leanVM

Contents

1 Introduction **4**

2 VM specification **4**

3 Proving primitives **6**

4 Committing the witness **9**
- 4.1 Multilinear stacking . . . 9
- 4.2 Committing to booleans via Ring-Switching . . . 9

5 Arithmetization framework **9**
- 5.1 M3 model . . . 10
- 5.2 Balancing the bus: grand product . . . 10
- 5.3 Proving the grand product . . . 11
  - 5.3.1 Grand-Product argument via GKR . . . 11
  - 5.3.2 Two layers at a time . . . 11
  - 5.3.3 Batching several grand-products . . . 12
- 5.4 Stacking the leaves . . . 12
- 5.5 Table sumcheck . . . 13
- 5.6 Example: proving elliptic curve multiplication . . . 14
  - 5.6.1 Statement . . . 14
  - 5.6.2 Arithmetization . . . 14
  - 5.6.3 The interactive protocol, unrolled . . . 15

6 Bus interactions **16**
- 6.1 VM State . . . 16
- 6.2 The lookup protocol . . . 17
- 6.3 Memory . . . 19
- 6.4 Bytecode . . . 19
- 6.5 The index column . . . 19

7 The instruction tables **19**
- 7.1 `XOR` . . . 19
- 7.2 `MUL_NATIVE` . . . 20
- 7.3 `SET_CONSTANT` . . . 20
- 7.4 `DEREF` . . . 20
- 7.5 `JUMP` . . . 21
- 7.6 `KECCAK` . . . 21

8 End-to-end protocol **22**
8.1 Bytecode encoding . . . 22
8.2 Public input . . . 23
8.3 Padding rows . . . 23
8.4 Fiat Shamir instantiation . . . 24
8.5 The unrolled protocol . . . 24

9 ISA programming **27**
9.1 Hints . . . 27
9.2 Division and inequality . . . 27
9.3 Functions . . . 27
9.4 Loops . . . 28
9.5 Conditionals . . . 28
9.6 Checking that a value is in the subfield K . . . 28
9.7 Range checks . . . 28
9.8 Hashing with Keccak . . . 29
9.9 Match statements . . . 29

10 Ethereum use cases **29**
10.1 Recursive aggregation . . . 29
10.1.1 The public input . . . 29
10.1.2 Checking coverage . . . 30
10.1.3 Deferred evaluation claims . . . 30
10.2 Data availability: LeanDA . . . 30
10.2.1 The commitment . . . 31
10.2.2 Codeword membership . . . 32

A Ring switching **32**
A.1 Setting . . . 32
A.2 The reduction . . . 33
A.3 Which maps are sound . . . 33
A.4 The map we draw . . . 34
A.5 Evaluating the weight . . . 35
A.6 Cost . . . 35
A.7 Credits . . . 35

B The polynomial commitment scheme: WHIR /Ligerito **36**
B.1 Preliminaries . . . 36
B.1.1 Codes, interleaving, and list decoding . . . 36
B.1.2 PCS and round-by-round soundness . . . 37
B.2 Reed–Solomon codes over binary fields . . . 37
B.3 The protocol . . . 38
B.4 Round-by-round soundness . . . 39
B.4.1 Proximity tools . . . 39
B.4.2 Proof of the main theorem . . . 41
B.5 Optimizations . . . 43
B.6 The Johnson bound . . . 43

C Proving Keccak by GKR **44**
C.1 Credits . . . 44
C.2 Overview . . . 44
C.3 Keccak-f[1600] . . . 44
C.4 Packing . . . 44
C.5 Example: chi on one layer . . . 45
C.6 The protocol . . . 46
C.7 Committing the middle state . . . 47
C.8 Soundness . . . 48
C.9 Proof of the embedding lemma . . . 48

D Novel polynomial basis and additive NTT **49**
D.1 Additive domains and subspace polynomials . . . 49
D.2 Novel basis and Reed-Solomon encoding . . . 49
D.3 The additive NTT . . . 50
D.4 Column weights . . . 51
D.5 Duality and codeword membership . . . 51

1 Introduction

Ethereum's consensus layer currently uses BLS signatures, its execution layer ECDSA, and its data availability layer KZG. All three are based on elliptic curves, which do not survive a quantum computer [Sho97]. Hash-based cryptography, both signatures and snarks (the hash-based ones are sometimes called STARKs [BSBHR18]), is a promising candidate for Ethereum's post-quantum transition:

- BLS → XMSS [HBG+18] (using each Ethereum slot as the nonce, statefulness is no longer an issue when attesting: double signing is already a slashable offense). But XMSS has no native aggregation, unlike BLS, so we need a snark to handle it.
- ECDSA → SPHINCS+ [BHK+19]. Drawback: dozens of times bigger. Again, snark-based aggregation is a natural solution to mitigate signature size increase.
- KZG → Reed–Solomon encode each blob, and provide the Merkle root of the codeword. A snark guarantees that the root commits to a correct encoding.

leanVM is a candidate as the snark machinery required above. It's a verifiable virtual machine (commonly a zkVM, for "zero-knowledge virtual machine", though leanVM is not (yet) zk), with a minimalistic ISA (inspired by Cairo [GPR21]). It is based on binary fields, inspired by [DP26], currently using Keccak-256 [BDPV11], Ethereum's hash function.

2 VM specification

Define the following two finite fields:

$$K = \mathbb{F}_{2^{64}} = \mathbb{F}_2[x]/(x^{64} + x^4 + x^3 + x + 1), \qquad E = K[y]/(y^3 + y + 1), \qquad |E| = 2^{192}.$$

And fix a generator g of the multiplicative group $K^\times$, of order $2^{64} - 1$.

An element of K is a polynomial in x of degree below 64, written in hex with the coefficient of x^i in bit i, so $\mathsf{2}$ is x and $\mathsf{3}$ is $x + 1$. An element of E is three of those, its coefficients on $1, y, y^2$.

LeanVM ISA, also known as leanISA, is inspired by [GPR21]. The Virtual Machine takes as input a bytecode (padded to a power of two, $N_{\text{prog}} = 2^{\kappa_{\text{bc}}}$) and some public input $\mathsf{input} \in \{0,1\}^{256} \simeq (\mathsf{input}_0, \ldots, \mathsf{input}_3) \in (\{0,1\}^{64})^4$. The Virtual Machine then initializes its two K-valued registers:

- the program counter $\mathbf{pc} \leftarrow 1_K$
- the frame pointer $\mathbf{fp} \leftarrow 1_K$

Then, initialize the write-once memory: a buffer, of size $N_{\text{mem}} = 2^{\kappa_{\text{mem}}}$, containing E-valued *memory words* (192 bits). The memory is indexed *in the exponent* of g (64 bits). The first memory word is $\mathbf{m}[\mathsf{g}^0] \in E$, the second is $\mathbf{m}[\mathsf{g}^1]$, the third is $\mathbf{m}[\mathsf{g}^2]$, etc. The bytecode addressing follows the same convention. Each memory access beyond $\mathsf{g}^{N_{\text{mem}}-1}$ (resp. $\mathsf{g}^{N_{\text{prog}}-1}$ for the bytecode) is invalid.

The public input fixes the first two memory words $\mathbf{m}[\mathsf{g}^0] = \mathsf{input}_0 + \mathsf{input}_1 \cdot y \in E$ and $\mathbf{m}[\mathsf{g}^1] = \mathsf{input}_2 + \mathsf{input}_3 \cdot y \in E$.

The Virtual Machine execution loop is the following:

1. fetch the instruction inst at address $\mathbf{pc}$.
2. execute inst:

- assert the memory constraints (e.g. $\mathbf{m}[\mathbf{fp} \cdot o_C] = \mathbf{m}[\mathbf{fp} \cdot o_A] + \mathbf{m}[\mathbf{fp} \cdot o_B]$ for XOR)
- update **pc** and **fp** (all instructions except `JUMP` set $\mathbf{pc} \leftarrow \mathbf{pc} \cdot \mathsf{g}$, advancing to the next instruction, and leave $\mathbf{fp} \leftarrow \mathbf{fp}$ unchanged)

3. if $(\mathbf{pc}, \mathbf{fp})$ equals the final state $(\mathbf{pc}_{\mathsf{final}}, \mathbf{fp}_{\mathsf{final}}) := (\mathsf{g}^{N_{\text{prog}}-1}, \mathsf{g}^0)$, exit the loop (successful execution). Otherwise go back to 1.

Write-once memory The first time a memory word is set defines its value for the rest of the execution, i.e. it cannot be modified again in the future.

When proving the VM execution via a snark, the memory is committed (by an untrusted prover) at the beginning of the execution (with all the values already filled): every instruction then simply consists in reading a bunch of memory cells, and asserting constraints on them (*write-once memory* is also sometimes called *read-only memory*).

Non determinism For a given pair of bytecode and public input, there is more than one valid execution.

In particular, the memory log-size κ_{mem} is a non deterministic parameter, freely chosen at the beginning of execution, with the constraint $16 \leq \kappa_{\text{mem}} \leq 32$.

Operands and addressing. An instruction is an opcode followed by a list of K-valued operands. An operand is either an *fp-relative reference*, an offset j into the current frame encoded as the g-power $o = \mathsf{g}^j$, or one K-lane of an *immediate* (k_0, k_1, k_2 in `SET_CONSTANT`). With the frame based at $\mathbf{fp} = \mathsf{g}^{\text{base}}$, a reference names the cell

$$\mathbf{m}[\mathbf{fp} \cdot o], \qquad \mathbf{fp} \cdot o \;=\; \mathsf{g}^{\text{base}} \cdot \mathsf{g}^{\,j} \;=\; \mathsf{g}^{\,\text{base}+j}.$$

Instruction set. The machine implements six instructions.

Instruction	Operands	Semantics
`XOR`	$[o_A, o_B, o_C]$	$\mathbf{m}[\mathbf{fp} \cdot o_C] = \mathbf{m}[\mathbf{fp} \cdot o_A] + \mathbf{m}[\mathbf{fp} \cdot o_B]$ (addition in E)
`MUL_NATIVE`	$[o_A, o_B, o_C]$	$\mathbf{m}[\mathbf{fp} \cdot o_C] = \mathbf{m}[\mathbf{fp} \cdot o_A] \cdot \mathbf{m}[\mathbf{fp} \cdot o_B]$ (multiplication in E)
`SET_CONSTANT`	$[o, k_0, k_1, k_2]$	$\mathbf{m}[\mathbf{fp} \cdot o] = k_0 + k_1 y + k_2 y^2$
`DEREF`	$[o_1, o_2, o_3;\ \mathsf{mode}]$	$\mathbf{m}[\mathbf{m}[\mathbf{fp} \cdot o_1] \cdot o_2] = \mathsf{src}(\mathsf{mode})$ (see below)
`JUMP`	$[o_c, o_d, o_f]$	conditional jump (see below)
`KECCAK`	$[o_{m_0}, o_{m_1}, o_{m_2}, o_{m_3}, o_{tail}, o_{b8}, o_{cap}, o_{out}, odd_{in}, odd_{out}]$	one Keccak permutation (see below)

`DEREF` stores, at the dereferenced address $\mathbf{m}[\mathbf{fp} \cdot o_1] \cdot o_2$, a value chosen by a *store mode* $\mathsf{mode} \in \{\texttt{deref_cell}[o_3], \texttt{deref_pc}, \texttt{deref_fp}\}$:

$$\mathsf{src}(\mathsf{mode}) \;=\; \begin{cases} \mathbf{m}[\mathbf{fp} \cdot o_3] & \mathsf{mode} = \texttt{deref_cell}[o_3] \quad \text{(the local cell at offset } o_3 \in K\text{)}, \\ \mathsf{g}^2 \cdot \mathbf{pc} & \mathsf{mode} = \texttt{deref_pc} \quad \text{(the current program counter advanced by two)}, \\ \mathbf{fp} & \mathsf{mode} = \texttt{deref_fp} \quad \text{(the current frame pointer)}. \end{cases}$$

Additionally, it asserts $\mathbf{m}[\mathbf{fp} \cdot o_1] \in K$.

`JUMP` reads a condition $c = \mathbf{m}[\mathbf{fp} \cdot o_c]$, and branches on whether it is zero. When $c \neq 0$ it transfers control, $\mathbf{pc} \leftarrow \mathbf{m}[\mathbf{fp} \cdot o_d]$ and $\mathbf{fp} \leftarrow \mathbf{m}[\mathbf{fp} \cdot o_f]$; when $c = 0$ it defaults to $\mathbf{pc} \leftarrow \mathbf{g} \cdot \mathbf{pc}$ and $\mathbf{fp} \leftarrow \mathbf{fp}$. Additionally, it asserts $\mathbf{m}[\mathbf{fp} \cdot o_c], \mathbf{m}[\mathbf{fp} \cdot o_f], \mathbf{m}[\mathbf{fp} \cdot o_d] \in K$ (unconditionally).

`KECCAK` applies the Keccak-$f[1600]$ permutation [BDPV11] to a state of 25 lanes of 64 bits held in thirteen cells, two lanes per cell, top limb zero. It reads the cells $\mathbf{m}[\mathbf{fp} \cdot o_{m_0}], \ldots, \mathbf{m}[\mathbf{fp} \cdot o_{m_3}]$, $\mathbf{m}[\mathbf{fp} \cdot \mathbf{g}^k \cdot o_{tail}]$ $(k < 4)$, $\mathbf{m}[\mathbf{fp} \cdot o_{b8}]$ and $\mathbf{m}[\mathbf{fp} \cdot \mathbf{g}^k \cdot o_{cap}]$ $(k < 4)$, in this order, and writes the result to $\mathbf{m}[\mathbf{fp} \cdot \mathbf{g}^k \cdot o_{out}]$ $(k < 13)$. Cells are read in the layout $odd_{in} \in \{0, 1\}$ and written in the layout odd_{out}:

- even (0): cell $c < 8$ holds lanes $(2c, 2c + 1)$, cell 8 holds $(16, \cdot)$;
- odd (1): cell 0 holds $(\cdot, 0)$, cell $1 \leq c \leq 8$ holds $(2c - 1, 2c)$;
- both: cell $9 \leq c < 13$ holds $(2c - 1, 2c)$.

The 64-bit of the cell 8 (even) or 0 (odd) (denoted $\cdot$ above) is ignored for the input and set to zero for the output.

3 Proving primitives

Definition 3.1 (Iverson bracket)**.** For a proposition P, $[P] = 1$ if P holds and $[P] = 0$ otherwise.

Definition 3.2 (Multiset)**.** A *multiset* $\{\{\cdots\}\}$ over a set S is a multiplicity function $S \to \mathbb{N}$; $\uplus$ adds multiplicities.

Definition 3.3 (Multilinear extension)**.** A *multilinear* is given by its 2^κ values on the boolean hypercube, a function $f : \{0,1\}^\kappa \to E$. Its *multilinear extension* $\widetilde{f} : E^\kappa \to E$ is the unique polynomial of degree at most one in each variable agreeing with f on $\{0,1\}^\kappa$; we identify the two freely.

Definition 3.4 (Equality indicator)**.** For $X, Y \in E^\kappa$,

$$\mathrm{eq}(X, Y) = \prod_j (X_j Y_j + (1 + X_j)(1 + Y_j)) = \prod_{j=1}^{\kappa} (1 + X_j + Y_j).$$

eq is multilinear, and $\mathrm{eq}(X, Y) = [X = Y]$ on the boolean hypercube $\{0,1\}^{2\kappa}$.

Definition 3.5 (Inner product over the boolean hypercube)**.** For a, b on $\{0,1\}^\kappa$, $\langle a, b \rangle = \sum_{x \in \{0,1\}^\kappa} a(x)\, b(x)$.

Fact 3.6 (eq carries the interpolation weights)**.** *For an arbitrary point $r \in E^\kappa$,*

$$\widetilde{f}(r) = \sum_{x \in \{0,1\}^\kappa} f(x)\ \mathrm{eq}(r, x) = \langle f, \mathrm{eq}(r, \cdot) \rangle.$$

Binding only the first $\ell \leq \kappa$ variables gives the same identity with the rest left free: for $r \in E^\ell$ and $x \in \{0,1\}^{\kappa - \ell}$,

$$\widetilde{f}(r, x) = \sum_{u \in \{0,1\}^\ell} \mathrm{eq}(r, u)\, f(u, x),$$

so a partial evaluation is again a multilinear, its values the $\mathrm{eq}(r, \cdot)$-weighted combination of the rows $f(u, \cdot)$.

Lemma 3.7 (Weight of a suffix)**.** *Read $x \in \{0,1\}^\kappa$ as an integer, low bit first, and let $\mathsf{pad}_n(r) = \sum_{x \geq n} \mathrm{eq}(r, x)$, the multilinear extension of $[x \geq n]$. Then $\mathsf{pad}_0 = 1$, $\mathsf{pad}_{2^\kappa} = 0$, and for $n < 2^\kappa$, with top bit $n_{\kappa-1}$,*

$$\mathsf{pad}_n(r) \;=\; (1 - n_{\kappa-1})\, r_{\kappa-1} + \mathrm{eq}(n_{\kappa-1}, r_{\kappa-1})\, \mathsf{pad}_{n \bmod 2^{\kappa-1}}(r_{<\kappa-1}),$$

so pad_n costs κ steps, unrolled from the low bit.

Proof. Split on the top bit $x_{\kappa-1}$. If $n_{\kappa-1} = 0$, every x with $x_{\kappa-1} = 1$ counts; if $n_{\kappa-1} = 1$, none with $x_{\kappa-1} = 0$ does. The others count when their low bits do. □

Lemma 3.8 (Every element outside K is a cubic generator)**.** *For $\alpha \in E \setminus K$, the set $\{1, \alpha, \alpha^2\}$ is a K-basis of E.*

Proof. Degrees multiply in the tower $K \subseteq K(\alpha) \subseteq E$, so $[E : K(\alpha)]\,[K(\alpha) : K] = [E : K] = 3$. That degree is prime and $[K(\alpha) : K] > 1$ because $\alpha \notin K$, hence $[K(\alpha) : K] = 3$ and $K(\alpha) = E$. The minimal polynomial of α over K then has degree 3, so $1, \alpha, \alpha^2$ are K-independent, and three independent elements of a space of K-dimension 3 are a basis. □

Lemma 3.9 (Schwartz–Zippel)**.** *A nonzero $P \in E[X_1, \ldots, X_\kappa]$ of total degree at most d satisfies*

$$\Pr_{r \leftarrow E^\kappa}[P(r) = 0] \;\leq\; \frac{d}{|E|}.$$

A nonzero multilinear has total degree at most κ, so it vanishes at a random point with probability at most $\kappa/|E|$.

Proof. Induction on κ. For $\kappa = 1$ a nonzero univariate of degree at most d has at most d roots, so a uniform r_1 is one of them with probability at most $d/|E|$. For $\kappa > 1$ write P in its last variable,

$$P(X_1, \ldots, X_\kappa) = \sum_{k=0}^{d} X_\kappa^k\, Q_k(X_1, \ldots, X_{\kappa-1}),$$

and let k^* be the largest k with $Q_k \neq 0$. Then Q_{k^*} is nonzero in $\kappa - 1$ variables of total degree at most $d - k^*$, so by induction $Q_{k^*}(r_{<\kappa}) = 0$ with probability at most $(d - k^*)/|E|$. Fix $r_{<\kappa}$ with $Q_{k^*}(r_{<\kappa}) \neq 0$: then $X_\kappa \mapsto P(r_{<\kappa}, X_\kappa)$ has degree exactly k^*, hence at most k^* roots, and a uniform r_κ is one with probability at most $k^*/|E|$. Adding the two bounds gives $d/|E|$. □

Corollary 3.10 (Testing an identity at one random point)**.** *Let $A, B \in E[X_1, \ldots, X_\kappa]$ have total degree at most d, and let $r \leftarrow E^\kappa$ be drawn after both are fixed. If $A \neq B$ then*

$$\Pr_{r}[A(r) = B(r)] \;\leq\; \frac{d}{|E|}.$$

Proof. Apply Lemma 3.9 to $A - B$, nonzero of total degree at most d. □

Fact 3.11 (Sumcheck [LFKN92])**.** *Over κ rounds, sumcheck reduces a claim $\sum_{x \in \{0,1\}^\kappa} P(x) = c^{(0)}$, on a polynomial P of degree at most d in each variable, to one evaluation $P(\chi)$ at a random $\chi \in E^\kappa$. Round j carries the running claim $c^{(j-1)}$: the prover sends the univariate*

$$h_j(Y) = \sum_{x \in \{0,1\}^{\kappa - j}} P(\chi_1, \ldots, \chi_{j-1}, Y, x)$$

of degree at most d, the verifier checks $h_j(0) + h_j(1) \stackrel{?}{=} c^{(j-1)}$, samples $\chi_j \leftarrow E$ and takes $c^{(j)} = h_j(\chi_j)$. It ends holding $c^{(\kappa)} = P(\chi)$, for the caller to settle. A false $c^{(0)}$ survives with probability at most $d\kappa/|E|$.

Definition 3.12 (Zerocheck)**.** To prove that a polynomial P vanishes on $\{0,1\}^\kappa$, the verifier samples $r \in E^\kappa$ uniformly and the two parties run sumcheck on

$$\sum_{x \in \{0,1\}^\kappa} P(x)\ \mathrm{eq}(r,x) \stackrel{?}{=} 0.$$

By Fact 3.6 that sum is $\widetilde{g}(r)$, for g the restriction of P to the boolean hypercube: the zero polynomial if P vanishes there, and otherwise a nonzero multilinear, which a random r sends to 0 with probability at most $\kappa/|E|$ (Lemma 3.9).

Lemma 3.13 (Partially fixed zerocheck [DT24])**.** *Let $g : \{0,1\}^\ell \times \{0,1\}^m \to \mathbb{F}_2$ be a nonzero Boolean multilinear. Fix $a \in E^\ell$ such that the values $\{\mathrm{eq}(a,b) : b \in \{0,1\}^\ell\}$ are linearly independent over $\mathbb{F}_2$. If $s \leftarrow E^m$ is sampled uniformly after g is fixed, then*

$$\Pr_s[\widetilde{g}(a,s) = 0] \le \frac{m}{|E|}.$$

Proof. Partially evaluate the first ℓ variables at a. By Fact 3.6, the resulting multilinear $h : \{0,1\}^m \to E$ has values

$$h(y) = \widetilde{g}(a,y) = \sum_{b \in \{0,1\}^\ell} g(b,y)\,\mathrm{eq}(a,b).$$

Because g is nonzero, there is some $y \in \{0,1\}^m$ for which the coefficients $\{g(b,y) : b \in \{0,1\}^\ell\}$ are not all zero. These coefficients lie in $\mathbb{F}_2$, so the assumed $\mathbb{F}_2$-linear independence of the values $\mathrm{eq}(a,b)$ gives $h(y) \neq 0$. Thus h is a nonzero multilinear. Its extension satisfies $\widetilde{h}(s) = \widetilde{g}(a,s)$, and Lemma 3.9 gives the claimed probability bound. □

Two savings on a round message [Gru24]. A round polynomial of degree D travels as its coefficients, and would take $D+1$ of them, but the round check $h_j(0) + h_j(1) \stackrel{?}{=} s_{j-1}$ already pins one to the incoming claim, so D are enough. The same idea applies to the known factor: a zerocheck's summand carries $\mathrm{eq}(r,\cdot)$, which the verifier can form itself, so only the cofactor travels and it is one degree lower. A zerocheck round on a degree-d cofactor therefore costs d field elements.

Definition 3.14 (Inner-product commitment scheme)**.** A *weighted claim* about a multilinear $g : \{0,1\}^M \to K$ is a pair (W,c) asserting $\langle W, g\rangle = c$, with $c \in E$ and the weight $W : \{0,1\}^M \to E$ *MLE-friendly*: the verifier can evaluate $\widetilde{W}$ anywhere in $\mathrm{poly}(M)$ operations. An *inner-product commitment scheme* commits to g and later proves any such claim,

$$\langle W, g\rangle = \sum_{w \in \{0,1\}^M} W(w)\,g(w) = c.$$

The commitment is over K, the opening over E. An evaluation claim $\widetilde{g}(r) = c$ is the case $W = \mathrm{eq}(r,\cdot)$, by Fact 3.6.

We use WHIR [ACFY25] / Ligerito [NA25] as inner-product commitment scheme (see annex B). The two are essentially the same scheme when WHIR is instantiated with an interleaved code and Ligerito with a Reed–Solomon code.

4 Committing the witness

4.1 Multilinear stacking

Suppose we want to commit to T multilinears $P_1, \dots, P_T$ over boolean hypercubes of various sizes. We could commit to each independently, at the cost of a large cumulative opening proof. Instead they are concatenated into a large multilinear polynomial q, which we commit to with an inner-product commitment scheme (Definition 3.14), with a protocol to reduce opening claims on P_i to a claim on q. Order them by size, $\kappa_1 \geq \cdots \geq \kappa_T$ variables, and concatenate their values end to end,

$$q \;=\; P_1 \,\|\, P_2 \,\|\, \dots \,\|\, P_T \,\|\, 0,$$

zero-padded to length 2^M, with $N_{\text{stack}} := \sum_i 2^{\kappa_i}$ the number of placed field elements and $M = \lceil \log_2 N_{\text{stack}} \rceil$. Placed largest first, P_i starts at an offset that is a multiple of 2^{κ_i}; writing $\mathsf{sel}_i \in \{0,1\}^{M-\kappa_i}$ for the high bits of that offset,

$$\widetilde{P}_i(z) \;=\; \widetilde{q}(z, \mathsf{sel}_i) \qquad (z \in \{0,1\}^{\kappa_i}).$$

Conversely, a claim on the stack splits along its pieces: writing $\zeta = (\zeta_{<\kappa}, \zeta_{\geq\kappa})$ for the split of $\zeta \in E^M$ after κ coordinates,

$$\widetilde{q}(\zeta) \;=\; \sum_i \mathrm{eq}(\mathsf{sel}_i, \zeta_{\geq\kappa_i})\, \widetilde{P}_i(\zeta_{<\kappa_i}), \tag{1}$$

the zero padding contributing nothing. An evaluation claim $\widetilde{P}_i(\zeta) = c$ is therefore a claim on the stack,

$$\widetilde{q}(\zeta, \mathsf{sel}_i) \;=\; \sum_{w \in \{0,1\}^M} \mathrm{eq}\big((\zeta, \mathsf{sel}_i), w\big)\, \widetilde{q}(w) \;=\; c,$$

a weighted sum over $\widetilde{q}$ with weight $W(w) = \mathrm{eq}((\zeta, \mathsf{sel}_i), w)$.

Batching. By the identity above, the claims the protocol raises on the P_i transfer to a list of claims $\langle W_j, \widetilde{q}\rangle = c_j$ on the stack, with E-valued weights and targets. The verifier then samples $\lambda \in E$, and it remains to prove

$$\Big\langle \sum_j \lambda^{j-1} W_j,\; \widetilde{q} \Big\rangle \stackrel{?}{=} \sum_j \lambda^{j-1} c_j$$

via the PCS (Annex B).

