Archive Certificates for ML-DSA-87 Authorizations from an Exact Bounded Relation

Yacine Alloul
Qlyphs
yacine@qlyphs.com

Preprint, version 2 · 8 November 2026

Abstract. A blockchain that signs every transaction with FIPS 204 ML-DSA-87 keeps 4,627 bytes of signature per transaction forever. We describe and measure an *archive certificate*: one hash-based proof, for a range of already validated blocks, of knowledge of canonical FIPS 204 signatures on exactly the recorded messages under the keys resolved from chain state. Fresh blocks keep native verification. The construction starts from an exact bounded relation for ML-DSA-87 verification once the high bits $u = w'_1$ are fixed. The main certificate keeps u hidden and proves the SHAKE256 challenge hash inside the proof, by a GKR argument for Keccak-f[1600] over the same WHIR commitment as the lattice relation.

For 65,536 signatures we measured 104.09 bytes per signature in a Johnson-regime family, which has a classical bound only, and 126.89 in a unique-decoding family, against 4,627 bytes raw. The price is a slow prover and a verifier slower than native verification. At 2^{64} queries and 2^{20} verification submissions, under the compiler, coding and extractor-efficiency hypotheses specified in Section 11, the classical random-oracle extraction error is below $2^{-188.07}$ for the Johnson family and below $2^{-201.99}$ for unique decoding. In the quantum random oracle model we state, for the unique-decoding family only, an extraction-failure bound below $2^{-129.67}$ at 2^{64} quantum queries and 2^{20} verification submissions; it is not a work factor.

Lean 4 checks Lemmas 1 to 4 and the WHIR steps at the scopes detailed in Section 12, the registered interactive theorem's conditions K1–K4, exact ledger arithmetic, and conditional compiler compositions. The extractor's efficiency condition K5 is derived in a defined bit-cost model from two implementation premises that remain open. The link between the interactive predicate and the ledger is proved within an abstract compiler interface; concrete compiled security games and correspondence with the Rust verifier remain outside the checked result. The bundles identified in Section 12 are also verified with AXLE. An external review is outstanding, and certificate-only bootstrap remains disabled under the deployment conditions of Section 11.12.

Keywords: ML-DSA, FIPS 204, arguments of knowledge, LogUp, GKR, Keccak, WHIR, Fiat–Shamir, quantum random oracle, blockchain history.

1 Introduction

1.1 Signatures that outlive their purpose

ML-DSA-87, the category 5 parameter set of FIPS 204 [1], has 2,592-byte public keys and 4,627-byte signatures [X]. In a chain where it is the only transaction signature, blocks of up to 128 transactions every two seconds (the laboratory configuration measured in this paper) carry up to 64 signatures per second, a full 2,048-signature range spans 32 seconds, and the raw signatures alone come to as much as 25.6 GB per day [D].

A signature does its work when its block is validated. Afterwards it is evidence: a node that joins later and wants to check history from the genesis anchor needs it, or something that convinces it just as well. A node that has already validated a block can delete the signatures for itself; that needs no cryptography. The hard case is the new node. For it a replacement must be an argument of *knowledge*: an adversary that can produce an accepted replacement must be able to produce the signatures themselves. Knowledge extraction is the basis for reducing a forged history to an ML-DSA-87 forgery; the complete authorization-security reduction, including state and history binding, remains future work (Section 11.11). Plain soundness ("valid signatures exist") is not enough, since a valid signature exists on every message the key holder could sign.

1.2 This work

We build such a replacement, an **archive certificate**, and measure it on real ML-DSA-87 signatures from an unmodified signer (libcrux-ml-dsa [2]). It is designed for a chain in which every transaction carries one ML-DSA-87 signature [3], but nothing in it depends on that chain beyond the record format of Section 2.2. Our contributions:

1. **An exact bounded relation** for ML-DSA-87 verification once $u = w_1'$ is fixed: an interval form of `UseHint` with exact endpoint corrections, proved in Lean and independently tested exhaustively, and a ternary region selector whose square is the hint bit (Section 3).
2. **A public-u certificate** (profile QLYAUTH7) specialised to that relation: one commitment, paired LogUp leaves against one shared table, random projections modulo q before lifting to the proof field, integer lifts with a proven no-wrap bound, and one sumcheck that carries every remaining identity (Section 4). It is the baseline, at 1,310 bytes per signature [M].
3. **A hidden-u certificate** that proves the challenge hash inside the same proof: bit planes of u in the commitment, a data-parallel GKR argument for the nine Keccak-f[1600] permutations of each challenge, a masked output check, a degree-3 input link for the sponge, and one WHIR opening of 29 values (Section 5). In the Johnson family it is 1.88 times smaller than the public-u certificate at 2,048 signatures [D] and reaches 104.09 bytes per signature at 65,536 [M], with a classical bound only; the unique-decoding family, which also has the quantum bound, takes 126.89 [M].
4. **A full-history Fiat–Shamir transcript** registered round by round, with a round registry exported by the verifier, and two transcript defects that our internal review found, which we report as lessons: Merkle node hashing shared across trees, and a projection decoder that silently zero-padded at large N (Section 6).
5. **Two parameter families** (Section 7): Johnson-regime WHIR with a classical claim only, and unique-decoding WHIR, which also carries a quantum claim.
6. **Measurements and adversarial tests** (Sections 8 to 10), including a prover-only optimisation that writes byte-identical certificates (Section 8.5), runs inside the nodes of three-node devnets (Section 8.7), and **a security analysis** (Section 11) in which every statement is labelled proven, cited, tested, or open, with the parts machine-checked in Lean listed in Section 12.

