

The two streams are ordered by increasing q. If $p = \mathsf{ins}_j(q, 0)$, the DIF stage F_j replaces

$$(X_p, X_{p+2^j}) \longmapsto (X_p + X_{p+2^j}, (X_p - X_{p+2^j})\theta_j(q)),$$
$$\theta_j(q) = \omega^{(q \bmod 2^j)2^{n-j-1}}.$$

A stable split produces $\mathsf{Part}_j(X)$, a paired butterfly consumes equal offsets of its two components, and a merge restores natural index order. Thus their composition is precisely F_j. No table of roots is needed: initialize $\zeta_j = \omega^{2^{n-j-1}}$, multiply a current twiddle by ζ_j, and reset it to one after every 2^j pairs.

Lemma 4.1 (Coordinate convention)**.** *Executing stages $n-1, \ldots, 0$ on natural input returns, at position q,*

$$(\mathsf{F}_0 \circ \cdots \circ \mathsf{F}_{n-1})(X)_q = \mathsf{NTT}_{N,\omega}(X)_{\mathsf{bitrev}_n(q)}. \tag{4}$$

Proof. For $M = N/2$, the first stage produces $a_t = X_t + X_{t+M}$ and $d_t = (X_t - X_{t+M})\omega^t$. Even frequencies of the full transform are $\mathsf{NTT}_{M,\omega^2}(a)$, and odd frequencies are $\mathsf{NTT}_{M,\omega^2}(d)$, since $\omega^M = -1$. The remaining stages operate separately on those halves with the root ω^2. Induction from the length-one identity gives frequencies $2\mathsf{bitrev}_{n-1}(q)$ in the first half and $2\mathsf{bitrev}_{n-1}(q) + 1$ in the second, exactly the stated order. □

The fused boundary. Separate consecutive SBM stages materialize natural order only to split it again. Define $\mathsf{Bridge}_{j,k} = \mathsf{Split}_k \circ \mathsf{Merge}_j$ semantically, but realize it without that intermediate vector: enumerate $p = 0, \ldots, N-1$, read the next item from input component $\mathsf{bit}_j(p)$, and append it to output component $\mathsf{bit}_k(p)$.

Theorem 4.2 (Bridge equality and interface-local optimum)**.** *For every Y, $\mathsf{Bridge}_{j,k}(\mathsf{Part}_j(Y)) = \mathsf{Part}_k(Y)$. On materialized inputs and fresh materialized outputs the schedule uses N reads, N writes, $O(1)$ field registers, and $M_{\mathrm{meta}} + O(\log N)$ auxiliary bits. No exact algorithm in this payload-opaque cursor/builder interface uses fewer than $2N$ field-record transfers on arbitrary inputs.*

Proof. Positions with a given bit j occur in their component in natural order, so the next selected input is exactly Y_p. Stable routing by bit k gives the prescribed output subsequences. Every item is read and written once. If an algorithm did not read an arbitrary input item, changing only that item would leave its observations unchanged while changing its required output. It must therefore read all N inputs and write all N fresh output items. This lower bound does not apply to aliased outputs or nonmaterialized views. □

Endpoint fusion. After stage $b < n$, merge $\mathsf{Part}_b(Y)$ into one aligned block of B field elements at a time, execute stages $b-1, \ldots, 0$ inside that block, and write it. Denote this operation by $\mathsf{MergeBatch}_b$. Every remaining butterfly changes only a bit below b, so both operands lie in that block. Consequently

$$\mathsf{MergeBatch}_b(\mathsf{Part}_b(Y)) = (\mathsf{F}_0 \circ \cdots \circ \mathsf{F}_{b-1})(Y).$$

It reads and writes each item once with $B + O(1)$ field memory. At $b = 0$ it is just a merge. At $b = n$ the entire transform is defined separately as a single full-block read, computation and write.

For $b < n$, our complete schedule is

$$\begin{aligned} S_{n-1} &= \mathsf{Split}_{n-1}(X), \\ Q_j &= \mathsf{PairedBfly}_j^F(S_j) \quad (n > j \geq b), \\ S_{j-1} &= \mathsf{Bridge}_{j,j-1}(Q_j) \quad (n > j > b), \\ \mathsf{FSBM}_{N,B}^{\mathrm{DIF}}(X) &= \mathsf{MergeBatch}_b(Q_b). \end{aligned} \tag{5}$$

Theorem 4.3 (Fused DIF and exact schedule saving)**.** *On the successful sequential interface of Definition 3.2, this schedule computes*

$$\mathsf{FSBM}^{\mathrm{DIF}}_{N,B}(X)_q = \mathsf{NTT}_{N,\omega}(X)_{\mathsf{bitrev}_n(q)}. \tag{6}$$

With materialized input and output it uses

$$\begin{aligned} P_{\mathsf{FSBM}} &= 2(n-b)+1, \\ I_{\mathbb{F}} &= 2N[2(n-b)+1], \\ M_{\mathbb{F}} &\le B + O(1). \end{aligned}$$

Its other RAM is $M_{\mathrm{aux}} \le M_{\mathrm{meta}} + O(\log N)$*;* $M_{\mathsf{D}} = 0$*. Field work is* $O(N \log N)$*, and external scratch and simultaneous field storage are* $O(N)$ *under last-use reclamation. Against the unfused schedule that uses one full pass for each of the* b *low local stages and split/butterfly/merge for each high stage, the saving is exactly*

$$[b + 3(n-b)] - [2(n-b)+1] = n-1$$

passes, or $2N(n-1)$ *transfers.*

Proof. Induct downward on the high stages. The initial split establishes the first partition; the butterfly implements its stage; each bridge only changes representation. Endpoint fusion completes exactly the low stages. The coordinate lemma proves the output equation.

For $n > b$ there are one initial split, $n-b$ butterflies, $n-b-1$ bridges and one endpoint, all full passes. For $n = b$ the one-block case is one pass. There is at most one B-element block; other field state is constant. After publishing a successor, reclaim its owned predecessor unless another consumer retains it, giving constant-many extent-N representations here. Root setup costs $O(n^2)$ for high stages and at most $O((N/B)bn)$ if recomputed within low-stage blocks; both are $O(Nn)$, as are the butterflies and twiddle advances. For $b < n$, the $n-b-1$ bridges each save one pass, while the endpoint saves b. An endpoint-only schedule uses $3(n-b)$ passes, so boundary fusion's additional saving is $n-b-1$, not $n-1$. At $b = n$, one full-block pass replaces n local-stage passes. Section B.2 separates these schedules and input policies. □

4.1 A Shared Execution Contract for Code Encoders

The routing primitive preserves coordinates independently of the paired map's coefficients, giving a reusable encoder interface.

Theorem 4.4 (Ordered paired-stage execution)**.** *Let* $\mathbb{K}$ *be any finite field. An encoder on* $N = 2^n$ *records executes* $d \ge 1$ *stages, with stage descriptors read sequentially. Stage* ℓ *pairs positions differing in bit* j_ℓ*, applies* $\phi_{\ell,q} : \mathbb{K}^2 \to \mathbb{K}^2$ *at offset* q*, with* $0 \le j_\ell < n$*, and preserves both output positions. Suppose each map and source reader uses constant field state and* $O(\log(N+d))$ *non-field workspace. Count all resident interface descriptors and reader metadata in* M_{meta}*. Coefficients are available in increasing* q *through a replayable sequential source or a generator with* $O(1)$ *field state and* $O(\log(N+d))$ *auxiliary state. Then the boundary-fused schedule computes exactly the ordered encoder with* $2d+1$ *full payload passes, versus* $3d$ *for separate split/map/merge stages. It saves* $d-1$ *passes, uses* $O(1)$ *field registers,* $M_{\mathrm{meta}} + O(\log(N+d))$ *auxiliary bits and* $O(N)$ *live codeword storage. External parameter storage, traffic and generation work are additional, identical charges in both schedules.*

Proof. Split at j_1, apply the first paired map, and use $\mathsf{Bridge}_{j_\ell, j_{\ell+1}}$ between maps; merge at j_d. The bridge theorem preserves each position and the offset at which the next map consumes it. Induction therefore preserves every stage value, including when stage bits repeat or increase. One split, d maps, $d-1$ bridges and one merge cost $2d+1$ passes. Each retains constant field state; reclaiming a predecessor after publication bounds live payload by constant-many length-N streams. The same coefficient sequence is consumed once per stage in either schedule. The result includes $d = 1$, with zero saving. □

Multiplicative-coset RS and FRI. In an odd-characteristic field with an order-N root ω_N, for $g \neq 0$, scaling coefficient u_t to $u_t g^t$ gives ordered evaluations at $g\omega_N^{\mathsf{bitrev}_n(q)}$. Fuse this scaling into FSBM's input pass: a running power of g uses constant state and adds $O(m)$ field work, with no additional payload pass. Thus Theorem 4.3 and Corollary 5.2 retain their traffic counts. In this order, multiplicative radix-2 FRI pairs are adjacent. For values v_0, v_1 at $x, -x$, challenge α produces

$$(v_0 + v_1)/2 + \alpha(v_0 - v_1)/(2x).$$

The next ordered domain has parameters g^2, ω_N^2. This is a sequential fold; Section B.3 derives its coordinates and retained-layer accounting for these multiplicative domains [7].

BaseFold-form encoders. BaseFold's foldable generator combines two child codewords using diagonal matrices [29, Definition 5]. Fix the recursive basis that places the two message halves consecutively. With a length-two repetition base, level j combines child positions as $(a_q + t_j(q)b_q, a_q + t'_j(q)b_q)$, where $t_j(q) \neq t'_j(q)$. Starting from adjacent repetitions, stages $j = 1, \ldots, n-1$, for $n \geq 2$, implement that generator in its natural column order. Under the coefficient-access premise above, boundary fusion therefore saves $n-2$ passes. No root of unity is required. Stored diagonals satisfy the premise by replaying the level's ordered parameter stream once per recursive block. The base and diagonals follow BaseFold; this result supplies their ordered sequential execution under the selected construction's distance and setup conditions.

Any commitment routine consuming the same ordered codeword can replace its encoder without changing its encoded leaves or Merkle root. The complete protocol and ROM theorem below concern Weir's four-site RS PCS. The code-family extension covers encoding and multiplicative FRI folding; complete FRI or BaseFold PCS composition also requires opening, transcript and security analysis.

5 From Fused Transforms to RS Commitments

The code-based commitment used in Shockwave separates row encoding, column commitment, and linear collapses [14, 15]. Only the first step is an FFT. We must also preserve column order, authentication paths, repeated queries and the lifetime of committed coefficients.

The four instances. The four coefficient vectors summarized in Section 3.1 and defined exactly in Section C are $Z_{\mathrm{var}}, Z_{\mathrm{deref}}, Z_{\mathrm{ops}}, Z_{\mathrm{mem}}$. For site X, use

$$\begin{aligned}(e_{\mathrm{var}}, e_{\mathrm{deref}}, e_{\mathrm{ops}}, e_{\mathrm{mem}}) &= (v, o+3, o+4, h+1),\\ r_X &= 2^{\lceil e_X/2 \rceil - 4},\\ m_X &= 2^{\lfloor e_X/2 \rfloor + 4}, \qquad N_X = 2m_X.\end{aligned} \tag{7}$$

The admitted shapes ensure $7 \leq e_X \leq 45$, so all exponents are integral and nonnegative and $N_X \leq 2^{27}$. In this section omit the site suffix and denote the codeword matrix by E. Reshape Z

row-major as $a_{im+t} = u_i[t]$, $i < r, t < m$. The ordered Reed–Solomon row code [23] is

$$\mathsf{E}[i,q] = \mathsf{RS}(u_i)[q] = \sum_{t<m} u_i[t]\omega_N^{t\,\mathsf{bitrev}_n(q)}, \quad q < N. \tag{8}$$

The evaluation points are distinct; permutation by bit reversal does not change the code distance $N - m + 1$.

Corollary 5.1 (Row replacement and generic row traffic)**.** *Apply fused DIF to $u_i\|0^{N-m}$, one row at a time, with local $B_X = 2^{b_X} \le N_X$. It returns exactly Equation (8). Writing each row directly to its final batch-output segment uses $B_X + O(1)$ field RAM, $O(N_X)$ scratch and $O(r_X N_X)$ total external field storage. For materialized zero-padded input the traffic is $2rN[2(n-b)+1]$; for an implicit-zero source it is*

$$I_{\mathbb{F}} = r[m + N + 4N(n-b)].$$

Auxiliary RAM is $M_{\text{meta}} + O(\log(rN))$.

Proof. Zero coefficients drop out of Equation (6), giving the ordered codeword in Equation (8). The same field block is reused for every row. For implicit input the first pass reads m items and writes N; the other $2(n-b)$ passes each transfer $2N$. At $b = n$ there is only the first full-block phase. There is no second copy of the completed row. □

Corollary 5.2 (Half-rate zero-tail endpoint)**.** *For $m = N/2$ and $b < n$, produce the first stage's paired streams directly from u, emitting u_q and $u_q\omega_N^q$. The output code is unchanged and batch field traffic becomes*

$$I_{\mathbb{F}} = r[4N(n-b) - N/2].$$

There are $2(n-b)$ materializing phases, the first of signature (m, N), so these are not all full extent-N passes.

Proof. The first butterfly has second input zero. Its two outputs are the displayed expressions in increasing q. There remain $n-b-1$ bridges, $n-b-1$ butterflies and one endpoint, each costing $2N$. Adding $m + N$ gives the formula. □

Definition 5.3 (Local commitment and opening maps)**.** Commit retains the coefficient rows, encoded rows and ordered Merkle levels as immutable external objects. Relayout stably sorts row-major records $(i, q, \mathsf{E}[i,q])$ by column q, preserving row order within a column. Stable radix partitions on all bits of q realize this in $O(rN \log N)$ record transfers and constant record buffers. With the common Weir hash $\mathcal{H}_B$, the Merkle maps are

$$H_0[q] = \mathcal{H}_B(0^{32}\|\mathsf{Enc}_{\mathbb{F}}(\mathsf{E}[0,q])\|\cdots\|\mathsf{Enc}_{\mathbb{F}}(\mathsf{E}[r-1,q])), \tag{9}$$

$$H_{\ell+1}[j] = \mathcal{H}_B(H_\ell[2j]\|H_\ell[2j+1]). \tag{10}$$

The prefix 0^{32} here means 32 zero bytes. Hash the complete ordered byte string. A path for q contains $H_\ell[(q \operatorname{div} 2^\ell) \operatorname{xor} 1]$ for $\ell = 0, \ldots, n-1$; left/right folding follows the bits of q.

For a point $z = z_L\|z_R$, with lengths $\log_2 r, \log_2 m$, and caller claim y, an opening first binds the context specified in Definition D.2. Append protocol name `polynomial evaluation proof`, then request r field coordinates with label `$1//DT`. The label DT selects degree-test randomness. Store the resulting η once. Compute

$$p_{\text{rand}}[t] = \sum_{i<r} \eta_i a_{im+t}, \qquad p_{\text{eval}}[t] = \sum_{i<r} E_{z_L}[i] a_{im+t}.$$

Append the random coefficients in increasing t with $\texttt{\$1//PR}$, then the evaluation coefficients with $\texttt{\$1//PE}$, as individual scalar events without vector wrappers. Request $Q = 309$ column indices with $\texttt{\$1//CO}$, in occurrence order, with replacement. PR and PE identify the random and evaluation collapses; CO identifies column-opening queries. For each occurrence retain its column and path, even if its index repeats. These operations return the typed proof in Table 11; they do not append its serialization to the transcript.

For each collapse use Scribe's alternating-accumulator stream pattern: initialize a length-m zero stream, then for each row replace it by a fresh stream with entries $s_t + \eta_i a_{im+t}$. Reclaim the old accumulator after publication. Use the same method with $E_{z_L}[i]$ for the other collapse. A restart replays the stored η, never requests new randomness. This avoids keeping a length-m accumulator in RAM or transposing the coefficient table.