4.2 Committing to booleans via Ring-Switching

The Keccak GKR (Annex C) terminates with a claim about the inputs (and intermediate layer) at the bit-level. Those bits are committed packed, (interpeting each 64 consecutive bits as a K-element), as the K-valued multilinear q_{keccak} in κ_{keccak} variables, which occupies one region of the stack in §4.1. Ring switching [DP26] turns 64 bit-level evaluations claims into one single claim on q_{keccak} (Annex A).

5 Arithmetization framework

The arithmetization is inspired by the M3 model (multi-multiset matching) [Irr]. Executed rows are unordered, and an instance is accepted exactly when every table's constraints hold and the bus balances.

5.1 M3 model

Tables and columns. A column of height 2^κ is a function $\{0,1\}^\kappa \to K$. A table is a group of columns of equal height. The arithmetization fixes the number of tables N_{tab} and, for each table T_j, its number of columns $N_{\text{col},j}$, its $N_{\text{cons},j}$ constraints and its $N_{\text{flushes},j}$ bus flushes. The table heights are not fixed: the prover announces each table's row count $N_{\text{exec},j}$ at the beginning of the proof, and T_j has height 2^{τ_j}, τ_j the least with $N_{\text{exec},j} \le 2^{\tau_j}$ ($\tau_{\text{keccak}} \ge 9$ for `KECCAK`, whose rows are the permutations the Keccak reduction proves, at least 2^9 of them, Annex C). Reading a row x as an integer, low bit first, the rows $x < N_{\text{exec},j}$ are the *executed* rows, and the others are *padding* rows (§8.3).

Constraints. The constraints of T_j are $C_{j,1}, \dots, C_{j,N_{\text{cons},j}}$, each a polynomial over K of degree at most d ($d = 2$ throughout) in the $N_{\text{col},j}$ columns, required to vanish on every row in $\{0,1\}^{\tau_j}$.

The bus. All interactions (between tables) share a single *bus*, a channel carrying tuples of up to $m = 16$ K-elements (shorter ones are zero-padded). Table T_j has $N_{\text{flushes},j}$ flushes $\mathbf{f}_{j,1}, \dots, \mathbf{f}_{j,N_{\text{flushes},j}}$, each is either a *push* or a *pull*. Each one of the m coordinates in each tuple is a fixed polynomial of degree at most d in the table's columns, so every executed row emits one tuple $\mathbf{f}_{j,f}(x)$ per flush. A few *boundary* tuples are pushed (resp. pulled) by default to the bus (for instance for the VM state $(\mathsf{ST}, \mathbf{pc}, \mathbf{fp})$, the initial $(\mathsf{ST}, \mathsf{g}^0, \mathsf{g}^0)$ is pushed and the final $(\mathsf{ST}, \mathsf{g}^{N_{\text{prog}}-1}, \mathsf{g}^0)$ is pulled).

Balance. The bus balances when its pushed tuples and its pulled tuples form the same multiset (3.2).

Domain separation. When several interactions share the bus, the first coordinate of every tuple is a *domain separator*, a fixed constant naming its interaction: $\mathsf{ST} = \mathsf{g}^0$ for VM state, $\mathsf{MEM} = \mathsf{g}^1$ for memory, $\mathsf{BC} = \mathsf{g}^2$ for bytecode. (They are needed since all interactions share a common bus.)

5.2 Balancing the bus: grand product

Both multisets (*pulls* and *pushes*) carry the same number of tuples, and padding each with factors of 1, which change no product, makes that number a power of two, 2^μ.

Fingerprint. The verifier samples $\alpha = (\alpha_0, \dots, \alpha_3) \in E^4$ and maps each tuple $\mathbf{f} \in K^{16}$ to a single E-element:

$$\pi_\alpha(\mathbf{f}) \;=\; \sum_{\iota \in \{0,1\}^4} \mathrm{eq}(\alpha, \iota)\, \mathbf{f}_\iota,$$

the sixteen slots indexed by their bits, low bit first, and smaller tuples padded with 0.

Product check. The verifier samples $\beta \in E$, and checks:

$$\prod_{\mathbf{f}\ \text{pushed}} (\beta - \pi_\alpha(\mathbf{f})) \;\stackrel{?}{=}\; \prod_{\mathbf{f}\ \text{pulled}} (\beta - \pi_\alpha(\mathbf{f})). \tag{2}$$

Theorem 5.1. *The protocol above has perfect completeness (equal pulled and pushed multisets always make (2) valid) and soundness error $4 \cdot 2^\mu/|E|$ (if the pulled and pushed multisets differ, the probability that (2) passes over uniform $\alpha \in E^4$ and $\beta \in E$).*

Lemma 5.2 (A product identity is a multiset identity)**.** *For a multiset P of tuples write*

$$\Pi_P(A, X) \;=\; \prod_{\mathbf{f} \in P} (X - \pi_A(\mathbf{f})),$$

a polynomial in the five formal variables $A = (A_0, \ldots, A_3)$ and X, with coefficients in K. The two sides of (2) are $\Pi_P(\alpha, \beta)$ for P the pushed multiset and the pulled one.

For finite multisets P, Q of width-m tuples over K, $\Pi_P = \Pi_Q$ if and only if $P = Q$.

Proof of Lemma 5.2. Distinct tuples have distinct fingerprints $\pi_A(\mathbf{f})$, the $\mathrm{eq}(A, \iota)$ being linearly independent. So Π_P is a product of the monic, degree-one, hence irreducible factors $X - \pi_A(\mathbf{f})$, and unique factorization in $K[A][X]$ recovers P from it. □

Proof of Theorem 5.1. Completeness. Equal pulled and pushed multisets make (2) hold for every α and β, so completeness is immediate.

Soundness. Different pulled and pushed multisets make $\Pi_P \neq \Pi_Q$, by Lemma 5.2, and (2) tests them at the one point $(\alpha_0, \ldots, \alpha_3, \beta)$. Both sides have degree at most $4 \cdot 2^\mu$ in $(\alpha_0, \ldots, \alpha_3, \beta)$, since there are at most 2^μ factors, of the form $\beta - \pi_\alpha(\mathbf{f})$, each of total degree at most 4. Applying Corollary 3.10 gives a soundness error of $4 \cdot 2^\mu / |E|$. □

5.3 Proving the grand product

Take one side of (2). Its tuples give its 2^μ leaves $V_0(x) = \beta - \pi_\alpha(\mathbf{f}_x)$, the ones past the last tuple being 1. Multiply the leaves two by two, then multiply the results two by two, and so on: after μ layers one value is left, the root $P = \prod_x V_0(x)$. Layer 0 is the leaves, layer i has $2^{\mu-i}$ nodes, and layer μ is the root. The prover announces P. GKR proves it by walking back down the tree, turning a claim about one layer into a claim about the layer below, until it reaches the leaves and §5.4 takes over.

5.3.1 Grand-Product argument via GKR

Write $\widetilde{V}_i : E^{\mu-i} \to E$ for the multilinear extension of layer i. Every node of layer i is the product of its two children,

$$V_i(x) = V_{i-1}(0, x)\, V_{i-1}(1, x), \qquad x \in \{0,1\}^{\mu-i},$$

so by Fact 3.6 a claim on $\widetilde{V}_i$ at a point r is a sum over the hypercube,

$$\widetilde{V}_i(r) = \sum_{x \in \{0,1\}^{\mu-i}} \mathrm{eq}(r, x)\, \widetilde{V}_{i-1}(0, x)\, \widetilde{V}_{i-1}(1, x),$$

which sumcheck (Fact 3.11) reduces to the two values $\widetilde{V}_{i-1}(0, \chi)$ and $\widetilde{V}_{i-1}(1, \chi)$ at one random χ. The prover sends them and a fresh challenge u combines them multilinearly, leaving the single claim $\widetilde{V}_{i-1}(u, \chi)$: the same kind of claim, one layer lower. The walk starts at the root, where the claim is the announced P itself, and μ such steps end on the leaves.

5.3.2 Two layers at a time

For efficiency, we contract two binary levels at once, a node being the product of its four grandchildren:

$$V_i(x) = \prod_{a,b \in \{0,1\}} V_{i-2}(a, b, x).$$

The same sumcheck then reduces $\widetilde{V}_i(r)$ to the four values $\widetilde{V}_{i-2}(a, b, \chi)$, which the prover sends and two fresh challenges (u_0, u_1) combine into one claim $\widetilde{V}_{i-2}(u_0, u_1, \chi)$, two layers down.

If μ is odd, GKR first processes the root-most binary layer, which has no sumcheck rounds, and then proceeds entirely in radix four.

5.3.3 Batching several grand-products

For recursion friendliness, the N_{side} grand products, numbered $s = 1, \ldots, N_{\text{side}}$, are proven in one pass, at one sumcheck per layer instead of N_{side}.

Pad every tree with 1-leaves up to the depth μ of the deepest. The verifier then samples at every layer a combiner $\lambda \in E$, and each layer runs a single sumcheck on the random linear combination of the N_{side} identities,

$$\sum_{s=1}^{N_{\text{side}}} \lambda^{s-1} \widetilde{V}_i^s(r) = \sum_{x \in \{0,1\}^{\mu-i}} \text{eq}(r, x) \sum_{s=1}^{N_{\text{side}}} \lambda^{s-1} \prod_{a,b \in \{0,1\}} \widetilde{V}_{i-2}^s(a, b, x)$$

All N_{side} trees leave the pass on claims at one shared terminal point ζ.

5.4 Stacking the leaves

GKR (§5.3) leaves one claim per side s: the value of $\widetilde{V}_0^s(\zeta)$, the extension of that side's leaves at the terminal point $\zeta \in E^\mu$. This subsection reduces it to one sum per table, over its rows, of a polynomial of degree at most $d = 2$ in its columns: exactly what the *table sumcheck* (§5.5) proves.

Blocks. The leaves of side s are a stack (§4.1) whose pieces are its *blocks*: one per flush of each table on that side, and one per boundary set (§5.1, §6.2). Block b holds 2^{κ_b} leaves $L_b(x)$ and sits at the public selector sel_b; for the block of a flush $\mathbf{f}_{j,f}$, $\kappa_b = \tau_j$, and $L_b(x)$ is $\beta - \pi_\alpha(\mathbf{f}_{j,f}(x))$ on an executed row; an honest prover puts 1 on a padding row. One difference with the stacking at commitment (§4.1) is the padding: 1s rather than 0s.

$$\widetilde{V}_0^s(\zeta) \;=\; \sum_b \text{eq}(\mathsf{sel}_b, \zeta_{\geq \kappa_b})\, \widetilde{L}_b(\zeta_{<\kappa_b}) \;+\; \underbrace{\Big(1 - \sum_b \text{eq}(\mathsf{sel}_b, \zeta_{\geq \kappa_b})\Big)}_{\text{unused leaves}}, \tag{3}$$

both sums over the blocks of side s.

Who settles what. Three kinds of terms:

- *Unused leaves.* Public: the verifier evaluates them.
- *Boundary blocks.* Every slot of their tuples is public (a constant, a public input, or the index column of §6.5) or a single committed column, so $\widetilde{L}_b(\zeta_{<\kappa_b})$ is a linear combination of their evaluations at $\zeta_{<\kappa_b}$. The verifier computes the public ones; the prover sends each committed one, as a claim deferred to the PCS (§4.1).
- *Table blocks.* Nothing is sent for them: the table sumcheck proves them, except for a public term coming from their padding rows (below).

The verifier subtracts the first two kinds and that public term from the claimed value; the difference, rem_s, is what the tables owe.

What a table owes. Let b be a block of T_j, so $\kappa_b = \tau_j$, and let $\phi_b(x) = \beta - \pi_\alpha(\mathbf{f}_{j,f}(x))$, a polynomial of degree at most d in T_j's columns (on the count side, $\phi_b(x)$ is the count itself, and $\phi_b(\mathbf{p}_j) = 0$). The verifier checks the claim on block b against the leaves

$$L_b(x) \;=\; \phi_b(x) + [x \geq N_{\text{exec},j}]\,(1 - \phi_b(\mathbf{p}_j)),$$

where $\mathbf{p}_j$ is a public row (§8.3). They are defined by the committed columns, whatever they hold. On an executed row $L_b(x) = \phi_b(x)$. An honest prover fills padding rows with $\mathbf{p}_j$, so there $L_b(x) = \phi_b(\mathbf{p}_j) + 1 - \phi_b(\mathbf{p}_j) = 1$, the leaf it puts in GKR; Proposition 8.1 covers a padding row holding anything else. The indicator $[x \geq N_{\text{exec},j}]$ only multiplies a constant, which the verifier computes, so by Fact 3.6

$$\widetilde{L}_b(\zeta_{<\tau_j}) \;=\; \sum_{x\in\{0,1\}^{\tau_j}} \mathrm{eq}(\zeta_{<\tau_j}, x)\,\phi_b(x) \;+\; (1-\phi_b(\mathbf{p}_j))\,\mathsf{pad}_{N_{\text{exec},j}}(\zeta_{<\tau_j}). \tag{4}$$

The last term is the public term above: $\mathsf{pad}_{N_{\text{exec},j}}$ is the multilinear extension of $[x \geq N_{\text{exec},j}]$, which the verifier computes in τ_j steps (Lemma 3.7). The first term sums ϕ_b over every row, padding rows included, with no indicator, so the table sumcheck proves it as if there were no padding. The blocks of T_j share the weight $\mathrm{eq}(\zeta_{<\tau_j}, \cdot)$ and differ only by selector, so together they contribute one sum, of

$$B_j^s(x) \;=\; \sum_{b\in T_j} \mathrm{eq}(\mathsf{sel}_b, \zeta_{\geq\tau_j})\,\phi_b(x), \tag{5}$$

of degree at most d in T_j's columns, with coefficients the verifier builds from α, β and the selectors. What remains of side s is

$$\sum_{j=1}^{N_{\text{tab}}} \sum_{x\in\{0,1\}^{\tau_j}} \mathrm{eq}(\zeta_{<\tau_j}, x)\,B_j^s(x) \;\stackrel{?}{=}\; \mathsf{rem}_s, \qquad s \in \{1,\ldots,N_{\text{side}}\}. \tag{6}$$

5.5 Table sumcheck

Proving constraints via zerocheck. $C_{j,i}$ is a polynomial in T_j's columns, so it is a function $\{0,1\}^{\tau_j} \to K$ that must vanish on every row. Following Fact 3.6, this can be proved by sampling $r \in E^{\tau_j}$ uniformly at random and running sumcheck on:

$$\widetilde{C_{j,i}}(r) \;=\; \sum_{x\in\{0,1\}^{\tau_j}} \mathrm{eq}(r, x)\,C_{j,i}(x) \;\stackrel{?}{=}\; 0$$

Zerocheck point recycling. Nothing forces r to be sampled just before the zerocheck: it may come from a previous sub-protocol, as long as it results from uniform sampling occurring after the table columns are committed. We take $r = \zeta_{<\tau_j}$, the first τ_j coordinates of the GKR terminal point ζ (§5.3). Validity of the constraints is thus reduced to:

$$\sum_{x\in\{0,1\}^{\tau_j}} \mathrm{eq}(\zeta_{<\tau_j}, x)\,C_{j,i}(x) \;\stackrel{?}{=}\; 0, \qquad \text{for } j \in \{1,\ldots,N_{\text{tab}}\},\ i \in \{1,\ldots,N_{\text{cons},j}\}. \tag{7}$$

The bus claims join in. §5.4 left one claim per side at that same point, (6). Both (7) and (6) are built from $\mathrm{eq}(\zeta_{<\tau_j}, \cdot)$-weighted sums over one table's rows of a polynomial of degree at most d in its columns: we batch them in a single sumcheck.

Back-loaded batched sumcheck. The verifier samples $\xi \in E$ and gives each claim its own power of it: ξ^{o_j+i-1} for $C_{j,i}$, with $o_j = N_{\text{cons},1} + \cdots + N_{\text{cons},j-1}$ numbering the constraints table by table, and then $\xi^{N_{\text{cons}}+s-1}$ for side s, with $N_{\text{cons}} = \sum_j N_{\text{cons},j}$ the constraint count. A sumcheck

runs over one cube, so write $\tau_{\max} = \max_j \tau_j$, let $X_{<\tau_j} = (X_0, \ldots, X_{\tau_j - 1})$ be the variables T_j uses, and multiply in the ones it does not:

$$F(X_0, \ldots, X_{\tau_{\max}-1}) \;=\; \sum_{j=1}^{N_{\text{tab}}} \mathrm{eq}(\zeta_{<\tau_j}, X_{<\tau_j}) \Big(\sum_{i=1}^{N_{\text{cons},j}} \xi^{\,o_j + i - 1} C_{j,i} + \sum_{s=1}^{N_{\text{side}}} \xi^{\,N_{\text{cons}} + s - 1} B_j^s \Big)(X_{<\tau_j}) \prod_{k=\tau_j}^{\tau_{\max}-1} X_k.$$

Since $\sum_{x \in \{0,1\}^m} x_0 \cdots x_{m-1} = 1$, the padded variables contribute a factor 1, so summing F over $\{0,1\}^{\tau_{\max}}$ gives back what each table owes. We thus run a single sumcheck:

$$\sum_{x \in \{0,1\}^{\tau_{\max}}} F(x) \;\stackrel{?}{=}\; \sum_{s=1}^{N_{\text{side}}} \xi^{\,N_{\text{cons}} + s - 1} \mathsf{rem}_s,$$

the verifier computing the right side itself. Each summand of F has individual degree $d + 1 = 3$, the constraint contributing d and the eq factor one, so the soundness error is $(3\tau_{\max} + N_{\text{side}} + N_{\text{cons}})/|E|$: $3\tau_{\max}$ for the rounds (Fact 3.11) and $N_{\text{side}} + N_{\text{cons}}$ for the ξ-batching (Corollary 3.10). Rounds bind $X_{\tau_{\max}-1}$ first, so T_j joins only at round $\tau_{\max} - \tau_j$: hence *back-loaded*. The end yields one claim per column, T_j's at $\chi_{<\tau_j}$, settled by the PCS opening (§4.1).

5.6 Example: proving elliptic curve multiplication

5.6.1 Statement

The claim. Over K, with $\mathsf{g} = 2$, fix the elliptic curve $t^2 + e\,t = e^3 + \mathsf{g}$ and the two public points

$$G = (e_G, t_G) = (\mathtt{1},\ \mathtt{9d7b84b595b2e2e6}),$$
$$Q = (e_Q, t_Q) = (\mathtt{bbde619e481ba9dd},\ \mathtt{c7dc5bdf1e3477f9}).$$

The group law gives:

$$[2](e,t) = (e',\ e^2 + (w+1)\,e'), \qquad w = e + t/e, \qquad e' = w^2 + w,$$
$$(e,t) + G = (e',\ w\,(e+e') + e' + t), \qquad w = (t + t_G)/(e + e_G), \qquad e' = w^2 + w + e + e_G.$$

The statement is: $Q = [66067]G$. We prove it via a double-and-add walk. Since $66067 = 2^{16} + 2^9 + 2^4 + 2 + 1$, there are sixteen doublings and four additions of G.

5.6.2 Arithmetization

The tables.

- $T_1 = \mathtt{DOUBLE}$, height 2^4.

  Columns: $\mathsf{tag}, e, t, w, e'$.
  Constraints: $C_{1,1} : w\,e = e^2 + t$ and $C_{1,2} : e' = w^2 + w$.
  Flushes: PULL (tag, e, t) and PUSH $(\mathsf{tag}^2,\ e',\ e^2 + (w+1)\,e')$.

- $T_2 = \mathtt{ADD}$, height 2^2.

  Columns: $\mathsf{tag}, e, t, w, e', v$.
  Constraints: $C_{2,1} : v\,(e + e_G) = 1$, $C_{2,2} : w\,(e + e_G) = t + t_G$ and $C_{2,3} : e' = w^2 + w + e + e_G$.
  Flushes: PULL (tag, e, t) and PUSH $(\mathsf{g} \cdot \mathsf{tag},\ e',\ w\,(e + e') + e' + t)$.

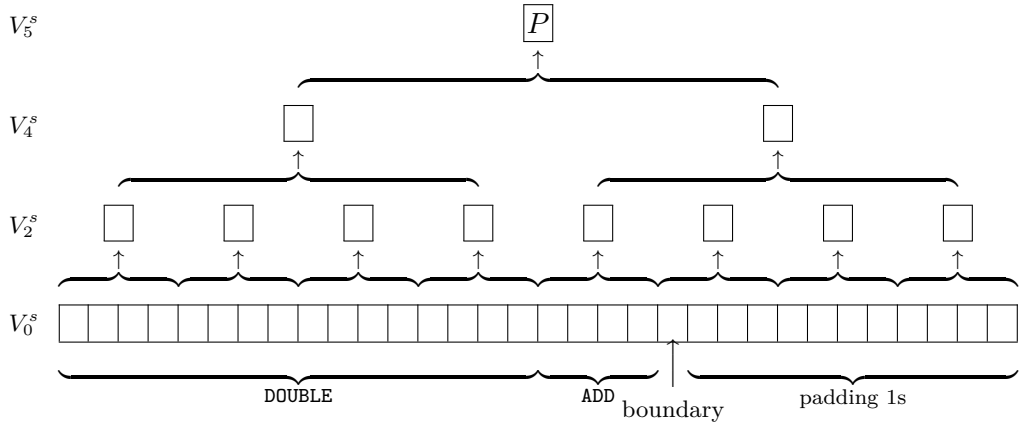
The column v in T_2 is required to prevent a row at the point G, where $e + e_G = 0$ and $w\,(e + e_G) = t + t_G$ holds for any w. So $N_{\text{tab}} = 2$ and $N_{\text{cons}} = 5$.

Boundary flushes. PUSH (g, e_G, t_G) and PULL $(\mathsf{g}^{66067}, e_Q, t_Q)$.

The existence of T_1 and T_2 verifying the constraints, and balancing the bus (including the boundary flushes) proves the statement.

5.6.3 The interactive protocol, unrolled

1. **Prover.** Commits the eleven columns. In this example, the two log heights are fixed, $\tau_1 = 4$ and $\tau_2 = 2$.
2. **Verifier.** Samples and sends $\alpha_0, \alpha_1, \alpha_2, \alpha_3$ and β.
3. **Prover.** Builds the leaves $\beta - \pi_\alpha(\mathbf{f})$, $16 + 4 + 1 = 21$ per side (PUSH / PULL), and pads to $\mu = 5$ with eleven 1s. The two sides share the same layout (largest blocks first):



   The boundary block is leaf 20, and the three selectors are (0), $(0, 0, 1)$ and $(0, 0, 1, 0, 1)$.
4. **Prover.** Sends one root P shared by both sides.
5. **Prover & Verifier.** For each side, run Grand-Product GKR on the tree above: μ is odd, so a binary layer first, then two radix-four steps. The sumchecks of each side are batched together, with final point $\zeta \in E^5$.
6. **Verifier.** Evaluates leaf 20, whose entries are public, and the weight of the eleven padding leaves. Subtracting both from $\widetilde{V}_0^s(\zeta)$ leaves $\mathsf{rem}_{\text{PUSH}}$ and $\mathsf{rem}_{\text{PULL}}$.
7. **Verifier.** Samples and sends the batching scalar ξ.
8. **Prover & Verifier.** Seven claims are now open, and every one of them is a sum over one table's rows against its own weight $\mathrm{eq}(\zeta_{<\tau_j}, \cdot)$. The five constraints have to vanish, and the two sides have to return what step 6 left:

$$\sum_{x \in \{0,1\}^{\tau_j}} \mathrm{eq}(\zeta_{<\tau_j}, x)\, C_{j,i}(x) \stackrel{?}{=} 0 \qquad \text{for the five } C_{j,i},$$

$$\sum_{j=1}^{2} \sum_{x \in \{0,1\}^{\tau_j}} \mathrm{eq}(\zeta_{<\tau_j}, x)\, B_j^s(x) \stackrel{?}{=} \mathsf{rem}_s \qquad \text{for } s \in \{\text{PUSH}, \text{PULL}\},$$

where B_j^s is what T_j owes on side s. Writing a column's value at row x as $e(x)$ and so on:

$$B_1^{\text{PUSH}}(x) = \mathrm{eq}((0), \zeta_{\geq 4})\Big(\beta - \pi_\alpha(\mathsf{tag}(x)^2,\ e'(x),\ e(x)^2 + (w(x)+1)\,e'(x))\Big),$$
$$B_1^{\text{PULL}}(x) = \mathrm{eq}((0), \zeta_{\geq 4})\Big(\beta - \pi_\alpha(\mathsf{tag}(x),\ e(x),\ t(x))\Big),$$
$$B_2^{\text{PUSH}}(x) = \mathrm{eq}((0,0,1), \zeta_{\geq 2})\Big(\beta - \pi_\alpha(\mathsf{g}\cdot\mathsf{tag}(x),\ e'(x),\ w(x)\,(e(x)+e'(x)) + e'(x) + t(x))\Big),$$
$$B_2^{\text{PULL}}(x) = \mathrm{eq}((0,0,1), \zeta_{\geq 2})\Big(\beta - \pi_\alpha(\mathsf{tag}(x),\ e(x),\ t(x))\Big).$$

They run over different cubes, $\{0,1\}^4$ for `DOUBLE` and $\{0,1\}^2$ for `ADD`, so `ADD`'s part is multiplied by X_2X_3, the two variables it does not use, whose sum over $\{0,1\}^2$ is 1 and so leaves its own sum untouched:

$$\begin{aligned} F(X_0,\dots,X_3) = {} & \mathrm{eq}(\zeta_{<4}, (X_0,\dots,X_3))\Big(\xi^0 C_{1,1} + \xi^1 C_{1,2} + \xi^5 B_1^{\text{PUSH}} + \xi^6 B_1^{\text{PULL}}\Big) \\ & + \mathrm{eq}(\zeta_{<2}, (X_0,X_1))\Big(\xi^2 C_{2,1} + \xi^3 C_{2,2} + \xi^4 C_{2,3} + \xi^5 B_2^{\text{PUSH}} + \xi^6 B_2^{\text{PULL}}\Big) X_2X_3. \end{aligned}$$

One sumcheck over $\{0,1\}^4$ then proves all seven claims at once:

$$\sum_{x\in\{0,1\}^4} F(x) \stackrel{?}{=} \xi^5 \mathsf{rem}_{\text{PUSH}} + \xi^6 \mathsf{rem}_{\text{PULL}}.$$

9. **Prover.** Sends its eleven column values: $\widetilde{\mathsf{tag}}, \widetilde{e}, \widetilde{t}, \widetilde{w}, \widetilde{e'}$ at $\chi = (\chi_0,\dots,\chi_3)$ for `DOUBLE`, and $\widetilde{\mathsf{tag}}, \widetilde{e}, \widetilde{t}, \widetilde{w}, \widetilde{e'}, \widetilde{v}$ at $\chi_{<2} = (\chi_0,\chi_1)$ for `ADD`.
10. **Verifier.** Rebuilds every $C_{j,i}$ and B_j^s from those eleven values, forms $F(\chi)$, and checks it against the sumcheck's last round.
11. **Prover & Verifier.** Verify each of the eleven column claims against their initial commitment.

6 Bus interactions

The machine runs three interactions on the bus, differentiated by their domain separators (§5.1): the VM state, the read-only memory, and the public bytecode. The last two are lookups using offline memory checking [BEG+94].