Positioning. Lattice-based aggregation of Falcon is far ahead on size and speed. Aerie [4], over the Akita lattice commitment [5], aggregates 32,768 Falcon-512 signatures in 0.65 s at 42.1 bytes per signature with salts, and 65,536 Falcon-1024 signatures in 1.86 s at about 41 bytes per signature, on a 16-thread laptop, with a verifier faster than native verification [X]. Our smallest figure, 104.09 bytes per signature at 65,536, took 3,075.5 s to prove on a shared 24-

vCPU server and verifies more slowly than native verification [M]. Aerie covers Falcon, whose hash input is public, and its knowledge soundness is proved in the classical random-oracle model; the Akita authors name a quantum random-oracle analysis as the main open soundness issue of their commitment [X]. LayerZero has announced ML-DSA aggregation in Aerie without figures [6] [X], and Hekate [7] proves exact ML-DSA-87 verification, SHAKE included; we found no comparable archive-compression measurement for it (Section 13). Our contribution is therefore narrower than "aggregation": a hash-based argument of knowledge for batches of unmodified FIPS 204 ML-DSA-87 signatures that proves the challenge hash inside the proof, is smaller than the raw signatures (at 65,536 signatures, 44 times for family J and 36 times for the unique-decoding family [D]), and has a stated classical bound in the proven Reed–Solomon regimes and, for its unique-decoding family only, a stated quantum random-oracle bound. We know of no other published measurement with all of these properties, and we claim no priority over the announced ML-DSA work in Aerie.

Main results. Sizes and times are measured [M] on real ML-DSA-87 signatures (Table 3, Table 4); security figures are analytic bounds [A] (Table 11), not measurements.

	Family J	Family U
Bytes per signature, $N = 2048$[a]	697.4 [M]	1,330.7 [D]
Bytes per signature, $N = 16{,}384$	210.6 [M]	254.8 [M]
Bytes per signature, $N = 65{,}536$	**104.09** [M]	**126.89** [M]
Raw ML-DSA-87 signature	4,627 bytes [X]	
Smaller than raw, $N = 65{,}536$	44.45 times, 97.8 % [D]	36.5 times, 97.3 % [D]
Prover, $N = 65{,}536$, first prover	3,075.5 s [M]	3,095.3 s [M]
Prover, $N = 65{,}536$, optimised[b]	not run	677.4, 749.7 s [M]
Verifier, $N = 65{,}536$, fresh process[c]	17.3–20.0 s [M]	16.6–17.5 s [M]
Classical extraction error, 2^{64} queries, 2^{20} submissions	below $2^{-188.07}$ [A]	below $2^{-201.99}$ [A]
Quantum extraction failure, 2^{64} quantum queries, 2^{20} submissions	not claimed	below $2^{-129.67}$ [A]

J: Johnson-regime WHIR, classical bound only; U: unique-decoding WHIR, classical and quantum bounds (Section 7). [a] Standalone statements; the larger N are chain ranges. [b] Same certificates byte for byte, on two other hosts; four later reruns took 782.5 to 1,967.9 s (Section 8.6) [M]. [c] Native verification of 65,536 signatures takes about 4.6 s [D]. Times are contended wall-clock figures on shared 24-vCPU servers. The bounds hold uniformly over the audited sizes 128 to 65,536, rest on the hypotheses of Section 11, and are failure probabilities at a stated budget, not work factors.

1.3 What we do not claim

We do not claim the smallest or the fastest aggregate: lattice-based Falcon aggregation is smaller and faster. We do not claim faster verification: checking a certificate costs several times native verification of the same signatures. We do not claim zero knowledge, data availability, a reduction of fresh-block traffic, or that certificate-only bootstrap preserves the NIST category 5 security of ML-DSA-87. Certificates sit outside consensus: they never enter block validity or fork choice, and certificate-only bootstrap is disabled (Section 11.12). Every bound below is a failure probability at a stated resource budget, not a work factor, and every bound rests on hypotheses we list. No quantum bound is claimed for the public-u profile or for the Johnson

family, so the 104.09 bytes per signature of the Johnson family go with a classical bound only; the quantum bound goes with the 126.89 bytes per signature of the unique-decoding family. At least half of either figure is the public challenge digest $\tilde{c}_i$, 64 bytes per signature, which our accumulator design requires and which the proof does not compress (Section 8.3). The hidden-u profiles are laboratory profiles; apart from the devnet of Section 8.7, which runs the unique-decoding profiles under its own identifiers, no deployment uses them.

1.4 Evidence conventions

Every number is tagged by how it was obtained. [M] **measured**: produced by running the code, with a transcript. [D] **derived**: arithmetic on cited inputs. [A] **analysis**: an error bound or count from the security analysis, recomputed by a script. [X] **specification**: a parameter of a profile, of a standard, or a figure reported by another work. A claims ledger kept with the paper source maps every number to the file and line it comes from, and a script checks that each cited line still holds the quoted text. Statements of the security analysis carry a second label, defined in Section 11.1: `proven`, `cited`, `tested` or `open`.

The initial measurements used two hardware configurations. The public-u profile and the first hidden-u build were measured on one Apple M3 Pro laptop (11 logical CPUs, 18 GiB) [X] that other jobs shared, at one-minute load averages of 16 to 120. The final hidden-u profiles were measured in a rented container limited to 24 logical CPUs and 24 GB of memory on an Intel Xeon 6975P-C host [X], also shared, at one-minute load averages of 17 to 41. The optimised prover of Section 8.5 was measured later in containers with the same quota on three other shared hosts, one Intel Xeon Platinum 8581C and two AMD EPYC 9655P [X], and its 65,536-signature unique-decoding run was repeated from a fresh build on three more, one Intel Xeon 6975P-C and two AMD EPYC 9655P (hosts D, E and F of Section 8.6) [X]. Times are therefore contended wall-clock figures, comparable only between runs on the same host; byte counts are deterministic for a given statement.