Lemma 5.4 (Local correctness and equality of openings)**.** *The local RW maps equal their direct polynomial/matrix definitions. With common fixed oracle functions, equal incoming transcripts and equal caller inputs, they return the same root, outgoing transcript and typed opening. For $y = \widetilde{a}(z)$, every local verifier check in Definition D.3 passes.*

Proof. Row equality and stable relayout give identical ordered columns; induction up the tree gives identical nodes and root. After j rows the random accumulator is $(\sum_{i<j} \eta_i a_{im+t})_{t<m}$, by induction on updates. The evaluation accumulator has the same invariant with E_{z_L}. This proves both complete coefficient vectors, including their order. Equal preceding event words give equal direct oracle responses; thus PR, PE, CO and every repeated query occurrence agree. For every q, linearity of Equation (8) gives

$$\mathsf{RS}(p_{\mathrm{rand}})[q] = \sum_i \eta_i \mathsf{E}[i, q],$$

and similarly for p_{eval}. Tensor factorization gives

$$\sum_t E_{z_R}[t] p_{\mathrm{eval}}[t] = \widetilde{a}(z).$$

The paths are from the same tree. These are all local checks. □

6 A Whole-Prover RW Schedule

The schedule realizes Weir's phase order (Section 3.1 and Definition C.13), including indexing, four retained commitments, the transcript and canonical output. All large objects are external streams. Table 3 shows how their interfaces preserve the protocol and bound live state.

Indexing commits ops and mem; proving commits and opens var, then commits deref and opens deref/ops/mem. Each state survives its last opening. The prototype retains all four through serialization; that additional storage is included in its measured costs.

Addressed operations by sequential sorting. Use a stable bottom-up merge sort on records with $O(1)$ field/digest payloads and $O(\log T)$-bit keys. Distribute consecutive runs onto two alternating input streams; merge their next pair of runs with two head records, distributing completed output runs alternately to two fresh streams. After each level, reclaim the old runs. Run lengths double until one remains. This is a sequential construction with $O(T \log T)$ record transfers, $O(T)$ scratch records and $O(\log T)$ counter bits.

To gather $U[a_j]$, sort requests (a_j, j), merge them against the indexed source $(i, U[i])$, and sort the resulting $(j, U[a_j])$ by j. Retain the current source value while serving repeated requests

Reused block	Failure if an obligation is omitted	Weir invariant and guarantee
Sort/join; MSB fold	Address order permutes table positions; an LSB fold eliminates the wrong variable.	Tag and restore request ordinals; pair table halves. Values and MSB coordinates match the logical tables and messages.
Commit/open; composition	A root alone cannot answer later sampled openings; retaining its inputs in RAM breaks the workspace bound.	Keep coefficient rows, codeword columns and Merkle levels external through their last consumer; charge them alongside active workspace.
Streaming messages/output	Absorbing fields in wire order changes challenges when event order differs.	Preserve event order; assemble external sections later in declaration order. Oracle keys and canonical proof bytes match.

Table 3: Protocol-specific obligations for the whole schedule. Sorting, folding and composition are reused; the guarantees are proved by Lemma 6.1 and Theorems D.6 and 6.2.

for that address. Bounds checks precede lookup. For requests $(1, 0)$, an address-order join emits $(U[0], U[1])$; restoring ordinals emits $(U[1], U[0])$. Their multilinear evaluations are respectively $(1-r)U[0] + rU[1]$ and $(1-r)U[1] + rU[0]$: multiset equality does not preserve this claim. For Xz, gather assignment values by column, multiply by sparse values, sort by row and original ordinal, and reduce each row, emitting zero for absent rows. Construct M_X in phase two similarly, gathering row equality weights and reducing by column. Every stored occurrence, including duplicates and explicit zeros, contributes in its defined order.

For preprocessing, sort the padded operations by $(d, a^d_{X,k}, X, k)$. A within-address running rank is its read timestamp; the group's size is its audit count. Sort timestamps back to (d, X, k), and merge counts with $0, \ldots, H-1$ to fill zero audits. In particular padded address-zero operations increment timestamps. Dynamic dereferences use the same gather against μ_ρ, μ_κ.

Lemma 6.1 (Ordered adapters)**.** *The sort/gather procedures produce exactly the tables in Definitions C.2 and C.3 and Lemma C.6. The following RW sumcheck and product procedures preserve every polynomial, ordered message and challenge of their logical definitions, for any fixed oracle functions.*

Proof. Stable merging preserves the order within an equal key; induction on run size proves sorting correctness. After merging at address a, the current source is $U[a]$, giving every request that value. Restoration by the unique original ordinal gives the requested ordered vector. Reducing exactly the sparse contributions proves the matrix formulas. Rank within an address group counts precisely the preceding (X, k) occurrences, proving timestamps and audits.

For a sumcheck table U of length $2L$, SplitLR materializes its low and high halves without changing their internal order. Zip supplies $(U[i], U[L+i])$. A constant number of Reduce accumulators computes $g(0), g(2)$, and, if needed, $g(3)$. The entering scalar determines $g(1)$ and the same coefficient interpolation as Definition C.5. Only those constant-many coefficients are resident during absorption and the challenge request. A second traversal writes $(1-r)U[i] + rU[L+i]$ to fresh streams. Hence the next table and compressed polynomial are exact. The table lengths decrease geometrically; replays cost a constant number of scans per level. Borrowed original tables remain immutable; only owned folded copies are reclaimed. Scribe's Eq.Next enumerates Boolean vectors in lexicographic order, which is our increasing MSB-index order, so it generates exactly $E_z[i]$ for the same ordered point z. This enumeration is independent of sumcheck's variable-elimination order. Coordinates

zero and one remain valid.

For a product tree, apply SplitLR, Zip and multiplication to make $V_{j,t}$ from $V_{j,t+1}$. Retain all levels needed by the root-to-leaf proving order. Their total size is less than twice their leaves. At proving layer t, use the just-proved MSB sumcheck adapter and then prepend the layer challenge, as specified by Definition C.7. Total table lengths over these layers are geometric too. The separate dot triples are included only in the last layer. The construction implements Weir's polynomial/message transitions using sequential storage. □

Commitments, collapses and openings. Run each FSBM encoder serially with one local block, then sort into column order. Stream each entire leaf-hash input onto the external query tape. Parent levels are made by sequential paired reads and are retained as level-tagged streams. The sum of their sizes is less than twice the leaf level. The coefficient rows remain available independently of codeword columns.

The two alternating-accumulator collapses of Definition 5.3 each take $O(rm)$ field transfers. Generate DT once into an immutable stream and restart the equality generator for the other collapse. Retain both resulting vectors for transcript absorption and proof output; store CO responses once, in occurrence order. For each of the Q occurrences, scan codeword columns to select its column, and scan each stored Merkle level to select its sibling. This schedule costs $O(Q(rN + N))$ record reads. It preserves duplicates and uses constant digest buffers.

Transcript and output. Represent the logical append-only word by a published immutable prefix W_{old} and a separate builder for application events since the last challenge. At a challenge, publish that event segment. Scan W_{old} and the segment into two fresh builders: the exactly framed query, and the next transcript prefix. Publish the query and submit it. Only after its answer, append the response descriptor and value to the next prefix, publish that prefix, and reclaim the old owned prefix and segment. The length counters determine the framing before either scan. Thus no reader observes an unfinished builder and no published stream is mutated. Copying a prefix is deterministic work, not a second oracle call; its constant-many copies are charged to that query's length. Any final unqueried event suffix is joined once. Opening tuples, PR/PE coefficients and the initial context are streamed in their defined order. Each query is consumed from these exact byte streams.

The typed proof tree need not reside in RAM. One serializer tags bounded chunks by canonical offsets, sorts them and emits the exact body; Lemma C.14 gives those offsets. Alternatively, retain the sections externally through the last challenge and scan them in declaration order. Both preserve PR/PE transcript order while emitting PE before PR in the proof. Absorbing PE first would change the full-prefix keys. The stated storage bounds cover either schedule; the prototype uses delayed serialization.

Theorem 6.2 (Whole index-and-prove resource bound)**.** *For an admitted satisfying* Weir *input, use the bounded-descriptor realization and external-query-tape model of Section 3. Choose $B_X = 2^{b_X} \le N_X$ for each site and put $B = \max_X B_X$. Conditional on successful faithful storage, the above serial schedule returns the canonical proof of the reference prover and satisfies*

$$M_{\mathbb{F}} = O(B + \log T), \qquad M_{\mathsf{D}} = O(1),$$
$$M_{\text{aux}} = O(\log^2 T + Q \log T).$$

External live storage contains $O(T)$ field, digest and routing records, and $O(T \log T)$ additional bytes. Field work and field/digest record traffic cost $O(T \log T + QT)$. Byte work and traffic cost

$$O(T \log^2 T + \mathcal{L}_{\text{oracle}}).$$

There are J_{ver} direct challenge requests, with J_{ver} as in Equation (23); stored responses are replayed. Their exact full-prefix input lengths are included in $\mathcal{L}_{\mathrm{oracle}}$. The bound includes indexing and proof output, but not witness generation, the verifier, or the knowledge extractor. The profile fixes $Q = 309$.

Proof. The four coefficient lengths are $V, 8O, 16O, 2H$; their sum and their half-rate encoded lengths are $O(T)$. All sparse and assignment arrays are constant-many $O(T)$-length streams. Sort/merge passes use constant record buffers and geometric product/folding tables have $O(T)$ total live records. Each commitment's Merkle levels have geometric size and there are four commitments. Opening values, collapse vectors and proof chunks number $O(Q\sum_X(r_X+\log N_X)+\sum_X m_X+\log^2 T) = O(T)$ for fixed Q. To see the last bound, each balanced shape has $r_X, m_X = O(\sqrt{T})$ and all logarithmic powers here are $O(T)$. Consequently the simultaneous large typed stores are $O(T)$, even with borrowed originals, pending proof output and a sorting successor alive together. Routing keys cost $O(\log T)$ bits per record.

The sum of primitive table work is $O(T)$ for folding, products, Merkle levels and collapses. In particular, a product batch's total sumcheck table length is $\sum_{t<L} O(2^t) = O(2^L)$, even though it emits $\sum_{t<L} t$ round records. Sorting and FSBM cost $O(T\log T)$. The occurrence-wise opening scans cost $O(QT)$. Equality-tensor work satisfies Scribe's linear bound per generated table; alternatively computing its factors separately gives the same conservative $O(T\log T)$ total upper bound. Tagged proof sorting has at most $O(T)$ chunks. Lemma C.14 charges their offset computation. Record keys and counters have $O(\log T)$ bits, giving the conservative $O(T\log^2 T)$ ordinary byte cost outside oracle keys.

The nonchallenge application payloads and initial context have $O(T)$ bytes. Each of the J_{ver} challenge responses adds a descriptor of $O(\log T)$ bytes and a fixed-size value. Thus $|W| = O(T + J_{\mathrm{ver}}\log T) = O(T\log T)$. One temporary query stream at a time fits the same bound. All prefix copies, framing and oracle input traversals are charged by their submitted lengths $\mathcal{L}_{\mathrm{oracle}}$; no quasilinear bound on that quantity is inferred. Once a field payload is serialized, subsequent prefix copies read its byte stream.

At most one B_X field block is resident. Caller points and equality generation have $O(\log T)$ field state; polynomial coefficients, product-batch claims and merge heads add constants. Hash inputs reside on query streams; only constant-many digests are buffered. For descriptors, the constant-many source/commitment tables remain live through their last consumers. The four Merkle trees and two constant-width product batches retain $O(\log T)$ level handles in total. One active sort uses four run streams; one active fold or collapse adds constant-many owned successors alongside borrowed inputs. Transcript copying holds only the old prefix, event segment, query and new-prefix builder, and proof chunks use a constant number of streams. Hence at most $O(\log T)$ descriptors are simultaneously resident. Folds, rows, sorting levels and oracle events create only polynomially many streams; ordinal identifiers and bounded counters therefore each require $O(\log T)$ bits. Query indices add at most $O(Q\log T)$ bits. Adding this simultaneously live state proves the RAM bounds under Scribe's composition contract.

The adapter lemma, local PCS correctness and deterministic serialization establish equality of each produced value. No stream permutation changes event order. Lifecycle induction in Theorem D.6 gives the canonical output and acceptance. □

7 Correctness and Knowledge Soundness

For honest replacement, ordered stream equality gives identical FSBM coordinates, commitment columns and Merkle roots; the MSB and sparse adapters give identical algebraic messages; the

common oracle keys then give identical challenges. Canonical assembly gives exact bytes. This is Theorem D.6, with whole-schedule costs in Theorem 6.2.

For malicious provers, an accepted proof lets an extractor recover a satisfying witness, except with the bounded error below.

Definition 7.1 (Classical-ROM experiment and budgets)**.** Fix an admitted instance, a, G, par, and honestly index it to obtain K. A classical adversary receives the index, queries the independent public oracles of Definition D.2, and outputs one pair (x, b). Verify with the same oracle tables and context $C(x) = (a, \mathsf{par}, K, G, x)$. Auxiliary information is oracle-independent, or its generator runs after indexing as part of the monitored, charged adversary. Malformed inputs reject; x may be chosen after the queries.

Use deterministic budgets: q_C bounds adversarial field/index queries including repeats; q_B bounds all distinct hash queries, including indexing, auxiliary-input generation and verification; L_B bounds the total bytes of submitted hash/challenge inputs, public inputs and final outputs, charged before the relevant answers. Let J_{ver} count the verifier's field/index requests (Equation (23)) and put $K_C = q_C + J_{\mathrm{ver}}$. Define

$$\epsilon_B = \min\{1, q_B(L_B + q_B)/2^{256}\}, \qquad \kappa_{\mathrm{net}}(L) = L(3L+1)/2,$$

$$E_{\mathrm{alg}} = 4c + 1 + 2(v+1) + 2(H + 3O) + \kappa_{\mathrm{net}}(o) + \kappa_{\mathrm{net}}(h) + 8.$$

Theorem 7.2 (Adaptive knowledge guarantee)**.** *In the stated* Weir *classical-ROM experiment there is a straight-line oracle simulator/extractor, with extracted witness w^*, satisfying*

$$\begin{aligned}\Pr[V_{\mathsf{Weir}}(C(x), b) = 1 \wedge \\ ((\mathsf{inst}, x), w^*) \notin \mathcal{R}_{\mathrm{R1CS}}] \le \epsilon_{\mathsf{Weir}},\end{aligned}$$

$$\begin{aligned}\epsilon_{\mathsf{Weir}} = \min\{1, \epsilon_B + K_C[(E_{\mathrm{alg}} + N_{\mathrm{var}} + N_{\mathrm{deref}})/p \\ + 2(3/4)^{309} + ((m_{\mathrm{ops}} - 1)/N_{\mathrm{ops}})^{309} \\ + ((m_{\mathrm{mem}} - 1)/N_{\mathrm{mem}})^{309}]\}.\end{aligned}$$

Its additional work is polynomial in the numeric dimensions and charged query/byte budgets, as quantified in Section E.1. The same bound applies to acceptance of a public input with no witness.

The probability jointly charges acceptance and an invalid extracted witness.

Reading the error terms. The site dimensions are fixed by Equation (7). Var and deref are prover-created: random row combination contributes $(N_{\mathrm{var}} + N_{\mathrm{deref}})/p$, and far virtual matrices contribute $2(3/4)^{309}$. Ops and mem are honestly indexed codewords; a false evaluation agrees on at most $m_X - 1$ columns, giving $((m_X - 1)/N_X)^{309}$. Algebraic root events contribute E_{alg}/p. The union bound charges adaptive exposure at K_C full-prefix requests and fresh hash hits through ϵ_B.

An illustrative budget. At $(C, V, O) = (4096, 262144, 32768)$, hypothetical budgets $q_C = 2^{32}, q_B = 2^{40}, L_B = 2^{64}$ give $J_{\mathrm{ver}} = 2056$, $E_{\mathrm{alg}} = 1{,}246{,}175$ and a full bound near $2^{-95.247}$. A candidate $Q = 388$ gives $2^{-128.035}$ at those budgets, adding 634,528 proof bytes and 316 verifier requests. The protocol remains fixed at 309. Section F analyzes other budgets, including hash floors, within the ideal-oracle experiment.

Proof outline. The dependency chain is canonical grammar (Definition D.3), prefix recognition (Lemma E.9), causal freezing (Lemma E.11), PCS error sets at fixed prefixes (Lemma E.12), first-exposure charging (Lemma E.10), then the final union bound.