6.1 VM State

The state interaction uses domain separator $\mathsf{ST} = \mathsf{g}^0$, and carries the 2 registers, program counter and frame pointer: $(\mathsf{ST}, \mathbf{pc}, \mathbf{fp})$. Each instruction row pulls its current state and pushes its successor $(\mathsf{ST}, \mathrm{next}(\mathbf{pc}), \mathrm{next}(\mathbf{fp}))$; the bus is preseeded by pushing the initial state $(\mathsf{ST}, \mathbf{pc}_{\text{initial}}, \mathbf{fp}_{\text{initial}}) = (\mathsf{ST}, \mathsf{g}^0, \mathsf{g}^0)$ and pulling the final state $(\mathsf{ST}, \mathbf{pc}_{\text{final}}, \mathbf{fp}_{\text{final}}) = (\mathsf{ST}, \mathsf{g}^{N_{\text{prog}}-1}, \mathsf{g}^0)$ (boundary conditions).

Proposition 6.1 (Balance implies the existence of a valid execution)**.** *Let $\mathcal{R}$ be a finite multiset of rows, each pulling a state $\mathsf{pull}(r)$ and pushing a state $\mathsf{push}(r)$, and suppose the state channel balances:*

$$\{\{\mathsf{push}(r) : r \in \mathcal{R}\}\} \uplus \{\{(\mathbf{pc}_{\text{initial}}, \mathbf{fp}_{\text{initial}})\}\} \;=\; \{\{\mathsf{pull}(r) : r \in \mathcal{R}\}\} \uplus \{\{(\mathbf{pc}_{\text{final}}, \mathbf{fp}_{\text{final}})\}\}.$$

Then $\mathcal{R}$ splits into rows $r_1, \dots, r_L$ with

$$\mathsf{pull}(r_1) = (\mathbf{pc}_{\text{initial}}, \mathbf{fp}_{\text{initial}}), \qquad \mathsf{push}(r_i) = \mathsf{pull}(r_{i+1}), \qquad \mathsf{push}(r_L) = (\mathbf{pc}_{\text{final}}, \mathbf{fp}_{\text{final}}),$$

$\mathsf{push}(r_i) \neq (\mathbf{pc}_{\text{final}}, \mathbf{fp}_{\text{final}})$ *for $i < L$, and rows forming closed walks.*

Proof. Build the directed multigraph G on the states: one edge $\mathsf{pull}(r) \to \mathsf{push}(r)$ per row, plus one edge e from $(\mathbf{pc}_{\text{final}}, \mathbf{fp}_{\text{final}})$ to $(\mathbf{pc}_{\text{initial}}, \mathbf{fp}_{\text{initial}})$.

Fix a state x. Its in-degree in G counts the rows with $\mathsf{push}(r) = x$, plus e if $x = (\mathbf{pc}_{\text{initial}}, \mathbf{fp}_{\text{initial}})$. Its out-degree counts the rows with $\mathsf{pull}(r) = x$, plus e if $x = (\mathbf{pc}_{\text{final}}, \mathbf{fp}_{\text{final}})$. Those two counts are the multiplicities of x on the two sides of the hypothesis, so they are equal: G is balanced at every state.

A finite balanced multigraph is a union of edge-disjoint closed walks: follow unused edges from any vertex, which balance always permits on entering one, until a vertex repeats; remove the closed walk found and repeat. Deleting e from the closed walk that contains it leaves $r_1, \dots, r_L$, cut at its first visit of the final state (what follows that visit is a closed walk of its own). □

6.2 The lookup protocol

We use a similar idea to [DP23, Protocol. 4.27].

Setting. A *lookup array* $\mathbf{L}$ holds 2^κ entries, entry $\mathbf{L}[i]$ being a tuple of N_{comp} elements of K. Its *components* $\mathbf{L}_0, \dots, \mathbf{L}_{N_{\text{comp}}-1}$ are the multilinears of height 2^κ holding one coordinate of the entry each: they are either committed or public. A *read* is a row of a table (§5.1) naming an address $a \in K$ and a value $v \in K^{N_{\text{comp}}}$, and is correct when $a = \mathsf{g}^i$ and $v = \mathbf{L}[i]$ for some $i < 2^\kappa$.

Tuple and columns. The bus interaction uses a tuple of the form $(\mathsf{sep}, \mathit{addr}, \mathit{count}, \mathit{val})$: a domain separator naming the array, the address, the count, then the entry in coordinates $3, \dots, N_{\text{comp}} + 2$. Every higher coordinate is zero, so $N_{\text{comp}} \leq m - 3$ for the bus width m (§5.1). Take every part of a read to be a committed K-valued column of the reading table: its address, its count, and the N_{comp} components of the value read. Committed on top of $\mathbf{L}$ is one finalize-count K-valued column $\mathit{count}^{\text{fin}}$ of height 2^κ.

Flush rules. With $A[i]$ the number of reads of address g^i:

- **Seed:** for each $i < 2^\kappa$, PUSH $(\mathsf{sep}, \mathsf{g}^i, 1, \mathbf{L}[i])$. A boundary block owned by no table, its address coordinate the index column of §6.5.
- **Read** of v at a with count count: PULL $(\mathsf{sep}, a, \mathit{count}, v)$ and PUSH $(\mathsf{sep}, a, \mathsf{g} \cdot \mathit{count}, v)$, both flushes of the reading row.
- **Finalize:** for each i, PULL $(\mathsf{sep}, \mathsf{g}^i, \mathit{count}^{\text{fin}}[i], \mathbf{L}[i])$. An honest prover commits $\mathit{count}^{\text{fin}}[i] = \mathsf{g}^{A[i]}$, the count the address has reached.

Completeness. An honest prover gives the t-th read of g^i the count g^t and commits $\mathit{count}^{\text{fin}}[i] = \mathsf{g}^{A[i]}$: both sides then carry the counts $\mathsf{g}^0, \dots, \mathsf{g}^{A[i]}$ at that address, so the bus balances.

Lemma 6.2 (Invariant count multisets)**.** *Let $\mathcal{N}$ be a finite multiset over K with $\mathsf{g} \cdot \mathcal{N} = \mathcal{N}$. If $|\mathcal{N}| < 2^{64} - 1$, every element of $\mathcal{N}$ is 0.*

Proof. Let $\mathrm{mult} : K \to \mathbb{N}$ be the multiplicity function of $\mathcal{N}$ (Definition 3.2). Multiplication by g is a bijection of K and $\mathsf{g} \cdot \mathcal{N}$ has multiplicity function $c \mapsto \mathrm{mult}(\mathsf{g}^{-1}c)$, so the hypothesis reads $\mathrm{mult}(\mathsf{g}c) = \mathrm{mult}(c)$ for all c. Every nonzero c is a power of g, so iterating this gives one value mult_1 on all of $K^\times$, which has $2^{64} - 1$ elements. Then $|\mathcal{N}| = \mathrm{mult}(0) + \mathrm{mult}_1(2^{64} - 1) < 2^{64} - 1$ forces $\mathrm{mult}_1 = 0$. □

Theorem 6.3 (Lookup correctness)**.** *Let the flush rules above be the only flushes carrying* $\mathbf{L}$*'s separator, and write* (a_j, c_j, v_j) *for the address, count and value that read* j *evaluates to. If the bus balances, every* $c_j \neq 0$*, and there are fewer than* $2^{64} - 1$ *reads, then every read is correct:* $(a_j, v_j) = (\mathsf{g}^i, \mathbf{L}[i])$ *for some* $i < 2^\kappa$*.*

Proof. Fix a read pair (a, v) that is *not* an entry of the array, so $(a, v) \neq (\mathsf{g}^i, \mathbf{L}[i])$ for every $i < 2^\kappa$. Balance equates the multiset of pushed tuples with the multiset of pulled ones; keep in each only the tuples carrying $\mathbf{L}$'s separator, address a and entry v, and the two are still equal. A seed and a finalization sit at the pairs $(\mathsf{g}^i, \mathbf{L}[i])$, different from (a, v), so neither survives. A read at (a, v) does, on both sides at once: it pulls its count c and pushes $\mathsf{g}c$. So the survivors are exactly the reads at (a, v).

Writing $\mathcal{N}$ for the multiset of counts of the reads at (a, v), balance therefore says $\mathsf{g} \cdot \mathcal{N} = \mathcal{N}$. There are fewer than $2^{64} - 1$ of them, so Lemma 6.2 makes every count in $\mathcal{N}$ zero; every count is nonzero by hypothesis, so $\mathcal{N}$ is empty. No read happens at such an invalid (a, v).

□

Corollary 6.4 (Addresses are in range)**.** *Under the same hypotheses every read address is an entry address* g^i*,* $i < 2^\kappa$*: no read at any other address,* 0 *included, occurs in an accepted proof (except with negligible soundness error).*

Nothing checks the finalize counts. $\mathit{count}^{\mathrm{fin}}$ appears nowhere in Theorem 6.3 or its proof. A wrong value (such as 0 for instance) cannot make a bad read pass; it can only unbalance the bus at that entry, making the proof invalid.

Meeting the hypotheses. Grand-product GKR checks balance (§5.2). It remains to check that all counts are nonzero, and the $N_{\mathrm{read}} < 2^{64} - 1$ read bound, as follows.

The count product. All counts are nonzero exactly when their product is. The prover therefore sends that product, one E-element, and the verifier checks it is nonzero. It is proven like any other grand product, by GKR over the stacking of every count column (§5.4), batched with the two other GKRs of the bus (push / pull) (§5.3).

Instance caps. The verifier checks the announced sizes against public caps (§8.5): at most 2^{32} rows per table, at most 2^{32} memory cells (§6.3) and at most 2^{32} bytecode instructions (§6.4). A row reads at most twenty-seven times (`KECCAK`: twenty-six cells and its bytecode entry), so the six tables flush $N_{\mathrm{read}} \leq 27 \cdot 6 \cdot 2^{32} < 2^{40}$ reads, far under the $2^{64} - 1$ that Lemma 6.2 requires.

Relaxing the hypotheses. A bus entry may be any polynomial of degree at most $d = 2$ in the row's columns (§5.1), so none of the three (address, count, value) need be committed. Example 1: we don't need to commit memory addresses of the form $\mathbf{fp} \cdot o$ (§6.3) when $\mathbf{fp}$ and o are already committed. Example 2: once the pull counter *count* is committed, no need to commit to the corresponding push counter $\mathsf{g} \cdot \mathit{count}$.

6.3 Memory

The memory is an array of $N_{\text{mem}} = 2^{\kappa_{\text{mem}}}$ cells, $16 \le \kappa_{\text{mem}} \le 32$ (§2). Bus interaction uses domain separation $\mathsf{sep} = \mathsf{MEM} = \mathsf{g}^1$. A memory word is 192-bit (i.e. one E-element), so $N_{\text{comp}} = 3$: memory is committed as three K-valued multilinears $\mathbf{m}_0, \mathbf{m}_1, \mathbf{m}_2$ in κ_{mem} variables, with

$$\mathbf{m} = \mathbf{m}_0 + y\,\mathbf{m}_1 + y^2\,\mathbf{m}_2.$$

6.4 Bytecode

The bytecode is an array of $N_{\text{prog}} = 2^{\kappa_{\text{bc}}}$ instructions, $\kappa_{\text{bc}} \le 32$ (§8.1). Bus interaction uses domain separation $\mathsf{sep} = \mathsf{BC} = \mathsf{g}^2$. An instruction is an opcode plus ten operands, so $N_{\text{comp}} = 11$: with the separator, the address and the count ahead of them, this is the widest entry on the bus at 14 coordinates, which is what forces the four index bits, hence the $m = 16$ slots of §5.1. Coordinate 3 is the opcode, naming the instruction:

$$\mathsf{op}_{\mathsf{XOR}} = \mathsf{g}^0, \quad \mathsf{op}_{\mathsf{MUL}} = \mathsf{g}^1, \quad \mathsf{op}_{\mathsf{SET}} = \mathsf{g}^2, \quad \mathsf{op}_{\mathsf{DRF}} = \mathsf{g}^3, \quad \mathsf{op}_{\mathsf{JMP}} = \mathsf{g}^4, \quad \mathsf{op}_{\mathsf{KEC}} = \mathsf{g}^5.$$

The program is public, i.e. not committed, the verifier can evaluate it directly.

6.5 The index column

The seed and finalization blocks run over every address $\mathsf{g}^0, \ldots, \mathsf{g}^{2^\kappa - 1}$, so the verifier needs the extension of that *index column* idx at the point ζ (§5.4). Writing w for the bits of i, $\mathsf{g}^{\,i} = \prod_k (\mathsf{g}^{\,2^k})^{w_k}$, whose k-th factor is $1 + w_k(1 + \mathsf{g}^{\,2^k})$, which gives the efficient MLE formula:

$$\widetilde{\mathrm{idx}}(\zeta) \;=\; \prod_{k=0}^{\kappa-1} \left(1 + \zeta_k \left(1 + \mathsf{g}^{\,2^k}\right)\right),$$

7 The instruction tables

Each instruction has its own table.

Write $\mathsf{read}\,(\mathsf{sep}, \mathit{addr}, \mathit{count}, \mathit{val}_0, \mathit{val}_1, \ldots)$ for the pair PULL $(\mathsf{sep}, \mathit{addr}, \mathit{count}, \mathit{val}_0, \mathit{val}_1, \ldots)$, PUSH $(\mathsf{sep}, \mathit{addr}, \mathsf{g} \cdot \mathit{count}, \mathit{val}_0, \mathit{val}_1, \ldots)$: every flush below but the state's.

7.1 `XOR`

`XOR` $[o_A, o_B, o_C]$ asserts $\mathbf{m}[\mathbf{fp} \cdot o_C] = \mathbf{m}[\mathbf{fp} \cdot o_A] + \mathbf{m}[\mathbf{fp} \cdot o_B]$, the 192-bit field sum (in E).

- **Columns:** $\mathbf{pc}, \mathbf{fp}$; operands o_A, o_B, o_C; the two read values $(v_{A,i})_{i\in\{0,1,2\}}$ and $(v_{B,i})_{i\in\{0,1,2\}}$; memory counts r_A, r_B, r_C; bytecode count r_{bc}.
- **Constraints:** none.
- **Flushes:**
  - **State:** PULL $(\mathsf{ST}, \mathbf{pc}, \mathbf{fp})$, PUSH $(\mathsf{ST}, \mathsf{g} \cdot \mathbf{pc}, \mathbf{fp})$.
  - **Bytecode:** $\mathsf{read}\,(\mathsf{BC}, \mathbf{pc}, r_{\text{bc}}, \mathsf{op}_{\mathsf{XOR}}, o_A, o_B, o_C, 0, 0)$.
  - **Memory:**

$$\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp} \cdot o_X, r_X, v_{X,0}, v_{X,1}, v_{X,2}) \quad (X \in \{A, B\}),$$
$$\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp} \cdot o_C, r_C, v_{A,0} + v_{B,0}, v_{A,1} + v_{B,1}, v_{A,2} + v_{B,2}).$$

7.2 MUL_NATIVE

MUL_NATIVE $[o_A, o_B, o_C]$ asserts $\mathbf{m}[\mathbf{fp} \cdot o_C] = \mathbf{m}[\mathbf{fp} \cdot o_A] \cdot \mathbf{m}[\mathbf{fp} \cdot o_B]$, the 192-bit field multiplication (in E).

- **Columns:** $\mathbf{pc}, \mathbf{fp}$; operands o_A, o_B, o_C; the two read values $(v_{A,i})_{i\in\{0,1,2\}}$ and $(v_{B,i})_{i\in\{0,1,2\}}$; memory counts r_A, r_B, r_C; bytecode count r_{bc}.
- **Constraints:** none.
- **Flushes:**
  - **State:** PULL $(\mathsf{ST}, \mathbf{pc}, \mathbf{fp})$, PUSH $(\mathsf{ST}, \mathsf{g} \cdot \mathbf{pc}, \mathbf{fp})$.
  - **Bytecode:** $\mathsf{read}\,(\mathsf{BC}, \mathbf{pc}, r_{\mathrm{bc}}, \mathsf{op}_{\mathsf{MUL}}, o_A, o_B, o_C, 0, 0)$.
  - **Memory** (the destination carries $v_A v_B$ in E, of degree two). With $y^3 = y + 1$, put

$$p_0 = v_{A,0}v_{B,0}, \quad p_1 = v_{A,0}v_{B,1} + v_{A,1}v_{B,0}, \quad p_2 = v_{A,0}v_{B,2} + v_{A,1}v_{B,1} + v_{A,2}v_{B,0},$$

$$p_3 = v_{A,1}v_{B,2} + v_{A,2}v_{B,1}, \qquad p_4 = v_{A,2}v_{B,2},$$

    twelve products over the nine pairs $v_{A,i}v_{B,j}$:

$$\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp} \cdot o_X, r_X, v_{X,0}, v_{X,1}, v_{X,2}) \quad (X \in \{A, B\}),$$
$$\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp} \cdot o_C, r_C, p_0 + p_3, p_1 + p_3 + p_4, p_2 + p_4).$$

7.3 SET_CONSTANT

SET_CONSTANT $[o, k_0, k_1, k_2]$ asserts $\mathbf{m}[\mathbf{fp} \cdot o] = k_0 + k_1 y + k_2 y^2$.

- **Columns:** $\mathbf{pc}, \mathbf{fp}$; operand o; immediate $(k_i)_{i\in\{0,1,2\}}$; memory count r; bytecode count r_{bc}.
- **Constraints:** none.
- **Flushes:**
  - **State:** PULL $(\mathsf{ST}, \mathbf{pc}, \mathbf{fp})$, PUSH $(\mathsf{ST}, \mathsf{g} \cdot \mathbf{pc}, \mathbf{fp})$.
  - **Bytecode:** $\mathsf{read}\,(\mathsf{BC}, \mathbf{pc}, r_{\mathrm{bc}}, \mathsf{op}_{\mathsf{SET}}, o, k_0, k_1, k_2, 0)$.
  - **Memory:** $\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp} \cdot o, r, k_0, k_1, k_2)$.

7.4 DEREF

DEREF $[o_1, o_2, o_3;$ mode$]$ reads a pointer and asserts the cell it points to. The row reads three cells:

- the *pointer* p, at $\mathbf{fp} \cdot o_1$;
- the *target* v_2, at the dereferenced address $p \cdot o_2$;
- the *local* value v_3, at $\mathbf{fp} \cdot o_3$.

The store asserts that v_2 equals the source chosen by the mode mode (§2): the local cell v_3, the return address $\mathsf{g}^2 \cdot \mathbf{pc}$, or the frame pointer $\mathbf{fp}$. The mode is two boolean flags $(f_{\mathbf{pc}}, f_{\mathbf{fp}})$: $(0,0)$ for deref_cell$[o_3]$, $(1,0)$ for deref_pc, $(0,1)$ for deref_fp.

- **Columns:** $\mathbf{pc}, \mathbf{fp}$; operands o_1, o_2, o_3; flags $f_{\mathbf{pc}}, f_{\mathbf{fp}}$; pointer p (a single K-limb); the local value $(v_{3,i})_{i\in\{0,1,2\}}$; memory counts r_1, r_2, r_3; bytecode count r_{bc}.
- **Constraints:** none.
- **Flushes:**
  - **State:** PULL $(\mathsf{ST}, \mathbf{pc}, \mathbf{fp})$, PUSH $(\mathsf{ST}, \mathsf{g}\cdot\mathbf{pc}, \mathbf{fp})$.
  - **Bytecode:** $\mathsf{read}\,(\mathsf{BC}, \mathbf{pc}, r_{\mathrm{bc}}, \mathsf{op}_{\mathsf{DRF}}, o_1, o_2, o_3, f_{\mathbf{pc}}, f_{\mathbf{fp}})$.
  - **Memory**, the target giving v_3, $\mathsf{g}^2\cdot\mathbf{pc}$ or $\mathbf{fp}$ at the three flag settings:

$$\begin{aligned}&\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp}\cdot o_1, r_1, p, 0, 0),\\ &\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp}\cdot o_3, r_3, v_{3,0}, v_{3,1}, v_{3,2}),\\ &\mathsf{read}\,(\mathsf{MEM}, p\cdot o_2, r_2, \bar{f}\,v_{3,0} + f_{\mathbf{pc}}\,(\mathsf{g}^2\cdot\mathbf{pc}) + f_{\mathbf{fp}}\,\mathbf{fp},\ \bar{f}\,v_{3,1},\ \bar{f}\,v_{3,2}),\qquad \bar{f} = 1 + f_{\mathbf{pc}} + f_{\mathbf{fp}}.\end{aligned}$$

7.5 JUMP

JUMP $[o_c, o_d, o_f]$ performs a conditional jump on whether the condition $v_{\mathrm{cond}} = \mathbf{m}[\mathbf{fp}\cdot o_c]$ is nonzero. It transfers to $\mathbf{pc} \leftarrow v_{\mathbf{pc}} = \mathbf{m}[\mathbf{fp}\cdot o_d]$, $\mathbf{fp} \leftarrow v_{\mathbf{fp}} = \mathbf{m}[\mathbf{fp}\cdot o_f]$ when $v_{\mathrm{cond}} \neq 0$, and defaults to $\mathbf{pc} \leftarrow \mathsf{g}\cdot\mathbf{pc}$, $\mathbf{fp} \leftarrow \mathbf{fp}$ otherwise. $v_{\mathrm{cond}}, v_{\mathbf{fp}}, v_{\mathbf{pc}}$ must be K-valued. The branch is taken according to a boolean indicator column $b = [\,v_{\mathrm{cond}} \neq 0\,]$, certified by a prover-supplied inverse w and 2 constraints:

- **Columns:** $\mathbf{pc}, \mathbf{fp}$; operands o_c, o_d, o_f; the condition v_{cond}, the destination $v_{\mathbf{pc}}$ and the frame $v_{\mathbf{fp}}$; inverse w (honest prover sets $w = v_{\mathrm{cond}}^{-1}$ if $v_{\mathrm{cond}} \neq 0$, otherwise $w = 0$); boolean indicator b; memory counts r_c, r_d, r_f; bytecode count r_{bc}.
- **Constraints:** $b = v_{\mathrm{cond}}\cdot w$ and $v_{\mathrm{cond}}\,(b+1) = 0$
- **Flushes:**
  - **State:** PULL $(\mathsf{ST}, \mathbf{pc}, \mathbf{fp})$, PUSH $(\mathsf{ST}, b\,v_{\mathbf{pc}} + b\,(\mathsf{g}\cdot\mathbf{pc}) + \mathsf{g}\cdot\mathbf{pc},\ b\,v_{\mathbf{fp}} + b\,\mathbf{fp} + \mathbf{fp})$.
  - **Bytecode:** $\mathsf{read}\,(\mathsf{BC}, \mathbf{pc}, r_{\mathrm{bc}}, \mathsf{op}_{\mathsf{JMP}}, o_c, o_d, o_f, 0, 0)$.
  - **Memory:**

$$\begin{aligned}&\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp}\cdot o_c, r_c, v_{\mathrm{cond}}, 0, 0),\\ &\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp}\cdot o_d, r_d, v_{\mathbf{pc}}, 0, 0),\\ &\mathsf{read}\,(\mathsf{MEM}, \mathbf{fp}\cdot o_f, r_f, v_{\mathbf{fp}}, 0, 0).\end{aligned}$$

7.6 KECCAK

One row is one permutation (§2), proved by the GKR of Annex C rather than by constraints. A row has 115 columns:

- $\mathbf{pc}$, $\mathbf{fp}$, the eight operand offsets, the two layout flags and *free*, and 27 read counts, one per cell read or written and one for the bytecode.
- in_ℓ, the 25 lanes of the state the row reads from memory.
- mid_ℓ, the 25 lanes of the state after 12 rounds. Committing it is an optimization and nothing more: it halves the depth of the reduction. No flush reads it.

- out_ℓ, the 25 lanes the row writes back. These are *virtual*: the flushes read them, but nothing commits them, and the claims the table sumcheck leaves on them are deferred to the GKR, which reduces them to claims on in_ℓ and mid_ℓ.

A limb holding lane a in the even layout and b in the odd one is the degree-2 form $a + \mathit{odd}\,(a+b)$, a limb holding no lane being the column *free* when read and 0 when written. The top limb is a literal 0.

- **Constraints:** none. The permutation is proved by the GKR of Annex C.
- **Flushes:**
  - **State:** PULL $(\mathsf{ST}, \mathbf{pc}, \mathbf{fp})$, PUSH $(\mathsf{ST}, \mathsf{g}\cdot\mathbf{pc}, \mathbf{fp})$.
  - **Bytecode:** $\mathsf{read}\,(\mathsf{BC}, \mathbf{pc}, r_{\mathrm{bc}}, \mathsf{op}_{\mathsf{KEC}}, o_{m_0}, o_{m_1}, o_{m_2}, o_{m_3}, o_{tail}, o_{b8}, o_{cap}, o_{out}, \mathit{odd}_{in}, \mathit{odd}_{out})$.
  - **Memory:** cell $c < 13$ is read at a_c under odd_{in} and written at $\mathsf{g}^c\mathbf{fp}\cdot o_{out}$ under odd_{out}, where

$$a_c = \begin{cases} \mathbf{fp}\cdot o_{m_c} & c < 4, \\ \mathsf{g}^{c-4}\mathbf{fp}\cdot o_{tail} & 4 \le c < 8, \\ \mathbf{fp}\cdot o_{b8} & c = 8, \\ \mathsf{g}^{c-9}\mathbf{fp}\cdot o_{cap} & 9 \le c < 13, \end{cases}$$

$$\begin{aligned} &\mathsf{read}\,(\mathsf{MEM}, a_c, r_c, u_c, v_c, 0) && (c < 9), \\ &\mathsf{read}\,(\mathsf{MEM}, a_c, r_c, \mathsf{in}_{2c-1}, \mathsf{in}_{2c}, 0) && (9 \le c < 13), \end{aligned}$$

  the writes the same with out_ℓ and r_{13+c}. The rate cells take the forms u_c, v_c above; the four capacity cells hold lanes 17 to 24 in either layout.

8 End-to-end protocol

8.1 Bytecode encoding

The bytecode (of $N_{\mathrm{prog}} = 2^{\kappa_{\mathrm{bc}}}$ instructions) is encoded as one K-valued multilinear P in $\kappa_{\mathrm{bc}} + 4$ variables:

1. lay instruction z out as 16 slots of K: the opcode in slot 3, the operands and immediate lanes in slots $4, \ldots, 13$, and 0 everywhere else.
2. read the resulting $2^{\kappa_{\mathrm{bc}}} \times 16$ table as P, the low κ_{bc} variables indexing the instruction and the high four its slot, bits low first.