2 Background

2.1 ML-DSA-87 verification

Let $q = 8380417$, $n = 256$, $(k, \ell) = (8, 7)$, $\gamma_1 = 2^{19}$, $\gamma_2 = (q-1)/32 = 261888$, $a = 2\gamma_2 = 523776$, $\beta = 120$, $\omega = 75$, $\tau = 60$ and $d = 13$ [X]. A public key is (ρ, t_1); a signature is $(\tilde{c}, z, h)$ with $z \in R_q^\ell$ and a hint vector $h \in \{0,1\}^{kn}$ of weight at most ω. Throughout, pk is a canonically encoded 2,592-byte public key, each coefficient of t_1 lies in $\{0, ..., 1023\}$, and `ctx` is a byte string of length at most 255. All polynomial products below are in $R_q = \mathbb{Z}_q[X]/(X^{256}+1)$. In pure mode with context `ctx`, verification computes the NTT-domain matrix $\hat{A} = \mathrm{ExpandA}(\rho)$ and its coefficient-domain form $A = \mathrm{NTT}^{-1}\left(\hat{A}\right)$, with the inverse NTT applied entrywise, $\mathrm{tr} = H(\mathrm{pk})$, $\mu = H(\mathrm{tr} \parallel 0 \parallel |\mathrm{ctx}| \parallel \mathrm{ctx} \parallel M)$, in which the mode byte 0 and the context length are each encoded in one byte, $c = \mathrm{SampleInBall}(\tilde{c})$ and

$$w' = Az - c \cdot 2^d t_1, \quad w_1' = \mathrm{UseHint}(h, w'),$$

and accepts if and only if the encodings are canonical, $\|z\|_\infty < \gamma_1 - \beta$, the hint weight is at most ω, and $\tilde{c} = H(\mu \parallel \mathrm{w1Encode}(w_1'))$, where H is SHAKE256 with 64-byte output [8]. The challenge hash takes w_1' as input, and w_1' depends on z: this loop is what makes ML-DSA harder to aggregate than Falcon, whose hash-to-point input is public. The hash input

$\mu \parallel \mathrm{w1Encode}(w'_1)$ has $64 + 1024 = 1088$ bytes, exactly eight blocks of the 136-byte SHAKE256 rate, so each challenge costs nine Keccak-f[1600] permutations, the ninth for the padding block [D].

2.2 Chain records and the accumulator

An `AuthRecord` holds a version, the 20-byte sender, a key epoch, a context and a payload. The signed message is a fixed domain string, the 64-byte network identifier and the record; the key is never carried with the record but resolved from state by (sender, key epoch). Each block header carries a 64-byte accumulator

$$A[b] = H(\text{accumulator}, \text{net} \parallel b \parallel A[b-1] \parallel \text{auth_root}[b]),$$

where $H(D, \cdot)$ denotes SHAKE256 under the domain string D and $\text{auth_root}[b]$ hashes, for every record of block b, the record, its state-resolved public key and the challenge digest $\tilde{c}_i$ [X]. A node validating a fresh block verifies every signature natively and takes $\tilde{c}_i$ from the first 64 bytes of the signature. The accumulator binds a later certificate to the records, state-resolved public keys and challenge digests of the signatures the chain accepted: the certificate must reproduce every $\tilde{c}_i$, hence every $A[b]$. In our laboratory chain the headers are synthetic (SHA3-256 of parent, height and accumulator); proof of work, transaction roots and bootstrap are not built [X].

2.3 Tools

We use multilinear extensions over Boolean hypercubes and the sumcheck protocol [9]; the logarithmic-derivative lookup argument LogUp [10] and its fractional GKR form [11], [12]; Gruen's splitting of the equality factor [13]; the WHIR polynomial commitment [14] as implemented in Plonky3 0.8.0 [15]; and the BCS transform of interactive oracle proofs [16] in the full-history form analysed by Chiesa and Fenzi [17], with the quantum analysis of Chiesa, Di, Hu and Zheng [18]. The proof field is Goldilocks, $p = 2^{64} - 2^{32} + 1$, with challenges in $K = \mathbb{F}_p[X]/(X^5 - 3)$ [X], so $|K| = p^5 > 2^{319}$.

3 An exact bounded relation

3.1 Fixed high bits and the interval form of UseHint

Fix $u = w'_1 \in \{0, \ldots, 15\}^{kn}$ for each signature: 4 bits per coefficient, 1,024 bytes once encoded. Define the centred residual $e = \mathrm{ctr}_q(w' - au)$, where ctr_q maps $\mathbb{Z}_q$ to $[-(q-1)/2, (q-1)/2]$. The key arithmetic fact is $q = 16a + 1$ [D]: the sixteen high-bit classes tile $\mathbb{Z}_q$ in steps of a, with one extra residue at the wrap-around.

Lemma 1 (interval form). proven in Lean (Section 12); independently tested, exhaustively (below). For every $r \in \mathbb{Z}_q$ and $u \in \{0, \ldots, 15\}$, let $e = \mathrm{ctr}_q(r - au)$. There exists $h \in \{0, 1\}$ with $\mathrm{UseHint}(h, r) = u$ if and only if

$$-a + 1 - [u = 0] \le e \le a + [u = 15].$$

When it exists, h is unique, and $h = 0$ exactly when $-\gamma_2 + 1 - [u = 0] \le e \le \gamma_2$.