Freeze before answers. A structural monitor recognizes full-prefix keys and their ancestor cuts, including guessed histories. Before the exposing answer, the extractor unfolds each var/deref root from existing hash-log preimages, fills unresolved branches with zero columns and caches the virtual matrix by the entire cut. Var freezes at τ (ξ if $c = 0$); deref at (β, γ). Ops/mem are honestly indexed. Outside the hash-hit event, authenticated columns equal these snapshots. Near matrices are uniquely decoded within $N/4$ errors; the others retain a far flag.

Charge first exposure. Snapshots and entering claims fix each block's bad-response set before its exposing answer. The full prefix and counter distinguish response-tree branches and levels. The actual future oracle path is fresh and product-uniform, even when the exposing key contains guesses. Selection or abandonment preserves the pointwise bound; at most K_C adversary-plus-verifier requests expose prefixes.

Follow acceptance to a witness. Outside hash and PCS events, accepted openings give true evaluations of decoded var/deref and honest ops/mem. With no algebraic root event, caller reductions, memory timestamps and product/dot checks establish the sparse claims. The two R1CS sumchecks then test $w^* \| (1, x, 0)$, where w^* is the decoded var table flattened in row order. Acceptance of an unsatisfying assignment entails an R1CS, memory, product or caller root event; their pointwise losses sum to E_{alg}/p. Table 12 identifies each nonzero polynomial, its fixed data and degree charge. Exposure charging and the PCS/hash losses give the bound; the extractor and complete event inclusions are in Section E.1. □

8 Evaluation

We evaluate completed workload under a fixed RAM quota, isolated transform savings and transfer to other ordered code encoders.

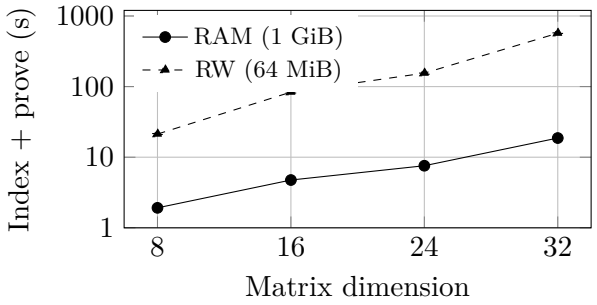
Inputs, environment and accounting. Shapes, supplied witness length and buffers follow Section 3. The prototype uses BN254-Fr, half-rate RS encoding, four commitments, $Q = 309$ column openings, SHA-256 and a common verifier. Each study freezes its source, binary, inputs and condition schedule.

The primary and capacity studies use sequential fresh Linux/ARM64 containers on a 32 GiB M1 Pro host, one prover thread, a one-CPU quota and no swap. Guest and host caches are uncontrolled; software/storage details are in Section B. Index+prove excludes input loading and verification; phase timers are nested. Prover RSS includes preparation's high-water but excludes verification and RW proof readback; capped completion includes both. We distinguish logical field transfers, live external payload and cgroup memory including page cache. MB/GB are decimal; caps use MiB/GiB. Every job rebuilds the index; witness generation and proof transfer are unmeasured.

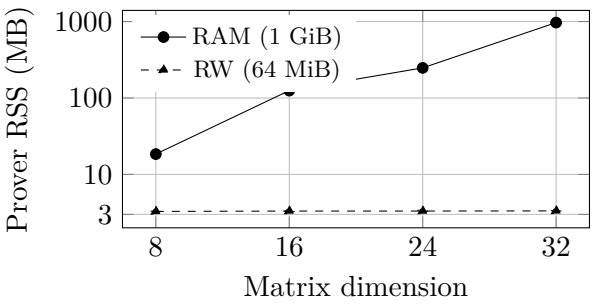
8.1 Capacity Under a Fixed RAM Budget

Matrix dimensions 8/16/24/32 compare RAM and RW at 64 MiB, with RAM 1 GiB as a latency comparator. Each configuration has three fresh processes, rotated conditions and a 1200 s timeout. RW selects the largest buffer valid at all four sites.

At 64 MiB, RW verifies all 12 attempts; RAM verifies 3/3 at dimension 8 and has nine cgroup-confirmed OOMs at the larger sizes. Every private witness slot is referenced and every padded



64 MiB completion	RAM	RW
Matrix 8	3/3	3/3
Matrix 16	0/3	3/3
Matrix 24	0/3	3/3
Matrix 32	0/3	3/3



64 MiB completion	RAM	RW
Poseidon 4	0/3	3/3
Poseidon 8	0/3	3/3
Poseidon 16	0/3	3/3

Figure 2: Matrix capacity curves: medians with min/max, three fresh processes per point; logarithmic vertical axes. Times compare operational configurations, RAM 1 GiB and RW 64 MiB. The tables report completed/attempted jobs at the common 64 MiB cap, including readback and verification; failed cells are cgroup-confirmed OOMs. RSS excludes verification/readback; dimensions and other costs are in Table 5.

constraint independently checked. Table 5 separates active rows and supplied witness slots from padded dimensions. The largest tested RW input has $(C, V, O) = (65536, 65536, 131072)$, 33,793 active rows and $W = 35840$: 64 times the padded constraints of the largest successful RAM point in this tested grid.

RAM at 1 GiB verifies 12/12; successful configurations share per-input proof hashes, lengths and oracle/query counts. Figure 2 compares their costs and capped outcomes. At dimension 32, RW uses 3.33 MB prover RSS and 9.51 minutes index+prove, versus RAM's 969.57 MB and 18.69 s. RW retains 1.228 GB logical live payload and emits a 5.892 MB proof; verification takes 14.32 s and field/index query keys total 3.936 GB. These medians fit an illustrative offline admission policy: 64 MiB per job, ten minutes for indexing/proving, 15 s for verification and 6 MB for proof delivery. A service could bound concurrent memory use with per-job quotas. Scratch must cover payload, file metadata and query streams; RAM 1 GiB offers lower latency when available. The complete RW schedule enables this capacity; fusion reduces traffic.

Poseidon chains of 4/8/16 hashes add 27 separately frozen attempts. About 94% of padded constraints are active and every private slot is referenced (Table 5). RW 64 MiB and RAM 1 GiB each verify 9/9 with matching proofs/query counts; RAM 64 MiB has nine confirmed OOMs. The largest RW chain takes 490.82 s index+prove with 3.31 MB prover RSS; Table 5 gives timing ranges and all external, proof and verification costs.

8.2 Isolating Transform Savings

We hold DIF coordinates and output order fixed across unfused, boundary-only, endpoint-only and combined schedules. Figure 3 shows their measured traffic at $B = 4$. For materialized input and $b < n$, the counts match Theorem 4.3: boundary fusion saves $n - b - 1$ passes and endpoint batching saves b, for $n - 1$ combined.

We test $n \in \{10, 13, 15\}$, $b \in \{2, n-2, n-1, n\}$, all four schedules, and both arbitrary materialized

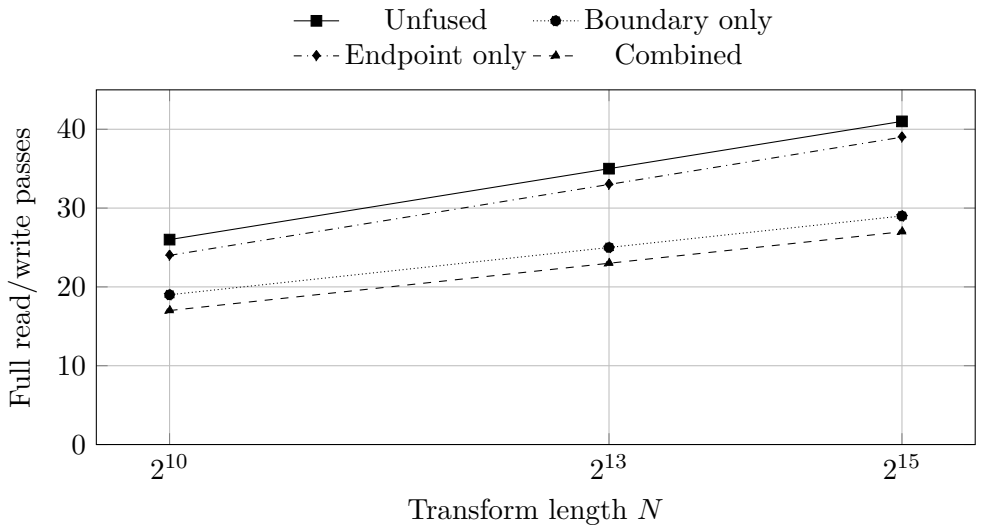


Figure 3: Encoder traffic at $B = 4$ on materialized input; a pass is $2N$ field-record transfers. All four schedules return identical output bytes at each length, verified in three runs per point.

and half-rate inputs, three processes per condition. All 312 output hashes and read/write counts agree with the transforms and formulas, including separate zero-tail controls. At $N = 2^{15}, B = 4$, boundary fusion alone removes 12 of the endpoint-only schedule's 39 passes (30.8%). Its repetition-paired encoder-time ratio is 0.694 [0.686,0.706]. At $B = N/2$ and $B = N$, no high–high bridge remains. Sections B.1 and B.2 give the full formulas, input controls and timing variation.

8.3 Transfer Across Ordered Code Families

An independent file engine tests the contract of Theorem 4.4 at $N = 32, 256, 1024, 4096$, three coefficient seeds and separate/fused schedules. The 36 cases cover coset RS over BN254-Fr and BaseFold-form encoding over BN254-Fr and the secp256k1 base field, which has no FFT roots at these lengths. Both schedules match RAM references and exact traffic counts. At $N = 4096$, boundary fusion reduces RS from 36 to 25 passes (30.6%) and the recursive encoder from 33 to 23 (30.3%), for the materialized encoder payload. Including streamed input expansion gives 29.9%/29.6% record savings; replaying two stored diagonal streams gives 25.5% for the recursive encoder (Table 10). Generated coefficients instead add per-pair work with constant field state.

For 12 coset RS inputs, folding matches all 105 reference states and their Merkle nodes across 93 folds; all 420 sampled path occurrences authenticate, including duplicates. Section B.3 fixes the coordinates and validation algorithms.

8.4 Same-Protocol Structured Inputs

Matrix sizes 8/16 and Poseidon chains of 2/4 hashes reference every supplied private slot; all padded constraints are independently checked. Poseidon uses the arkworks 0.5 BN254-Fr gadget, rate two, 8 full/57 partial rounds and exponent 5.

Each fixture compares RAM, endpoint-only and combined RW at $B = 4$ and the largest buffer valid at all four PCS sites; legacy unfused RW adds a continuity baseline at that larger buffer. The matched schedules share half-rate input, the batch-output builder and paper transcript. All 72

structured runs verify with matching per-input proof hashes, sizes, oracle counts and query bytes; traffic differences match the four-site encoder formulas.

Boundary routing's whole-prover increment. Table 4 separates encoder savings from the smaller whole-prover gain. On Poseidon 4 at $B = 1024$, median times are 5.21 s for encoding, 34.09 s for relayout and 9.99/6.69 s for query construction/prefix advancement. At larger buffers, field savings fall to 0.72–2.42%; index+prove ratio ranges cross one in three of four fixtures (Table 8). Query-key costs are analyzed in Section F.

	Reduction (%)		
Input	Encode time	Index+prove	Field I/O
Matrix 8	27.3	2.7 [1.9,2.8]	5.94
Matrix 16	28.4	4.3 [4.2,4.5]	5.97
Poseidon 2	29.1	4.3 [3.9,4.4]	6.21
Poseidon 4	29.3	4.7 [4.6,6.6]	6.13

Table 4: Boundary-fusion savings in the primary 2 GiB study at $B = 4$: combined versus endpoint-only, with matched half-rate input and row output. Times are median repetition-paired reductions, three fresh processes per condition; brackets give descriptive min/max. Field I/O counts whole-prover logical reads+writes.

Separate capped completion check. Under a separate 64 MiB cap, RAM is OOM-killed in all three attempts for each of matrix 16 and Poseidon 4; endpoint-only and combined each verify in all three. Their costs are in Table 9; this cohort is separate from the primary 2 GiB timings and Section 8.1.

8.5 Witness Coverage and Shape Controls

Eight synthetic cases add 120 verified runs. Filled cases reference 122–127 of 127 private slots and retain RW/RAM RSS ratios 0.101–0.316. They satisfy $A = C$ with a constant-one middle matrix. These tautological relations isolate access coverage. Dense seed 11's larger-buffer time ratio is 1.200 [0.992,1.335]; two repetitions have unexplained extra transcript time. Tables 7 and 8 report shapes, padding and all paired effects.

8.6 Earlier Studies

Historical Shockwave at `caa46e564377c6d` matches ordered inputs, field, half-rate code and 309 queries. Its BLAKE3/Merlin transcript differs from Weir's SHA-256/full-prefix keys [15, 21, 25]. Its verified cap comparisons are whole-system evidence under those differing profiles (Table 6); earlier scaling, buffer and cache controls remain in Section B.

9 Conclusion

Weir closes a complete RW index-and-prove schedule with explicit live-state and oracle-byte costs, canonical proof equality and an adaptive ROM knowledge bound. Its ordered encoder contract also covers coset RS and recursive foldable generators. At 64 MiB, RW completes matrix dimension 32 and Poseidon 16 hashes; RAM completes only matrix dimension 8 in the tested grid. The measured

latency and external costs define an offline operating choice for prepared inputs and sequential scratch storage. Workers can select RAM, RW and local buffers with a fixed proof and verifier.

References

[1] Alok Aggarwal and Jeffrey Scott Vitter. The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9):1116–1127, 1988.

[2] Scott Ames, Carmit Hazay, Yuval Ishai, and Muthuramakrishnan Venkitasubramaniam. Ligero: Lightweight sublinear arguments without a trusted setup. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, pages 2087–2104. ACM, 2017. doi:10.1145/3133956.3134104.

[3] David H. Bailey. FFTs in external or hierarchical memory. *The Journal of Supercomputing*, 4(1):23–35, 1990.

[4] Anubhav Baweja, Pratyush Mishra, Tushar Mopuri, Karan Newatia, and Steve Wang. Scribe: Low-memory SNARKs via read-write streaming. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 5751–5770, Baltimore, MD, August 2026. USENIX Association. URL: https://www.usenix.org/conference/usenixsecurity26/presentation/baweja.

[5] Anubhav Baweja, Pratyush Mishra, Tushar Mopuri, Karan Newatia, and Steve Wang. Scribe: Low-memory SNARKs via read-write streaming. Cryptology ePrint Archive, Paper 2024/1970, full version, July 2026. July 29, 2026 revision. URL: https://eprint.iacr.org/2024/1970.

[6] Mihir Bellare and Phillip Rogaway. Random oracles are practical: A paradigm for designing efficient protocols. In *Proceedings of the 1st ACM Conference on Computer and Communications Security*. ACM, 1993. URL: https://web.cs.ucdavis.edu/~rogaway/papers/ro-abstract.html.

[7] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Fast Reed–Solomon interactive oracle proofs of proximity. In *45th International Colloquium on Automata, Languages, and Programming (ICALP 2018)*, volume 107 of *Leibniz International Proceedings in Informatics*, pages 14:1–14:17. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2018. doi:10.4230/LIPIcs.ICALP.2018.14.

[8] Eli Ben-Sasson, Dan Carmon, Yuval Ishai, Swastik Kopparty, and Shubhangi Saraf. Proximity gaps for Reed–Solomon codes. Electronic Colloquium on Computational Complexity, Report TR20-083, Revision 3, July 2021. July 3, 2021 revision. URL: https://eccc.weizmann.ac.il/report/2020/083/revision/3/download.

[9] Alexander R. Block, Justin Holmgren, Alon Rosen, Ron D. Rothblum, and Pratik Soni. Public-coin zero-knowledge arguments with (almost) minimal time and space overheads. Cryptology ePrint Archive, Paper 2020/1425, 2020. URL: https://eprint.iacr.org/2020/1425, doi:10.1007/978-3-030-64378-2_7.

[10] Jonathan Bootle, Alessandro Chiesa, Yuncong Hu, and Michele Orrù. Gemini: Elastic SNARKs for diverse environments. In *Advances in Cryptology – EUROCRYPT 2022*, volume 13276, pages 427–457. Springer, 2022.