Instruction	3	4	5	6	7	8	9	10	11	12	13
`XOR`	$\mathsf{op}_{\mathsf{XOR}}$	o_A	o_B	o_C	0	0	0	0	0	0	0
`MUL_NATIVE`	$\mathsf{op}_{\mathsf{MUL}}$	o_A	o_B	o_C	0	0	0	0	0	0	0
`SET_CONSTANT`	$\mathsf{op}_{\mathsf{SET}}$	o	k_0	k_1	k_2	0	0	0	0	0	0
`DEREF`	$\mathsf{op}_{\mathsf{DRF}}$	o_1	o_2	o_3	$f_{\mathbf{pc}}$	$f_{\mathbf{fp}}$	0	0	0	0	0
`JUMP`	$\mathsf{op}_{\mathsf{JMP}}$	o_c	o_d	o_f	0	0	0	0	0	0	0
`KECCAK`	$\mathsf{op}_{\mathsf{KEC}}$	o_{m_0}	o_{m_1}	o_{m_2}	o_{m_3}	o_{tail}	o_{b8}	o_{cap}	o_{out}	odd_{in}	odd_{out}

So $P(z, 1, 1, 0, 0)$ is instruction z's opcode, $P(z, 0, 0, 1, 0)$ its first operand, and $P(z, 0, 0, 0, 0) = P(z, 1, 1, 1, 1) = 0$.

8.2 Public input

The first two memory cells must hold the public input, $\mathbf{m}[\mathsf{g}^0] = \mathsf{input}_0 + \mathsf{input}_1 y$ and $\mathbf{m}[\mathsf{g}^1] = \mathsf{input}_2 + \mathsf{input}_3 y$ (§2). They sit at $(\underbrace{0,\dots,0}_{\kappa_{\mathrm{mem}}})$ and $(1,\underbrace{0,\dots,0}_{\kappa_{\mathrm{mem}}-1})$, so it suffices to check the following (degree 1) polynomial equality:

$$\widetilde{\mathbf{m}}(X,0,\dots,0) \;=\; (1+X)(\mathsf{input}_0 + \mathsf{input}_1 y) + X(\mathsf{input}_2 + \mathsf{input}_3 y).$$

Recall that the memory $\mathbf{m} \in E^{\le 1}[X_1,\dots,X_{\kappa_{\mathrm{mem}}}]$ is committed as three limbs $\mathbf{m}_0, \mathbf{m}_1, \mathbf{m}_2 \in K^{\le 1}[X_1,\dots,X_{\kappa_{\mathrm{mem}}}]$, with $\mathbf{m} = \mathbf{m}_0 + y\,\mathbf{m}_1 + y^2\,\mathbf{m}_2$ (§6.3). Since $1, y, y^2$ is a basis of E over K, the equality holds, as polynomials in X, exactly when it holds limb by limb, and every limb's right side is a line the verifier knows. The verifier samples $r_m \in E$, computes

$$c_0 = (1+r_m)\,\mathsf{input}_0 + r_m\,\mathsf{input}_2, \qquad c_1 = (1+r_m)\,\mathsf{input}_1 + r_m\,\mathsf{input}_3,$$

and pools the claims $c_0, c_1, 0$ on $\mathbf{m}_0, \mathbf{m}_1, \mathbf{m}_2$ at $(r_m, 0, \dots, 0)$, the last being 0 because both public cells have top limb zero. The prover sends nothing: the claims are functions of the public input, which seeds the transcript, and of r_m, drawn after the commitment. If either cell is wrong, some limb's two sides are distinct polynomials of degree at most one in r_m, so its claim is true with probability at most $1/|E|$ (Lemma 3.9), and the opening rejects a false claim.

8.3 Padding rows

An honest prover fills the padding rows of T_j with the public row $\mathbf{p}_j$: zero, except the twenty-five `KECCAK` output lanes (§7.6), which hold the instruction applied to a zero state. This is only for completeness: $\mathbf{p}_j$ satisfies the constraints (`JUMP`'s vanish at zero), and the Keccak reduction (Annex C) proves that permutation like any other. When the recursion guest needs a larger commitment (§10.1), the prover grows the memory, not a table.

The bus does not depend on what a padding row holds:

Proposition 8.1 (Padding rows flush nothing)**.** *Let P (resp. Q) be the multiset of tuples pushed (resp. pulled) by the executed rows and the boundary. If $P \neq Q$, the bus check passes with probability at most $4 \cdot 2^{\mu}/|E|$, plus the soundness errors of GKR, the table sumcheck and the PCS.*

Proof. GKR ends on a claim about the leaves L_b of §5.4, which the table sumcheck and the PCS opening check. The committed columns fix these leaves once α and β are drawn, before GKR's challenges, so up to those errors the bus check passes only if they have equal push and pull products.

Let x be a padding row of block b, whatever it holds. Block b flushes the tuple $\mathbf{f}_{j,f}$, and $\phi_b(x) = \beta - \pi_\alpha(\mathbf{f}_{j,f}(x))$, where π_α is the fingerprint of §5.2. Let $\mathbf{d} = \mathbf{f}_{j,f}(x) - \mathbf{f}_{j,f}(\mathbf{p}_j)$ be the difference between the row's tuple and the honest one. Since π_α is linear, the leaf of x is

$$L_b(x) \;=\; 1 + \phi_b(x) - \phi_b(\mathbf{p}_j) \;=\; 1 - \pi_\alpha(\mathbf{d}).$$

It does not depend on β, and it is a nonzero polynomial in α: the weights $\mathrm{eq}(\alpha, \iota)$ sum to 1, so $L_b(x) = \pi_\alpha(\mathbf{1} - \mathbf{d})$ with $\mathbf{1}$ the all-one tuple, and $\mathbf{1} - \mathbf{d}$ is 1 in slot 0, where both tuples hold the same domain separator.

So the push product is $\Pi^{\mathsf{pad}}_{\mathrm{PUSH}}(\alpha)\,\Pi_P(\alpha,\beta)$, where $\Pi^{\mathsf{pad}}_{\mathrm{PUSH}}$ multiplies the padding leaves, a nonzero polynomial in α alone, and Π_P multiplies the leaves $\beta - \pi_\alpha(t)$, $t \in P$ (Lemma 5.2). Likewise, the pull product is $\Pi^{\mathsf{pad}}_{\mathrm{PULL}}\,\Pi_Q$. Suppose $P \neq Q$, and that the two products are the same polynomial. Their highest powers of β then match: $\Pi^{\mathsf{pad}}_{\mathrm{PUSH}}\beta^{|P|}$ and $\Pi^{\mathsf{pad}}_{\mathrm{PULL}}\beta^{|Q|}$, so $|P| = |Q|$ and $\Pi^{\mathsf{pad}}_{\mathrm{PUSH}} = \Pi^{\mathsf{pad}}_{\mathrm{PULL}}$.

Dividing by it gives $\Pi_P = \Pi_Q$, so $P = Q$ by Lemma 5.2, a contradiction. The two products are thus different polynomials, of degree at most $4 \cdot 2^\mu$, and agree at the random (α, β) with probability at most $4 \cdot 2^\mu / |E|$ (Corollary 3.10). □

The count tree pads the same way, and whatever its padding leaves are, $R_c \neq 0$ still forces every executed row's counts to be nonzero (§6.2).

8.4 Fiat Shamir instantiation

The transcript is a duplex sponge [BDPA11] in overwrite mode, after Plonky3's duplex challenger [Pol], over Keccak-f[1600] (§2). The *rate* is lanes $0, \ldots, 15$ and the *capacity* lanes $16, \ldots, 24$, 576 bits.

Seed. The first permutation takes, from the zero state, the 256-bit IV in lanes $0, \ldots, 3$ and the 256-bit public input in lanes $4, \ldots, 7$. The IV is the hash of a fixed tag naming the proof system's version, `leanvm-v1`, followed by the bytecode: the tag has a fixed length and comes first, and the padding binds the preimage's length, so no length field is needed. The statement is bound before any challenge is drawn.

Absorbing. A scalar of E sent by the prover waits until a permutation takes it, at most four of them. A permutation writes the i-th waiting scalar over lanes $4i, \ldots, 4i + 3$, as its three K-limbs followed by a domain tag (1 for a sent scalar, 4 for a grinding nonce), zeroes the rest of the rate and keeps the capacity. A fifth scalar first permutes the four waiting. The tags are nonzero, so a sent zero never reads as the zeroed rate.

Squeezing. Every challenge is a permutation of its own, taking whatever waits; the challenge is the first three lanes of its output, read as an element of E. So a round's message and its challenge cost one permutation, and the recursion guest replays a permutation as one `KECCAK` instruction whose message cells are the waiting scalars and whose capacity cells are the previous output's.

Grinding. A proof of work of b bits absorbs the prover's nonce with its tag and draws a challenge, whose low b bits must be zero; that challenge is used for nothing else. With $b = 0$ the nonce must be zero.

Merkle trees hash with Keccak-256, apart from the sponge.

8.5 The unrolled protocol

Each reduction below adds evaluation claims on columns to a shared *claim pool*, proved by the final PCS Opening phase.

Setup. The Fiat-Shamir initial state (§8.4) is seeded by the public input and by the IV, the hash of the version tag and the bytecode. The prover sends the memory log-size κ_{mem}, the six row counts $N_{\text{exec},j}$ and the rate exponent $\nu_0 \in \{1, \ldots, 4\}$ of the commitment (Annex B); the verifier checks them against the caps (§6.2) and derives the layout of every stack (PCS and GKR leaves).

Commitment.

1. **Prover.** Stack (§4.1) every column, the three memory limbs, the two finalize (memory / bytecode) counts, and the committed Keccak states q_{keccak} (§7.6), into q and sends its commitment (Annex B).

2. **Verifier.** Receives the commitment to q.

Bus.

1. **Verifier.** Samples and sends the fingerprint challenge $\alpha \in E^4$ and the multiset challenge $\beta \in E$ (§5.2).

2. **Prover.** Forms the push, pull, and *count* leaf vectors (§5.4; on executed rows, the count leaves are the count columns themselves, §6.2), stacked into blocks and padded with 1s. Sends one bus root R, then the count root R_c.

3. **Prover & Verifier.** Run the one batched GKR (§5.3) over all three trees. For the bus, the verifier checks each side's product against R; one R for both sides is the balance check (§5.2). For the counts it checks $R_c \neq 0$ (no read self-cancels, §6.2). The pass yields three leaf claims $\widetilde{V}_0^s(\zeta)$ at one shared ζ.

4. **Prover & Verifier.** Decompose each leaf claim (§5.4). The verifier forms every public part itself: the block selectors, the constant entries, the index column (§6.5), the padding term of (4), and the program's whole share of the two bytecode blocks, one evaluation at (ζ, α) (§8.1). Three blocks per side belong to no table: the state boundary, plus each array's (memory / bytecode) seed (push side) or finalization (pull side). For their committed columns the prover sends one evaluation each, entering the claim pool. Every other block belongs to a table, and is discharged by the table sumcheck (§5.5).

Table sumcheck. One run proves the six tables' constraints together with their share of the bus (§5.5).

1. **Verifier.** Samples and sends the batch challenge $\xi \in E$. Its first $N_{\text{cons}} = \sum_j N_{\text{cons},j}$ powers weight the constraints, a disjoint range per table; the last N_{side} weight the bus forms, one per side and shared by every table. It draws no point: the batch runs at the bus point, T_j tested at $\zeta_{<\tau_j}$.

2. **Prover & Verifier.** Run the sumcheck, $\tau_{\max} = \max_j \tau_j$ rounds with T_j joining at round $\tau_{\max} - \tau_j$, against the target the verifier derived from the three leaf claims. Its summand has individual degree three and the round check pins one of the four coefficients (§3), so a round costs three E-elements.

3. **Prover & Verifier.** The prover sends one value per column of every table. The verifier rebuilds each $C_{j,i}$ and B_j^s from them and checks that they reproduce the sumcheck's final value; the column values then enter the claim pool, T_j's at $\chi_{<\tau_j}$. `KECCAK`'s fifty value lanes are columns of its table but have no region of their own in the stack. An input lane is a slot of q_{keccak}, so its claim is a claim on q_{keccak} at the same point followed by the slot's bits (§7.6). The claims on the output lanes leave the pool: they are the Keccak reduction's claims on the output.

Public input.

1. **Verifier.** Samples and sends $r_m \in E$.
2. **Verifier.** Computes c_0, c_1 from the public input and pools $(c_0, c_1, 0)$ as claims on $\mathbf{m}_0, \mathbf{m}_1, \mathbf{m}_2$ at $(r_m, 0, \ldots, 0)$ (§8.2). The prover sends nothing.

KECCAK validity.

1. **Verifier.** Samples and sends the first claim's challenges, $r_{\text{seg}} \in E$ and $r_{\text{lane}} \in E^5$. The claim is made at the `KECCAK` table's point $\chi_{<\tau_{\text{keccak}}}$ (§C.6, §C.7).
2. **Prover & Verifier.** The prover sends the part of the first claim on the committed state after twelve rounds; the part on the output is the table's claims on its output lanes. The parties run the layers of the reduction with the middle state committed (§C.6, §C.7): twelve, one per round of a segment, which take the claim to one on the committed states twelve rounds earlier (§C.6).
3. **Prover & Verifier.** Open the part on the committed state: a sumcheck over the slot leaves one evaluation of q_{keccak}, an ordinary column claim, which joins the pool last (§C.7).
4. **Prover & Verifier.** Open the last claim: a sumcheck over the permutations of a pack, then over the slot, leaves the 64 bit slices of q_{keccak} at one point, which the prover sends (§C.6).
5. **Prover & Verifier.** The verifier samples the six challenges of the ring-switching map, and the ring switching of §4.2 reduces those 64 values to a weighted-sum claim on the q_{keccak} region of the stack, with an MLE-friendly E-valued weight (Definition 3.14).

Opening.

1. **Prover & Verifier.** Each pooled claim is one evaluation of a column at a point, its value already supplied; via the stacking selectors (§4.1) it becomes a weighted sum $\sum_w W_j(w)\,\widetilde{q}(w) = c_j$ over the stacked witness. (The ring-switched claim arrives already in this form, its weight supported on the q_{keccak} region.)
2. **Verifier.** Sends one batching challenge $\lambda \in E$, drawn after every claim value is bound. Claim j takes the weight λ^{j-1}, the ring-switched claim first and the pooled column claims after: the J weights are distinct powers of the one challenge (Theorem B.7).
3. **Prover & Verifier.** Fold the J claims into $W_\lambda = \sum_j \lambda^{j-1} W_j$ and $C_\lambda = \sum_j \lambda^{j-1} c_j$, and run the inner-product PCS opening on $\sum_w W_\lambda(w)\,\widetilde{q}(w) = C_\lambda$ (Annex B), the verifier evaluating W_λ itself. This one opening discharges every pooled claim; there is no separate reduction sumcheck.
4. **Verifier.** Accepts if every check above passed: the caps, the one bus root for both sides, the nonzero count root, every sumcheck's rounds and final value, the Keccak reduction's checks, and the PCS opening.

9 ISA programming

How the zkDSL compiler (`crates/lean_compiler`; surface language in `crates/lean_compiler/zkDSL.md`) programs the six-instruction ISA. Two themes run through every pattern: *write-once memory is an assertion mechanism* (a second write of a cell is an equality check, an unwritten cell is prover-chosen), and *indices live in the exponent* (a logical index i is the element g^i, so incrementing, offsetting, and address formation are single field multiplications).

9.1 Hints

The prover's nondeterminism enters through write-once cells that are still unset when execution needs them. Some are explicit advice (a fresh allocation pointer g^{base}, a witness stream, a computed-advice builtin); others are back-solved by an instruction whose remaining operands are known. In every case the resulting word is unconstrained except by the instructions and memory equalities that consume it, so a program must check all security-relevant advice.

9.2 Division and inequality

The zkDSL's single-slash division $q = a/b$ uses write-once back-solving: it emits `MUL_NATIVE` with q as the one unset input and a as the already-written output. Witness generation fills $q = a\,b^{-1}$, while the ordinary multiplication table proves $qb = a$. It costs one VM instruction. Division by zero is undefined. Double slash `//` remains compile-time integer floor division for sizes and indices; it is not a field operation.

The assertion `assert a != b` computes $x = a + b$ with `XOR`, takes a hinted $w = x^{-1}$, forms $p = x\,w$ with `MUL_NATIVE`, and writes 1 to p with `SET_CONSTANT`. Memory being write-once, the two writes to p agree only if $p = 1$, and $x = 0$ forces $p = 0$ whatever the hint.

9.3 Functions

A frame is fp-relative with layout $[\mathit{retpc}, \mathit{retfp}, \text{args}, \text{rets}, \text{locals}]$. A call: the callee frame pointer is hinted into a fresh cell *nfp*; the arguments and the caller's $\mathbf{fp}$ are stored through *nfp* with `DEREF`s (modes `cell` and `fp`); a `DEREF` in mode `pc` stores the return address $\mathsf{g}^2 \cdot \mathbf{pc}$ (the instruction after the transfer); one `JUMP` enters the callee. Return is a single `JUMP` to $\mathbf{m}[\mathbf{fp} \cdot 1]$ with frame pointer $\mathbf{m}[\mathbf{fp} \cdot \mathsf{g}]$. Cost: $n_{\text{args}} + n_{\text{rets}} + 4$ instructions per call.

The callee frame pointer being hinted, the following convention is crucial: *a function's internal logic should only depend on its inputs, not on where its frame happens to land*, so that the value of *nfp* does not impact its execution.

The ISA makes it possible to use values of $\mathbf{fp}$ outside the range $\{\mathsf{g}^0, \ldots, \mathsf{g}^{2^{\kappa_{\text{mem}}}-1}\}$. As long as every memory access of the form $\mathbf{m}[\mathbf{fp} \cdot o]$ (for some offset o) is in range, i.e. $\mathbf{fp} \cdot o = \mathsf{g}^i$ with $i < 2^{\kappa_{\text{mem}}}$, the execution is valid. Yet, reasoning about such out-of-range values of $\mathbf{fp}$ is tricky. The initial $\mathbf{fp}$ is in range, since the state boundary condition fixes $\mathbf{fp}_0 = \mathsf{g}^0$ (§6.1), but nothing guarantees it for the ones a `JUMP` sets. We thus suggest the following programming convention: whenever a `JUMP` updates $\mathbf{fp}$, the program should guarantee that the new value is in range. The calling convention above does it for free:

- On a call, $\mathbf{fp} \leftarrow \mathbf{m}[\mathbf{fp} \cdot o_f]$, where $\mathbf{m}[\mathbf{fp} \cdot o_f] = \mathit{nfp}$ is hinted, hence arbitrary. But the callee frame was initialized with the `DEREF` $\mathbf{m}[\mathit{nfp}] \leftarrow \mathsf{g}^2 \cdot \mathbf{pc}$, whose memory access at address *nfp* forces $\mathit{nfp} = \mathsf{g}^i$ with $i < 2^{\kappa_{\text{mem}}}$ (Corollary 6.4).

- On a return, $\mathbf{fp} \leftarrow \mathbf{m}[\mathbf{fp} \cdot \mathsf{g}]$ reads back what the caller wrote there, i.e. its own **fp**, in range by induction.

9.4 Loops

A counted loop keeps its counter in the exponent: it starts at g^{lo}, advances by one multiplication $i \leftarrow \mathsf{g} \cdot i$, and exits on reaching g^{hi}. The body is lowered as a tail-recursive helper function whose exit test *is* the recursive call's `JUMP` condition (the `XOR` $i + \mathsf{g}^{\mathrm{hi}}$ is nonzero exactly while the loop must continue), so an iteration costs one call plus two instructions, with no is-zero gadget. Loop state is threaded through captured arguments or a heap buffer.

9.5 Conditionals

`if/else` on a field equality is one `XOR` plus one conditional `JUMP`: the taken jump goes to whichever block the test should *not* fall into (= falls into *then*, $\neq$ into *else*), so no negation is ever computed. Intra-function jump targets are bytecode addresses $\mathsf{g}^{\mathrm{entry}+i}$, backpatched at layout.

A taken `JUMP` reloads **fp** from a cell, and the ISA has no **fp**-read; a branching function therefore materializes its own **fp** once: a `DEREF` in mode `fp` writes **fp** into a fresh one-cell heap buffer, a `DEREF` in mode `cell` copies it back into the frame (2 instructions, amortized; in `main`, $\mathbf{fp} = \mathsf{g}^0 = 1$ is already a constant). Bindings made inside a branch are local to it; branches communicate through write-once cells: only one branch executes, so both may write the *same* cell.

9.6 Checking that a value is in the subfield K

To check that $\mathbf{m}[\mathbf{fp} \cdot o_1]$ and $\mathbf{m}[\mathbf{fp} \cdot o_2]$ are both in K, it suffices to emit `JUMP` $[o_0, o_1, o_2]$ with $\mathbf{m}[\mathbf{fp} \cdot o_0] = 0$. Indeed, the jump falls through, while `JUMP` unconditionally requires all three values it reads to be in K.

9.7 Range checks

The range check *in the exponent* proves $x \in \{\mathsf{g}^0, \ldots, \mathsf{g}^{k-1}\}$, i.e. $\log_{\mathsf{g}} x < k$, in 3 cycles:

1. `DEREF` through x: the dereferenced address $x \cdot \mathsf{g}^0$ rides the memory interaction, which balances only for the $2^{\kappa_{\mathrm{mem}}}$ addresses $\mathsf{g}^0, \ldots, \mathsf{g}^{2^{\kappa_{\mathrm{mem}}}-1}$ (§6.3): the bus itself proves $x = \mathsf{g}^e$ with $e < 2^{\kappa_{\mathrm{mem}}}$;

2. `MUL` $x \cdot y$ into a write-once cell holding the constant g^{k-1}: the runner back-solves the complement $y = \mathsf{g}^{k-1-e}$ (§9.1), and the double write asserts $x \cdot y = \mathsf{g}^{k-1}$;

3. `DEREF` through y: proves $y = \mathsf{g}^f$, $f < 2^{\kappa_{\mathrm{mem}}}$.

Then $e + f \equiv k - 1 \pmod{2^{64} - 1}$ with $e, f < 2^{\kappa_{\mathrm{mem}}} \leq 2^{32}$, so $e + f < 2^{33} < 2^{64} - 1$ and the congruence is an equality over the integers; a "negative" $k - 1 - e$ would wrap to $\approx 2^{64} \gg 2^{\kappa_{\mathrm{mem}}}$, so $e \leq k - 1$, for every memory size the prover may announce, provided $k \leq 2^{16}$ (the minimum, §2). The two `DEREF`s only touch their targets, whose values nothing reads; the constant cell is one amortized `SET` per distinct bound per frame.

9.8 Hashing with Keccak

The machine's hash is Keccak-256 [BDPV11], Ethereum's: the 256-bit hash of FIPS 202 [Nat15] with padding first byte `0x01` instead of `0x06`. The message is padded with `01 00`* `80` (or `81`) to a multiple of 136 bytes, and each block is XORed into lanes $0, \ldots, 16$ before a permutation.

A block is 8.5 cells: block k of a message in consecutive cells is the nine cells from $\lfloor 17k/2 \rfloor$, starting at the high limb of the first when k is odd. These cells line up with the state cells of the layout of k's parity (§2), the extra half cell falling on the free limb. So block k costs at most nine `XOR`s into the previous output, written in that layout, and one `KECCAK` reading it in that layout. The first block is the state itself, with a zero capacity, and the last writes the even layout, whose cells $0, 1$ are the digest. A padding byte is a constant cell, or one `XOR` into a message cell. Constant cells are set once per frame, the zero capacity and the common tail sharing the run $[\texttt{0x01}, 0, 0, 0, 0]$; they are never left unwritten, since an unwritten cell holds whatever the prover chooses.

A list of n claims, 64 bytes each, has runtime length. Seventeen claims are four whole block pairs, so a loop absorbs seventeen claims per iteration, carrying the state through a heap chain, and a `match` on the $r < 17$ remaining claims places the padding. The split $n = 17b + r$ is prover advice, pinned by the range checks $r < 17$, $b < 2^{16}$ and $\mathtt{g}^{17b+r} = \mathtt{g}^n$.

9.9 Match statements

A `match` on $x = \mathtt{g}^j$, $j < n$, runs block j through a jump table: n cells, cell j holding block j's address. Tables live in `main`'s frame, which is always at address $\mathtt{g}^0$, so cell j of a table at offset t is at $x \cdot \mathtt{g}^t$. A `DEREF` reads that cell and a `JUMP` goes there: 3 cycles, for any n. The program fills all tables before `main` starts, one `SET` per cell. The dispatch does not check $j < n$. Untrusted values (like hints) of x must be range-checked first (§9.7).

10 Ethereum use cases

An aggregate proof certifies signature claims and the correct encoding of blob data. One guest program establishes these claims by checking raw signatures, checking a blob matrix, or recursively verifying child proofs of the same program.

10.1 Recursive aggregation

A signature claim identifies what was signed and by whom. For both leanXMSS, two layers of XMSS [HBG+18], and SPHINCS+ [BHK+19], it is a (public key, message) pair. Each leanXMSS signature carries the leaf index it was made at, so its Merkle path is its own, and the guest derives that signature's addresses from the index's bits; the claim does not record it.

A node may publish any subset of the signature claims supported by its raw signatures and verified children. The default is their sorted, deduplicated union. The essential condition is coverage: every published claim must be justified, either directly at this node or recursively through a child.

The DA claim is at most one LeanDA root `root`, paired with the challenge `z` of its membership test (§10.2.2). A node that checks blobs publishes their root; otherwise it may pass on one child's root, or none.

10.1.1 The public input

Digests bind the two variable-length claim lists into a fixed-size statement. The VM's public input is the hash (Keccak-256, §9.8) of the following fields, in order:

32 bytes	Fiat-Shamir IV
32 bytes	leanXMSS claim list digest
32 bytes	leanSPHINCS claim list digest
32 bytes	DA digest $H(\mathsf{root}, \mathsf{z})$ (zeros if none)
$(\kappa_{\mathrm{bc}} + 5) \times 24$ bytes	deferred bytecode point and value

The deferred claim lets recursion postpone an expensive polynomial evaluation until the final verification (§10.1.3).

Each list digest is the hash of that scheme's sorted claims, 64 bytes per (public key, message) pair (§9.8). Every field above has a fixed width, so the preimage parses one way and the statement needs no length fields.

10.1.2 Checking coverage

Every published claim needs support from a direct check or a recursively verified child. The guest builds a table with one region for leanXMSS and one for SPHINCS+. Each region starts with the claims the parent publishes, followed by extra slots for duplicates and omitted claims.

Each occurrence of a supporting claim needs its own slot. If two children prove the same published claim, one occurrence uses its published slot and the other uses an extra slot. Every occurrence of a claim the parent omits uses an extra slot. These claims are still verified; only the published prefixes enter the parent's list digests.

The prover supplies the extra claims and assigns each supporting occurrence to a slot. The guest checks that they match, confines the assignment to the correct region, and writes a distinct counter value there. Write-once memory prevents slot reuse. Requiring as many writes as slots ensures that every slot is covered, including every published claim.

10.1.3 Deferred evaluation claims

Verifying a child requires one evaluation of a fixed polynomial: the bytecode multilinear, in $\kappa_{\mathrm{bc}} + 4$ variables (§8.1). Computing it inside the guest would be very expensive. Instead, the guest carries one evaluation claim in its statement, a point and a value. This expensive computation is moved 'outside of the snark': only the root's claim is checked, natively, by the final verifier. Nothing else is deferred, the Keccak argument's weights being cheap to evaluate in the guest (Annex C).