The bracket $[P]$ is 1 if P holds and 0 otherwise. The endpoint corrections come from FIPS 204 `Decompose`, which maps the residues next to $q - 1$ to high bits 0 with a low part shifted by one; an interval form without them is wrong at exactly those residues. We checked Lemma

1 exhaustively against FIPS 204 Algorithms 36 to 40, written independently in C: $16q = 134{,}086{,}672$ comparisons, no existence mismatch, no non-unique hint and no hint-rule mismatch over the 16,760,834 admissible cases [M]. The count is what the lemma predicts: $2a$ residuals per value of u, plus one for $u = 0$ and one for $u = 15$, $16 \cdot 2a + 2 = 16{,}760{,}834$ [D]. The independent C enumeration is regression evidence over the full finite domain. The interval statement and the ternary cover are also proved in Lean, as described in Section 12. On 128 real signatures, none of the 262,144 residuals fell outside its interval and the interval rule rebuilt every hint bit [M].

3.2 Ternary regions

The three sub-intervals of Lemma 1 are the region of no hint and the two regions on either side (Figure 1). We index them with $S \in \{-1, 0, 1\}$, so that the hint is $h = S^2$:

S	**Residual interval**	$h = S^2$	**Width** $\Delta(u, S)$
-1	$[-a + 1 - d_0, -\gamma_2 - d_0]$	1	$\gamma_2 - 1$
0	$[-\gamma_2 + 1 - d_0, \gamma_2]$	0	$a - 1 + d_0$
$+1$	$[\gamma_2 + 1, a + d_{15}]$	1	$\gamma_2 - 1 + d_{15}$

Table 1: The ternary regions, with $d_0 = [u = 0]$ and $d_{15} = [u = 15]$. The width is the interval length minus one; a residual lies in region S if and only if $e = Y + \text{start}(u, S)$ with $0 \le Y \le \Delta(u, S)$.

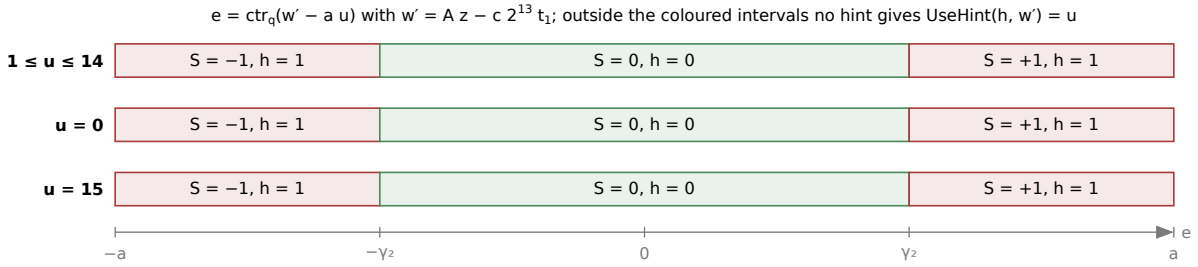


Figure 1: The three regions of Lemma 1, drawn to scale on the residual axis $[-a - 1, a + 1]$. Red regions carry a hint bit, the green region does not. The $u = 0$ row starts one step lower, at $-a$, and the $u = 15$ row ends one step higher, at $a + 1$: these are the two wrap-around corrections of `Decompose`. Each is one residue out of about 10^6, too small to be visible at this scale; Table 1 gives the exact endpoints.

Both $\text{start}(u, S)$ and $\Delta(u, S)$ are quadratic polynomials in S whose coefficients depend only on the class of u (0, 15, or interior). For example $\Delta = \Delta_0 + \Delta_1 S + \Delta_2 S^2$ with $\Delta_0 = a - 1 + d_0$, $\Delta_1 = d_{15}/2$ and $\Delta_2 = -\gamma_2 - d_0 + d_{15}/2$ [X]. For each fixed u, the map $e \mapsto (S, Y)$ is a bijection from admissible residuals onto $\{(S, Y) : S \in \{-1, 0, 1\}, 0 \le Y \le \Delta(u, S)\}$; the implementation's self-test checks the mapping on all 16,760,834 admissible pairs (e, u) [M].

3.3 The certified relation

For records $i = 0, \ldots, N - 1$ in replay order, with $(\text{pk}_i, \text{ctx}_i, M_i)$ resolved from state, the verifier derives μ_i, the NTT-domain matrix $\hat{A}_i = \text{ExpandA}(\rho_i)$, its coefficient-domain form $A_i = \text{NTT}^{-1}\big(\hat{A}_i\big)$, and $c_i = \text{SampleInBall}(\tilde{c}_i)$. The certificate claims that for every i there are integer vectors $z_i \in \mathbb{Z}^{\ell n}$ and $e_i \in \mathbb{Z}^{kn}$, high bits $u_i \in \{0, \ldots, 15\}^{kn}$ and hints $h_i \in \{0, 1\}^{kn}$ with

$$A_i z_i - c_i 2^{13} t_{1,i} - a u_i \equiv e_i \pmod q, \quad \|z_i\|_\infty \le 524167, \quad e_{i,t} \text{ in the region of } \big(u_{i,t}, h_{i,t}\big),$$

$$\sum_t h_{i,t} = \nu_i \le 75, \quad \tilde{c}_i = H(\mu_i \,\|\, \text{w1Encode}(u_i)).$$

Lemma 2 (equivalence). proven given Lemma 1. Any such witness yields a canonical FIPS 204 signature $(\tilde{c}_i, z_i, h_i)$ on $(\mathrm{pk}_i, \mathrm{ctx}_i, M_i)$, and every valid signature yields such a witness.