[11] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In *Advances in Cryptology – EUROCRYPT 2023*, pages 499–530. Springer, 2023.

[12] James W. Cooley and John W. Tukey. An algorithm for the machine calculation of complex fourier series. *Mathematics of Computation*, 19(90):297–301, 1965.

[13] Matteo Frigo, Charles E. Leiserson, Harald Prokop, and Sridhar Ramachandran. Cache-oblivious algorithms. In *40th Annual Symposium on Foundations of Computer Science*, pages 285–297. IEEE, 1999.

[14] Alexander Golovnev, Jonathan Lee, Srinath Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. Cryptology ePrint Archive, Report 2021/1043, full version, 2021. URL: https://eprint.iacr.org/2021/1043.

[15] Alexander Golovnev, Jonathan Lee, Srinath T. V. Setty, Justin Thaler, and Riad S. Wahby. Brakedown: Linear-time and field-agnostic SNARKs for R1CS. In *Advances in Cryptology – CRYPTO 2023, Part II*, pages 193–226. Springer, 2023. doi:10.1007/978-3-031-38545-2_7.

[16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016*, pages 305–326. Springer, 2016. doi:10.1007/978-3-662-49896-5_11.

[17] Yanpei Guo, Xuanming Liu, Kexi Huang, Wenjie Qu, Tianyang Tao, and Jiaheng Zhang. DeepFold: Efficient multilinear polynomial commitment from Reed–Solomon code and its application to zero-knowledge proofs. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 3497–3516. USENIX Association, 2025. URL: https://www.usenix.org/conference/usenixsecurity25/presentation/guo-yanpei.

[18] Jia-Wei Hong and Hsiang-Tsung Kung. I/O complexity: The red–blue pebble game. In *Proceedings of the 13th Annual ACM Symposium on Theory of Computing*, pages 326–333. ACM, 1981.

[19] Kyeongtae Lee, Jihye Kim, and Hyunok Oh. Sluice: Prove-phase bounded-memory Groth16 via read-write streaming. Cryptology ePrint Archive, Paper 2026/1758, 2026. URL: https://eprint.iacr.org/2026/1758.

[20] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. doi:10.1145/146585.146605.

[21] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. BLAKE3: One function, fast everywhere. Cryptographic hash-function specification, 2020. URL: https://github.com/BLAKE3-team/BLAKE3-specs/blob/master/blake3.pdf.

[22] Christodoulos Pappas and Dimitrios Papadopoulos. Hobbit: Space-efficient zkSNARK with optimal prover time. In *34th USENIX Security Symposium (USENIX Security 25)*, pages 4917–4936. USENIX Association, 2025. URL: https://www.usenix.org/conference/usenixsecurity25/presentation/pappas.

[23] Irving S. Reed and Gustave Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8(2):300–304, 1960.

[24] Srinath Setty. Spartan: Efficient and general-purpose zkSNARKs without trusted setup. In *Advances in Cryptology – CRYPTO 2020*, pages 704–737. Springer, 2020.

[25] Srinath Setty, Riad S. Wahby, et al. Spartan: Shockwave implementation branch. GitHub repository, 2022. Commit caa46e564377c6dea8463f8d1fa37a4785016ad5. URL: https://github.com/conroi/Spartan/tree/caa46e564377c6dea8463f8d1fa37a4785016ad5.

[26] Jeffrey Scott Vitter. External memory algorithms and data structures: Dealing with massive data. *ACM Computing Surveys*, 33(2):209–271, 2001.

[27] Barry WhiteHat, Marta Bellés, and Jordi Baylina. ERC-2494: Baby jubjub elliptic curve. Ethereum Improvement Proposals, ERC-2494, 2020. URL: https://eips.ethereum.org/EIPS/eip-2494.

[28] Hadas Zeilberger, Binyi Chen, and Ben Fisch. BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. In *Advances in Cryptology – CRYPTO 2024, Part X*, pages 138–169. Springer, 2024. doi:10.1007/978-3-031-68403-6_5.

[29] Hadas Zeilberger, Binyi Chen, and Ben Fisch. BaseFold: Efficient field-agnostic polynomial commitment schemes from foldable codes. Cryptology ePrint Archive, Paper 2023/1705, 2026. Full version, revised July 10, 2026. URL: https://eprint.iacr.org/2023/1705.

[30] Zongyang Zhang, Weihan Li, Yanpei Guo, Kexin Shi, Sherman S. M. Chow, Ximeng Liu, and Jin Dong. Fast RS-IOP multivariate polynomial commitments and verifiable secret sharing. In *33rd USENIX Security Symposium (USENIX Security 24)*, pages 3187–3204. USENIX Association, 2024. URL: https://www.usenix.org/conference/usenixsecurity24/presentation/zhang-zongyang.

A Code and Data Availability

A Rust prototype, canonical proof vectors, differential tests and earlier experiment materials are distributed through https://anonymous.4open.science/r/weir-artifact/. The primary fusion/coverage and capacity/component measurements are separate studies, each identified by source snapshots, input hashes, condition schedules and per-process records. Sections B and F report their methods, results and cost accounting.

B Supplementary Measurements

Execution environment. The primary and capacity studies use an Apple M1 Pro (10 cores, 32 GiB RAM, Apple AP1024R 1 TB SSD), Docker Desktop 29.5.2/Linux 6.12.76 and Rust 1.96.0 release builds. Input and scratch files use an APFS bind mount. Each container has one prover thread and a one-CPU quota, without CPU pinning or swap. Guest and host caches are uncontrolled.

The matrix scale/calibration plans contain 27/9 attempts; the separate Poseidon plan contains 27. All use the same frozen binary, image, one-CPU quota and 1200 s timeout. The Poseidon plan fixes 4/8/16 hashes and rotates RAM 64 MiB, RW 64 MiB and RAM 1 GiB across three repetitions; the second repetition reverses input order. Each plan's timing aggregates use its own fresh processes, including the size-4 Poseidon point. Their 63 attempts give 45 verified proofs and 18 cgroup-confirmed RAM OOMs. Active rows and supplied witness coverage were independently computed from canonical sparse entries.

Input	Active rows	C	V	O	W	B
Matrix 8	577	1024	1024	2048	704	1024
Matrix 16	4353	8192	8192	16384	4864	2048
Matrix 24	14401	16384	16384	32768	15552	4096
Matrix 32	33793	65536	65536	131072	35840	8192
Poseidon 4	961	1024	1024	32768	962	1024
Poseidon 8	1921	2048	2048	65536	1922	1024
Poseidon 16	3841	4096	4096	131072	3842	2048

	Index+prove s: median [min,max]	
Input	RAM 1 GiB	RW 64 MiB
Matrix 8	1.92 [1.92,1.93]	21.27 [21.00,21.64]
Matrix 16	4.75 [4.75,4.76]	83.52 [81.03,83.95]
Matrix 24	7.57 [7.56,7.58]	155.10 [149.73,155.41]
Matrix 32	18.69 [18.57,18.76]	570.60 [566.73,576.83]
Poseidon 4	5.94 [5.94,5.98]	131.30 [130.70,133.17]
Poseidon 8	8.81 [8.79,8.86]	252.21 [251.97,255.83]
Poseidon 16	14.78 [14.73,14.80]	490.82 [481.33,495.15]

	Prover RSS MB		RW costs			
Input	RAM 1 GiB	RW 64 MiB	Payload MB	Proof MB	Verify s	Query GB
Matrix 8	18.44	3.28	19.20	1.150	1.99	0.544
Matrix 16	124.95	3.31	153.53	2.406	4.70	1.275
Matrix 24	247.62	3.31	308.34	3.215	7.08	1.934
Matrix 32	969.57	3.33	1228.08	5.892	14.32	3.936
Poseidon 4	227.30	3.49	302.53	2.696	5.57	1.527
Poseidon 8	449.91	3.31	605.05	3.610	7.52	2.035
Poseidon 16	894.48	3.31	1210.10	4.899	11.04	3.049

Table 5: Workload shape and capacity-study costs. Active rows have a nonzero aggregate matrix coefficient; C is padded. Costs are medians of three verified RAM 1 GiB and RW 64 MiB runs per input; timing brackets give descriptive min/max. Common-cap outcomes appear in Figure 2. Every supplied private slot is referenced; all padded constraints were independently checked. Query GB counts cumulative field/index key bytes. Payload is the peak simultaneous logical external data; RSS excludes verification/readback.

Fixture	(C, V, O)	Shockwave 64 MiB	First tested success	Weir 64 MiB	Weir external peak
Matrix 16×16	$(8192, 8192, 16384)$	0/3	128 MiB, 3/3	3/3	153.5 MB
Poseidon 4 hashes	$(1024, 1024, 32768)$	0/3	256 MiB, 3/3	3/3	302.5 MB
Poseidon 8 hashes	$(2048, 2048, 65536)$	0/1	512 MiB, 1/1	1/1	605.1 MB

Table 6: Historical Shockwave/Weir comparison under the differing profiles of Section 8.6: verified/attempted runs. All failed cells have cgroup-confirmed OOM. Poseidon 4 also fails at 128 MiB and Poseidon 8 at 128/256 MiB (one run each). "First tested success" denotes the first verified cap among the tested settings. External peak includes live input and intermediate payload.

B.1 Primary Fusion and Coverage Study

The frozen schedule has 104 encoder and 64 common-cap prover conditions, each repeated three times, plus six 64 MiB conditions with three attempts each. The 300-minute budget, 180 s encoder/1200 s prover timeouts, CPU quota, image/binary/source/input hashes and condition order were fixed before execution. The 522 attempts finish in 135.57 summed process minutes: 516 successes and six confirmed RAM OOMs. Separate pilots are excluded from every aggregate.

Input coverage and storage convention. Table 7 reports supplied private slots separately from capacity and insertion to O. The four structured fixtures have no explicit zero coefficients or duplicate pairs before padding. In the balanced synthetic matrices, A/C each contain 94 explicit zeros and 17–23 duplicate excess entries; B has 128 nonzero entries and 896 inserted zeros. The dense A/C matrices each contain 373 explicit zeros and 250–285 duplicate excess entries, while B receives 3968 inserted zeros. References use nonzero coefficients after modular duplicate aggregation. Coverage distinguishes public/constant slots, supplied private slots and private padding.

Input	(C, V, O)	W	Referenced W	Padded private	Stored A/B/C
Bal. filled/1	$(128, 128, 1024)$	127	125	1	1024/128/1024
Bal. filled/11	$(128, 128, 1024)$	127	122	1	1024/128/1024
Bal. filled/7	$(128, 128, 1024)$	127	126	1	1024/128/1024
Bal. one/1	$(128, 128, 1024)$	1	1	125	1024/128/1024
Dense filled/1	$(128, 128, 4096)$	127	127	1	4096/128/4096
Dense filled/11	$(128, 128, 4096)$	127	127	1	4096/128/4096
Dense filled/7	$(128, 128, 4096)$	127	127	1	4096/128/4096
Legacy ratio	$(128, 8192, 1024)$	1	0	461	1024/128/1024
Matrix 16	$(8192, 8192, 16384)$	4864	4864	0	8449/4353/4097
Matrix 8	$(1024, 1024, 2048)$	704	704	0	1089/577/513
Poseidon 2	$(512, 512, 16384)$	482	482	0	4486/8485/480
Poseidon 4	$(1024, 1024, 32768)$	962	962	0	8970/16973/960

Table 7: Input coverage in the matched-schedule study. W is supplied private witness length, whereas V is padded half-capacity. Referenced slots have nonzero aggregate coefficients after duplicate aggregation. Stored counts precede insertion to length O; each matrix receives $O-$stored explicit zero entries. All padded constraints were independently checked.

The larger buffers are 1024, 2048, 512 and 1024 for matrix 8/16 and Poseidon 2/4. At these buffers the sitewise N/B, in var, deref, ops, mem order, is respectively $(1, 4, 4, 2)$, $(1, 4, 8, 2)$, $(1, 16, 32, 2)$ and $(1, 16, 16, 2)$. Thus the larger-buffer contrast still has bridges at some sites, but none at var or mem. Combined RW has 3.18–3.38 MB prover RSS versus RAM's 18.43–227.30 MB, with paired index+prove ratios 9.77–20.84.

Zero-tail controls and timing variability. At $N = 8192, B = 4$, moving from padded to implicit-zero input and then to the half-rate first stage each removes 8192 reads and 8192 writes for every schedule. The half-rate/implicit-zero boundary-only encoder ratio is 1.105 [1.005,1.170]; for the combined schedule it is 0.969 [0.683,1.168]. The standalone timer excludes initial coefficient publication, output collection/hash and cleanup, but includes padded publication when that input policy explicitly performs it. Dense seed 11 at $B = 256$ has lower encoding time in every paired repetition, yet two runs have extra transcript query-construction and prefix-advancement time.

Proofs, query bytes and logical counters stay fixed; the cause of the timing variation remains unidentified.

Input	$B_{\max}$	RAM s	Combined s at $B_{\max}$ [min,max]
Matrix 8	1024	1.84	17.97 [17.74,18.03]
Matrix 16	2048	4.54	73.18 [73.04,73.59]
Poseidon 2	512	3.90	64.87 [64.85,65.91]
Poseidon 4	1024	5.74	119.07 [118.89,142.54]
Bal. filled/1	256	1.15	12.02 [12.01,12.02]
Bal. filled/11	256	1.15	12.00 [12.00,12.06]
Bal. filled/7	256	1.15	12.05 [12.00,12.07]
Bal. one/1	256	1.15	12.04 [11.99,12.05]
Dense filled/1	256	1.96	23.21 [23.15,23.25]
Dense filled/11	256	1.96	28.99 [23.26,31.50]
Dense filled/7	256	1.95	23.13 [23.11,23.20]
Legacy ratio	2048	1.97	17.98 [17.95,18.04]

	Combined/endpoint time ratio		Combined/RAM
Input	$B = 4$ [min,max]	$B_{\max}$ [min,max]	RSS at $B_{\max}$
Matrix 8	0.973 [0.972,0.981]	1.001 [0.976,1.005]	0.173
Matrix 16	0.957 [0.955,0.958]	0.998 [0.963,1.000]	0.026
Poseidon 2	0.957 [0.956,0.961]	0.983 [0.824,0.983]	0.029
Poseidon 4	0.953 [0.934,0.954]	0.987 [0.986,1.148]	0.015
Bal. filled/1	0.984 [0.971,0.993]	0.991 [0.987,0.994]	0.307
Bal. filled/11	0.986 [0.983,0.987]	0.994 [0.992,1.002]	0.316
Bal. filled/7	0.985 [0.973,0.988]	0.994 [0.992,0.995]	0.313
Bal. one/1	0.986 [0.980,0.992]	0.997 [0.986,0.997]	0.310
Dense filled/1	0.963 [0.961,0.969]	0.988 [0.986,0.989]	0.101
Dense filled/11	0.968 [0.959,0.969]	1.200 [0.992,1.335]	0.101
Dense filled/7	0.967 [0.964,0.967]	0.985 [0.985,0.988]	0.102
Legacy ratio	0.972 [0.963,0.981]	1.001 [0.992,1.007]	0.114

Table 8: Primary 2 GiB study: all 12 inputs and both matched buffer points, three processes per condition. Times are medians; ratios pair repetition labels and brackets give descriptive min/max. Time ratios compare combined with endpoint-only at the indicated buffer; RSS ratios compare combined at $B_{\max}$ with RAM. Each input compares RAM, endpoint-only/combined at both buffers and legacy unfused at $B_{\max}$ for the four structured inputs. All 192 runs verify.

B.2 Separate Fusion and Input Controls

For materialized input and $b < n$, the unfused, boundary-only, endpoint-only and combined schedules use $b + 3(n - b)$, $2(n - b) + b + 1$, $3(n - b)$ and $2(n - b) + 1$ full read/write passes, respectively. Each of the $n - b - 1$ high–high boundaries changes from merge/split to one bridge. Endpoint batching combines the last merge and the b low stages. At $b = 0$, endpoint batching saves no pass. At $b = n - 1$, there is no high–high boundary. At $b = n$, schedules with endpoint batching use one full-block pass; schedules without it use n separate local passes.