A node with n children combines the claim each child carries with the new claim produced while verifying it, $2n$ claims on the one polynomial. One sumcheck over its $\kappa_{\mathrm{bc}} + 4$ variables reduces their random combination to a single evaluation, which is the claim the node carries. A node with no child carries the claim at the all-zeros point. The Fiat-Shamir for the batching and the sumcheck is seeded by every child's statement and both of its claims, the child count and each child's words padded with zeros to whole rates of the sponge (§8.4).

10.2 Data availability: LeanDA

Erasure coding makes data recoverable from a subset of its encoded symbols. Sampling checks whether those symbols can be retrieved, but does not establish that they form a valid encoding. An approach using leanVM was introduced in Risitano's presentation and developed in LeanDA [MWKR26] by Meng, Wagner, Kadianakis and Risitano: commit to an encoded matrix and prove every row is a codeword, then sample authenticated cells. A formal security proof can be found in [Wag26].

Our construction uses Keccak-256 and the additive Reed-Solomon encoder over K already used by the PCS, with a different commitment (§10.2.1) and codeword membership test (§10.2.2) from LeanDA's. The polynomial basis, transform, and membership lemmas are collected in Annex D.

10.2.1 The commitment

A blob is a polynomial in $K[X]$ of degree less than k, represented by its k evaluations on $\mathcal{U}_{\log_2 k}$. Its Reed-Solomon encoding is the vector of evaluations on the larger domain $\mathcal{D} = \mathcal{U}_{\log_2 n}$, where $n = 2k$. The first k symbols are the original blob, so the encoding is systematic; the remaining symbols provide redundancy. Any k encoded evaluations recover the blob.

One commitment covers a nonempty matrix of N_{blob} encoded blobs, one per row.

Split each encoded row into cells of N_{cell} symbols. The guest fixes these dimensions, matching the byte sizes of an EIP-4844 blob and an EIP-7594 sampling cell:

k	2^{14} symbols	128 KiB of payload per row
n	2^{15} symbols	256 KiB per encoded row
N_{cell}	2^8 symbols	2 KiB per cell
n/N_{cell}	128 cells	any 64 reconstruct a row
N_{blob}	1 to 1024	dynamic parameter

Zero codewords pad the row count to the next power of two. The root binds this padded matrix.

Let H denote the hash and $e_{i,j}$ the hash of cell j in row i. Each row has a Merkle tree of its own: its leaves are $e_{i,0}, \ldots, e_{i,127}$, each inner node is the hash of its two children, and its root is root_i. The left child of root_i covers the first k/N_{cell} cells, the original blob. The commitment root is the root of the Merkle tree whose leaves are the row roots of the padded matrix, each padding row having the zero codeword's root (Figure 1). A sampled cell is authenticated by its path to its row's root, then that root's path to root.

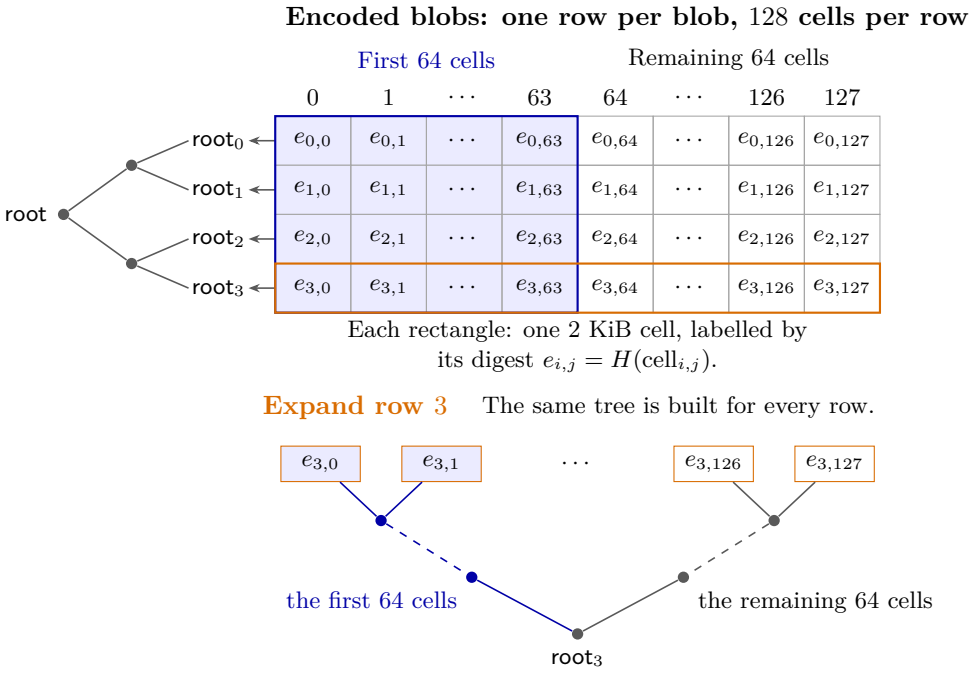


Figure 1: The commitment to 4 encoded blobs.

10.2.2 Codeword membership

A row is valid exactly when it consists of evaluations of a polynomial of degree less than k. On our additive domain at rate 1/2, such rows are orthogonal to every codeword, and this condition also characterizes them: the code is self-dual (Corollary D.9). This gives an efficient membership test.

The challenge is one element $\mathsf{z} \in E$, drawn from the VM's transcript (§8.4) with the hash of `leanDA/rs-membership/v1` as IV and root as public input. It defines L by (24) and $\mathbf{L} = (L(x))_{x \in \mathcal{D}}$. The statement carries $H(\mathsf{root}, \mathsf{z})$, z serialized as its three little-endian 64-bit limbs and a zero word.

The guest recomputes root, takes z as a hint, computes $\mathbf{L}$ and checks

$$\langle \mathbf{L},\, w_i \rangle = \sum_{x \in \mathcal{D}} L(x) w_i(x) \stackrel{?}{=} 0$$

for every row w_i. Factor j of (24) depends only on the bits of x from j up, so $\mathbf{L}$ is a product tree over those bits, built once for all rows.

The native verifier derives z from root outside the SNARK, as the security proof requires, and checks the proof against $H(\mathsf{root}, \mathsf{z})$. A node passing on a child's root passes its digest unchanged.

Every valid row passes. Under a uniform challenge, a fixed invalid matrix passes with probability at most $(k-1)/|E| < 2^{-178}$ (Proposition D.10).

A Ring switching

Ring switching was introduced in [DP26]. We present a variant tailored to our use case: transforming a PCS for K into a PCS for $\mathbb{F}_2$ (transforming a K-valued multilinear PCS, with κ_{keccak} variables and opening claims in E, into a commitment protocol to 64 $\mathbb{F}_2$-valued multilinears, each with κ_{keccak} variables and sharing opening claims in E).

A.1 Setting

The two fields are those of §2: $K = \mathbb{F}_2[x]/(x^{64} + x^4 + x^3 + x + 1)$ inside $E = K[y]/(y^3 + y + 1)$. As $\mathbb{F}_2$-vector spaces, K has basis $x^0, \ldots, x^{63}$, and E has the 192 products

$$e_w = x^i y^j \in E, \qquad w = 64j + i, \quad 0 \le i < 64, \quad 0 \le j < 3.$$

The witness is 64 bit-valued multilinears $Q_0, \ldots, Q_{63} : \{0,1\}^{\kappa_{\text{keccak}}} \to \{0,1\}$. We commit to them packed, one K element per point:

$$q_{\text{keccak}} : \{0,1\}^{\kappa_{\text{keccak}}} \to K, \qquad q_{\text{keccak}}(u) = \sum_{i=0}^{63} Q_i(u)\, x^i.$$

The commitment scheme proves statements of one shape. Given a target $T \in E$ and an MLE-friendly weight $W : \{0,1\}^{\kappa_{\text{keccak}}} \to E$ (Definition 3.14), an opening proves

$$\sum_{u \in \{0,1\}^{\kappa_{\text{keccak}}}} W(u)\, q_{\text{keccak}}(u) = T.$$

The weight W and target T are E-valued, the committed values are K-valued.

What arrives is not of that shape. It is 64 claimed values $s_i \in E$, one per bit polynomial, at a point $r \in E^{\kappa_{\text{keccak}}}$ they all share.

$$s_i \stackrel{?}{=} \widetilde{Q_i}(r) = \sum_{u \in \{0,1\}^{\kappa_{\text{keccak}}}} \text{eq}(r, u)\, Q_i(u), \qquad 0 \le i < 64,$$

Ring switching [DP26] turns those 64 claims into a single statement the opening can prove.

A.2 The reduction

Fix an $\mathbb{F}_2$-linear map $\Phi : E \to E$:

$$\begin{aligned}\sum_{u\in\{0,1\}^{\kappa_{\text{keccak}}}} \Phi\big(\mathrm{eq}(r,u)\big)\, q_{\text{keccak}}(u) &= \sum_{i=0}^{63} x^i \sum_{u\in\{0,1\}^{\kappa_{\text{keccak}}}} Q_i(u)\, \Phi\big(\mathrm{eq}(r,u)\big) \\ &= \sum_{i=0}^{63} x^i\, \Phi\Big(\sum_{u\in\{0,1\}^{\kappa_{\text{keccak}}}} Q_i(u)\ \mathrm{eq}(r,u)\Big) = \sum_{i=0}^{63} x^i\, \Phi(s_i).\end{aligned} \tag{8}$$

The reduction is:

1. once the s_i are fixed, the verifier draws Φ at random;
2. it computes the target $T = \sum_{i=0}^{63} x^i \Phi(s_i)$;
3. the commitment scheme opens q_{keccak} against the weight $W : \{0,1\}^{\kappa_{\text{keccak}}} \to E$, $W(u) = \Phi\big(\mathrm{eq}(r,u)\big)$, with target T; if that opening accepts, the verifier accepts the 64 claims.

Everything now turns on which Φ we can afford to draw.

A.3 Which maps are sound

The error, transposed. Suppose the s_i are wrong. The errors $\delta_i = s_i - \widetilde{Q_i}(r) \in E$, for $0 \le i < 64$, are not all zero, and they are fixed before Φ is drawn. By (8) the target the verifier computes differs from the honest one by $\sum_{i=0}^{63} x^i \Phi(\delta_i)$, so the opening succeeds only if

$$\sum_{i=0}^{63} x^i\, \Phi(\delta_i) = 0.$$

Expand each error in the $\mathbb{F}_2$-basis of E, say $\delta_i = \sum_{w=0}^{191} \delta_{i,w} e_w$ with $\delta_{i,w} \in \mathbb{F}_2$, and exchange the two sums:

$$\sum_{i=0}^{63} x^i\, \Phi(\delta_i) = \sum_{i=0}^{63} x^i \sum_{w=0}^{191} \delta_{i,w}\, \Phi(e_w) = \sum_{w=0}^{191} \Phi(e_w)\, \Delta_w, \qquad \Delta_w := \sum_{i=0}^{63} \delta_{i,w} x^i \in K. \tag{9}$$

The x^i are an $\mathbb{F}_2$-basis of K, so some $\Delta_w \neq 0$. The cheating prover succeeds if $\sum_{w=0}^{191} \Phi(e_w)\, \Delta_w = 0$. We must thus draw Φ to make this event unlikely.

Linear maps are Frobenius sums. Squaring is additive in characteristic 2, so

$$\Phi(a) = \sum_{k=0}^{191} c_k\, a^{2^k}, \qquad a \in E, \quad c_k \in E,$$

is $\mathbb{F}_2$-linear for any choice of coefficients. Every $\mathbb{F}_2$-linear map $E \to E$ arises this way, and exactly once. Distinct tuples of (c_k) give distinct maps: their difference would be a nonzero polynomial of degree at most 2^{191} with 2^{192} roots. There are $|E|^{192}$ tuples and as many maps. Call $\mathcal{S} = \{k : c_k \neq 0\} \subseteq \{0, \ldots, 191\}$ the *support* of Φ. Substituting into the check above,

$$\sum_{w=0}^{191} \Phi(e_w)\, \Delta_w = \sum_{w=0}^{191} \sum_{k\in\mathcal{S}} c_k\, e_w^{2^k}\, \Delta_w = \sum_{k\in\mathcal{S}} c_k\, \Omega_k, \qquad \Omega_k := \sum_{w=0}^{191} e_w^{2^k}\, \Delta_w \in E. \tag{10}$$

The Ω_k depend only on the error, the c_k only on the randomness. So we need the map $K^{192} \to E^{|\mathcal{S}|}$, $\Delta \mapsto (\Omega_k)_{k\in\mathcal{S}}$, to be injective: a nonzero error has to leave some $\Omega_k \neq 0$ for the coefficients to act on.

Sixty-four terms. That map is K-linear, since the Δ_w enter linearly and the Frobenius touches only the constants e_w. It sends K^{192}, of K-dimension 192, into $E^{|\mathcal{S}|}$, of K-dimension $3|\mathcal{S}|$, so injectivity needs $|\mathcal{S}| \geq 64$. Take $\mathcal{S} = \{0, \ldots, 63\}$ and suppose $\Omega_k = 0$ for all $k < 64$. Two steps force $\Delta = 0$.

First, split Ω_k over K. Write $w = 64j + i$, so that $e_w^{2^k} = x^{i2^k} \left(y^{2^k}\right)^j$ and

$$\Omega_k = \sum_{j=0}^{2} (y^{2^k})^j \, \Omega_{j,k}, \qquad \Omega_{j,k} := \sum_{i=0}^{63} \Delta_{64j+i} \, x^{i2^k} \in K, \qquad 0 \leq j < 3.$$

Squaring is a bijection of K onto itself, so $y^{2^k} \in K$ would force $y \in K$, hence $E = K[y] = K$. By Lemma 3.8 applied to y^{2^k}, the set $\{1, y^{2^k}, (y^{2^k})^2\}$ is a K-basis of E, and $\Omega_k = 0$ forces $\Omega_{0,k} = \Omega_{1,k} = \Omega_{2,k} = 0$.

Second, read the 64 equations $\Omega_{j,k} = 0$, at fixed j, as a single polynomial. Put

$$D_j(v) = \sum_{i=0}^{63} \Delta_{64j+i} \, v^i \in K[v], \qquad \text{so that} \qquad \Omega_{j,k} = D_j(x^{2^k}).$$

Then D_j has degree at most 63 and vanishes at the 64 points $x, x^2, x^4, \ldots, x^{2^{63}}$. Those are distinct, because x has degree 64 over $\mathbb{F}_2$ and so has 64 distinct conjugates. A nonzero polynomial of degree at most 63 cannot have 64 roots, so $D_j = 0$ and every Δ_{64j+i} vanishes. A nonzero error therefore leaves some $\Omega_k \neq 0$.

The count is tight on both sides: 64 terms give 64 equations $\Omega_k = 0$ over E, which is 192 equations over K, against exactly 192 unknowns $\Delta_w \in K$.

A.4 The map we draw

We need a random Φ of support $\{0, \ldots, 63\}$, cheap to apply. Instead of drawing its 64 coefficients independently, we derive them from six challenges.

Draw $f_0, \ldots, f_5 \leftarrow E$ once the s_i are fixed, and compose six two-term maps: for an input $a \in E$, put $a_0 = a$ and

$$a_{p+1} = a_p + f_p \, a_p^{2^{2^{5-p}}} \in E \quad (0 \leq p < 6), \qquad \Phi(a) = a_6. \tag{11}$$

Each stage is $\mathbb{F}_2$-linear, so Φ is. Write $k = \sum_{p=0}^{5} k_p \, 2^{5-p}$ for the binary digits of k. The composition expands to

$$\Phi(a) = \sum_{k=0}^{63} c_k \, a^{2^k}, \qquad c_k = \prod_{p \,:\, k_p = 1} f_p^{2^{(k \bmod 2^{5-p})}}. \tag{12}$$

Stage p fires when $k_p = 1$ and contributes a factor f_p, which every later stage that fires raises to its own Frobenius power; those shifts sum to k mod 2^{5-p}. Each k from 0 to 63 occurs exactly once, so the c_k are 64 distinct monomials in $f_0, \ldots, f_5$, one for every Frobenius power the previous section requires.

Soundness. Fix a nonzero error before the challenges are drawn. The previous section gives some $\Omega_k \neq 0$, and distinct monomials cannot cancel, so

$$\sum_{k=0}^{63} c_k \, \Omega_k$$

is a nonzero polynomial in $f_0, \dots, f_5$. Its total degree is at most the largest degree of a c_k, reached at $k = 63$ where every bit is set:

$$2^{31} + 2^{15} + 2^7 + 2^3 + 2 + 1 < 2^{32},$$

so by Schwartz–Zippel (Lemma 3.9) the check passes with probability less than $2^{32}/2^{192} = 2^{-160}$.

A.5 Evaluating the weight

An opening reduces $\sum_{u \in \{0,1\}^{\kappa_{\text{keccak}}}} W(u)\, q_{\text{keccak}}(u) = T$ to a claim about q_{keccak} at a random point $r' \in E^{\kappa_{\text{keccak}}}$, and the verifier must evaluate the multilinear extension of W at r', written $\widetilde{W}(r')$, by itself. One fact turns that into a short formula.

Fact (Frobenius passes through eq *on the boolean hypercube).* Squaring respects sums and products and fixes bits, so for $u \in \{0,1\}^{\kappa_{\text{keccak}}}$, by Definition 3.4,

$$\mathrm{eq}(r,u)^{2^k} = \prod_{n=1}^{\kappa_{\text{keccak}}} (1 + r_n + u_n)^{2^k} = \prod_{n=1}^{\kappa_{\text{keccak}}} (1 + r_n^{2^k} + u_n) = \mathrm{eq}(r^{2^k}, u), \qquad r^{2^k} := (r_1^{2^k}, \dots, r_{\kappa_{\text{keccak}}}^{2^k}).$$

Expand W at r' with (12) and this fact:

$$\begin{aligned}\widetilde{W}(r') &= \sum_{u \in \{0,1\}^{\kappa_{\text{keccak}}}} W(u)\ \mathrm{eq}(r',u) = \sum_{u \in \{0,1\}^{\kappa_{\text{keccak}}}} \Phi(\mathrm{eq}(r,u))\ \mathrm{eq}(r',u) \\ &= \sum_{k=0}^{63} c_k \sum_{u \in \{0,1\}^{\kappa_{\text{keccak}}}} \mathrm{eq}(r^{2^k}, u)\ \mathrm{eq}(r',u).\end{aligned}$$

The inner sum is $\mathrm{eq}(r^{2^k}, r')$, since $\mathrm{eq}(u, \cdot\,)$ is its own multilinear extension for every u. By Definition 3.4 again,

$$\boxed{\widetilde{W}(r') \;=\; \sum_{k=0}^{63} c_k \prod_{n=1}^{\kappa_{\text{keccak}}} (1 + r_n^{2^k} + r'_n).}$$

A.6 Cost

One application of Φ takes 63 squarings and 6 multiplications: the Frobenius ladder climbs $32 + 16 + 8 + 4 + 2 + 1$ steps, and each stage multiplies once by its challenge. The same recurrence run on the coefficient table gives the c_k of (12): stage p raises its 2^p entries to the $2^{2^5 - p}$ power and multiplies each by f_p; the raising is 32 squarings whatever p is.

	squarings	multiplications
one application of Φ	63	6
the target T, 64 applications	4032	384
the coefficients c_k, once	192	63
the weight $\widetilde{W}(r')$	$63\kappa_{\text{keccak}}$	$64\kappa_{\text{keccak}}$

Everything above is in E. The 64 products $x^i\, \Phi(s_i)$ in the target are left out: x^i lies in K, so they are cheaper, and stepping x^i to x^{i+1} is a shift.

A.7 Credits

The construction above rests on generalizing ring switching [DP26] to extension degrees that are not powers of two, an idea of Lev Soukhanov, *[[alloc]init].* The map Φ follows [BRW26].

B The polynomial commitment scheme: WHIR /Ligerito

A self-contained description of the WHIR/Ligerito polynomial commitment scheme [ACFY25, NA25] over binary fields, with a round-by-round soundness analysis in the Johnson list-decoding regime. The two schemes coincide when WHIR is instantiated with interleaved codes and Ligerito with the Reed-Solomon codes of Annex D, which gives the novel polynomial basis and additive NTT used for encoding. Following Flock [BRW26], we do not use out-of-domain (OOD) sampling at commitment, so our PCS is only list binding. The two fields are those of §2: we commit to a K-valued multilinear, and opening claims are in E.

Symbols.

here	in [ACFY25, NA25, BSCH+25]	meaning
M_i	m_i	variables of the multilinear at level i
Λ	r	number of levels
ℓ_i	ℓ_i, ν	fold factor, and the interleaving exponent
ρ_i	ρ_i	code rate
ν_i	R_i	rate exponent, $\rho_i = 2^{-\nu_i}$
γ_i	γ_i	proximity radius
η_i	η_i	Johnson slack
θ	θ	MCA parameter (denoted m in [BSCH+25])
$\delta_{\min}$	δ	relative minimum distance
λ	λ, α	batching challenge, as in §4.1
χ_j	s_j	sumcheck fold challenge
$c^{(j)}$	σ_j	running sumcheck claim after round j
J_i	T_i	number of claims, as in §8.5
ω_i	t_i	query count
$\mathcal{Q}_i$	J_i	the sampled columns
n	n	code block length
$\mathcal{D}$	S	evaluation domain
$Z^{(i)}$	$C^{(i)}$	oracle, an interleaved word (C_j is a constraint)
N_{rows}	(none)	rows of $Z^{(0)}$ the prover commits, $N_{\text{rows}} \le 2^{\ell_0}$
N_{stack}	(none)	field elements placed in the stack, as in §4.1
$\mathcal{L}_\gamma$	λ_γ	list of codewords within γ
W, c	w, v	weight and value of a claim, as in §4.1
f^*	p	final plaintext multilinear
c_{ood}	y	out-of-domain answer (y generates E over K)
A_{mca}	a	numerator of the MCA error
$\mathcal{V}_i$, $\widehat{\mathcal{V}}_i$	W_i, $\widehat{W}_i$	subspace vanishing polynomial, normalized
$\mathcal{X}_j$	X_j	novel-basis polynomial (X_i is a sumcheck variable)
ψ_d	q_d	linearized map of the NTT (q is the witness stack)
B	B	butterfly array
$\mathcal{P}$	P	message polynomial (P_i is a committed column)
e_c	β_c	the fixed $\mathbb{F}_2$-basis, as in Annex A

B.1 Preliminaries

B.1.1 Codes, interleaving, and list decoding

We use the Reed-Solomon code $\mathrm{RS}[E, \mathcal{D}, k]$ of Definition D.2, with $|\mathcal{D}| = n$, dimension $k = 2^\kappa$, and rate $\rho = k/n$. Messages are indexed by $\{0,1\}^\kappa$; the linear encoder Enc of §B.2 maps them onto the code and preserves K-valued words.

Definition B.1 (Interleaved words, lists, folds)**.** An *interleaved word* is a matrix $Z \in E^{\{0,1\}^\ell \times \mathcal{D}}$; it is an *interleaved codeword* if every row $Z[u,\cdot]$ is a codeword ($u \in \{0,1\}^\ell$). Z *agrees* with an interleaved codeword U on $\mathcal{A} \subseteq \mathcal{D}$ if $Z[u,x] = U[u,x]$ for all $u \in \{0,1\}^\ell$ and all $x \in \mathcal{A}$; it is γ-*close* to U if they agree on some $\mathcal{A}$ with $|\mathcal{A}| \geq (1-\gamma)n$; and $\mathcal{L}_\gamma(Z)$ is the set of interleaved codewords γ-close to Z. A single word *agrees with the code* on $\mathcal{A}$ if some codeword equals it there, and Z *agrees with the interleaved code* on $\mathcal{A}$ if some interleaved codeword equals its rows there. An interleaved codeword U *decodes* to the multilinear $g_U : \{0,1\}^{\ell+\kappa} \to E$ with $U[u,\cdot] = \mathrm{Enc}(g_U(u,\cdot))$; distinct U decode to distinct g_U, hence distinct $\widetilde{g_U}$. For $\chi \in E^j$, $j \leq \ell$, the *fold* $Z_\chi \in E^{\{0,1\}^{\ell-j} \times \mathcal{D}}$ is $Z_\chi[u,x] := \sum_{\iota \in \{0,1\}^j} \mathrm{eq}(\chi,\iota)\, Z[(\iota,u),x]$ (for $j = \ell$, a single word in $E^{\mathcal{D}}$).

Fact B.2 (Folding commutes with decoding)**.** *If U is an interleaved codeword then so is U_χ, with $\widetilde{g_{U_\chi}} = \widetilde{g_U}(\chi,\cdot)$ (linearity of* Enc *plus Fact 3.6).*

Theorem B.3 (Johnson bound)**.** *For every interleaved word Z of* $\mathrm{RS}[E,\mathcal{D},2^\kappa]$ *(rate ρ) and every $\eta \in (0, 1-\sqrt{\rho})$: $|\mathcal{L}_{1-\sqrt{\rho}-\eta}(Z)| \leq \frac{1}{2\eta\sqrt{\rho}}$. (see §B.6 for a self-contained proof)*

B.1.2 PCS and round-by-round soundness

Definition B.4 (L-list binding, informal)**.** A *polynomial commitment scheme* (PCS) is *L-list binding* if every commitment transcript cm determines a set $\mathcal{G}_{\mathsf{cm}}$ of at most L multilinear polynomials, not necessarily efficiently computable, such that for every unbounded prover and every collection of opened claims: if no member of $\mathcal{G}_{\mathsf{cm}}$ satisfies all the claims, the verifier rejects except with small probability. $L = 1$ is standard binding.

Definition B.5 (RBR soundness, informal; cf. [CCH+19, ACFY25])**.** A public-coin protocol is *round-by-round (RBR) sound with errors* $(\varepsilon_1, \varepsilon_2, \dots)$, one per verifier message, if there is an *invariant* on partial transcripts (not necessarily efficiently computable) marking the positions where a cheating prover has already lost:

(i) on a false statement, the invariant holds before any message is sent;

(ii) the prover alone cannot escape it: it survives every prover message;

(iii) the i-th verifier message lets it escape with probability at most ε_i;

(iv) if it survives to the end of the interaction, the verifier rejects.

Interactive soundness error is then at most $\sum_i \varepsilon_i$. After Fiat–Shamir, it is $\max_i \varepsilon_i$ per random-oracle query [CCH+19, BSCS16].

B.2 Reed–Solomon codes over binary fields

The encoder $\mathrm{Enc}_{\kappa,\nu}$ of Definition D.3 evaluates 2^κ novel-basis coefficients on an additive domain of size $n = 2^{\kappa+\nu}$. Here $\nu \geq 1$ and $\kappa + \nu \leq 64$. The protocol uses two properties, proved in Annex D:

- *fast encoding, for the prover:* $\mathrm{Enc}_{\kappa,\nu}$ is computed by the additive NTT with $\frac{1}{2}\kappa n$ multiplications and fewer than $(\kappa+1)\,n$ additions (Proposition D.6);
- *MLE-friendly columns, for the verifier:* for every $x \in \mathcal{D}$, the weight $W_x : \{0,1\}^\kappa \to K$, defined by $\mathrm{Enc}_{\kappa,\nu}(g)[x] = \langle W_x, g \rangle$ (the relation being valid for arbitrary g), admits an MLE efficiently evaluable anywhere in E^κ in $O(\kappa)$ field operations (Lemma D.7).