The forward direction is Lemma 1 applied coefficientwise: the region bounds give $|e| \le a + 1 < (q-1)/2$, so e is the centred residual $\mathrm{ctr}_q(w' - au)$ of Lemma 1. It also uses $|z| \le 524167 < \gamma_1 - \beta$ [D], the weight bound, the hash equality, and the canonical hint packing (sorted indices, zero padding). The converse is immediate from verification. Three remarks. First, the hint count is a constraint *per signature*: an epoch-wide budget $\sum_i \nu_i \le 75N$ would be a different relation and would not be sound, since it admits fixed public records, with their published hint counts, that no list of accepted signatures matches (two records with counts 76 and 0, say). Second, the lemma gives a *canonical* signature; it does not say that the extracted signature is byte-identical to the one broadcast at inclusion, and nothing in this paper depends on that. Third, the relation leaves a choice of what to publish. Publishing u_i turns the last condition into a native hash computation (Section 4); publishing only $\tilde{c}_i$ moves it into the proof (Section 5).

4 The public-u certificate (QLYAUTH7)

This profile publishes $U_i = \mathrm{w1Encode}(u_i)$ with each signature. Every verifier then recomputes $\tilde{c}_i = H(\mu_i \parallel U_i)$ and c_i natively, and no hash needs to be proven. We describe the profile frozen at $N = 2048$ briefly: it is the baseline, and the hidden-u certificate of Section 5 reuses each of its components.

Witness and commitment. Each signature owns 2,048 cells $b = i \cdot 2048 + t$. Three columns are committed: $X = z + 524167$ on the first 1,792 cells (the eighth slot is unconstrained padding), the selector S, and $Y = e - \mathrm{start}(u, S)$ on all 2,048 cells. The multiplicities of the shared table $\{0, ..., 2^{20} - 1\}$ are committed as two columns of 2^{19} entries [X]. One WHIR commitment holds the stacked polynomial of 24 variables, with starting rate 1/16, the Johnson-bound regime, target 272 bits, no grinding, and 141, 80, 62 and 51 queries per phase [X].

Ranges by paired LogUp leaves. The bounds become four lookups per cell: X and $1048334 - X$ (so $|z| \le 524167$), and Y and $\Delta(u, S) - Y$ (so e lies in region S). Each complementary pair costs one fractional leaf, because

$$\frac{1}{\theta - x} + \frac{1}{\theta - (\Delta - x)} = \frac{2\theta - \Delta}{\theta^2 - \theta\Delta + x(\Delta - x)}.$$

Table entries are paired the same way. The ideal law of the lookup challenge θ is uniform on $K \setminus \mathbb{F}_p$. The registered decoder forms three candidates from the accepted words of its pool, padded with zeros when the pool runs short, takes the first that has a non-zero coordinate outside the constant one, and returns X if none has; its output therefore always lies outside $\mathbb{F}_p$, so no leaf denominator of a base-field witness vanishes, and its deviation from the ideal law is part of the coin-decoder distance (Section 6). At $N = 2048$ the 2^{19} table leaves sit exactly in the tree positions of the ignored X padding, so the tree has exactly 2^{23} leaves [X]. Two facts make the integer conclusion exact. The two members of each pair are below 2^{20}, so their integer sum is below $2^{21} \ll p$: no field wrap can fake a membership. The lookups count at most $7680N$ occurrences, below 2^{29} at $N = 65{,}536$ [D], far below the characteristic p, as the LogUp argument requires. And the lookups do not imply $S \in \{-1, 0, 1\}$: the value $S = 1/2$ with a suitable Y

passes every lookup. Ternarity is therefore a separate zerocheck $\sum_b \mathrm{eq}(\xi, b)\big(S_b^3 - S_b\big) = 0$, at a random point ξ, tested in isolation (Section 9).

Projection modulo q before lifting. The relation lives modulo q, and the proof field is $\mathbb{F}_p$. The verifier draws, per trial j, $\alpha_j \in \mathbb{F}_q^N$ and $\eta_j \in \mathbb{F}_q^{kn}$ and computes natively

$$r_{j,i} = \mathrm{ctr}\big(\alpha_{j,i}\eta_j\big), \quad s_{j,i} = \mathrm{ctr}\Big(\alpha_{j,i}A_i^\dagger\eta_j\Big), \quad d_{j,i} = \alpha_{j,i}\langle\eta_j, b_i\rangle \bmod q,$$

with $b_i = c_i 2^{13} t_{1,i} + a u_i$ and $A^\dagger$ the adjoint in the *coefficient* domain: the transpose of A with each entry f replaced by $f^*(X) = f(X^{-1}) = f_0 - \sum_{k=1}^{255} f_{256-k}X^k$. In the NTT domain of FIPS 204 this is the module transpose together with the slot reversal $m \mapsto 255 - m$; transposing only the module indices is wrong. For each chunk of 512 signatures and each trial the prover sends one integer $V_{c,j} \in [-2^{39}, 2^{39})$ [X] and the proof checks

$$\sum_{i \in c}\big(\langle s_{j,i}, z_i\rangle - \langle r_{j,i}, e_i\rangle\big) - \mathrm{ctr}\left(\sum_{i \in c} d_{j,i} \bmod q\right) = qV_{c,j}.$$

Once the lookups give $|z| \le 524167$ and $|e| \le 523777$, the difference of the two sides is bounded in absolute value by 8,923,706,090,701,451,264, below p, so the identity over $\mathbb{F}_p$ implies the identity over $\mathbb{Z}$. One lift over 2,048 signatures with this quotient range would not fit [M]. For a fixed non-zero residual matrix, one trial misses it with probability at most $2/q - 1/q^2$, so fourteen trials give $(2/q - 1/q^2)^{14} \approx 2^{-307.98}$ [A]. On 128 real signatures, with the coefficient-domain adjoint all 14 trials pass and with the plain transpose none does; adding 1 to a coefficient of z or e is caught in 448 of 448 checks, while adding q to a coefficient of e passes all 448: only the range argument binds the integer witness, as intended [M].