Input	RW path	Index+prove s [min,max]	Payload MB	Proof MB	Verify s	Query GB
Matrix 16	Endpoint	73.20 [73.16,73.41]	153.53	2.406	4.48	1.275
Matrix 16	Combined	72.78 [72.63,81.22]	153.53	2.406	4.48	1.275
Poseidon 4	Endpoint	120.98 [120.81,122.88]	302.53	2.696	5.35	1.527
Poseidon 4	Combined	118.94 [118.91,119.09]	302.53	2.696	5.36	1.527

Table 9: Separate 64 MiB completion study: medians of three verified processes per RW cell, with descriptive time min/max. RAM has three cgroup-confirmed OOMs per input. Buffers are 2048 for matrix 16 and 1024 for Poseidon 4. Completion includes proof readback and verification; index+prove excludes both. Payload is peak simultaneous logical external data; Query GB counts cumulative field/index keys. These processes are distinct from the 2 GiB primary timings and later capacity studies.

Zero-tail and row-output controls. For the half-rate PCS input, the matched schedules consume $N/2$ coefficients and append directly to the same batch-output builder, with one publication of each completed row. The zero-tail policy is held fixed within a schedule comparison. Let P be the corresponding materialized schedule's pass count above. The per-row reads and writes are

Policy	Reads	Writes
Publish padded input	$NP + N/2$	$NP + N$
Implicit zero split	$NP - N/2$	NP
Half-rate first stage	$N(P-1) - N/2$	$N(P-1)$

The first row includes publication of the padded input. The second reads only the coefficients at the first split but publishes both partitions. The third combines that split and the first butterfly. For $b = n$, there is no split: both implicit policies fill zeros in the allowed N-field block. Reads are $NP - N/2$, writes are NP, with $P = 1$ or $P = n$ as specified above.

Legacy fused/unfused whole-prover measurements also differ in zero-tail treatment and completed-row copying; their contrast measures the joint change. The matched controls isolate each switch.

Correctness checks. A separate native smoke study checks 128 encoder processes across two lengths, four buffers per length, four schedules and four input policies (including arbitrary materialized input). It covers $B = 1$, one or multiple high–high boundaries, and $B = N$. A further 25 processes at $(C, V, O) = (4, 128, 16)$, seed 1 and the paper transcript comprise one RAM reference and all four matched schedules under three zero-tail policies at $B = 1,256$. Every prover verifies and has the same proof hash, proof size, oracle request counts and query bytes. Whole-prover field-traffic differences equal the sum of the four sites' derived encoder traffic differences. These are single-process correctness checks without a memory cap; their elapsed times are excluded from the performance comparisons above.

B.3 Extended Encoder Coordinates and Validation

Coset and folding coordinates. Let $f(X) = \sum_{t<m} u_t X^t$. Evaluating the coefficient-scaled polynomial $\sum_t u_t g^t X^t$ at ω_N^k gives $f(g\omega_N^k)$, proving the coset claim. At half rate the first-stage outputs are $u_q g^q$ and $u_q g^q \omega_N^q$; both powers advance sequentially. No scaled coefficient table is retained.

Encoder / coefficients	Payload MB Separate	Fused	Expansion MB
Coset RS, generated	9.437	6.554	0.197
Recursive, generated	8.651	6.029	0.197
Recursive, two streams	8.651	6.029	0.197

Encoder / coefficients	Parameter reads MB	Generator calls	Total record traffic Separate MB	Fused MB
Coset RS, generated	0.000	24576	9.634	6.750
Recursive, generated	0.000	22528	8.847	6.226
Recursive, two streams	1.442	0	10.289	7.668

Table 10: Derived encoder traffic at $N = 4096$, starting from $N/2$ prepared message records. Payload counts match the component measurements; streamed expansion and parameter costs are calculated. Coefficients are generated per pair or read from two prepared diagonal streams. The latter occupy 0.262 MB; fresh publication adds that many write bytes. An RS generator call includes root derivation/checks and one twiddle power (four modular powers); a recursive call uses one SHA-256 value and negation. Generator work is separate from record MB. All records are 32 bytes. Reference-array and protocol costs are outside this encoder accounting.

For $q < N/2$, $\mathsf{bitrev}_n(2q) = \mathsf{bitrev}_{n-1}(q)$ and $\mathsf{bitrev}_n(2q+1) = \mathsf{bitrev}_{n-1}(q) + N/2$. Adjacent values thus evaluate f at $x, -x$, with $x = g\omega_N^{\mathsf{bitrev}_{n-1}(q)}$. Writing $f(X) = f_0(X^2) + Xf_1(X^2)$, the displayed fold evaluates $f_0 + \alpha f_1$ at x^2, in bit-reversed order on the next coset. Induction gives the complete folding trace. Folding from length N to length two reads $2N - 4$ field records and writes $N - 2$. Retaining all fold layers uses fewer than $2N$ field records. Retained binary Merkle levels over these layers use fewer than $4N$ digests. Building a level by adjacent hashes needs constant digest state. Query retrieval can scan retained levels and append path records in a prescribed occurrence order; these scans and output costs are separate from encoder traffic. Computing each bit-reversed domain point by exponentiation uses $O(\log N)$ field work in the validation prototype.

Recursive generator coordinates. Take $G_0 = (1\ 1)$ and

$$G_j = \begin{pmatrix} G_{j-1} & G_{j-1} \\ G_{j-1}T_{j-1} & G_{j-1}T'_{j-1} \end{pmatrix}.$$

Here the diagonals have length 2^j; our stage labels use $t_j = \mathrm{diag}(T_{j-1})$. For message halves u, v, set $a = uG_{j-1}, b = vG_{j-1}$. Their encoding is $(a + bT_{j-1}, a + bT'_{j-1})$. Induction from adjacent repetitions gives precisely the ascending paired stages in Section 4.1, with global pair offset reduced modulo 2^j within each recursive block. This fixes the row permutation permitted by the foldable-code definition; a different message basis requires its own adapter. Sequential execution uses the following explicit diagonal-access contract. For this recurrence, offset q consumes diagonal entry $q \bmod 2^j$. Reopening a length-2^j parameter stream at each block realizes exactly that sequence with constant field state. The two diagonal families contain $2\sum_{j=1}^{n-1} 2^j = 2N - 4$ field records in total. Each of the $d = n - 1$ stages consumes $N/2$ pairs, hence two stored diagonals incur dN parameter reads in either schedule. The scalar maps retain constant field state and bounded counters; source identities and cursors count in M_{meta}.

Total encoder accounting. Table 10 starts from a prepared message stream of $m = N/2$ records. Streaming coset scaling/zero padding or base repetition reads m records and writes N, so total field traffic is

$$I_{\mathbb{F}} = 2NP + \tfrac{3}{2}N + R_{\text{par}},$$

where P is the payload-pass count. For generated coefficients $R_{\text{par}} = 0$; two stored diagonals give $R_{\text{par}} = dN$. Publishing fresh diagonal streams adds $2N - 4$ writes; their prepared storage is already charged separately in the table. Generator calls, detailed in Table 10, contribute CPU work separately from these record totals. Each generator retains constant field state, one digest and logarithmic counters/input buffers.

Component experiment. The independent Python file engine implements separate split/map/merge and boundary-fused schedules. Every payload access is a forward 32-byte read or append. Input expansion is outside the counted encoder: scaled coefficients followed by zeroes for RS, and adjacent repetition for the recursive code. RS references use direct Horner evaluation at the two smaller sizes and a natural-frequency DIT FFT at the larger sizes, followed by an explicit reference permutation. Recursive-code references directly evaluate the block-generator recurrence in RAM. Sizes, seeds and code families follow Section 8.3. Nonzero diagonals are deterministic SHA-256-derived fixtures with $t'_j = -t_j$. These fixed fixtures validate ordered equality and traffic. Security of a PCS using the recursive code additionally requires its construction's setup and distance conditions.

A coefficient-folding reference checks every field value and Merkle node, with duplicate and boundary query occurrences. Hashing uses scalar SHA-256 leaves and ordered concatenation at internal nodes. The harness retains RAM reference arrays; algorithmic RAM is established by Theorem 4.4. Four query occurrences per RS state include duplicates and boundary indices; aggregate counts are reported in Section 8.3.

B.4 Historical Measurements

Archived scaling/buffer controls fix $V = 64C, O = 8C$, seed 1, and combine boundary, endpoint and zero-tail changes. Their snapshot path uses append-only-file prefixes; Section 6 uses the immutable stream interface.

Smaller matrix sizes 8 and 12 complete at 64 MiB in both provers (one run each). Matrix 16 has 107.7 MB original RSS versus 3.23–3.30 MB for Weir; Poseidon 4 has 184.77–184.79 MB versus 3.34–3.35 MB.

The 96 macOS cache-policy and twelve Linux guest-eviction processes preserve proofs and logical traffic while OS/guest reads and times vary. Linux synchronization is matched; uncontrolled host cache limits the counters to guest/OS traffic.

C Algebraic Protocol Specification

We specify the tables, messages and checks needed to instantiate the reused blocks of Section 3. The input relation, field, coordinates and scalar encoding are defined in Section 3.

Definition C.1 (Public generator tuple)**.** The context binds an ordered tuple

$$G = \left(2^{\lceil v/2 \rceil}, 2^{\lceil (s+4)/2 \rceil}, 2^{\lceil (h+1)/2 \rceil}, 2^{\lceil (s+3)/2 \rceil}\right), \quad 0 \le s \le 63. \tag{11}$$

The integer s is independent of o; these retained public parameters are unused by the dense PCS equations.

Definition C.2 (Preprocessing and committed tables)**.** Pad each sparse list to length O with $(0,0,0)$. Let $a^\rho_X, a^\kappa_X, \nu_X$ be its row addresses, column addresses and values. Order accesses by (X,k), $A<B<C$, and set

$$t^d_{X,k} = \#\{(Y,j)<(X,k) : a^d_{Y,j} = a^d_{X,k}\},$$
$$A^d[u] = \#\{(X,k) : a^d_{X,k} = u\}.$$

Their field embeddings are τ^d_X, α_d; all addresses and counts are below p, since $H \le 2^{44}$ and $3O < 2^{43}$. With each three-block group ordered A,B,C, define

$$\begin{aligned} Z_{\text{var}} &= w^+, & Z_{\text{mem}} &= \alpha_\rho \| \alpha_\kappa, \\ Z_{\text{ops}} &= (a^\rho_X)_X \| (\tau^\rho_X)_X \| (a^\kappa_X)_X \\ &\quad \| (\tau^\kappa_X)_X \| (\nu_X)_X \| 0^O, \\ Z_{\text{deref}} &= (\delta^\rho_X)_X \| (\delta^\kappa_X)_X \| 0^{2O}. \end{aligned} \tag{12}$$

Here grouped vectors are concatenated, and stored addresses denote their canonical field embeddings.

Definition C.3 (Dynamic equality memories)**.** For $r_x \in \mathbb{F}^c, r_y \in \mathbb{F}^{v+1}$, let $r^+_x = 0^{h-c}\|r_x$, $r^+_y = 0^{h-v-1}\|r_y$, and

$$\mu_\rho = E_{r^+_x}, \quad \mu_\kappa = E_{r^+_y}, \quad \delta^X_d[k] = \delta^d_X[k] = \mu_d[a^d_{X,k}].$$

The sparse claim is $y^{\text{sparse}}_X = \sum_{k<q_X} \nu_{X,k} E_{r_x}[a_{X,k}] E_{r_y}[b_{X,k}]$.

Lemma C.4 (Sparse claims and memory identity)**.** *For every challenge pair,*

$$y^{\text{sparse}}_X = \sum_{k<O} \nu_X[k]\delta^X_\rho[k]\delta^X_\kappa[k]. \tag{13}$$

Fix a direction and $\beta,\gamma \in \mathbb{F}$, and put $f_u(t) = u + \beta\mu_d[u] + \beta^2 t - \gamma$. Define $I[u] = f_u(0)$, $A[u] = f_u(A^d[u])$, $R_X[k] = f_{a^d_{X,k}}(t^d_{X,k})$, and $W_X[k] = f_{a^d_{X,k}}(t^d_{X,k}+1)$. Then

$$(\prod_u I[u])(\prod_{X,k} W_X[k]) = (\prod_{X,k} R_X[k])(\prod_u A[u]). \tag{14}$$

Proof. Leading zero tensor coordinates select the active address ranges; padded entries contribute zero through $\nu_X[k]$. At an address with a accesses, both products contain exactly $f_u(0),\dots,f_u(a)$. Multiplication over addresses proves the identity without cancellation, including zero factors. □

C.1 Sumcheck and R1CS transitions

Definition C.5 (Table sumcheck)**.** Fix a round count $k \ge 0$, tables $U_j \in \mathbb{F}^{2^k}$ for $1 \le j \le a$, an incoming transcript, and degree bound $D \in \{2,3\}$. Let Φ have total degree at most D in the table entries: the forms used are AB, $A(BC-D')$, ABC, and fixed linear combinations of triple products; D' is a table. Starting with claim e, in each round of current length $2L$ set

$$U^{[X]}_j[i] = U_j[i] + X(U_j[L+i] - U_j[i]),$$
$$g(X) = \sum_{i<L} \Phi(U^{[X]}_1[i],\dots,U^{[X]}_a[i]).$$

Interpolate degree-$\le D$ polynomial p from $(g(0), e-g(0), g(2))$ at $(0,1,2)$, adding $g(3)$ at 3 for cubic rounds. Emit all $D+1$ coefficients, including zeros. Absorb `poly/UniPoly_begin`, coefficients

in increasing degree under `coeff`, and `poly/UniPoly_end`. Request r with `challenge_nextround`, replace U_j by $U_j^{[r]}$, and set $e = p(r)$. Store only $(c_0, c_2, \ldots, c_D)$ in that round's proof. Return proof, ordered challenges, terminal tables/claim and transcript; $k = 0$ returns empty vectors and unchanged inputs.

Lemma C.6 (Sumcheck correctness and R1CS transitions)**.**

(a) *If the initial claim is $\sum_u \Phi(U_1[u], \ldots, U_a[u])$, every round polynomial is the true partial sum, and the final tables are $\widetilde{U}_j(r)$, for every challenge sequence. The verifier reconstructs the absorbed coefficients by*

$$c_1 = e - 2c_0 - \sum_{j=2}^{D} c_j. \tag{15}$$

Its terminal check is $e = \Phi(\widetilde{U}_1(r), \ldots, \widetilde{U}_a(r))$.

(b) *For a satisfying assignment $z = z^{\mathrm{assn}}$ and any $\tau \in \mathbb{F}^c$, phase one uses tables (E_τ, Az, Bz, Cz), degree three, c rounds, expression $A(BC - D')$, and claim zero. At its point r_x, let $a_X = \widetilde{Xz}(r_x)$. For subsequent coefficients ξ_A, ξ_B, ξ_C, define*

$$M_X[j] = \sum_{k : b_{X,k} = j} \nu_{X,k} E_{r_x}[a_{X,k}] \qquad (j < 2V),$$
$$M[j] = \sum_{X = A,B,C} \xi_X M_X[j], \qquad e_2 = \sum_X \xi_X a_X.$$

Phase two uses (z, M), degree two, $v+1$ rounds, expression AB, and claim e_2. Both honest sumcheck invariants hold. Write $r_y = (b) \| r'_y$ and $z_{\mathrm{pub}} = (1) \| x \| 0^{V-I-1}$. The phase-one terminal check is $e = \widetilde{E}_\tau(r_x)(a_A a_B - a_C)$; the phase-two check is

$$\begin{aligned} e = & \left(\sum_X \xi_X \widetilde{X}(r_x, r_y) \right) \\ & \cdot \left((1-b)\widetilde{w^+}(r'_y) + b \widetilde{z}_{\mathrm{pub}}(r'_y) \right), \end{aligned} \tag{16}$$

where $\widetilde{X}(r_x, r_y) = \sum_k \nu_{X,k} E_{r_x}[a_{X,k}] E_{r_y}[b_{X,k}]$.

Proof. (a) MSB splitting gives $g(0) + g(1) = e$, hence $p = g$; folding gives next sum $g(r)$. Induction yields terminal evaluations. Reconstruction solves the endpoint sum, giving identical replay.