B.3 The protocol

Protocol B.6 (Whir/Ligerito)**.**

Parameters. The scheme commits to $f : \{0,1\}^M \to K$ across $\Lambda \geq 1$ *levels* $i = 0, \ldots, \Lambda - 1$ plus a plaintext final level Λ: variable counts $M = M_0 > M_1 > \cdots > M_\Lambda \geq 0$ with fold factors $\ell_i := M_i - M_{i+1} \geq 1$ and M_Λ small (e.g. $M_\Lambda \leq 5$); per level, a rate $\rho_i = 2^{-\nu_i}$, $\nu_i \geq 1$, giving the encoder $\mathrm{Enc}_i := \mathrm{Enc}_{\kappa_i,\nu_i}$ of §B.2 with $\kappa_i := M_{i+1}$, i.e. the code $\mathcal{C}_i := \mathrm{RS}[E, \mathcal{D}_i, 2^{\kappa_i}]$ over its domain $\mathcal{D}_i \subseteq K$ of size $n_i = 2^{\kappa_i+\nu_i}$ (requires $\kappa_i + \nu_i \leq 64$); and a query count ω_i. (The proximity radii $\gamma_i \in (0, 1 - \sqrt{\rho_i})$ appear only in the analysis.)

Commit$(f : \{0,1\}^M \to K)$: **P** sends the oracle $Z^{(0)}[u, \cdot] := \mathrm{Enc}_0(f(u, \cdot))$, $u \in \{0,1\}^{\ell_0}$, whose entries are K elements. No OOD sample is taken.

Open$(\{(W_t, c_t)\}_{t=1,\ldots,J_0})$, $J_0 \geq 1$ weighted claims about f: set $f^{(0)} := f$. For $i = 0, 1, \ldots, \Lambda - 1$:

1. *(batch)* **V** sends $\lambda \leftarrow E$; the *batched claim* is $(W, c^{(0)}) := (\sum_{t=1}^{J_i} \lambda^{t-1} W_t,\ \sum_t \lambda^{t-1} c_t)$.
2. *(sumcheck [LFKN92])* For $j = 1, \ldots, \ell_i$: **P** sends $h_j \in E[X]$ with $\deg h_j \leq 2$, honestly $h_j(X) = \sum_x \widetilde{W}(\chi^{<j}, X, x)\, \widetilde{f^{(i)}}(\chi^{<j}, X, x)$ where $\chi^{<j} := (\chi_1, \ldots, \chi_{j-1})$; **V** checks $h_j(0) + h_j(1) \stackrel{?}{=} c^{(j-1)}$, samples $\chi_j \leftarrow E$ and sends it to the prover; both set $c^{(j)} := h_j(\chi_j)$.
   Set $\chi := (\chi_1, \ldots, \chi_{\ell_i})$, $c' := c^{(\ell_i)}$, and, for $x \in \{0,1\}^{M_{i+1}}$: $f^{(i+1)}(x) := \widetilde{f^{(i)}}(\chi, x)$, $W'(x) := \widetilde{W}(\chi, x)$. The *residual claim* (W', c') is a weighted claim about the folded multilinear $f^{(i+1)} : \{0,1\}^{M_{i+1}} \to E$.
3. If $i < \Lambda - 1$:
   (a) *(commit)* **P** sends the oracle $Z^{(i+1)}[u, \cdot] := \mathrm{Enc}_{i+1}(f^{(i+1)}(u, \cdot))$, $u \in \{0,1\}^{\ell_{i+1}}$.
   (b) *(ood)* **V** sends $z \leftarrow E^{M_{i+1}}$; **P** sends $c_{\mathrm{ood}} \in E$ (honestly $\widetilde{f^{(i+1)}}(z)$).
   (c) *(query)* **V** sends ω_i i.i.d. columns $\mathcal{Q}_i = (x_q)_{q=1,\ldots,\omega_i} \leftarrow \mathcal{D}_i$; it reads $Z^{(i)}[\cdot, x_q]$ and computes $c_q := \sum_u \mathrm{eq}(\chi, u)\, Z^{(i)}[u, x_q] = Z^{(i)}_\chi[x_q]$; honestly $c_q = \mathrm{Enc}_i(f^{(i+1)})[x_q]$, by Fact B.2.
   (d) Claims for level $i + 1$ (so $J_{i+1} := \omega_i + 2$): the residual claim (W', c'), the OOD claim $(\mathrm{eq}(z, \cdot), c_{\mathrm{ood}})$, and the ω_i *consistency claims* $(W^{(i)}_{x_q}, c_q)$, where $W^{(i)}_x$ are the column weights of Enc_i (§B.2): the q-th asserts $\mathrm{Enc}_i(f^{(i+1)})[x_q] = c_q$.
4. If $i = \Lambda - 1$: *(final)* **P** sends the multilinear $f^* : \{0,1\}^{M_\Lambda} \to E$; **V** sends $\omega_{\Lambda-1}$ i.i.d. columns $\mathcal{Q}_{\Lambda-1} \leftarrow \mathcal{D}_{\Lambda-1}$ and computes c_q from $Z^{(\Lambda-1)}$ as in 3(c). Consistency is again a weighted claim, $\mathrm{Enc}_{\Lambda-1}(f^*)[x_q] = \langle W^{(\Lambda-1)}_{x_q}, f^* \rangle$ (Lemma D.7), so the level closes like the others: **V** samples and sends $\lambda \leftarrow E$, batching the $\omega_{\Lambda-1}$ consistency claims with the residual claim (W', c') into one claim about f^*; M_Λ further sumcheck rounds discharge it, and **V** closes by evaluating $\widetilde{f^*}$ at their point from the table it holds.

V accepts iff every check above passed.

All weights in the protocol are MLE-friendly and multilinear, so every sumcheck summand $\widetilde{W} \cdot \widetilde{f^{(i)}}$ has individual degree at most 2. At level 0 that summand pairs an E-valued weight with the K-valued f; every later level is E throughout.

Completeness is perfect: the honest prover passes every check ($h_j(0) + h_j(1) = c^{(j-1)}$ by definition of h_j, $c_q = \mathrm{Enc}_i(f^{(i+1)})[x_q]$ by Fact B.2, the OOD and residual claims by construction, the final checks as identities).

B.4 Round-by-round soundness

Fix a commitment $\mathsf{cm} = Z^{(0)}$ and input claims $\{(W_t, c_t)\}_t$, and define the relation

$$\mathsf{R}_{\mathrm{open}} := \Big\{ (Z^{(0)}, \{(W_t, c_t)\}_t) \ : \ \exists U \in \mathcal{L}_{\gamma_0}(Z^{(0)}) \text{ with } \langle W_t,\, g_U \rangle = c_t \text{ for all } t \Big\}.$$

The commitment binds the prover to the list $\mathcal{G}_{\mathsf{cm}} := \{\widetilde{g_U} : U \in \mathcal{L}_{\gamma_0}(Z^{(0)})\}$, of size at most $L_0 := \frac{1}{2\eta_0\sqrt{\rho_0}}$ for $\gamma_0 = 1 - \sqrt{\rho_0} - \eta_0$, by the Johnson bound (Theorem B.3).

Theorem B.7 (RBR soundness)**.** *Suppose that for every level i: $\gamma_i = 1 - \sqrt{\rho_i} - \eta_i$ with $\eta_i \in (0, 1 - \sqrt{\rho_i})$, $L_i := \frac{1}{2\eta_i\sqrt{\rho_i}}$, and $\epsilon_i := A_{\mathrm{mca},i}/|E|$ is the mutual correlated agreement error of Theorem B.9 below for $\mathcal{C}_i$ at radius γ_i. Then the Protocol B.6 is a L_0-list binding PCS, and its opening phase $\mathsf{R}_{\mathrm{open}}$ has the following RBR soundness error at each verifier message:*

$$\begin{aligned}
&\text{batching, level } i: && \varepsilon_i^{\mathrm{batch}} &&= \frac{(J_i - 1)\, L_i}{|E|},\\
&\text{fold challenge } \chi_j,\ \text{level } i: && \varepsilon_{i,j}^{\mathrm{fold}} &&= \frac{2L_i}{|E|} + \epsilon_i,\\
&\text{OOD challenge entering level } i \geq 1: && \varepsilon_i^{\mathrm{ood}} &&= \binom{L_i}{2} \cdot \frac{M_i}{|E|},\\
&\text{query message } \mathcal{Q}_i: && \varepsilon_i^{\mathrm{query}} &&= (1 - \gamma_i)^{\omega_i},\\
&\text{final batching challenge } \lambda: && \varepsilon^{\mathrm{fin}} &&= \frac{\omega_{\Lambda-1}}{|E|},\\
&\text{each of the } M_\Lambda \text{ closing rounds}: && \varepsilon^{\mathrm{tail}} &&= \frac{2}{|E|}.
\end{aligned}$$

Here J_0 is the number of input claims and $J_i = \omega_{i-1} + 2$ for $i \geq 1$. The RBR error*, which governs the Fiat–Shamir compilation (§B.1.2), is the maximum of these entries over all rounds.*

The remainder of this section proves the theorem: §B.4.1 states the coding-theoretic tools, §B.4.2 the invariant and the round-by-round argument.

B.4.1 Proximity tools

The engine is *mutual correlated agreement* (MCA) [ACFY25]. For a two-row word Z, recall $Z_\chi = (1 - \chi)Z[0, \cdot] + \chi\, Z[1, \cdot]$.[1]

Theorem B.8 (MCA in the unique-decoding regime; adapted from [BSCH+25, Cor. 1.4])**.** *Let* $\mathrm{RS}[E, \mathcal{D}, k]$ *have block length $n = |\mathcal{D}|$ and relative minimum distance $\delta_{\min} := 1 - (k-1)/n$. Suppose*

$$\delta_{\min} \geq 3\sqrt{\frac{2}{n}} \qquad \text{and} \qquad \gamma \in \left[\frac{\delta_{\min}}{3},\ \frac{\delta_{\min}}{2} - \frac{3}{\delta_{\min} n}\right].$$

Then, for every two-row word Z,

$$\Pr_{\chi \leftarrow E}\left[\exists \mathcal{A} \subseteq \mathcal{D},\ |\mathcal{A}| \geq (1-\gamma)n : \begin{array}{l} Z_\chi \text{ agrees with the code on } \mathcal{A}, \text{ and} \\ Z \text{ does not agree with the interleaved code on } \mathcal{A} \end{array} \right] \leq \frac{\gamma n + 1}{|E|}.$$

[1] [BSCH+25] state the results for the affine line $Z[0, \cdot] + \chi\, Z[1, \cdot]$; in characteristic 2, $Z_\chi = Z[0, \cdot] + \chi(Z[0, \cdot] + Z[1, \cdot])$, and this invertible change of pair maps codeword pairs to codeword pairs, so the formulations are equivalent.

We record Theorem B.8 only for comparison, we don't need it in the List-Decoding Regime, we instead need:

Theorem B.9 (MCA up to the Johnson bound; [BSCH+25, Thm. 4.6])**.** *Let* $\mathrm{RS}[E, \mathcal{D}, k]$ *have block length* $n = |\mathcal{D}|$ *and set* $\rho := (k-1)/n$*, the* slightly reduced rate *([BSCH+25] parametrize by the degree bound* $k-1$*, not the dimension* k*). Let* $\gamma \in (0, 1-\sqrt{\rho})$*, and set* $\eta := 1 - \sqrt{\rho} - \gamma$ *and* $\theta := \max\{\lceil \frac{\sqrt{\rho}}{\eta} \rceil, 3\}$ *(denoted* m *in [BSCH+25]). For every two-row word* Z*,*

$$\Pr_{\chi \leftarrow E}\left[\exists \mathcal{A} \subseteq \mathcal{D},\ |\mathcal{A}| \geq (1-\gamma)n : \begin{array}{l} Z_\chi \text{ agrees with the code on } \mathcal{A}, \text{ and} \\ Z \text{ does not agree with the interleaved code on } \mathcal{A} \end{array} \right] \leq \frac{A_{\mathrm{mca}}}{|E|},$$

where

$$A_{\mathrm{mca}} \;=\; \frac{2(\theta+\frac{1}{2})^5 + 3(\theta+\frac{1}{2})\gamma\rho}{3\rho^{3/2}} \cdot n \;+\; \frac{\theta+\frac{1}{2}}{\sqrt{\rho}}$$

We write $\epsilon := A_{\mathrm{mca}}/|E|$ for this MCA error at radius γ. MCA is exactly what Lemma B.10 needs: folding should not create *new* nearby codewords.

Lemma B.10 (Folding preserves lists)**.** *Let* $\mathrm{RS}[E, \mathcal{D}, k]$ *have rate* ρ*, let* $\gamma \in (0, 1-\rho)$*, and let* ϵ *be an MCA error at radius* γ*, i.e. a bound on the probability of Theorem B.9 for every two-row word. For every interleaved word* $Z \in E^{\{0,1\}^\ell \times \mathcal{D}}$*,* $\ell \geq 1$*:*

$$\Pr_{\chi \leftarrow E}\Big[\mathcal{L}_\gamma(Z_\chi) \neq \{U_\chi : U \in \mathcal{L}_\gamma(Z)\} \Big] \;\leq\; \epsilon.$$

The bound does not grow with ℓ. This is a theorem of Jo [Jo26, Thm. 4.4]: for a code linear over a field $\mathbb{F}$, and a fold whose two coefficients are functions of a seed drawn uniformly from a set of at most $|\mathbb{F}|$ elements, the MCA error of every row interleaving of the code equals that of the code itself, at every radius. Both hypotheses hold here, with $\mathbb{F} = E$:

- *The code is E-linear.* Its domain lies in K, but $\mathrm{RS}[E, \mathcal{D}, k]$ is an E-subspace of $E^{\mathcal{D}}$, and so is each of its row interleavings.
- *The seed set is no larger than the field.* The seed is the fold challenge χ, drawn from E itself, and the coefficients are $(1-\chi, \chi)$.

The theorem holds at every radius, hence in the list-decoding regime of Theorem B.9, and the event it bounds is the MCA event of that theorem, with one agreement set shared by the fold and by every row. A row-by-row union bound would instead pay $2^{\ell-1}\epsilon$.

Proof. $\supseteq$ holds always: $U \in \mathcal{L}_\gamma(Z)$ agreeing with Z on $\mathcal{A}$ gives U_χ agreeing with Z_χ on $\mathcal{A}$. For $\subseteq$: read Z as the two-row word $(Z[(0,\cdot),\cdot], Z[(1,\cdot),\cdot])$ over the $2^{\ell-1}$-fold row interleaving of the code, whose fold at χ is Z_χ, and consider its MCA event with respect to that interleaved code. By [Jo26, Thm. 4.4], that event has probability at most the MCA error of a two-row word of $\mathrm{RS}[E, \mathcal{D}, k]$, hence at most ϵ. Suppose it does not occur, and let $V \in \mathcal{L}_\gamma(Z_\chi)$ with common agreement set $\mathcal{A}$, $|\mathcal{A}| \geq (1-\gamma)n$. Then Z_χ agrees on $\mathcal{A}$ with the interleaved codeword V, so (as the event failed) there is an interleaved codeword W whose rows $W[(\iota,u),\cdot]$ equal $Z[(\iota,u),\cdot]$ on $\mathcal{A}$ for every $\iota \in \{0,1\}$ and $u \in \{0,1\}^{\ell-1}$; W lies in $\mathcal{L}_\gamma(Z)$. Finally $W_\chi[u,\cdot]$ agrees with $V[u,\cdot]$ on $\mathcal{A}$, and $|\mathcal{A}| \geq (1-\gamma)n > k$ while distinct codewords agree on fewer than k points, so $V = W_\chi$. □

Lemma B.11 (OOD separation)**.** *Let* Z *be an interleaved word with* $|\mathcal{L}_\gamma(Z)| \leq L$*, decoding to* M*-variable multilinears. Then* $\Pr_{z \leftarrow E^M}[\exists$ *distinct* $U, U' \in \mathcal{L}_\gamma(Z) : \widetilde{g_U}(z) = \widetilde{g_{U'}}(z)] \leq \binom{L}{2} M/|E|$.

Proof. Distinct interleaved codewords have distinct multilinear extensions; apply Lemma 3.9 to each difference and union bound. □

B.4.2 Proof of the main theorem

At level i, abbreviate $\chi^{\le j} := (\chi_1, \ldots, \chi_j)$ and $\chi^{<j} := \chi^{\le j-1}$; thus $\chi^{\le 0}$ is empty and $Z^{(i)}_{\chi^{\le 0}} = Z^{(i)}$. For $0 \le j \le \ell_i$, the *round-j claim* is the weighted claim

$$(\,\widetilde{W}(\chi^{\le j}, \cdot),\ c^{(j)}\,) \qquad \text{about multilinears } g : \{0,1\}^{M_i - j} \to E,$$

where W, $c^{(0)}$, and $c^{(j)} = h_j(\chi_j)$ are read off the transcript as in Protocol B.6; an interleaved codeword satisfies a claim if its decoded multilinear does. The round-0 claim is the batched claim, and the round-ℓ_i claim is the residual claim (W', c').

We now define the invariant required by Definition B.5, a condition on the partial transcripts of the opening. In one phrase: *every codeword close to the current fold violates the current claim.* Precisely, the list below gives the condition just after each verifier message; a transcript ending with prover messages inherits the condition of its last verifier message. Every condition also includes one implicit disjunct, the *escape clause*: *"some verifier check has already failed"* (such transcripts are rejected no matter what follows).

- *Start of level i* (before any message for $i = 0$, after the query message J_{i-1} otherwise): every $U \in \mathcal{L}_{\gamma_i}(Z^{(i)})$ violates at least one of the level's J_i claims (the input claims for $i = 0$, the claims of step 3(d) of Protocol B.6 otherwise).
- *After the batching challenge λ of level i, and after each of its fold challenges χ_j:* every $U \in \mathcal{L}_{\gamma_i}(Z^{(i)}_{\chi^{\le j}})$ violates the round-j claim (λ being the case $j = 0$).
- *After the OOD challenge z closing level i ($i < \Lambda - 1$):* the round-ℓ_i instance above holds, and the multilinear extensions decoded from $\mathcal{L}_{\gamma_{i+1}}(Z^{(i+1)})$ are pairwise distinct at z.
- *After the final query message $\mathcal{Q}_{\Lambda-1}$:* f^* violates one of the level's $\omega_{\Lambda-1} + 1$ claims.
- *After the final λ, and after each of the M_Λ rounds that follow (complete transcript):* f^* violates the running claim. The terminal one $\mathbf{V}$ checks against $\widetilde{f^*}$ itself, so it rejects.

Conditions (i), (ii), (iv) of Definition B.5 hold by inspection: on a false statement, the first item is exactly the failure of R_{open}; a prover message changes nothing an instance depends on, except possibly failing a check, which only switches the escape clause on; and the last item is precisely "some check failed", so such a complete transcript is rejected. The separation clause of the OOD instance is there because the query message consumes it: it pins the new oracle's list to at most one candidate consistent with the prover's answer. It remains to bound the escape probabilities (iii); in the proof below we assume all checks so far pass, since otherwise the escape clause already holds.

Proof of Theorem B.7. If the statement lies outside R_{open}, then before any message is sent every $U \in \mathcal{L}_{\gamma_0}(Z^{(0)})$ violates at least one input claim, so the invariant holds. We follow the protocol and bound, for each verifier message, the probability that the invariant fails to carry over. Every list below consists of interleaved words of $\mathcal{C}_i$ within radius γ_i, hence has size at most L_i by Theorem B.3.

Batching challenge λ. Assume that every $U \in \mathcal{L}_{\gamma_i}(Z^{(i)})$ violates at least one of the J_i claims (W_t, c_t). Fix such a U. Its discrepancy against the batched claim is $\sum_t \lambda^{t-1} d_t$ with $d_t := \langle W_t, g_U \rangle - c_t$, a polynomial in λ of degree less than J_i, nonzero because the d_t are not all zero. By Lemma 3.9 it vanishes with probability at most $(J_i - 1)/|E|$, so a union bound over the at most L_i list members leaves every U violating the round-0 claim except with probability $(J_i - 1)L_i/|E|$.

Fold challenge χ_j. Assume that every $U \in \mathcal{L} := \mathcal{L}_{\gamma_i}(Z^{(i)}_{\chi^{<j}})$ violates the round-$(j-1)$ claim. Since all checks so far pass (otherwise the verifier trivially rejects), $h_j(0) + h_j(1) = c^{(j-1)}$. For $U \in \mathcal{L}$, let

$$H_U(X) \;:=\; \sum_{x \in \{0,1\}^{M_i - j}} \widetilde{W}(\chi^{<j}, X, x)\, \widetilde{g_U}(X, x)$$

be the honest round polynomial for U; it has degree at most 2, since $\widetilde{W}$ and $\widetilde{g_U}$ are multilinear. Summing over $X \in \{0,1\}$ turns the multilinear evaluations back into values, so

$$H_U(0) + H_U(1) \;=\; \langle\, \widetilde{W}(\chi^{<j}, \cdot),\; g_U \,\rangle \;\neq\; c^{(j-1)} \;=\; h_j(0) + h_j(1),$$

the inequality being the assumed violation; hence $h_j \neq H_U$ as polynomials. Two bad events over χ_j:

- B_1: $h_j(\chi_j) = H_U(\chi_j)$ for some $U \in \mathcal{L}$. For each U, two distinct polynomials of degree at most 2 agree on at most 2 points, so $\Pr[B_1] \leq 2L_i/|E|$.
- B_2: $\mathcal{L}_{\gamma_i}(Z^{(i)}_{\chi^{\leq j}}) \neq \{U_{\chi_j} : U \in \mathcal{L}\}$, i.e. folding does not map the old list onto the new one. Lemma B.10, applied to the $2^{\ell_i - j + 1}$-row matrix $Z^{(i)}_{\chi^{<j}}$, gives $\Pr[B_2] \leq \epsilon_i$.

If neither occurs, every $V \in \mathcal{L}_{\gamma_i}(Z^{(i)}_{\chi^{\leq j}})$ is $V = U_{\chi_j}$ for some $U \in \mathcal{L}$, with $\widetilde{g_V} = \widetilde{g_U}(\chi_j, \cdot)$ by Fact B.2, so

$$\langle\, \widetilde{W}(\chi^{\leq j}, \cdot),\; g_V \,\rangle \;=\; \sum_{x \in \{0,1\}^{M_i - j}} \widetilde{W}(\chi^{<j}, \chi_j, x)\, \widetilde{g_U}(\chi_j, x) \;=\; H_U(\chi_j) \;\neq\; h_j(\chi_j) \;=\; c^{(j)},$$

using $\neg B_1$ for the inequality: every V violates the round-j claim. The total error is $2L_i/|E| + \epsilon_i$.

The level interface. After the last fold challenge of level $i < \Lambda - 1$, we know: every codeword within γ_i of the single row $c := Z^{(i)}_{\chi} \in E^{\mathcal{D}_i}$ violates the residual claim (W', c'). The prover now sends the new oracle $Z^{(i+1)}$.

OOD challenge z. After z, two properties must hold: the one above, which does not involve z; and pairwise distinctness at z of the multilinear extensions decoded from $\mathcal{L}_{\gamma_{i+1}}(Z^{(i+1)})$, a list already determined by the prover's message. The latter fails with probability at most $\binom{L_{i+1}}{2} M_{i+1}/|E|$ over z, by Lemma B.11.

Query message $\mathcal{Q}_i$. Assume both properties of the previous step; the prover has since sent its answer c_{ood}. The values $\widetilde{g_U}(z)$ are pairwise distinct across $\mathcal{L}_{\gamma_{i+1}}(Z^{(i+1)})$, so at most one member U^* can satisfy $\widetilde{g_{U^*}}(z) = c_{\text{ood}}$: every other member violates the OOD claim $(\text{eq}(z, \cdot), c_{\text{ood}})$, whatever $\mathcal{Q}_i$ is. If no such U^* exists, we are done. Otherwise write $g^* := g_{U^*}$, a multilinear on $\{0,1\}^{M_{i+1}}$, and distinguish two cases.

- $\text{Enc}_i(g^*)$ *agrees with c on at least $(1 - \gamma_i)n_i$ coordinates.* Then the codeword $\text{Enc}_i(g^*)$ lies in $\mathcal{L}_{\gamma_i}(c)$, hence violates the residual claim by assumption: $\langle W',\, g^* \rangle \neq c'$, whatever $\mathcal{Q}_i$ is.
- *Otherwise.* A uniform column $x \leftarrow \mathcal{D}_i$ satisfies $\text{Enc}_i(g^*)[x] = c[x]$ with probability less than $1 - \gamma_i$, so all ω_i i.i.d. queries do so simultaneously with probability less than $(1 - \gamma_i)^{\omega_i}$; and any disagreeing query x_q makes U^* violate the consistency claim $(W^{(i)}_{x_q},\, c_q)$, since $c_q = c[x_q]$.

Except with probability $(1 - \gamma_i)^{\omega_i}$, every member of $\mathcal{L}_{\gamma_{i+1}}(Z^{(i+1)})$ thus violates at least one of the J_{i+1} claims, which is exactly the assumption of the batching step at level $i + 1$.

Final level. At level $\Lambda - 1$ the fold challenges end as above: every codeword within $\gamma_{\Lambda-1}$ of $c := Z_\chi^{(\Lambda-1)}$ violates the residual claim. The prover sends f^*. If $\mathrm{Enc}_{\Lambda-1}(f^*)$ agrees with c on at least $(1-\gamma_{\Lambda-1})n_{\Lambda-1}$ coordinates then $\mathrm{Enc}_{\Lambda-1}(f^*) \in \mathcal{L}_{\gamma_{\Lambda-1}}(c)$, so f^* violates the residual claim; otherwise, except with probability $(1-\gamma_{\Lambda-1})^{\omega_{\Lambda-1}}$, some query disagrees and f^* violates that consistency claim. Either way it violates one of the $\omega_{\Lambda-1}+1$ batched claims, hence the batch itself except with probability $\omega_{\Lambda-1}/|E|$ (Lemma 3.9), with no union over a list: f^* is transmitted, not one of the codewords near an oracle. The M_Λ rounds that follow are a plain sumcheck (Fact 3.11) on a polynomial that $\mathbf{V}$ holds, so the terminal check against $\widetilde{f^*}$ fails.