Fractional GKR, one master sumcheck, openings. The 2^{23} leaves are summed by a fractional GKR tree of 23 layers; the verifier checks that the root numerator is 0 and the root denominator non-zero. Each layer round sends the two top coefficients (a_1, a_2) of Gruen's quadratic cofactor, and the verifier rebuilds $a_0 = c - x(a_1 + a_2)$ without division. An earlier version divided by x and rejected honest proofs when a challenge coordinate was 0, which a test found; the change also saved 10,160 bytes at $N = 2048$ [M]. One degree-4 sumcheck over 2^{22} cells then proves a random combination of the leaf links, the ternarity zerocheck, the 2,048 hint counts $\sum_t S_{i,t}^2 = \nu_i$ and the 56 projection identities, and WHIR opens five values of the one commitment [X]. The verifier accepts no prover-supplied public term.

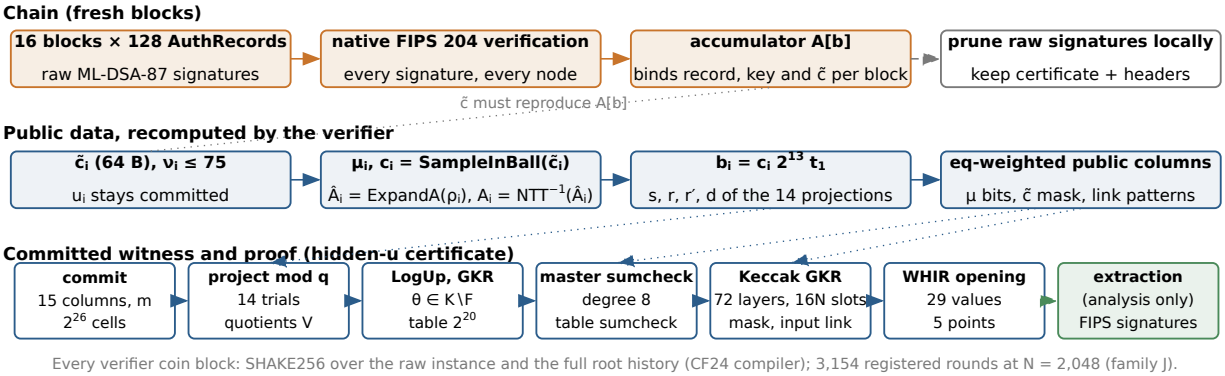
Binding to the chain. The certificate verifier checks header linkage, replays the records with keys from state, computes $\tilde{c}_i$, recomputes every $A[b]$ and compares it with the header, and only then checks the proof against a manifest that it derives from the headers. No key or statement digest is taken from the certificate.

At $N = 2048$ this certificate takes 2,682,945 bytes, 1,310.03 bytes per signature [M]. The public part, 1,024 bytes of u and one byte of ν per signature, is 78 % of it and sets a floor of 1,025 bytes per signature [D]. The next section removes it.

5 The hidden-u certificate

The public u costs 1,024 bytes per signature [X]; the challenge digest $\tilde{c}_i$ costs 64. The hidden-u certificate publishes $\tilde{c}_i$ and ν_i, keeps u_i committed, and proves $\tilde{c}_i = H(\mu_i \parallel \mathrm{w1Encode}(u_i))$

inside the same proof, over one WHIR commitment shared with the lattice part. Figure 2 shows the data flow.



Every verifier coin block: SHAKE256 over the raw instance and the full root history (CF24 compiler); 3,154 registered rounds at N = 2,048 (family J).

Figure 2: Data flow of the hidden-u certificate. Top: the chain verifies every fresh signature natively and binds its challenge digest into the header accumulator. Middle: what the certificate verifier recomputes from public data. Bottom: the committed witness and the proof phases, in transcript order. The extraction box is part of the analysis, not of the verifier.

5.1 Committed witness

Table A has 15 columns of $2^{11+\log_2 N}$ cells, cell $i \cdot 2048 + t$: X, Y and S as before; four bit planes $B_0, \ldots, B_3$ with $u_t = B_0 + 2B_1 + 4B_2 + 8B_3$; and $T_1, \ldots, T_8$, the sponge states of each signature after permutations 1 to 8, in the signed encoding $\mathrm{enc}(b) = 1 - 2b$ on the 1,600 state cells and 0 on the 448 padding cells [X]. That is 30,720 cells per signature [X]. Table B holds the multiplicities. The library stacks both tables into one multilinear polynomial in $\mathrm{nv} = \lceil \log_2(15 \cdot 2048N + 2^{20}) \rceil$ variables, 26 at $N = 2048$ [X].

The lattice part changes in three places.