(b) Vanishing residuals give phase one's claim. Reordering sums gives $\sum_j z[j] M_X[j] = a_X$ and $\widetilde{M_X}(r_y) = \widetilde{X}(r_x, r_y)$. Tensor factorization gives $\widetilde{z}(r_y) = (1-b)\widetilde{w^+}(r'_y) + b\widetilde{z}_{\mathrm{pub}}(r'_y)$ and $\widetilde{E}_\tau(r_x) = \prod_i (\tau_i r_{x,i} + (1-\tau_i)(1-r_{x,i}))$. Apply part (a) to both phases. The empty-product convention covers $c = 0$. □

C.2 Product-network reductions

Definition C.7 (Product tables and batched reduction)**.** Fix $L, P \geq 1$ and $D \geq 0$. The product inputs are $U_j \in \mathbb{F}^{2^L}$ for $j < P$. For each $a < D$, the three dot tables A_a, B_a, W_a have length 2^{L-1}. Set $V_{j,L} = U_j$ and, descending in $t < L$,

$$V_{j,t}[i] = V_{j,t+1}[i] V_{j,t+1}[2^t + i] \quad (i < 2^t).$$

Start with $c_{j,0} = V_{j,0}[0]$, $r_0 = ()$, and $d_a = \sum_i A_a[i] B_a[i] W_a[i]$. For $t = 0, \ldots, L-1$, let $A_{j,t}, B_{j,t}$ be the two halves of $V_{j,t+1}$, and $D_t = D$ only at $t = L-1$, zero otherwise.

1. Request $P + D_t$ scalars under `rand_coeffs_next_layer`, in order, obtaining λ_j.
2. Run t cubic sumcheck rounds with claim $e_t = \sum_{j<P} \lambda_j c_{j,t} + \sum_{a<D_t} \lambda_{P+a} d_a$ and expression
$$\Phi_t = \sum_{j<P} \lambda_j A_{j,t} B_{j,t} E_{r_t} + \sum_{a<D_t} \lambda_{P+a} A_a B_a W_a.$$
Let its point and proof be u_t, S_t, its product terminal values $a_{j,t}, b_{j,t}$, and final-layer dot values a'_a, b'_a, w'_a.
3. For each increasing j, absorb $a_{j,t}, b_{j,t}$ under `claim_prod_left`, `claim_prod_right`. At the last layer, for each increasing a, absorb a'_a, b'_a, w'_a under `claim_dotp_left`, `claim_dotp_right`, `claim_dotp_weight`.
4. Request ζ_t under `challenge_r_layer`; set $c_{j,t+1} = (1-\zeta_t)a_{j,t} + \zeta_t b_{j,t}$ and $r_{t+1} = (\zeta_t) \| u_t$. Store $(S_t, (a_{j,t})_j, (b_{j,t})_j)$.

Return the layer-record vector followed by $((a'_a)_a, (b'_a)_a, (w'_a)_a)$, together with r_L and the outgoing transcript. The final three vectors are empty for $D = 0$; $c_{j,L}$ is derived, not serialized.

Lemma C.8 (Layer invariant and terminal claims)**.** *For every honest challenge sequence, $c_{j,0} = \prod_i U_j[i]$ and $c_{j,t} = \widetilde{V}_{j,t}(r_t)$. Writing $\mathsf{eq}(r,u) = \prod_i (r_i u_i + (1-r_i)(1-u_i))$, the sumcheck terminal equation is*
$$e = \mathsf{eq}(r_t, u_t) \sum_{j<P} \lambda_j a_{j,t} b_{j,t} + \sum_{a<D_t} \lambda_{P+a} a'_a b'_a w'_a. \tag{17}$$
For $D = 2q$ consecutive half-triples of full tables $(\bar{A}_i, \bar{B}_i, \bar{W}_i)$, folding each terminal pair by $(x,y) \mapsto (1-\zeta_{L-1})x + \zeta_{L-1} y$ yields their full-table evaluations at r_L.

Proof. Disjoint-half multiplication gives the product roots; $\sum_z A_{j,t}[z] B_{j,t}[z] E_{r_t}[z] = c_{j,t}$ and the dot sums are d_a. Honest sumcheck gives the terminal equation; first-coordinate expansion gives folded product/dot claims. Induction and coefficient reconstruction give replay, including $L = 1$. □

Sparse instantiation and event order. The ops batch has $(P, D, L) = (12, 6, o)$, with product inputs
$$((R^X_\rho)_X, (W^X_\rho)_X, (R^X_\kappa)_X, (W^X_\kappa)_X), \quad X = A, B, C.$$
Its dot triples are the halves of $(\delta^X_\rho, \delta^X_\kappa, \nu_X)$, ordered
$$(A,0), (A,1), (B,0), (B,1), (C,0), (C,1).$$
The mem batch is $(4, 0, h)$ on $(I_\rho, A_\rho, I_\kappa, A_\kappa)$.

First absorb protocol name `Sparse polynomial product layer proof`. For row then column, absorb initialization, read, write and audit root products. Their labels concatenate `claim_`, the direction string (`row` or `col`), `_eval_`, and the kind (`init`, `read`, `write`, `audit`), in the stated order. Reads and writes use three-element scalar-vector wrappers. For $X = A, B, C$, absorb its two dot-half sums under `claim_eval_dotp_left`, `claim_eval_dotp_right`. Check Equation (14) in both directions and $d_{X,0} + d_{X,1} = y^{\text{sparse}}_X$. Run ops then mem on the continuing transcript, obtaining $r_{\text{ops}} \in \mathbb{F}^o, r_{\text{mem}} \in \mathbb{F}^h$. Writing $\Pi U = \prod_i U[i]$, ProductLayer's row/col fields store $(\Pi I_d, (\Pi R^X_d)_X, (\Pi W^X_d)_X, \Pi A_d)$, $d = \rho, \kappa$; its value field stores $((d_{X,0})_X, (d_{X,1})_X)$. Vectors use A, B, C order; serialization then stores mem before ops (Table 11).

Lemma C.9 (Instantiated terminal claims)**.** *Honest product outputs are the compression-table evaluations at $r_{\text{ops}}, r_{\text{mem}}$; recombined dot outputs are $(\widetilde{\delta}^X_\rho, \widetilde{\delta}^X_\kappa, \widetilde{\nu}_X)(r_{\text{ops}})$.*

Proof. Apply Lemma C.8 to these ordered instances, using Lemma C.4 for their initial claims. □

C.3 Caller evaluations

Lemma C.10 (Concatenation, binding and caller correctness)**.** (a) *For $s, t \geq 0$, $U_j \in \mathbb{F}^{2^t}$, and $Z = U_0 \| \cdots \| U_{2^s-1}$, put $e_j = \widetilde{U}_j(r)$. Then*

$$\widetilde{Z}(a\|r) = \sum_j E_a[j]\widetilde{U}_j(r) = \widetilde{e}(a). \tag{18}$$

Define

$$\mathsf{Bot}_\gamma(v)[j] = (1-\gamma)v[2j] + \gamma v[2j+1].$$

Applying $\mathsf{Bot}_{a_{s-1}}, \ldots, \mathsf{Bot}_{a_0}$ leaves $(\widetilde{e}(a))$.

(b) *For honest tables, $y_X = \widetilde{Z}_X(z_X)$ at all four sites.*

Proof. (a) Use $E_{a\|r}[j2^t+k] = E_a[j]E_r[k]$. Each bottom binding sums out the last Boolean coordinate; induction gives the stated reverse order, including empty coordinate lists.

(b) Var directly evaluates w^+ at $r_y[1\ldots]$. For deref, ops and mem, apply part (a) with lengths $(s,t) = (3,o),(4,o),(1,h)$ to the ordered blocks of Equation (12) and Definition C.11. Only the explicit merge-padding blocks are zero. □

C.4 Hash-layer values and checks

Definition C.11 (Hash-layer values)**.** For $d = \rho, \kappa$ and $X = A, B, C$, put

$$\begin{aligned} a_d^X &= \widetilde{[a_X^d]}(r_{\mathrm{ops}}), \quad t_d^X = \widetilde{\tau_X^d}(r_{\mathrm{ops}}), \\ e_d^X &= \widetilde{\delta_d^X}(r_{\mathrm{ops}}), \quad v_X = \widetilde{\nu_X}(r_{\mathrm{ops}}), \quad q_d = \widetilde{\alpha_d}(r_{\mathrm{mem}}). \end{aligned}$$

Brackets embed integer addresses in $\mathbb{F}$. Also define

$$\begin{aligned} \mathsf{id}_h(z) &= \sum_{i<h} [2^{h-1-i}] z_i, \\ m_\rho(z) &= \mathsf{eq}(r_x^+, z), \qquad m_\kappa(z) = \mathsf{eq}(r_y^+, z). \end{aligned}$$

Begin with protocol name `Sparse polynomial hash layer proof`. Execute deref, ops, mem caller reductions, each followed by its complete opening. Deref additionally begins with protocol name `Derefs evaluation proof`. Their block-evaluation vectors, in that order, are

$$\begin{aligned} &((e_\rho^X)_X, (e_\kappa^X)_X, 0, 0), \\ &((a_\rho^X)_X, (t_\rho^X)_X, (a_\kappa^X)_X, (t_\kappa^X)_X, (v_X)_X, 0), \\ &(q_\rho, q_\kappa). \end{aligned}$$

Absorb the deref vector under `evals_ops_val`, the ops vector under `claim_evals_ops`, and the mem vector under `claim_evals_mem`.

The prefix lengths are $3, 4, 1$.

Use `challenge_combine_n_to_one` for the first two and `challenge_combine_two_to_one` for the third. Reverse bottom binding gives the joint scalar. Absorb it under the respective label `joint_claim_eval`, `joint_claim_eval_ops`, or `joint_claim_eval_mem`. For deref and ops, append r_{ops} to the prefix; for mem, append r_{mem}. Open at this concatenated point. HashLayer stores row/col $((a_d^X)_X, (t_d^X)_X, q_d)$, $d = \rho, \kappa$, values $(v_X)_X$, derefs $((e_\rho^X)_X, (e_\kappa^X)_X)$, then openings ops, mem, deref. Vectors use A, B, C order.

Lemma C.12 (Hash-layer terminal identities)**.** *For honest tables and every challenge sequence,*

$$\widetilde{I}_d(r_{\mathrm{mem}}) = \mathsf{id}_h(r_{\mathrm{mem}}) + \beta m_d(r_{\mathrm{mem}}) - \gamma, \tag{19}$$

$$\widetilde{A}_d(r_{\mathrm{mem}}) = \mathsf{id}_h(r_{\mathrm{mem}}) + \beta m_d(r_{\mathrm{mem}}) + \beta^2 q_d - \gamma, \tag{20}$$

$$\widetilde{R}_d^X(r_{\mathrm{ops}}) = a_d^X + \beta e_d^X + \beta^2 t_d^X - \gamma, \tag{21}$$

$$\widetilde{W}_d^X(r_{\mathrm{ops}}) = a_d^X + \beta e_d^X + \beta^2 (t_d^X + 1) - \gamma. \tag{22}$$

The product-layer dot outputs are $(e_\rho^X, e_\kappa^X, v_X)_X$*. These equations and the joint opening claims are the consumer checks.*

Proof. Since $\sum_u E_z[u] = 1$, multilinear extension preserves affine combinations. The address extension is id_h; the equality-memory extensions are m_ρ, m_κ. Apply this to the compression formulas. Lemmas C.9 and C.10 give the dot and opening identities. □

C.5 Events and lifecycle

$\mathsf{App}(l, b)$ appends the label/payload of Definition D.2. Labels are literal ASCII: typographic breaks add no bytes and `$l` is literal. Protocol names use label `protocol-name`; scalar payloads use $\mathsf{Enc}_{\mathbb{F}}$. A vector emits `begin_append_vector`, its ordered scalar entries, and `end_append_vector`, all under its label even when empty. Polynomials use Definition C.5's events.

$\mathsf{PolyComm}(l, R)$ emits the following events in order:

$$(l, \texttt{poly_commitment_begin}),$$
$$(\texttt{poly_commitment_share}, R),$$
$$(l, \texttt{poly_commitment_end}).$$

DerefComm surrounds these with `begin_derefs_commitment` and `end_derefs_commitment`, both under `derefs_commitment`. Wrappers request no randomness and reset nothing.

Definition C.13 (Index-and-prove lifecycle)**.** Use Section 6's realization in the following order.

1. Index. Preprocess and commit ops then mem; retain their states and bind roots/dimensions in K. Emit no application event. Construct C and initialize W_0.

2. R1CS. Append `Spartan SNARK proof` and `R1CS proof`. Commit w^+, emitting

$$\mathsf{PolyComm}(\texttt{poly_commitment}, R_{\mathrm{var}}).$$

Request c scalars under `challenge_tau`; run phase one. Absorb a_X under `Xz_claim` in $X = A, B, C$ order; then request ξ_X under `challenege_Xz` in that order. Replace `X` by the letter; `challenege` is literal. Run phase two; open var at $r_y[1 \ldots]$.

3. Dereferences. Form dynamic tables; absorb each sparse claim under `Xr_claim`, $X = A, B, C$. Append `Sparse polynomial evaluation proof`. Commit Z_{deref}, emitting $\mathsf{DerefComm}(l, R_{\mathrm{deref}})$ for $l =$ `comm_poly_row_col_ops_val`. Request β, γ under `challenge_r_hash`.

4. Sparse proof. Append `Sparse polynomial evaluation proof` again. Run product batches ops then mem, followed by hash-layer callers/openings deref, ops, mem.

5. Output. Assemble Table 11 with its context header, adding no event. Retain each commitment through its opening; every stage continues the preceding transcript.

C.6 Canonical proof tree

S is a canonical scalar, R a 32-byte digest, and $V(T)$ a count-prefixed ordered vector (distinct from size V); U is uint64 and $B = (S, V(S), V(S), S)$. Table 11 fixes declaration order; Definition D.3 fixes all lengths. Type aliases in the table only abbreviate names; wrappers add no bytes and digests have no count prefix. Here Comm and DerefComm are R, Eval is Sparse, DerefEval is Open, SC is $V(\mathrm{CU})$, and CU is $V(S)$ except the linear coefficient.

Table 11: Canonical body grammar in declaration order. Sat/Eval are R1CS satisfaction/evaluation proofs; other names are subrecords. Execution follows the lifecycle.

Type	Ordered contents
SNARK	Sat, (S, S, S), Eval
Sat	Comm, SC, (S, S, S), SC, S, Open
Open	Opening, (U, U) split dimensions
Opening	U column count, $V(S)$ PE, $V(V(S))$ PR, V(Column)
Column	$V(S)$ column, $V(R)$ path
Sparse	DerefComm, Network
Network	ProductLayer, HashLayer
ProductLayer	B row, B col, $(V(S), V(S))$ values, Batch mem, Batch ops
Batch	V(Layer), $(V(S), V(S), V(S))$ dot claims
Layer	SC, $V(S)$ left, $V(S)$ right
HashLayer	$(V(S), V(S), S)$ row, $(V(S), V(S), S)$ col, $V(S)$ values, $(V(S), V(S))$ derefs, Open ops, Open mem, DerefEval

Lemma C.14 (Shape-directed sizes and offsets)**.** *Every proof leaf and count has a dimension-determined byte offset. The body can be assembled from offset-tagged chunks without retaining the proof tree or an offset table in RAM. Computing one offset takes $O(\log^2 T)$ bit work and $O(\log T)$ auxiliary bits.*

Proof. Let $s(\cdot)$ denote encoded byte length, excluding the context header. Scalars and digests have size 32, integers size 8, tuples sum their field sizes, and a vector adds 8 to the sum of its element sizes. In particular,

$$\begin{aligned} s(\mathrm{SC}_{D,k}) &= 8 + k(8 + 32D), \\ s(\mathrm{Layer}_{P,t}) &= 24 + 64P + 104t, \\ s(\mathrm{Batch}_{P,D,L}) &= 32 + L(24 + 64P) \\ &\quad + 52L(L-1) + 96D, \\ s(\mathrm{Open}_{r,m,N}) &= 56 + 64m \\ &\quad + Q(16 + 32r + 32\log_2 N). \end{aligned}$$

These sizes include all vector count prefixes, even for empty vectors. Here $s(\mathrm{Open}) = s(\mathrm{Opening}) + 16$, including both uint64 split dimensions. Add preceding tuple-field sizes and vector lengths to locate a leaf; the offset before layer t is $t(24 + 64P) + 52t(t-1)$. The grammar has constant nesting depth, so an offset uses a constant number of operations on $O(\log T)$-bit integers. Structural induction gives disjoint intervals covering the body; sorting tagged chunks by offset restores declaration order. □

D Canonical Context and Verification

Field and scalar encoding. The active protocol uses the BN254 scalar field of prime order [27]

$$p = 21888242871839275222246405745257275088 \cdot 10^{39} + 548364400416034343698204186575808495617.$$

It uses the root family $\omega_* = 5^{(p-1)/2^{28}}$ and $\omega_N = \omega_*^{2^{28-n}}$, for $n \le 28$. Require $\omega_*^{2^{28}} = 1$, $\omega_*^{2^{27}} \neq 1$.