□

B.5 Optimizations

The committed f is the stack of §4.1: N_{stack} field elements in K, then zeros up to 2^M. Let the row index be the stack's most significant ℓ_0 coordinates: $f(u,x) := q(x,u)$ for $u \in \{0,1\}^{\ell_0}$ and $x \in \{0,1\}^{M-\ell_0}$, §D. Row u is then the block of $2^{M-\ell_0}$ consecutive elements of q at offset $\langle u \rangle 2^{M-\ell_0}$, the padding is whole rows, and only $N_{\text{rows}} = \lceil N_{\text{stack}}/2^{M-\ell_0} \rceil$ of the 2^{ℓ_0} rows hold data. So we get the following optimizations at commitment:

- the FFT-encoder only runs on N_{rows} rows (and not the full 2^{ℓ_0}), since FFT-ing zeros gives zeros;
- a Merkle leaf only contains N_{rows} elements of K (proof size optimization), we keep the convention of hashing the full 2^{ℓ_0} values for simplicity, but we hash them in reverse order, starting from the padding zeros;
- the prover can thus preprocess the sponge state after hashing the whole 136-byte blocks of those zeros, and then hash the rest of each leaf starting from this state.

Conclusion: at commitment, both the FFT and the Merkle leaf hashing cost are proportional to N_{stack} (not its next power of two). Unfortunately we still pay for the padding inside the N_{stack} field elements, because some instruction counts are not powers of two. This could be prevented using the Jagged PCS [HJR+25].

B.6 The Johnson bound

We prove Theorem B.3 in its general form (arbitrary code).

Theorem B.12 (Johnson bound, agreement form)**.** *Let $\mathcal{C} \subseteq \Sigma^n$ be a code, over any alphabet Σ, in which two distinct codewords agree on at most ρn coordinates, and let $\eta \in (0, 1-\sqrt{\rho})$. Then every word $\mathbf{c} \in \Sigma^n$ has at most $\frac{1}{2\eta\sqrt{\rho}}$ codewords agreeing with it on at least $(\sqrt{\rho}+\eta)\,n$ coordinates.*

Proof. Let $U_1, \ldots, U_L$ be distinct codewords, each agreeing with $\mathbf{c}$ on at least $(\sqrt{\rho}+\eta)n$ coordinates. For each j, fix a set $\mathcal{A}_j$ of exactly $a := \lceil (\sqrt{\rho}+\eta)n \rceil$ coordinates on which U_j agrees with $\mathbf{c}$, and write $\varphi := a/n$, so $\sqrt{\rho}+\eta \le \varphi \le 1$. For a coordinate i, let $d_i := |\{j : i \in \mathcal{A}_j\}|$; then $\sum_i d_i = La = L\varphi n$.

Wherever two codewords both agree with $\mathbf{c}$ they agree with each other, so $|\mathcal{A}_j \cap \mathcal{A}_{j'}| \le \rho n$ for $j \ne j'$, and counting ordered pairs,

$$\sum_i d_i(d_i-1) \;=\; \sum_{j \ne j'} |\mathcal{A}_j \cap \mathcal{A}_{j'}| \;\le\; L(L-1)\,\rho n.$$

Cauchy–Schwarz on the vectors $(d_1, \ldots, d_n)$ and $(1, \ldots, 1)$ gives $\left(\sum_i d_i\right)^2 \le n \sum_i d_i^2$, i.e. $\sum_i d_i^2 \ge (L\varphi n)^2/n = L^2\varphi^2 n$; the left side above, which equals $\sum_i d_i^2 - \sum_i d_i$, is therefore at least $L^2\varphi^2 n - L\varphi n$.

Dividing by Ln and rearranging,

$$L\varphi^2 - \varphi \;\le\; (L-1)\,\rho \qquad\Longleftrightarrow\qquad L\,(\varphi^2-\rho) \;\le\; \varphi - \rho.$$

Finally $\varphi^2 - \rho \ge (\sqrt{\rho}+\eta)^2 - \rho = 2\eta\sqrt{\rho} + \eta^2 > 2\eta\sqrt{\rho}$ while $\varphi - \rho \le 1$, so $L \le \frac{1}{2\eta\sqrt{\rho}}$. □

C Proving Keccak by GKR

C.1 Credits

The design is based on Lev Soukhanov's Hashcaster 2 [S+26]. The packing is a reverse multiplication friendly embedding [CCXY18].

C.2 Overview

The prover initially commits to every input state (1600 bits) of the $2^{\tau_{\text{keccak}}}$ permutations being proven. Think of it as 25 K-valued multilinear polynomials in τ_{keccak} variables. At some point during the protocol, there will be some claim on the output, typically 25 evaluation claims on the 25 K-valued polynomials that represent the output state. The goal of GKR [GKR08] here is to reduce this output claim to an input claim with negligible soundness error.

Keccak-f is the composition of 24 rounds; each is a linear map followed by chi and the round constant, which we see as a multivariate polynomial of degree two in the state's 1600 variables over $\mathbb{F}_2$. GKR goes back one round at a time: a *layer* turns a claim on a round's output state into a claim on its input state.

C.3 Keccak-$f[1600]$

A state is 1600 bits: 25 lanes of 64 bits, lane ℓ in row $\lfloor \ell/5 \rfloor$ and column $\ell \bmod 5$ of a 5×5 grid. Position $j = z + 64\ell$ is bit z (*slice* z) of lane ℓ.

For a lane $0 \le \ell < 25$ and $0 \le k < 5$, $\ell^{\to k}$ is the lane in the same row as ℓ whose column is that of ℓ plus k, modulo 5, and $j^{\to k}$ is the same slice of that lane:

$$\ell^{\to k} = 5\lfloor \ell/5 \rfloor + ((\ell \bmod 5) + k) \bmod 5, \qquad (z + 64\ell)^{\to k} = z + 64\,\ell^{\to k}.$$

For example $3^{\to 1} = 4$ and $4^{\to 1} = 0$. The map $j \mapsto j^{\to k}$ is a bijection with inverse $j \mapsto j^{\to 5-k}$. Chi uses $k = 1, 2$.

Round $R \in \{1, \dots, 24\}$ maps a state a to

$$b = \mathsf{lin}(a), \qquad a'[j] = b[j] + (1 + b[j^{\to 1}])\, b[j^{\to 2}] + [\, j < 64 \,]\, \mathsf{RC}_R[j], \tag{13}$$

where lin is pi ∘ rho ∘ theta, the product is chi, and RC_R is FIPS 202's RC$[R-1]$, added by iota.

C.4 Packing

Lemma C.1 (Embedding)**.** *There exists $\mathbb{F}_2$-linear maps $\mathsf{emb} : \mathbb{F}_2^{16} \to \mathbb{F}_2^{\le 23}[X]$ and $\mathsf{proj} : \mathbb{F}_2^{\le 46}[X] \to \mathbb{F}_2^{16}$ with $\mathsf{proj}(\mathsf{emb}(u) \cdot \mathsf{emb}(v)) = u \wedge v$ for all u, v, $\wedge$ being the coordinatewise product.*

The proof is in §C.9. Let emb_i embed the unit vector at $i \in \{0,1\}^4$ and $\mathsf{emb}_{\mathbf{1}} = \sum_i \mathsf{emb}_i$, so that $\mathsf{proj}(\mathsf{emb}_{\mathbf{1}} \cdot \mathsf{emb}(u)) = u$. Each bit of $\mathsf{proj}(G)$ is a sum of coefficients of G, so proj applies just as well to a polynomial G with coefficients in E, giving $\mathsf{proj}(G) \in E^{16}$. For some weights $\sigma \in E^{16}$, we use the notation $\langle \sigma, \mathsf{proj}(G) \rangle = \sum_{i<16} \sigma_i \mathsf{proj}(G)_i$ (Definition 3.5).

The key idea is to *pack* the permutations in groups of sixteen. At one position, the bits of a pack's sixteen permutations form a vector $u \in \mathbb{F}_2^{16}$, which emb turns into a single polynomial. Chi's product of two bits then becomes, for the whole pack, one carry-less product of two such polynomials, from which proj reads back the sixteen products.

C.5 Example: chi on one layer

Here we focus only on chi; the linear map and the round constant are left to §C.6. Take $2^{\tau_{\text{keccak}}}$ permutations in *packs* of sixteen: permutation $t = (i, p)$ is number $i \in \{0,1\}^4$ of pack $p \in \{0,1\}^{\tau_{\text{keccak}}-4}$. With $a_{\text{in}}(j;t)$ the bits ($j < 1600$) of permutation t entering chi, those leaving it are

$$a_{\text{out}}(j;t) = a_{\text{in}}(j;t) + \big(1 + a_{\text{in}}(j^{\to 1};t)\big)\, a_{\text{in}}(j^{\to 2};t).$$

The verifier holds a claim on a weighted sum of them, with weights $\mathrm{eq}(r,p)$ on the packs ($r \in E^{\tau_{\text{keccak}}-4}$), σ_i within a pack and w_j on the positions:

$$c = \sum_{p\in\{0,1\}^{\tau_{\text{keccak}}-4}} \mathrm{eq}(r,p) \sum_{i\in\{0,1\}^4} \sigma_i \sum_{j<1600} \mathsf{w}_j\, a_{\text{out}}\big(j;(i,p)\big).$$

1. The product polynomial. In pack p at position j, let $S_{p,j} = \mathsf{emb}\big((a_{\text{in}}(j;(i,p)))_{i\in\{0,1\}^4}\big)$. Since $\mathsf{proj}(\mathsf{emb_1}\mathsf{emb}(u)) = u$ and $\mathsf{proj}(\mathsf{emb}(u)\mathsf{emb}(v)) = u \wedge v$, the polynomial $G_{p,j} = \mathsf{emb_1} S_{p,j} + (\mathsf{emb_1} + S_{p,j^{\to 1}})\, S_{p,j^{\to 2}}$ projects to the sixteen outputs of pack p at j. The prover sends the 47 coefficients of

$$G = \sum_{p\in\{0,1\}^{\tau_{\text{keccak}}-4}} \mathrm{eq}(r,p) \sum_{j<1600} \mathsf{w}_j\, G_{p,j},$$

and the verifier checks $\langle \sigma, \mathsf{proj}(G)\rangle = c$.

2. Random point. The verifier then evaluates G at a random point $\upsilon \in E$ (soundness error $46/|E|$) and compares it with the following.

$$\begin{aligned} G(\upsilon) &\stackrel{?}{=} \sum_{p\in\{0,1\}^{\tau_{\text{keccak}}-4}} \mathrm{eq}(r,p) \sum_{j<1600} \mathsf{w}_j\, G_{p,j}(\upsilon) \\ &= \sum_{p\in\{0,1\}^{\tau_{\text{keccak}}-4}} \mathrm{eq}(r,p) \sum_{j<1600} \mathsf{w}_j \Big(\mathsf{emb_1}(\upsilon)\, S_{p,j}(\upsilon) + \big(\mathsf{emb_1}(\upsilon) + S_{p,j^{\to 1}}(\upsilon)\big)\, S_{p,j^{\to 2}}(\upsilon)\Big) \\ &= \sum_{p\in\{0,1\}^{\tau_{\text{keccak}}-4}} \sum_{j<1600} \mathrm{eq}(r,p) \mathsf{w}_j \Big(\bar\sigma'\, S_{p,j}(\upsilon) + \big(\bar\sigma' + S_{p,j^{\to 1}}(\upsilon)\big)\, S_{p,j^{\to 2}}(\upsilon)\Big). \end{aligned}$$

with $\bar\sigma' = \mathsf{emb_1}(\upsilon)$.

3. Sumcheck. A degree-three sumcheck proves the right side above (first on the $\tau_{\text{keccak}} - 4$ bits of p, then on the 11 bits of j). It ends at a random pack $\chi^{\text{pack}} \in E^{\tau_{\text{keccak}}-4}$ and a random position $\chi^{\text{pos}} \in E^{11}$, and the verifier has to evaluate:

$$\mathrm{eq}(r,\chi^{\text{pack}})\ \widetilde{\mathsf{w}}(\chi^{\text{pos}})\ \big(\bar\sigma'\,\mathsf{v}_0 + (\bar\sigma' + \mathsf{v}_1)\,\mathsf{v}_2\big), \qquad \mathsf{v}_k = \sum_{j<1600} \mathrm{eq}(\chi^{\text{pos}}, j)\, S_{\chi^{\text{pack}},j^{\to k}}(\upsilon),$$

with $S_{\chi,j} = \sum_{p\in\{0,1\}^{\tau_{\text{keccak}}-4}} \mathrm{eq}(\chi,p)\, S_{p,j}$. The prover sends $\mathsf{v}_0, \mathsf{v}_1, \mathsf{v}_2$, which lets the verifier conclude the sumcheck. It remains to prove the correctness of $\mathsf{v}_0, \mathsf{v}_1, \mathsf{v}_2$.

4. One claim again. Each v_k is itself a weighted sum of bits. Since emb is linear, $S_{p,j} = \sum_i a_{\text{in}}(j;(i,p))\,\mathsf{emb}_i$, so $S_{p,j}(v) = \sum_i \sigma'_i\, a_{\text{in}}(j;(i,p))$ with $\sigma'_i = \mathsf{emb}_i(v)$; and re-indexing the sum over j by the bijection $j \mapsto j^{\to k}$, of inverse $j \mapsto j^{\to 5-k}$ (§C.3), carries the shift off the bits and onto the weights:

$$\mathsf{v}_k = \sum_{p\in\{0,1\}^{\tau_{\text{keccak}}-4}} \text{eq}(\chi^{\text{pack}},p) \sum_{i\in\{0,1\}^4} \sigma'_i \sum_{j<1600} \text{eq}(\chi^{\text{pos}}, j^{\to 5-k})\, a_{\text{in}}(j;(i,p)).$$

The verifier samples ϑ and batches the three into $c' = \mathsf{v}_0 + \vartheta\,\mathsf{v}_1 + \vartheta^2\,\mathsf{v}_2$, it remains to check:

$$c' \stackrel{?}{=} \sum_{p\in\{0,1\}^{\tau_{\text{keccak}}-4}} \text{eq}(\chi^{\text{pack}},p) \sum_{i\in\{0,1\}^4} \sigma'_i \sum_{j<1600} \mathsf{w}'_j\, a_{\text{in}}(j;(i,p)),$$

with $\mathsf{w}'_j = \text{eq}(\chi^{\text{pos}}, j) + \vartheta\ \text{eq}(\chi^{\text{pos}}, j^{\to 4}) + \vartheta^2\ \text{eq}(\chi^{\text{pos}}, j^{\to 3})$: a claim of the same form we had at the beginning. In other words, after adding the round constants and applying the linear layers, we can reiterate on the next GKR layer.

C.6 The protocol

We prove $2^{\tau_{\text{keccak}}}$ permutations of Keccak-f[1600] at once, indexed by $t \in \{0,1\}^{\tau_{\text{keccak}}}$ low bit first and written $t = (i,p)$ with $i \in \{0,1\}^4$ the low four bits and $p \in \{0,1\}^{\tau_{\text{keccak}}-4}$ the pack of §C.5.

The prover commits the inputs as one multilinear $q_{\text{keccak}} : \{0,1\}^{\tau_{\text{keccak}}} \times \{0,1\}^5 \to K$, slot $\ell < 25$ of its second coordinate holding input lane ℓ, bit z being the coefficient of x^z; nothing here weighs the seven slots left over, which may hold anything. Its bit slices Q_z satisfy $q_{\text{keccak}} = \sum_{z<64} Q_z\, x^z$ (Annex A), so bit z of input lane ℓ of permutation t is $Q_z(t,\ell)$.

Let $a_e(j;t)$ be bit j after e rounds (13) of permutation t, from its committed input, and $\mathsf{out}_\ell(t) = \sum_{z<64} a_{24}(z+64\ell;t)\,x^z$ its output lane ℓ. We are given values out^*_ℓ, claimed to be $\widetilde{\mathsf{out}_\ell}$ at one point $(r_{\text{low}}, r_{\text{pack}}) \in E^4 \times E^{\tau_{\text{keccak}}-4}$, and prove

$$\mathsf{out}^*_\ell = \widetilde{\mathsf{out}_\ell}(r_{\text{low}}, r_{\text{pack}}) \quad \text{for all } \ell < 25. \tag{14}$$

Claims. The example's claim, with its weights named: a *claim at depth* $d \le 24$ is (c,σ,r,w), with $c \in E$, *in-pack weights* $\sigma \in E^{16}$, $r \in E^{\tau_{\text{keccak}}-4}$ and *position weights* $\mathsf{w} \in E^{1600}$, asserting

$$c = \sum_{p\in\{0,1\}^{\tau_{\text{keccak}}-4}} \sum_{i\in\{0,1\}^4} \sum_{j<1600} \text{eq}(r,p)\,\sigma_i\,\mathsf{w}_j\ a_{24-d}(j;(i,p)). \tag{15}$$

For the first, at depth 0, the verifier samples $r_{\text{lane}} \in E^5$ once the out^*_ℓ are in the transcript and sets $\sigma_i = \text{eq}(r_{\text{low}}, i)$, $r = r_{\text{pack}}$, $\mathsf{w}_{z+64\ell} = \text{eq}(r_{\text{lane}}, \ell)\,x^z$ and $c = \sum_{\ell<25} \text{eq}(r_{\text{lane}},\ell)\,\mathsf{out}^*_\ell$: by Fact 3.6, (15) then reads $c = \sum_\ell \text{eq}(r_{\text{lane}},\ell)\,\widetilde{\mathsf{out}_\ell}(r_{\text{low}}, r_{\text{pack}})$, a random combination of the claims (14).

A layer. Layer $d < 24$ is the example run on round $24-d$: with $b = \mathsf{lin}(a_{23-d})$,

$$a_{24-d}(j;t) = b(j;t) + (1 + b(j^{\to 1};t))\,b(j^{\to 2};t) + [\,j<64\,]\,\mathsf{RC}_{24-d}[j], \tag{16}$$

which is chi on the bits of b, plus the round constant. Only the two ends change.

The constant. It is public and the same in every permutation, and $\text{eq}(r,\cdot)$ sums to one, so its whole share of the claim is $\bar\sigma \sum_{z<64} \mathsf{RC}_{24-d}[z]\,\mathsf{w}_z$ with $\bar\sigma = \sum_i \sigma_i$: the verifier computes that itself and moves it to its own side. Step 1 is then as in the example, with $S_{p,j}$ and G built on the bits of b:

$$\langle \sigma,\, \mathsf{proj}(G)\rangle + \bar\sigma \sum_{z<64} \mathsf{RC}_{24-d}[z]\,\mathsf{w}_z \ \stackrel{?}{=}\ c. \tag{17}$$

Steps 2 and 3 are unchanged: $\tau_{\text{keccak}}+7$ sumcheck rounds bind $G(v)$ (§3), $\tau_{\text{keccak}}-4$ over p at two E-elements each, then 11 over j at three, slice low and lane high (w_j and $S_{p,j\to k}$ vanish at lanes 25 to 31); with $\sigma'_i=\mathsf{emb}_i(v)$, $\bar\sigma'=\sum_i\sigma'_i$ and v_k as there, the verifier checks

$$\mathrm{eq}(r,\chi^{\text{pack}})\;\widetilde{\mathsf{w}}(\chi^{\text{pos}})\;\big(\bar\sigma'\,\mathsf{v}_0+(\bar\sigma'+\mathsf{v}_1)\,\mathsf{v}_2\big)\;\stackrel{?}{=}\;c^{(\tau_{\text{keccak}}+7)}. \tag{18}$$

The linear map. Step 4 leaves a claim on the bits of b, with weights $\mathsf{w}^\flat_j=\mathrm{eq}(\chi^{\text{pos}},j)+\vartheta\,\mathrm{eq}(\chi^{\text{pos}},j^{\to4})+\vartheta^2\,\mathrm{eq}(\chi^{\text{pos}},j^{\to3})$. As lin is linear, $\sum_j\mathsf{w}^\flat_j\,b(j;t)=\sum_j(\mathsf{lin}^\top\mathsf{w}^\flat)_j\,a_{23-d}(j;t)$: transpose it onto the weights and the claim is on a_{23-d}, the layer handing $(c',\sigma',\chi^{\text{pack}},\mathsf{lin}^\top\mathsf{w}^\flat)$ to depth $d+1$. Nobody ever writes those 1600 weights out: $\mathrm{eq}(\chi^{\text{pos}},\cdot)$ is a lane table times a slice table and the shifts only move lanes, so a claim's weights stay $\mathsf{lin}^\top$ of such a product, which is what $\widetilde{\mathsf{w}}$ is evaluated from.

The opening. The claim at depth 24 is on the committed inputs,

$$c=\sum_{i\in\{0,1\}^4}\sigma_i\sum_{o\in\{0,1\}^5}\sum_{z<64}\mathsf{w}^{\text{in}}(o,z)\,\widetilde{Q_z}(i,r,o),\qquad \mathsf{w}^{\text{in}}(\ell,\,z)=\mathsf{w}_{z+64\ell}, \tag{19}$$

w^{in} being zero on slots 25 and above. A degree-two sumcheck binds i (challenges $\chi^{\text{low}}\in E^4$), then the slot (challenges $\chi^{\text{in}}\in E^5$); the prover sends $s_0,\dots,s_{63}$ and the verifier checks the final claim against $\widetilde{\sigma}(\chi^{\text{low}})\sum_z\widetilde{\mathsf{w}^{\text{in}}}(\chi^{\text{in}},z)\,s_z$, leaving

$$s_z=\widetilde{Q_z}(\chi^{\text{low}},r,\chi^{\text{in}})\qquad(z<64), \tag{20}$$

which ring switching takes (Annex A).

C.7 Committing the middle state

This is an optimization to get smaller proofs and faster recursion: we also commit to each permutation's state after twelve rounds, its lane ℓ in slot $25+\ell$. Fifty lanes a permutation now, so a slot index takes six bits and q_{keccak} has $\kappa_{\text{keccak}}=\tau_{\text{keccak}}+6$ variables. A permutation becomes two runs of twelve rounds: half 0 from the committed input to the middle state, half 1 from there to the output. Writing $a_{m,e}$ for the state after e rounds of half $m<2$ from its committed input, what we prove on top of (14) is that half 0 lands where half 1 starts:

$$a_{0,12}(j;t)=a_{1,0}(j;t)\quad\text{for all } t \text{ and } j<1600. \tag{21}$$

Both halves end on a state, and we claim both at the one point $(r_{\text{low}},r_{\text{pack}})$ we are given: half 1 on the output, where the out^*_ℓ are, and half 0 on the middle state, whose value there the prover sends as c_{st}. Reusing that point is the point: one point, one claim. It is random and drawn after q_{keccak} is committed, so it pins the middle state as firmly as the output. That is fifty lane claims, which we fold into one by sampling r_{lane} over the lanes and r_{seg} over the two halves. The surviving claim carries a coordinate for the half, and twelve layers prove both at once: (15) sums over m too, on $a_{m,12-d}$; layer d crosses round $12(m+1)-d$ of each half, its constant weighted by $\mathrm{eq}(r_{\tau_{\text{keccak}}-4},m)$; the sumcheck gains a round; and depth 12 opens as in §C.6 over six slot rounds, with $\mathsf{w}^{\text{in}}(25m+\ell,z)=\mathrm{eq}(r_{\tau_{\text{keccak}}-4},m)\,\mathsf{w}_{z+64\ell}$: the segment rides the weights, so only r's pack coordinates enter the point.

Only c_{st} is still the prover's word. Comparing it with slots $25+\ell$ of the commitment, where half 1 starts, is exactly (21) at a random point: a sumcheck over the slot (challenges χ^{out}) does it, the prover sending the word c_{out} and the verifier checking the last running claim against it times the weight that slot carries, which leaves

$$\widetilde{q_{\text{keccak}}}(r_{\text{low}},r_{\text{pack}},\chi^{\text{out}})=c_{\text{out}}. \tag{22}$$

C.8 Soundness

Fix q_{keccak} before the protocol starts. For every prover, the probability that the verifier accepts with its two final claims (22) and (20) true of q_{keccak}, while (14) or (21) fails, is at most

$$\frac{37\tau_{\text{keccak}} + 902}{|E|} = \frac{(\tau_{\text{keccak}} + 6) + 12\,(3\tau_{\text{keccak}} + 72) + (12 + 20)}{|E|},$$

a term per challenge a false claim must survive (Lemma 3.9, Fact 3.11):

- the start, $\tau_{\text{keccak}} + 6$: r_{low} and r_{pack} hide a wrong middle state, r_{seg} and r_{lane} a wrong lane among the fifty the first claim combines;
- a layer, $3\tau_{\text{keccak}} + 72$: 46 for v, as G has degree 46; $3(\tau_{\text{keccak}} - 3)$ for the pack and segment rounds and 33 for the 11 over positions, all of degree three; 2 for ϑ, in which c' is quadratic;
- the openings, 12 and 20: the six slot rounds of the middle state's opening, then the four instance and six slot rounds of (19), all of degree two.

A round message counts as a whole polynomial of bounded degree, of which the compressed messages of §3 are a special case, and $(r_{\text{low}}, r_{\text{pack}})$ is drawn after q_{keccak} is committed. The bound holds over any field of characteristic two in which $x^0, \dots, x^{63}$ are independent over $\mathbb{F}_2$, as they are in E. Fiat-Shamir and the commitment scheme are not accounted for: the two claims are the output, for ring switching and the PCS to discharge.

C.9 Proof of the embedding lemma

Proof. Take the seven pairwise coprime irreducibles g below, each carrying the bits of $u \in \mathbb{F}_2^{16}$ listed under it, and let F be their product, of degree 23.

g	X	$X+1$	X^3+X+1	X^3+X^2+1	X^5+X^2+1	X^5+X^3+1	$X^5+X^3+X^2+X+1$
$\deg g$	1	1	3	3	5	5	5
bits	u_0	u_1	u_2, u_3	u_4, u_5	u_6, u_7, u_8	u_9, u_{10}, u_{11}	u_{12}, u_{13}, u_{14}

Writing a_0, a_1, a_2 for the bits g carries, set

$$h_g(u) = a_0, \qquad h_g(u) = a_0 + (a_0 + a_1)X, \qquad h_g(u) = a_0 + (a_0 + a_1 + a_2)X + a_2X^2$$

for $\deg g = 1$, 3 and 5, so that $h_g(u)$ has degree at most $(\deg g - 1)/2$ and takes the values a_0 at 0 and a_1 at 1, with a_2 as its coefficient of X^2. Let H_u be the polynomial of degree below 23 congruent to $h_g(u)$ modulo every g (Chinese remainder theorem), and put $\mathsf{emb}(u) = H_u + u_{15}F$, of degree at most 23. Let $\mathsf{proj}(P)$ read, g by g in the order of the table, the residue $R = P \bmod g$: its value at 0 when $\deg g = 1$, its values at 0 and 1 when $\deg g = 3$, those two and its coefficient of X^4 when $\deg g = 5$; the sixteenth bit is the coefficient of X^{46} in P. Both maps are $\mathbb{F}_2$-linear.

Let $P = \mathsf{emb}(u)\mathsf{emb}(v)$ and let b_0, b_1, b_2 be the bits of v carried by g. Modulo g, P is $h_g(u)\,h_g(v)$, of degree at most $\deg g - 1$, so that product is the residue R itself. Evaluation being multiplicative, $R(0) = a_0b_0$ and $R(1) = a_1b_1$; and for $\deg g = 5$ both factors have degree at most 2, so the coefficient of X^4 in R is a_2b_2. Those are the first fifteen bits of $u \wedge v$. Finally

$$P = H_uH_v + u_{15}FH_v + v_{15}FH_u + u_{15}v_{15}F^2,$$

whose first three terms have degree at most 45 while F^2 is monic of degree 46, so the coefficient of X^{46} in P is $u_{15}v_{15}$. □

D Novel polynomial basis and additive NTT

This annex collects the algebra shared by the PCS (Annex B) and LeanDA (§10.2): additive domains, the novel polynomial basis, its additive FFT (called an NTT over a finite field), and codeword membership. The basis and transform follow [LCH14], as presented in [DP26].