- **Region polynomials.** They depend on u only through $d_0 = [u = 0]$ and $d_{15} = [u = 15]$. With committed bits these are $d_0 = \prod_j (1 - B_j)$ and $d_{15} = \prod_j B_j$, products of committed columns *inside the summand* of the master sumcheck, which therefore has seven committed and seven public columns and total degree at most 8. Its top monomial $w_Y \cdot Y \cdot S^2 \cdot B_0 B_1 B_2 B_3$ receives $-\gamma_Q$ from the d_0 term and $+\gamma_Q/2$ from the d_{15} term, where γ_Q is the leaf-batch weight (Appendix C), so its coefficient is $-\gamma_Q/2$: the degree is exactly 8 when $\gamma_Q \neq 0$ and at most 7 when $\gamma_Q = 0$. Its terminal check evaluates the summand on the opened values; it never claims that the multilinear extension of d_0 equals $\prod_j (1 - \tilde{B}_j)$, which is false.
- **Booleanity.** $\sum_b \mathrm{eq}(\xi, b) \sum_j \gamma_{B_j} (B_j^2 - B_j) = 0$ joins the ternarity zerocheck, with four *independent* weights. A single weight would not suffice: in $\mathbb{F}_p$ both 5 and -3 are squares, so $B_0 = (1 + \sqrt{5})/2$ and $B_1 = (1 + \sqrt{-3})/2$ give $(B_0^2 - B_0) + (B_1^2 - B_1) = 1 - 1 = 0$ although neither is a bit.
- **Integer lift.** The u term now belongs to the witness side. Moving au there unreduced makes the lift wrap, since $1/2(q-1) \cdot 2048 \cdot 512 \cdot 523776 \cdot 15 \approx 3.45 \cdot 10^{19} > p$. The term therefore enters as $-\langle r'_{j,i}, u_i \rangle$ with $r' = \mathrm{ctr}(\alpha\eta a \bmod q)$, which is exact modulo q and keeps the bound at 8,923,771,996,974,608,384 $< p$ [M]. This needs $u \leq 15$, so Booleanity must stay an exact obligation until its zerocheck.

5.2 Proving the challenge hash

Keccak GKR. One data-parallel GKR argument runs the 72 layers of Keccak-f[1600] (24 rounds of three layers: θ-a of degree 5, θ-b and ρ-π-χ-ι of degree 3) [M] over $16N$ slots of 2,048 cells, slot $16i + g$ with $g \in \{0, ..., 15\}$. Slots $g < 9$ are the nine permutations of signature i; slots 9 to 15 hold the zero vector, which every layer maps to zero. In the signed encoding each layer is a polynomial map: a product of encodings is the encoding of the XOR, and χ is $A(1 - (1 + B)(1 - C)/2) = \mathrm{enc}(a_0 \oplus (\neg a_1 \wedge a_2))$ with $(A, B, C) = (\mathrm{enc}(a_0), \mathrm{enc}(a_1), \mathrm{enc}(a_2))$, checked on all eight inputs [M]. Each layer is reduced by one sumcheck, with $4 + \log_2 N$ normalized Gruen rounds over the slot variables and 11 rounds over the cell variables, then a batch of the gathered values and an 11-round wiring sumcheck down to a claim on the layer input [X]. At $N = 2048$ the Keccak GKR takes 384,000 bytes [M].

Masked output. The output of slot $g < 8$ must equal the committed T_{g+1}, and the first 512 cells (64 bytes) [X] of slot 8 must equal $1 - 2\tilde{c}_i[s]$ on cell $s < 512$, where $\tilde{c}_i[s]$ is bit s of the public digest; nothing else of the output is constrained. The prover sends the eight values $o_k = \tilde{T}_k(\xi_{\mathrm{out}})$ at a random point ξ_{out}, which become opening claims, and a degree-2 sumcheck of $\mathrm{eq}(\xi_{\mathrm{out}}, b) \cdot \mathrm{mask}(b) \cdot \mathrm{out}(b)$ reduces the masked claim to one claim on the last layer, where the Keccak GKR starts. The verifier evaluates the masked equality weight in closed form.

Input link. The input of slot g is the previous state XORed with the absorbed block, and in the signed encoding $\mathrm{enc}(s \oplus b) = \mathrm{enc}(s)(1 - 2b)$ for a bit b. The Keccak GKR ends with one claim on the permutation inputs at a point ξ_{link}. The link reduces that claim, minus a public part that carries the bits of μ and the zero initial state, by a degree-3 sumcheck over (cell, signature) of $W_a \sum_{g=1}^{7} \lambda_g T_g + \lambda_8 W_{a8} T_8 + W_b \sum_{g=1}^{7} \lambda_g T_g U_g + \lambda_0 W_c U_0$, where the W are public cell patterns (state copy, padding block, rate, and the part of block 0 that carries u), λ_g are per-slot equality weights, and U_g are the committed bits of u re-indexed to the absorbed positions, low nibble first. The same identity converts signs explicitly: a real zero bit maps to +1, while dummy slots and padding cells map to 0. It cannot be replaced by a product of evaluations, because the XOR of multilinear evaluations is not the evaluation of the XOR. In the bit encoding, take the one-variable tables $s = (0, 1)$ and $b = (1, 0)$, listed as $(f(0), f(1))$, at the point 2: the multilinear extension of $s \oplus b$ is 1 there, while $x + y - 2xy$ applied to the extensions of s and b gives 5 [M].

Bit wiring and opening. The link ends at a point ξ' with claimed values $U_g(\xi')$, and a 13-variable degree-2 sumcheck [X] maps them to one opening of $B_0, ..., B_3$. One WHIR proof then opens 29 values at five points: $X, Y, S, B_0, ..., B_3$ at the master point, $T_1, ..., T_8$ at the masked-output point and again at the input-link point, $B_0, ..., B_3$ at the wiring point, and the two multiplicity columns at the table point [X]. All 29 values and the five points, and in family J the out-of-domain point and its answer, are fixed before the batching challenge.

5.3 Binding

The chain verifier of the laboratory recomputes every $A[b]$ from the certificate's $\tilde{c}_i$ and absorbs $\tilde{c}$ into the raw instance in place of U. With ν_i the public part is 65 bytes per signature [D]. The canonical signatures rebuilt from X, S, the bit planes and the public $\tilde{c}_i$ were identical to the originals and accepted by libcrux, 2,048 of 2,048 [M]. Appendix B gives the layout.