$\mathsf{Enc}_{\mathbb{F}}(u)$ is the 32-byte little-endian encoding of the unique integer representative in $[0, p)$. No internal Montgomery limbs are wire values.

Definition D.1 (Public context)**.** With the field and roots above, set

$$\begin{aligned} C &= (a, \mathsf{par}, K, G, x), \\ K &= (C_{\mathrm{r1cs}}, V, I, 3, O, H, R_{\mathrm{ops}}, R_{\mathrm{mem}}), \\ \mathsf{par} &= (\mathsf{id}_{\mathsf{Weir}}, p, 28, \omega_*, 1, \mathsf{BR}, 1, 2, 309, 1, \\ &\qquad \mathsf{HashID}, \mathsf{OracleID}, \mathsf{WireID}). \end{aligned}$$

Here a is the application string, G is from Definition C.1, $x \in \mathbb{F}^I$, $\mathsf{id}_{\mathsf{Weir}} = \texttt{Weir/context/v2}$, and $\mathsf{BR} = \texttt{bit-reversed}$. The remaining entries specify forward sign, rate 1/2, 309 column queries and one degree test.

The identifiers select exactly the maps defined here:

$$\begin{aligned} \mathsf{HashID} &= \texttt{Weir-Merkle-RO256-v1}, \\ \mathsf{OracleID} &= \texttt{Weir-direct-oracles-v1}, \\ \mathsf{WireID} &= \texttt{Weir-canonical-wire-v1}. \end{aligned}$$

The honest indexer constructs K. Reject other parameters, inadmissible shapes or generators. The PCS shapes are Equation (7); verification-relevant application data belongs in a.

Encode $\mathsf{EC}(C)$ in tuple order: canonical scalars, 32-byte digests, and eight-byte unsigned little-endian integers. The modulus uses 32 unsigned little-endian bytes. Strings/vectors have eight-byte counts followed by bytes/encoded elements; lengths must fit. Private state and block size are absent.

Definition D.2 (Full-prefix transcript)**.** For a one-byte tag t, let $\mathsf{fr}(t, b) = t \| \mathsf{nat}(|b|) \| b$. Here $\mathsf{nat}(n)$ is k bytes `ff`, one byte `00`, then the minimal k-byte little-endian encoding of n; $k = 0$ for zero. Initialize

$$W_0 = \mathsf{fr}(0, \texttt{Weir/transcript/v2}) \| \mathsf{fr}(1, \mathsf{EC}(C)).$$

An application event (l, b) appends $\mathsf{fr}(2, l) \| \mathsf{fr}(3, b)$, in the order of Section C.

Use mutually independent public random functions $\mathcal{H}_B : \{0,1\}^* \to \{0,1\}^{256}$, $\mathcal{H}_{\mathbb{F}} : \{0,1\}^* \to \mathbb{F}$ and $\mathcal{H}_{[N]} : \{0,1\}^* \to [N]$ for admitted N, with uniform fresh and consistent repeated answers. Merkle hashing and the header use $\mathcal{H}_B$.

For each scalar request, use a global counter j, initially zero, and occurrence s. Encode it as $\mathsf{E}(s) = \mathsf{nat}(j) \| u \| v$. The one-byte site u is zero outside openings and $1, 2, 3, 4$ at var, deref, ops, mem; role v is zero for ordinary challenges, one for DT, and two for CO. The domain identifier D is byte `00` for $\mathbb{F}$, or `01` followed by $\mathsf{le}_{64}(N)$ for $[N]$. For literal label l, set

$$\begin{aligned} J &= \mathsf{fr}(5, \mathsf{E}(s)) \| \mathsf{fr}(6, l) \| \mathsf{fr}(7, D), \\ k &= \mathsf{fr}(4, W) \| J, \qquad z = \mathcal{H}_D(k). \end{aligned}$$

Append $\mathsf{fr}(8, J)\|\mathsf{fr}(9, \mathsf{E}_D(z))$ and increment j. Here E_D is the canonical scalar or eight-byte unsigned little-endian index encoding. Counters and transcripts never reset. Each DT vector makes r_X field requests in coordinate order; each CO vector makes 309 index requests with replacement. The full word, not a digest, occurs in every key; its resource realization is Sections 3 and 6.

Immediately before each opening's protocol marker and DT, append label `Weir/opening/v1` with the canonical tuple

$$(X, R_X, r_X, m_X, N_X, 1, 309, z_X, y_X).$$

The site uses the same one-byte tag $1, 2, 3, 4$ as $\mathsf{E}(s)$; $z_X \in \mathbb{F}^{e_X}$ and $y_X \in \mathbb{F}$. The verifier derives the root, dimensions and point from the commitment and caller equations. At var, y_X is the supplied `eval_vars_at_ry`; otherwise it is the bottom-binding result. PR precedes PE, and both precede all CO requests, as in Definition 5.3.

Definition D.3 (Canonical proof and verifier)**.** Use Table 11 with these lengths: phase-one/two SC have c cubic/$v+1$ quadratic rounds; a compressed degree-D polynomial has D entries. A (P, D, L) batch has L layers, with t cubic rounds and two P-entry vectors at layer t, plus three D-entry dot vectors. Ops/mem use $(12, 6, o)/(4, 0, h)$. Product read/write and dot-half vectors, and hash-layer address/timestamp/value/deref vectors, have length three; initialization/audit claims are scalars. At site X, split dimensions are $(\log_2 r_X, \log_2 m_X)$, column count is N_X, PE has m_X scalars, PR contains one m_X-entry vector, and the 309 CO records each contain r_X scalars and $\log_2 N_X$ digests. CO derives indices. Encode this tree as $\mathsf{E}_{\mathsf{Weir}}$, using the context's scalar, digest, integer, tuple and count-prefix conventions. Set

$$\begin{aligned} H_{\mathrm{ctx}}(b) &= \mathcal{H}_B(\mathsf{fr}(11, \texttt{Weir/context-hash/v2})\|b), \\ h_C &= H_{\mathrm{ctx}}(\mathsf{fr}(10, \mathsf{EC}(C))), \\ \mathsf{Ser}_{\mathsf{Weir}}(C, \pi) &= \texttt{Weir/proof/v2}\|h_C\|\mathsf{E}_{\mathsf{Weir}}(\pi). \end{aligned}$$

Before challenges validate C, magic, h_C, exact body length and shape-directed parse. Reject wrong counts/dimensions, noncanonical scalars, truncation and trailing bytes; unchecked counts cannot extend loops. Sizes follow Lemma C.14.

Replay Definition C.13. Reconstruct sumchecks by Lemma C.6; check every sumcheck/R1CS endpoint, memory/dot identity, product fold and caller/hash identity when its operands become available. Authenticate every CO column and test

$$\begin{aligned} \sum_i \eta_i C_q[i] &= \mathsf{RS}(p_{\mathrm{rand}})[q], \\ \sum_i E_{z_L}[i] C_q[i] &= \mathsf{RS}(p_{\mathrm{eval}})[q], \\ \sum_t E_{z_R}[t] p_{\mathrm{eval}}[t] &= y_X. \end{aligned}$$

RS encoding pads to length N_X with zeros. Any failed check rejects; only completion accepts.

Lemma D.4 (Canonical parsing and context separation)**.** *Context/body encodings are injective and uniquely decodable. Verification terminates on finite inputs with work polynomial in numeric dimensions and input length. Distinct contexts have distinct corresponding challenge keys; identical accepted bytes across them imply an H_{ctx} collision.*

Proof. Induct on the count-delimited typed grammar for injectivity; shape-bounded advancing loops terminate. Every key includes the full framed context. Equal accepted headers across distinct contexts give distinct framed preimages of the same hash. □

Definition D.5 (Reference and RW provers)**.** Both follow Definition C.13 and $\mathsf{Ser}_{\mathsf{Weir}}$: the reference evaluates finite maps directly, RW uses Definition 5.3 and Section 6. Their shared verifier/context/relation makes malicious verification games identical for fixed coins and oracles; honest provers are absent.

Theorem D.6 (Completeness and exact replacement)**.** *On an admitted satisfying input, common a, G, and common fixed oracles, the reference and RW provers terminate with identical index roots and canonical proof bytes, accepted by V_{Weir}. The physical claim assumes faithful successful stream operations.*

Proof. Preprocessing and PCS correctness give equal roots/context. Induct through events: Lemma 6.1 preserves tables/messages, local PCS correctness preserves collapses/paths, and equal keys give equal responses. Honest algebraic identities establish all checks; deterministic grammar serialization gives identical accepted bytes. Finite loops terminate. Faithful external streams realize these values; publication and last-use reclamation preserve the invariant. □

E Security Proof

We first derive pointwise algebraic error events under sequential independent verifier challenges. Section E.1 then accounts for adaptive public-oracle queries.

Lemma E.1 (Pointwise algebraic error bounds)**.** *Challenges are fresh, independent and uniform, drawn after the preceding messages in the prescribed order. Use κ_{net} from Definition 7.1.*

(a) Sumcheck. *Fix the tables and degree-D expression of Definition C.5. With a false initial claim, arbitrary adaptive round polynomials of degree at most D and the prescribed endpoint sums, the probability of a true terminal claim is at most* $\min(1, kD/p)$.

(b) R1CS. *Fix an unsatisfying assignment before the first challenge. Against arbitrary adaptive messages, with exact terminal access to its witness and instance tables, the two-phase verifier accepts with probability at most* $\min(1, (4c + 1 + 2(v + 1))/p)$.

(c) Memory. *Fix one direction's honest addresses, timestamps and audit counts, an arbitrary memory $\mu \in \mathbb{F}^H$, and arbitrary read tables $\delta_X \in \mathbb{F}^O$, before β, γ. Compress each initialization, audit, read and write triple (a, v, t) as $a + \beta v + \beta^2 t - \gamma$. If any read is false, its product identity holds with probability at most* $\min(1, 2(H + 3O)/p)$.

(d) Product/dot. *Fix the inputs of Definition C.7, with $D = 2q$ dot triples paired as halves of fixed full tables. An adversary supplies root and dot-sum claims before the first batching challenge, then arbitrary typed messages. If some initial claim is false, the probability that all checks pass and all final product and recombined dot claims are true is at most* $\min(1, \kappa_{\mathrm{net}}(L)/p)$. *For $D = 0$, the dot conditions are empty.*

(e) Callers. *Fix $U_j \in \mathbb{F}^{2^t}$, $Z = U_0 \| \cdots \| U_{2^s-1}$, $r \in \mathbb{F}^t$, and a claimed block vector e before $a \in \mathbb{F}^s$. If some $e_j \neq U_j(r)$, then* $\Pr[\widetilde{e}(a) = \widetilde{Z}(a\|r)] \leq \min(1, s/p)$. *The three callers contribute at most $8/p$ when their joint claims are checked against the fixed tables.*

Proof. (a) At a first false-to-true transition, the claimed polynomial and true partial sum have different endpoint sums. Their difference is nonzero and has at most D roots. Condition on the preceding history, then sum over the k possible transitions.

Challenge	Fixed before exposure	Nonzero error test	Total charge (in units of $1/p$)
R1CS τ	Assignment/residual table	Multilinear residual, degree c	c
Phase-one rounds	Entering claim and round polynomial	Claimed minus true partial sum, degree 3	$3c$
ξ	Reported terminal error vector	Random linear cancellation	1
Phase-two rounds	Entering claim and round polynomial	Claimed minus true partial sum, degree 2	$2(v+1)$
(β, γ)	First false memory direction and read tables	Compressed-product difference	$2(H+3O)$
Product layer t	Pending product/dot errors	Batch cancellation, cubic repair, child fold	$1+3t+1$
Caller selector	Claimed block errors	Multilinear error extension	$3+4+1$

Table 12: Pointwise algebraic events in Lemma E.1(b)–(e). History fixes the applicable error before each listed challenge; the set is empty when that error is zero. Summing product layers $t = 0, \ldots, L-1$ gives $\kappa_{\text{net}}(L)$; ops and mem contribute at $L = o, h$. First-exposure charging multiplies the combined E_{alg}/p by K_C in the ROM argument.

(b) By the polynomial root bound (induction from the univariate bound), the nonzero degree-c residual vanishes at τ with probability at most c/p. Otherwise phase one starts false; its cubic rounds contribute $3c/p$. A passing unrepaired terminal check entails a nonzero reported error vector in $(a_X)_X$, fixed before ξ. Uniform batching cancels it with probability $1/p$; otherwise phase two starts false and its quadratic rounds contribute $2(v+1)/p$. Exact terminal access forces rejection outside this union.

(c) Unique factorization in $\mathbb{F}[B, G]$ equates the products exactly when their triples match: factors $a + Bv + B^2 t - G$ are monic in $-G$, and $H, 3O < p$ gives injective embeddings. At an address with $s > 0$ accesses, timestamp zero matches initialization to the first read, $1, \ldots, s-1$ match each write to the next read, and s matches the last write to audit. For $s = 0$, initialization matches audit. Thus a false read makes the product difference nonzero, of degree at most $2(H+3O)$. Apply the root bound without canceling sampled factors.

(d) Pending errors are $c_{j,t} - \widetilde{V}_{j,t}(r_t)$ and dot-sum errors; a false dot sum persists until the last layer. At a first repair, the batching error vector is nonzero. Conditional on the preceding history, its cancellation costs $1/p$; otherwise cubic sumcheck repair costs $3t/p$. If both are avoided and the terminal check passes, a child pair has nonzero error (e_0, e_1). Choose its first occurrence before ζ_t. Its folded error $(1-\zeta_t)e_0 + \zeta_t e_1$ has at most one root, contributing $1/p$, including for paired dot halves at the final layer. Starting false and ending entirely true entails such a repair; summing $(2+3t)/p$ over $t < L$ gives $\kappa_{\text{net}}(L)/p$. All bounds use preceding histories, with no future-acceptance conditioning.

(e) By Lemma C.10, the difference is the multilinear extension of a nonzero error vector, of degree at most s. Apply the root bound at each caller's preceding history and sum for $s = 3, 4, 1$. □

Definition E.2 (Registered-table experiment)**.** Ops/mem are honestly indexed. The adversary fixes x and registers $w^+ \in \mathbb{F}^V$ before τ (ξ at $c = 0$); after r_x, r_y and sparse claims, it registers $Z_{\text{deref}} \in \mathbb{F}^{8O}$ before (β, γ). Its first six blocks are reads $\delta_{\tilde{d}}^X$; padding need not vanish. Registration fixes the whole table. Replace each PCS by exact $y_X = Z_X(z_X)$, omitting DT/CO/authentication; keep algebraic checks and fresh uniform challenges in order, rejecting malformed shapes. The sparse suffix starts from any fixed preceding history's points, claims and registered reads.

Theorem E.3 (Registered-table soundness)**.** (a) *If some entry sparse claim is false, the suffix accepts with probability at most* $\epsilon_{\text{sparse}} = \min(1, E_{\text{sparse}}/p)$*, where*

$$E_{\text{sparse}} = 2(H + 3O) + \kappa_{\text{net}}(o) + \kappa_{\text{net}}(h) + 8.$$

This bound holds uniformly over the registered-table entry prefixes. (b) *In Definition E.2, acceptance with an unsatisfying registered witness has probability at most* $\epsilon_{\text{alg}} = \min(1, E_{\text{alg}}/p)$*, with* E_{alg} *as in Definition 7.1.*

Proof. (a) Fix memories and reads before compression. If a read is false, select the first false direction before (β, γ); Lemma E.1(c) charges $2(H + 3O)/p$ once. Outside that event, a passing root identity entails a false ops/mem root claim. If all reads are true, a false sparse claim entails a false dot-half claim by Equation (13).