Fix the $\mathbb{F}_2$-basis $e_c := x^c$, $0 \le c < 64$, of K, the first 64 vectors of the basis of E in Annex A. In particular $e_0 = 1$. For a bit vector u, write $\langle u \rangle := \sum_i u_i 2^i$ for its integer index, with bit 0 first.

D.1 Additive domains and subspace polynomials

For $0 \le i \le 64$ let $\mathcal{U}_i := \langle e_0, \dots, e_{i-1} \rangle_{\mathbb{F}_2}$ and define the *subspace vanishing polynomial* $\mathcal{V}_i(X) := \prod_{u \in \mathcal{U}_i} (X + u)$, of degree 2^i.

Lemma D.1 (Subspace polynomials)**.** *Each $\mathcal{V}_i$ is linearized: it contains only powers X^{2^j}. For $i < 64$: (i) $\mathcal{V}_{i+1}(X) = \mathcal{V}_i(X)\big(\mathcal{V}_i(X) + \mathcal{V}_i(e_i)\big)$ with $\mathcal{V}_i(e_i) \neq 0$; (ii) $\mathcal{V}_i(X+Y) = \mathcal{V}_i(X) + \mathcal{V}_i(Y)$ as a polynomial identity; in particular the map $\mathcal{V}_i : K \to K$ is $\mathbb{F}_2$-linear.*

Proof. By induction on i; the base case is $\mathcal{V}_0(X) = X$. For (i), split the product defining $\mathcal{V}_{i+1}$ along $\mathcal{U}_{i+1} = \mathcal{U}_i \sqcup (e_i + \mathcal{U}_i)$:

$$\mathcal{V}_{i+1}(X) \;=\; \prod_{u \in \mathcal{U}_i} (X + u) \cdot \prod_{u \in \mathcal{U}_i} (X + e_i + u) \;=\; \mathcal{V}_i(X) \cdot \mathcal{V}_i(X + e_i) \;=\; \mathcal{V}_i(X)\big(\mathcal{V}_i(X) + \mathcal{V}_i(e_i)\big),$$

where the middle step recognizes the second product as $\mathcal{V}_i$ evaluated at $X + e_i$, and the last step is the identity (ii) for $\mathcal{V}_i$ with $Y = e_i$. Moreover $\mathcal{V}_i(e_i) \neq 0$, since the roots of $\mathcal{V}_i$ are exactly $\mathcal{U}_i$ and $e_i \notin \mathcal{U}_i$. The recurrence also preserves the linearized form: squaring doubles each exponent, and the other term is a scalar multiple of $\mathcal{V}_i$. For (ii) at $i+1$, write $\mathcal{V}_{i+1} = \mathcal{V}_i^2 + \mathcal{V}_i(e_i)\,\mathcal{V}_i$ by (i): the second term satisfies the identity by induction, and so does the first, since in characteristic 2

$$\mathcal{V}_i(X+Y)^2 \;=\; \big(\mathcal{V}_i(X) + \mathcal{V}_i(Y)\big)^2 \;=\; \mathcal{V}_i(X)^2 + \mathcal{V}_i(Y)^2. \qquad \square$$

For $i < 64$, let $\widehat{\mathcal{V}}_i := \mathcal{V}_i / \mathcal{V}_i(e_i)$ (the hat denotes normalization, not multilinear extension), so $\widehat{\mathcal{V}}_i(e_i) = 1$ and $\widehat{\mathcal{V}}_0 = X$ (using $e_0 = 1$). By linearity, $\widehat{\mathcal{V}}_i(x) = \sum_c x_c \widehat{\mathcal{V}}_i(e_c)$ for $x = \sum_c x_c e_c$, so given the values $\widehat{\mathcal{V}}_i(e_c)$ (precomputed via the recursion (i)), $\widehat{\mathcal{V}}_i$ is evaluated anywhere in K in at most 64 additions.

D.2 Novel basis and Reed-Solomon encoding

Definition D.2 (Reed-Solomon code)**.** For an additive domain $\mathcal{D} \subseteq K$ of size n and $1 \le k \le n$, let

$$\mathrm{RS}[E, \mathcal{D}, k] := \{(p(x))_{x \in \mathcal{D}} : p \in E[X],\ \deg p < k\}.$$

It has dimension k and rate $\rho = k/n$. Its K-valued subcode is $\mathrm{RS}[K, \mathcal{D}, k]$.

Definition D.3 (Novel basis and encoder [LCH14])**.** Let $0 \le \kappa < 64$, $\nu \ge 0$, and $\kappa + \nu \le 64$. For $0 \le j < 2^\kappa$ with bits j_i, let $\mathcal{X}_j(X) := \prod_i \widehat{\mathcal{V}}_i(X)^{j_i}$; then $\deg \mathcal{X}_j = j$, so $(\mathcal{X}_j)_{j < 2^\kappa}$ is a (triangular) basis of the polynomials of degree less than 2^κ. Let $\mathcal{D} := \langle e_0, \dots, e_{\kappa+\nu-1} \rangle_{\mathbb{F}_2}$ and, for a message $g : \{0,1\}^\kappa \to E$,

$$\mathcal{P}_g(X) := \sum_{u \in \{0,1\}^\kappa} g(u) \cdot \mathcal{X}_{\langle u \rangle}(X), \qquad \mathrm{Enc}_{\kappa,\nu}(g) := \big(\mathcal{P}_g(x)\big)_{x \in \mathcal{D}},$$

a linear injection onto $\mathrm{RS}[E, \mathcal{D}, 2^\kappa]$ of rate $\rho = 2^{-\nu}$; the $\mathcal{X}_j$ take their values on $\mathcal{D} \subseteq K$, so a K-valued message gives a K-valued word. (The message is read *directly* as novel-basis coefficients; no conversion is applied.)

The payload consists of polynomial coefficients. Any $k = 2^\kappa$ distinct evaluations determine those coefficients by interpolation, but an evaluation prefix need not equal the payload.

D.3 The additive NTT

The encoder is computed by the additive NTT of [LCH14], a Cooley-Tukey-style butterfly network in which the squaring map is replaced by degree-2 linearized maps ψ_d collapsing the domain, and the twiddles are evaluations of the $\widehat{\mathcal{V}}_d$. For $0 \le d < \kappa$ define

$$\psi_d(X) \;:=\; \frac{\mathcal{V}_d(e_d)^2}{\mathcal{V}_{d+1}(e_{d+1})} \cdot X(X+1).$$

Lemma D.4 (Chain structure)**.** *Each ψ_d is $\mathbb{F}_2$-linear with kernel $\{0,1\}$, and $\psi_d \circ \widehat{\mathcal{V}}_d = \widehat{\mathcal{V}}_{d+1}$, hence $\widehat{\mathcal{V}}_d = \psi_{d-1} \circ \cdots \circ \psi_0$. Setting $\mathcal{D}^{(d)} := \widehat{\mathcal{V}}_d(\mathcal{D})$ for $0 \le d \le \kappa$, each $\mathcal{D}^{(d)}$ is an $\mathbb{F}_2$-subspace of dimension $\kappa + \nu - d$. For $d < \kappa$, it contains $1 = \widehat{\mathcal{V}}_d(e_d)$, and ψ_d maps $\mathcal{D}^{(d)}$ onto $\mathcal{D}^{(d+1)}$ two-to-one with fibers $\{x, x+1\}$.*

Proof. $X(X+1) = X^2 + X$ is linear with kernel $\{0,1\}$. By Lemma D.1(i), $\widehat{\mathcal{V}}_{d+1} = \frac{\mathcal{V}_d(\mathcal{V}_d + \mathcal{V}_d(e_d))}{\mathcal{V}_{d+1}(e_{d+1})} = \frac{\mathcal{V}_d(e_d)^2}{\mathcal{V}_{d+1}(e_{d+1})} \widehat{\mathcal{V}}_d(\widehat{\mathcal{V}}_d + 1) = \psi_d \circ \widehat{\mathcal{V}}_d$; the chain follows from $\widehat{\mathcal{V}}_0 = X$. $\widehat{\mathcal{V}}_d|_{\mathcal{D}}$ is linear with kernel $\mathcal{U}_d \subseteq \mathcal{D}$, so its image has dimension $\kappa+\nu-d$, and $\psi_d(\mathcal{D}^{(d)}) = \widehat{\mathcal{V}}_{d+1}(\mathcal{D}) = \mathcal{D}^{(d+1)}$ with $\ker \psi_d = \{0,1\} \subseteq \mathcal{D}^{(d)}$. □

Lemma D.5 (Even-odd refinement)**.** *For $0 \le d \le \kappa$ define $\widehat{\mathcal{V}}_b^{(d)} := \psi_{d+b-1} \circ \cdots \circ \psi_d$ (and $\widehat{\mathcal{V}}_0^{(d)} := X$), and $\mathcal{X}_j^{(d)} := \prod_b (\widehat{\mathcal{V}}_b^{(d)})^{j_b}$, so $\mathcal{X}_j^{(0)} = \mathcal{X}_j$. If $d < \kappa$, $\mathcal{P} = \sum_{j<2^{\kappa-d}} a_j \mathcal{X}_j^{(d)}$, and $\mathcal{P}_\iota := \sum_j a_{2j+\iota} \mathcal{X}_j^{(d+1)}$, then*

$$\mathcal{P}(X) \;=\; \mathcal{P}_0(\psi_d(X)) + X \cdot \mathcal{P}_1(\psi_d(X)).$$

Proof. $\widehat{\mathcal{V}}_{b+1}^{(d)} = \widehat{\mathcal{V}}_b^{(d+1)} \circ \psi_d$, so $\mathcal{X}_{2j}^{(d)} = \mathcal{X}_j^{(d+1)} \circ \psi_d$ and $\mathcal{X}_{2j+1}^{(d)} = X \cdot \mathcal{X}_{2j}^{(d)}$; split the sum by parity. □

Additive NTT [LCH14]. *Input:* coefficients $a_0, \ldots, a_{2^\kappa - 1}$ of $\mathcal{P} = \sum_j a_j \mathcal{X}_j$. *Output:* $(\mathcal{P}(x))_{x \in \mathcal{D}}$, identifying $\{0,1\}^{\kappa+\nu} \cong \mathcal{D}$ via $v \mapsto x_v := \sum_c v_c e_c$.

1. Initialize $\mathsf{B}(v) := a_{\langle (v_0, \ldots, v_{\kappa-1}) \rangle}$ for all $v \in \{0,1\}^{\kappa+\nu}$ (the coefficient array, tiled 2^ν times).
2. For $d = \kappa - 1$ down to 0: for every v with $v_d = 0$, with $v' := v \oplus \mathbf{1}_d$ and twiddle $t := \sum_{c>d} v_c \widehat{\mathcal{V}}_d(e_c)$: $\mathsf{B}(v) \mathrel{+}= t \cdot \mathsf{B}(v')$; then $\mathsf{B}(v') \mathrel{+}= \mathsf{B}(v)$.
3. Output B; then $\mathsf{B}(v) = \mathcal{P}(x_v)$.

Proposition D.6 (Correctness and cost)**.** *The algorithm computes $\mathrm{Enc}_{\kappa,\nu}$ with exactly $\frac{1}{2}\kappa n$ multiplications and κn additions in the butterflies, plus fewer than n additions for the twiddles: t depends only on $(v_c)_{c>d}$, so stage d has $2^{\kappa+\nu-1-d}$ distinct twiddles, shared by the 2^d butterflies of a block and enumerable with one addition each, $\sum_{d<\kappa} 2^{\kappa+\nu-1-d} < n$ in total.*

Proof sketch. For $\ell \in \{0,1\}^d$ let $\mathcal{P}_\ell^{(d)} := \sum_{j<2^{\kappa-d}} a_{\langle \ell \rangle + 2^d j} \mathcal{X}_j^{(d)}$ collect the coefficients congruent to $\langle \ell \rangle$ modulo 2^d. Downward induction on d maintains the invariant that after stages $\kappa - 1, \ldots, d$, $\mathsf{B}(v) = \mathcal{P}^{(d)}_{(v_0,\ldots,v_{d-1})}(\widehat{\mathcal{V}}_d(x_v))$, where by linearity $\widehat{\mathcal{V}}_d(x_v) = v_d + t$ with t the twiddle of the algorithm (coordinates $c < d$ lie in $\ker \widehat{\mathcal{V}}_d = \mathcal{U}_d$, and $\widehat{\mathcal{V}}_d(e_d) = 1$). The base $d = \kappa$ is the tiled initialization.

For the step, let $\ell := (v_0, \ldots, v_{d-1})$: the order-$(d{+}1)$ even and odd parts of $\mathcal{P}^{(d)}_\ell$ are $\mathcal{P}^{(d+1)}_{(\ell,0)}$ and $\mathcal{P}^{(d+1)}_{(\ell,1)}$ (as $\langle \ell \rangle + 2^d(2j+\iota) = \langle (\ell,\iota) \rangle + 2^{d+1} j$), whose values at $\widehat{\mathcal{V}}_{d+1}(x_v) = \widehat{\mathcal{V}}_{d+1}(x_{v'})$ are the pre-stage entries $\mathsf{B}(v)$ and $\mathsf{B}(v')$; Lemma D.5 at the fiber $\{t, t+1\}$ of ψ_d over this point (Lemma D.4) gives $\mathcal{P}^{(d)}_\ell(t) = \mathsf{B}(v) + t\,\mathsf{B}(v')$ and $\mathcal{P}^{(d)}_\ell(t+1) = \mathcal{P}^{(d)}_\ell(t) + \mathsf{B}(v')$, exactly the butterfly. At $d = 0$, $\mathsf{B}(v) = \mathcal{P}(x_v)$. Each of the κ stages does $n/2$ butterflies of one multiplication and two additions. □

D.4 Column weights

A codeword coordinate is a linear combination of the message coefficients. The following factorization lets the PCS verifier evaluate that combination's multilinear weight without running a transform.

Lemma D.7 (Column weights are tensors)**.** *For $x \in \mathcal{D}$ let $W_x(u) := \prod_i \widehat{\mathcal{V}}_i(x)^{u_i}$. Then $\mathrm{Enc}_{\kappa,\nu}(g)[x] = \langle W_x,\, g \rangle := \sum_u W_x(u) g(u)$, and*

$$\widetilde{W_x}(r) \;=\; \prod_{i=0}^{\kappa-1} \big((1 + r_i) + r_i \widehat{\mathcal{V}}_i(x)\big),$$

so one evaluation of $\widetilde{W_x}$ costs $O(\kappa)$ multiplications and additions in total: the values $\widehat{\mathcal{V}}_i(x)$ follow the chain of Lemma D.4, $\widehat{\mathcal{V}}_0(x) = x$ and $\widehat{\mathcal{V}}_{i+1}(x) = \psi_i(\widehat{\mathcal{V}}_i(x))$, at two multiplications and one addition per step (the constants of the ψ_i are precomputed once, shared by all levels and queries), and the κ-factor product costs the same again.

Proof. The first identity is Definition D.3 expanded: $\mathcal{P}_g(x) = \sum_u g(u) \prod_i \widehat{\mathcal{V}}_i(x)^{u_i} = \langle W_x,\, g \rangle$. For the second, start from the definition of the multilinear extension and factor eq coordinate-wise:

$$\widetilde{W_x}(r) \;=\; \sum_{u \in \{0,1\}^\kappa} \mathrm{eq}(u, r)\, W_x(u) \;=\; \sum_{u \in \{0,1\}^\kappa} \prod_{i=0}^{\kappa-1} \mathrm{eq}(u_i, r_i)\, \widehat{\mathcal{V}}_i(x)^{u_i}.$$

Each factor of the summand depends on a single bit u_i, so by distributivity the sum over $\{0,1\}^\kappa$ factors into a product of one-bit sums:

$$\widetilde{W_x}(r) \;=\; \prod_{i=0}^{\kappa-1} \sum_{u_i \in \{0,1\}} \mathrm{eq}(u_i, r_i)\, \widehat{\mathcal{V}}_i(x)^{u_i} \;=\; \prod_{i=0}^{\kappa-1} \big((1 + r_i) \cdot 1 + r_i \cdot \widehat{\mathcal{V}}_i(x)\big),$$

using $\mathrm{eq}(0, r_i) = 1 + r_i$ and $\mathrm{eq}(1, r_i) = r_i$. □

D.5 Duality and codeword membership

Lemma D.8 (Summation on an additive domain)**.** *Let $\mathcal{D} = \mathcal{U}_{\log_2 n}$, with $n \geq 2$ a power of two. The derivative $\mathcal{V}'_{\log_2 n}$ is a nonzero constant. For every polynomial h of degree less than n,*

$$\sum_{x \in \mathcal{D}} h(x) = \mathcal{V}'_{\log_2 n}\, [X^{n-1}]\, h, \tag{23}$$

where $[X^{n-1}]\, h$ denotes the coefficient of X^{n-1} in h. In particular, the sum is zero if $\deg h \leq n - 2$.

Proof. The vanishing polynomial $\mathcal{V}_{\log_2 n}(X) = \prod_{x \in \mathcal{D}} (X - x)$ is linearized (Lemma D.1): its only powers of X are $X, X^2, X^4, \ldots$. In characteristic two, the derivative of every term except the X term is zero. Its derivative is therefore constant, equal to the coefficient of X. This constant is nonzero:

at any root $x \in \mathcal{D}$, differentiating the product gives $\mathcal{V}'_{\log_2 n}(x) = \prod_{y \in \mathcal{D} \setminus \{x\}} (x - y)$, a product of nonzero factors.

Now let h have degree less than n. Its values at the n domain points determine it by Lagrange interpolation:

$$h(X) = \sum_{x \in \mathcal{D}} \frac{h(x)}{\mathcal{V}'_{\log_2 n}(x)} \frac{\mathcal{V}_{\log_2 n}(X)}{X - x}.$$

Each quotient $\mathcal{V}_{\log_2 n}(X)/(X - x)$ is monic of degree $n - 1$, so its coefficient of X^{n-1} is 1. Taking that coefficient on both sides, and using the fact that the derivative is the same constant at every x, gives

$$[X^{n-1}]\, h = \sum_{x \in \mathcal{D}} \frac{h(x)}{\mathcal{V}'_{\log_2 n}(x)} = \frac{1}{\mathcal{V}'_{\log_2 n}} \sum_{x \in \mathcal{D}} h(x).$$

Multiplying by the nonzero derivative gives (23). □

Corollary D.9 (Dual code)**.** *For $1 \le k < n$ on the domain of Lemma D.8,*

$$\mathrm{RS}[E, \mathcal{D}, k]^{\perp} = \mathrm{RS}[E, \mathcal{D}, n - k].$$

At rate $1/2$, the code equals its dual.

Proof. If f and g have degrees less than k and $n - k$, then $\deg(fg) \le n - 2$. Lemma D.8 gives $\sum_{x \in \mathcal{D}} f(x)g(x) = 0$, so the two codes are orthogonal. Their dimensions sum to n, giving equality with the dual. □

Proposition D.10 (Membership with power weights)**.** *Let $k = 2^\kappa$, $n = 2k$, and $\mathcal{D} = \mathcal{U}_{\kappa+1}$. For $\mathsf{z} \in E$, define*

$$L(X) := \prod_{j < \kappa} (1 + \mathsf{z}^{2^j} \widehat{\mathcal{V}}_j(X)). \tag{24}$$

Its evaluation vector is a codeword. For any fixed $w \in E^{\mathcal{D}}$, the inner product $\langle L, w \rangle := \sum_{x \in \mathcal{D}} L(x)w(x)$ vanishes identically as a polynomial in z if and only if w is a codeword. If w is not a codeword and z is uniform, then

$$\Pr_{\mathsf{z}}[\langle L, w \rangle = 0] \le \frac{k - 1}{|E|}.$$

Proof. Since $\widehat{\mathcal{V}}_j = \mathcal{X}_{2^j}$, expanding (24) gives $L = \sum_{u < k} \mathsf{z}^u \mathcal{X}_u$, the binary digits of u picking its factors. Thus $\deg L < k$, and Corollary D.9 makes it orthogonal to every codeword.

Conversely, the coefficient of z^u in $\langle L, w \rangle$ is $\langle \mathcal{X}_u, w \rangle$. If the inner product vanished identically, w would be orthogonal to every basis codeword, hence would itself belong to the code. For an invalid w, it is therefore a nonzero polynomial in z of degree less than k. The probability bound follows from Lemma 3.9. □

References

[ACFY25] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed–solomon proximity testing with super-fast verification. In *Advances in Cryptology – EUROCRYPT 2025*, 2025. Cryptology ePrint Archive, Paper 2024/1586. URL: https://eprint.iacr.org/2024/1586.

[BDPA11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. Duplexing the sponge: Single-pass authenticated encryption and other applications. In *Selected Areas in Cryptography (SAC 2011)*, pages 320–337, 2011.

[BDPV11] Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche. The Keccak reference, version 3.0, 2011. URL: https://keccak.team/files/Keccak-reference-3.0.pdf.

[BEG+94] Manuel Blum, William S. Evans, Peter Gemmell, Sampath Kannan, and Moni Naor. Checking the correctness of memories. *Algorithmica*, 12(2–3):225–244, 1994.

[BHK+19] Daniel J. Bernstein, Andreas Hülsing, Stefan Kölbl, Ruben Niederhagen, Joost Rijneveld, and Peter Schwabe. The SPHINCS+ signature framework. In *ACM Conference on Computer and Communications Security (CCS 2019)*, 2019. URL: https://sphincs.org/.

[BRW26] Benedikt Bünz, Ron Rothblum, and William Wang. Flock: Fast proving for batch boolean computations. Cryptology ePrint Archive, Paper 2026/1329, 2026. URL: https://eprint.iacr.org/2026/1329.

[BSBHR18] Eli Ben-Sasson, Iddo Bentov, Yinon Horesh, and Michael Riabzev. Scalable, transparent, and post-quantum secure computational integrity. Cryptology ePrint Archive, Paper 2018/046, 2018. URL: https://eprint.iacr.org/2018/046.

[BSCH+25] Eli Ben-Sasson, Dan Carmon, Ulrich Haböck, Swastik Kopparty, and Shubhangi Saraf. On proximity gaps for reed–solomon codes. Cryptology ePrint Archive, Paper 2025/2055, 2025. URL: https://eprint.iacr.org/2025/2055.

[BSCS16] Eli Ben-Sasson, Alessandro Chiesa, and Nicholas Spooner. Interactive oracle proofs. In *Theory of Cryptography (TCC 2016-B)*, 2016.

[CCH+19] Ran Canetti, Yilei Chen, Justin Holmgren, Alex Lombardi, Guy N. Rothblum, Ron D. Rothblum, and Daniel Wichs. Fiat–shamir: From practice to theory. In *Symposium on Theory of Computing (STOC 2019)*, 2019.

[CCXY18] Ignacio Cascudo, Ronald Cramer, Chaoping Xing, and Chen Yuan. Amortized complexity of information-theoretically secure MPC revisited. In *Advances in Cryptology – CRYPTO 2018*, 2018.

[DP23] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. Cryptology ePrint Archive, Paper 2023/1784, 2023. URL: https://eprint.iacr.org/2023/1784, doi:10.1007/978-3-031-91134-7_4.

[DP26] Benjamin E. Diamond and Jim Posen. Polylogarithmic proofs for multilinears over binary towers. In *Advances in Cryptology – EUROCRYPT 2026*, pages 3–32. Springer, 2026. Cryptology ePrint Archive, Paper 2024/504. URL: https://eprint.iacr.org/2024/504, doi:10.1007/978-3-032-25336-1_1.

[DT24] Quang Dao and Justin Thaler. Constraint-packing and the sum-check protocol over binary tower fields. Cryptology ePrint Archive, Paper 2024/1038, 2024. URL: https://eprint.iacr.org/2024/1038.

[GKR08] Shafi Goldwasser, Yael Tauman Kalai, and Guy N. Rothblum. Delegating computation: Interactive proofs for muggles. In *Symposium on Theory of Computing (STOC 2008)*, pages 113–122, 2008.

[GPR21] Lior Goldberg, Shahar Papini, and Michael Riabzev. Cairo: a turing-complete STARK-friendly CPU architecture. Cryptology ePrint Archive, Paper 2021/1063, 2021. URL: https://eprint.iacr.org/2021/1063.

[Gru24] Angus Gruen. Some improvements for the piop for zerocheck. Cryptology ePrint Archive, Paper 2024/108, 2024. URL: https://eprint.iacr.org/2024/108.

[HBG+18] Andreas Hülsing, Denis Butin, Stefan-Lukas Gazdag, Joost Rijneveld, and Aziz Mohaisen. XMSS: eXtended Merkle Signature Scheme. RFC 8391, 2018. URL: https://datatracker.ietf.org/doc/html/rfc8391.

[HJR+25] Tamir Hemo, Kevin Jue, Eugene Rabinovich, Gyumin Roh, and Ron D. Rothblum. Jagged polynomial commitments (or: How to stack multilinears). Cryptology ePrint Archive, Paper 2025/917, 2025. URL: https://eprint.iacr.org/2025/917.

[Irr] Irreducible. Multi-multiset matching (M3). https://www.binius.xyz/basics/binius-v0/m3/. Binius documentation.

[Jo26] Sunghyeon Jo. Interleaving stability for mutual correlated agreement and curve decodability. Cryptology ePrint Archive, Paper 2026/891, 2026. URL: https://eprint.iacr.org/2026/891.

[LCH14] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han. Novel polynomial basis and its application to reed–solomon erasure codes. In *Foundations of Computer Science (FOCS 2014)*, 2014.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4), 1992.

[MWKR26] Long Meng, Benedikt Wagner, George Kadianakis, and Francesco Risitano. LeanDA: Design and benchmark. Ethereum Research, 2026. URL: https://ethresear.ch/t/leanda-design-and-benchmark/25642.

[NA25] Andrija Novakovic and Guillermo Angeris. Ligerito: A small and concretely fast polynomial commitment scheme. Cryptology ePrint Archive, Paper 2025/1187, 2025. URL: https://eprint.iacr.org/2025/1187.

[Nat15] National Institute of Standards and Technology. SHA-3 standard: Permutation-based hash and extendable-output functions. Federal Information Processing Standards Publication 202, U.S. Department of Commerce, 2015. doi:10.6028/NIST.FIPS.202.

[Pol] Polygon Zero. Plonky3. https://github.com/Plonky3/Plonky3. The `challenger` crate's `DuplexChallenger`.

[S+26] Lev Soukhanov et al. Hashcaster 2: Newfanged RMFE in polynomials, 2026. URL: https://github.com/levs57/hashcaster2.

[Sho97] Peter W. Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5):1484–1509, 1997.

[Wag26] Benedikt Wagner. Formally verified security for PQ-DAS / leanDA. Ethereum Research, August 2026. URL: https://ethresear.ch/t/formally-verified-security-for-pq-das-leanda/25746.