At each caller, tables, suffix point and block claims precede the prefix challenge. Outside the $8/p$ caller events, exact joint openings make every block evaluation true, including deref padding. Affine hash identities and dot recombination then make the final compression/product claims true even for false registered reads. Acceptance must therefore repair a false ops/mem batch input. Apply Lemma E.1(d) to both batches, with the mem bound valid after every ops history. Acceptance lies in this compression/caller/batch union, giving the sum.

(b) The witness precedes the first challenge. Acceptance with true sparse claims entails a root/repair event of Lemma E.1(b); otherwise its false sparse entry claim invokes part (a) at that preceding history. Their union has coefficient E_{alg}. □

E.1 Adaptive knowledge soundness

We prove Theorem 7.2 under Definition 7.1's experiment and budgets.

Lemma E.4 (Challenge inventory)**.** *A complete verification has*

$$J_{\text{ver}} = 2c + v + 13o + 5h + \frac{o(o-1) + h(h-1)}{2} + \sum_X r_X + 1256. \tag{23}$$

Its block order is τ, phase-one rounds, ξ, phase-two rounds, var DT/CO, (β, γ), ops layers, mem layers, then (caller prefix, DT, CO) at deref, ops, mem. A layer orders batching, sumcheck rounds, then folding.

Proof. R1CS/compression contribute $2c+v+6$. A (P, D, L) batch contributes $PL+D+L(L-1)/2+L$. Substitute ops $(12, 6, o)$ and mem $(4, 0, h)$, then add eight caller and $\sum_X r_X + 4 \cdot 309$ opening requests. Wrappers add none; failed checks truncate. Ordinary requests use $(0, 0)$, and site i's DT/CO use $(i, 1), (i, 2)$. □

Lemma E.5 (Online virtual matrix)**.** *Except on an event* Bad_B *of probability at most* ϵ_B*, simultaneously at every monitored commitment cut* U *with root* R *and shape* (r, N)*, the current hash log determines a fixed matrix* M_U*. Every later authenticated column for that root and shape equals the corresponding column of* M_U*. With indexed hash-log lookup, constructing it costs* $O(rN + N)$ *record work.*

Proof. Target every 32-byte window of submitted strings and every earlier hash output, including the current fresh query's input windows. Let Bad_B occur when a fresh answer hits a target. Conditional uniformity and at most $L_B + q_B$ targets per query give ϵ_B, covering collisions and later preimages of mentioned digests.

At a cut, invert the existing log (earliest entry per output) and unfold R by position to height $\log_2 N$. Internal inputs must be ordered 64-byte child pairs; leaf inputs must be 32 zero bytes followed by r canonical scalars. Fill unresolved/malformed branches with zero columns. Outside Bad_B, every later root preimage was already logged, since its digest was in the exposing key; uniqueness fixes it. Repeat for its already-mentioned children to bind every authenticated column. Unresolved branches cannot gain valid paths. Fixed heights handle overlapping leaf/node lengths when $r = 1$. The matrix need not hash to R; honestly indexed trees are already logged and the same argument binds their actual matrices. □

Snapshot decoding. For each site let $m = N/2$, $t = N/4$, and $\mathcal{C}$ be its ordered RS row code. Matrix distance counts columns differing in at least one row. A matrix is *near* if it is within t columns of a matrix with every row in $\mathcal{C}$.

Lemma E.6 (Deterministic snapshot decoding)**.** *A deterministic* $O(rN^3)$*-field-operation algorithm returns the unique near coefficient matrix, or* $\perp$ *if none exists.*

Proof. Distance $N - m + 1 > 2t$ gives uniqueness. For each row u, solve $Q_0(\alpha_j) = u_j E_0(\alpha_j)$ for all $j < N$, with $\deg Q_0 < m + t$ and monic degree-t locator E_0. A near codeword P supplies a locator extended to degree t; every solution satisfies $Q_0 = PE_0$, since their difference has degree below $m + t$ and at least $N - t = m + t$ roots. Reject if the system has no solution, division fails, the quotient has degree at least m, or the union of row mismatch sets exceeds t. Deterministic Gaussian elimination in r systems of size $O(N)$ costs $O(rN^3)$. □

Lemma E.7 (Imported RS proximity bound)**.** *For a fixed matrix* M *that is not near,*

$$\Pr_{\eta \leftarrow \mathbb{F}^r}[\mathrm{dist}(\eta^\top M, \mathcal{C}) \le N/4] \le N/p.$$

Proof. In [8, Theorem 3.1], set $(k, n, \ell, q) = (m - 1, N, r, p)$. Dimension $k + 1 = m$ permits radius $(1 - m/N)/2 = 1/4$; order-N roots provide distinct evaluation points. The source's relative column distance $> 1/4$ matches our integer distance $> N/4$. Bit reversal preserves this distance. □

Exposure of challenge blocks. A block is a sequence of requests with no intervening prover choice: DT, CO, τ, a batching/caller vector, or (β, γ); each sumcheck or fold request is a one-coordinate block. Fixed wrappers between coordinates belong to the block.

Definition E.8 (Structural prefix monitor)**.** For each field/index query (D, k), recognize canonical $k = \mathsf{fr}(4, W) \| J$ with no trailing bytes and an admitted context matching the game's a, par, K, G and domain. Parse Definition C.13's literal labels, event order, payload types and lengths, and request counter, site, role and domain; the current descriptor must be next and embedded responses well typed. Compare counts with available bytes and shapes before consuming them.

The monitor makes no oracle calls and performs no algebraic, authentication, header or oracle-consistency checks. Unrecognized keys receive normal, charged answers and expose no prefix. For every nonempty block begun in the word, report its position ι and exact starting prefix P_ι, including an unanswered current block, plus complete var/deref commitment cuts. Cache by position and entire prefix bytes; never parse payload substrings as transcripts.

Lemma E.9 (Recognition and causality)**.** *Every key on an accepting execution is recognized. A query exposes at most one prefix per fixed block or commitment position; exposing a descendant also exposes every ancestor cut. Thus first-exposure indices satisfy $T(U) \le T(P)$ for ancestors U of P. Arbitrary well-typed internal responses preserve recognition of subsequent keys within a block.*

Proof. Canonical framing, shapes and the counter uniquely parse the accepting verifier's full word, including every identical ancestor cut. Within a block no prover choice intervenes: each next wrapper and descriptor is determined for every typed response. Recognition does not test oracle consistency. □

Lemma E.10 (First-exposure bound)**.** *Fix a block position. At first exposure of each starting prefix P, choose a bad-response set B_P, measurable before answering the exposing query, of product-uniform measure at most e. The probability that final verification follows such a block with responses in its chosen set is at most $K_C e$.*

Proof. For fixed P, block keys form a response tree distinguished by full prefixes and counters. Before first exposure no such oracle/input pair was queried: it would already expose P by Lemma E.9. Another oracle's identical input reveals none of these values.

Let $\mathcal{F}_i^-$ contain prior queries/answers and the current submitted query with deterministic monitoring, before its answer. First exposure, P and B_P are measurable there. Conditional on this history, unqueried tree values are independent uniform; the path following actual oracle responses has product distribution and hits B_P with probability at most e. This is the actual path even when a descendant exposes guessed responses.

Later selection or abandonment can only select this fixed-path event. Charge it to first exposure, at most one prefix for this position per chronological query. The tower property and union bound over at most K_C queries, including final verification, give $K_C e$. There is no conditioning on acceptance or later hash events. □

Extractor and freezing. Lazily sample real oracles and run the adversary once. Before each challenge answer, process recognized cuts in temporal order: construct and decode each unseen var/deref snapshot from the current hash log and cache by the entire cut. Failure stores a far flag and zero table. Malformed keys and keys from other contexts receive normal charged answers; monitor final verification likewise. If final verification accepts, return the cached var table keyed by that verification's entire var cut, in row-major order. Return a zero witness if verification fails or has no admissible var cut. Length V is admitted by the relation.

Lemma E.11 (Causal freezing)**.** *All ancestor commitment snapshots used for a block's error set are fixed before its exposing query is answered, and reused at every descendant on the same path. Var is frozen by first exposure of τ when $c > 0$; deref is frozen by first exposure of (β, γ).*

Proof. All ancestors are reported before the answer. Their snapshots are existing cached values or deterministic functions of the current hash log, hence measurable in $\mathcal{F}_i^-$, including guessed histories. Descendants reuse identical cut bytes. At $c = 0$, var freezes at ξ; the omitted residual test is a nonzero constant and contributes no random error. Ops/mem are honestly indexed. □

Lemma E.12 (PCS errors on all exposed prefixes)**.** *Use cached matrices at every recognized DT/CO prefix, including prefixes with guessed history. Define these bad sets:*

1. *At DT, if the snapshot is far use* $\{\eta : \mathrm{dist}(\eta^\top M, \mathcal{C}) \le N/4\}$*; otherwise use the empty set.*
2. *At CO, test the prefix-determined scalar equality*

$$F_P : \quad \sum_t E_{z_R}[t]\, p_{\mathrm{eval}}[t] = y.$$

If it fails, use the empty set. Otherwise, for a far snapshot and nonexceptional displayed DT, let S_P *be the agreement set of* $\eta^\top M$ *and* $\mathsf{RS}(p_{\mathrm{rand}})$*. For a near snapshot decoded to* A *and false claim* $y \neq \widetilde{A}(z_L \| z_R)$*, use the agreement set of* $E_{z_L}^\top M$ *and* $\mathsf{RS}(p_{\mathrm{eval}})$*. Use the empty set in all other cases. The CO bad set is* S_P^{309}*.*

DT measure is at most N/p*; CO product measure is at most* $(3/4)^{309}$*. For an indexed site use only the near/false case with its known matrix: DT is empty and CO measure is at most* $\vartheta_X = ((m_X - 1)/N_X)^{309}$*. Outside* Bad_B*, acceptance with a far snapshot or false decoded/indexed evaluation entails one of these events.*

Proof. At DT the snapshot alone precedes the answer and fixes its bad set; η is the sampled response, not prior data. At CO, the snapshot and displayed $z, y, \eta, \mathrm{PR}, \mathrm{PE}$ all precede the answer and fix its bad set. The proximity lemma bounds DT. A far nonexceptional collapse agrees with any PR codeword at fewer than $3N/4$ columns. For a near false claim, F_P forces PE to differ from the decoded collapse; its nonzero degree-$< m$ difference has at most $m - 1$ roots, plus $N/4$ erroneous columns. Thus $|S_P| < 3N/4$; indexed matrices have only the $m - 1$ roots.

On acceptance F_P holds, and outside Bad_B columns match snapshots. A far snapshot entails exceptional DT or every CO index in its PR agreement set; a near false claim entails every CO index in its PE agreement set. This proves inclusion for every typed history, without independence assumptions about PR/PE or sites. □

Proof of Theorem 7.2. Lazy sampling preserves the game; monitoring adds no messages/queries and changes no answers. Include Bad_B in the failure union. Outside it, authenticated columns match cached var/deref and honest ops/mem. By Lemmas E.10 and E.12, the PCS union costs

$$K_C[(N_{\mathrm{var}} + N_{\mathrm{deref}})/p + 2(3/4)^{309} + \vartheta_{\mathrm{ops}} + \vartheta_{\mathrm{mem}}].$$

Outside this union, accepted openings give true table evaluations. For a false extracted witness, use the pointwise algebraic events of Lemma E.1(b)–(e). At each prefix, history and frozen tables fix the entering claims: choose the applicable nonzero error polynomial's zero set, or the empty set. Accepted sumchecks enforce endpoint sums; choose the first false memory direction before compression, and retain dot errors until the final-layer batch. The event coefficients sum to E_{alg}. By Lemma E.11, these sets precede exposure even for guessed histories. Exposure charging gives $K_C E_{\mathrm{alg}}/p$, with no conditioning on later openings or acceptance. Adding the hash and PCS terms proves the joint-failure bound for adaptive x bound in the context. The common verifier gives the RW guarantee; if no witness exists, every extracted assignment is false. □

Extraction cost. At most K_C snapshots are decoded per malicious site. Prefix and target-window processing is quadratic in the charged byte/query budgets; indexed hash-log lookup is constant-depth. With Lemma E.6, additional work is bounded by

$$O\left(L_B^2 + q_B^2 + K_C \sum_{X \in \{\mathsf{var},\mathsf{deref}\}} r_X N_X^3\right).$$

The word size is $O(\log(T + L_B + q_B + 2))$ bits, so bit-cost accounting also remains polynomial.

F Query Costs and Parameter Sensitivity

Key-byte totals and proof sizes match all 192 primary processes; maxima and alternative Q profiles are calculated (implemented $Q = 309$).

$\log_2 q_C$	$\log_2 q_B$	$\log_2 L_B$	$Q = 309$	388	427
16	40	64	111.202	143.978	151.995
32	40	64	95.247	128.035	144.214
32	64	80	95.247	112.000	112.000
32	80	96	80.000	80.000	80.000
48	40	64	79.247	112.035	128.221

Table 13: Full-bound exponents $-\log_2 \epsilon(Q)$ at $(C, V, O) = (4096, 262144, 32768)$ for the displayed hypothetical budgets and candidate Q.

F.1 Query Bytes

Table 14 groups 12 inputs with one public scalar and a 461-byte context. Structured allocator-peak medians are 0.822–0.829 MB. The field/index-key counter excludes Merkle inputs and rejection traversals, counting part of $\mathcal{L}_{\text{oracle}}$. Poseidon 4's four CO blocks contribute 91.6% of key bytes. Accumulated prefixes make mem's key longest. Fusion preserves keys; snapshot changes their storage (Section B.4).

Input	Max key MB	Query GB	Proof MB
Matrix 8	0.667	0.544	1.150
Matrix 16	1.600	1.275	2.406
Poseidon 2	1.367	1.058	2.035
Poseidon 4	1.828	1.527	2.696
Balanced	0.448	0.348	0.846
Dense	0.743	0.573	1.225
Legacy ratio	0.723	0.575	1.294

Table 14: Primary query/proof costs (192 matched processes). Maxima are reconstructed; balanced/dense group four fixtures/three seeds of identical lengths. MB/GB are decimal.

For $C = O = V = 2^v$, $8 \le v \le 41$, $I = V - 1$, take V explicit zeros per matrix: every public input admits an empty private witness. The context has $32I$ input bytes and $r_{\text{var}} = 2^{\lceil v/2 \rceil - 4}$ var DT requests. Hence $\mathcal{L}_{\text{oracle}} \ge r_{\text{var}} |\mathsf{EC}(C)| \ge 32 r_{\text{var}}(V - 1)$, growing as $V^{3/2}$ over admitted scales at resource size $T = 6V + O(1)$ for fixed-size application context a. At $V = 2^{20}$, var DT context copies alone exceed 2.147 GB; Theorem 6.2 charges these bytes separately.

F.2 Full-Bound and Proof-Cost Sensitivity

For candidate Q, set $J(Q) = J(309) + 4(Q - 309)$ and $K_C(Q) = q_C + J(Q)$. The full bound is

$$\epsilon(Q) = \min\{1, \epsilon_B + K_C(Q)[(E_{\text{alg}} + N_{\text{var}} + N_{\text{deref}})/p + 2(3/4)^Q + \theta^Q_{\text{ops}} + \theta^Q_{\text{mem}}]\},$$

with $\theta_X = (m_X - 1)/N_X$. Table 13 uses exact rationals and Definition 7.1's budgets. At 309 this is the theorem; other Q require matching parameters, parser, sampling, exposure monitor and proof vectors. Hash budgets impose a floor that increasing Q cannot remove. These are ideal-oracle bounds, not concrete security or attack costs.

By the grammar, $S(Q) - S(309) = (Q - 309)\sum_X(16 + 32r_X + 32\log_2 N_X)$. At Table 13's shape, each extra sample adds 8032 bytes: 388 adds 634,528 bytes (12.31%) and 316 requests. Recomputing equality weights/claims per occurrence costs $Q\sum_X \log_2 N_X$ parent hashes, $2Q\sum_X m_X$ Horner steps, leaf hashes and row dot products: algebraic PCS work $O(Q\sum_X(r_X + m_X))$, plus authentication/transcript work. Candidate latency is unmeasured.