6 A full-history Fiat–Shamir transcript

6.1 Registered rounds

The first implementation of the public-u certificate used the library's streaming sponge. Its interactive analysis goes through, but no published compiler theorem covers that particular stream. Every later profile uses, as written, the full-commitment-history BCS construction analysed by Chiesa and Fenzi [17]. The protocol is cast as k registered rounds (from here on k counts rounds; it is unrelated to the module rank $k = 8$). Each round commits one oracle: a real Merkle tree (the witness, and each WHIR fold) or a singleton tree holding one scalar prover message, whose root both sides recompute and which is never transmitted. Round i's coin block is one SHAKE256 output over a domain string, the profile, the framed raw instance and the complete root history:

$$C_i = \mathrm{SHAKE256}(\mathrm{dom} \parallel \mathrm{profile} \parallel F(1, x) \parallel \mathrm{Entry}(1) \parallel \ldots \parallel \mathrm{Entry}(i) \parallel F(3, \mathrm{Desc}_i) \parallel i \parallel L).$$

The analysis uses this formal block of L bytes, and L is encoded in the input. The implementation squeezes only the prefix that the round's decoders read; since a shorter SHAKE256 output on the same input is a prefix of a longer one, those bytes are the first bytes of C_i, and the decoders read the same values from either (Section 12). The framing of x is injective on canonical statements, under two conditions. Records carry public keys without a length field, so record boundaries are recoverable only because every key has exactly 2,592 bytes [X]; the implementation asserts this length. Message lengths are written in four bytes. The serialization is proved injective on canonical statements whose messages are shorter than 2^{32} bytes. The implementation does not yet enforce this domain: for a longer message, the length field contains its length modulo 2^{32}. The verifier should reject such messages by a checked length conversion before digesting and serializing the statement. No injectivity claim outside the stated domain is made here.

Previous coin blocks are not inputs, salts are fixed to zero, and fixed pools replace rejection loops: extension elements, θ, the $14(N + 2048)$ projection values of round 1 and the query indices are decoded from fixed segments, with their statistical distance from uniform added to the interactive error. The public instance is absorbed raw, not as a digest. The WHIR library's challenger is replaced by an adapter that serves only registered calls and poisons the transcript on any other call. QLYAUTH7 has 354 rounds [X]; the hidden-u profiles have 3,154 rounds (Johnson) and 3,150 (unique decoding) at $N = 2048$.

Round registry. The verifier exports its own round table: for every round the phase, the claim it maintains, the sumcheck degree before and after normalization, the message and its body length, the coin-block shape and the terminal identity; and at the end the opening shape, the coin decoders and a census. The export asserts every row against the table the verifier enforces. In honest runs at $N = 128$ and $N = 2048$ every logged round matched the registry in body length and coin prefix [M]. The security ledger of Section 11 reads its round table from these registries.

6.2 Two defects found by our own review

Both defects were found by an internal adversarial review of the transcript against the hypotheses of the compiler theorems, before any large-N measurement. Both are fixed in the final profiles.

Merkle node hashing shared across trees. QLYAUTH7 hashes leaves with the profile and the round, but internal nodes as $H(\text{0x81} \parallel L \parallel R)$, one function shared by every tree. The classical compiler uses one Merkle oracle and does not need more, but the quantum compiler of [18] (Construction 11.7) needs an independent vector-commitment oracle per tree. The final profiles hash nodes as SHAKE256-512(0x81 ‖ LE32(profile) ‖ LE16(tree round) ‖ L ‖ R), a 135-byte input that fits one SHAKE256 block [X]. Tests check that the same (L, R) under two tree rounds or two profiles hash differently and that fold-tree nodes hashed under the witness tree's round are rejected, and the registry records that every node hash of prover and verifier ran under its own tree's commit round [M]. The shared function was a gap between the code and a theorem's hypotheses, not a demonstrated attack.

A projection decoder that zero-padded. The 14 projection trials draw $n_q = 14(N + 2048)$ values [X] of $\mathbb{F}_q$ from 3-byte words by rejection, from a pool with a fixed slack of 512 words; when the pool ran short, the remaining values were set to zero. At $N \le 2048$ a short pool is far too unlikely to matter. But each word is rejected with probability $8191/2^{23}$, so the expected number of rejections grows with N while the slack did not: with 512 words of slack the decoder distance is $2^{-153.9}$ at $N = 16{,}384$ and about 1 at $N = 65{,}536$, where on average about 412 projection coefficients of the last trial would be set to zero [A]. A zero coefficient is a coin the prover can predict, and the analysis charges the decoder distance to every round. The final profiles use a slack of $512 + \lceil n_q/400 \rceil$ words (656 at $N = 2048$, 2,878 at $N = 65{,}536$), for which the probability of a short pool is at most 2^{-768} [A], and a short pool makes the prover refuse and the verifier reject instead of padding [M]. The pools of extension elements are strict as well, and on the verifier's side every strict pool behaves alike: a shortfall poisons the transcript, and the verifier cannot accept. The honest prover has no immediate abort for a short extension-element pool, but a certificate it produces anyway carries the transcript error and is rejected. In the experiment with independent uniform raw coin blocks, the probability that at least one strict pool is exhausted is at most the sum of the registered per-round decoder allowances, which is at most $2^{-572.75}$ for every registered profile (Section 11.5) [A]. This bounds the strict-pool contribution to honest rejection; the 2^{-768} bound covers only the projection pool. We do not claim perfect completeness.

The lesson: a decoder distance computed once at the frozen size does not transfer to larger N. It has to be recomputed for every registered shape, which the ledger now does from the registries. A third defect, in the first version of the security argument rather than in the transcript, is described in Section 11.5.

7 Parameter families

The hidden-u certificate comes in two laboratory families that share the interactive protocol and differ only in the WHIR regime (Table 2).

- **Family J (Johnson)**, profiles `0xfff910xx`: WHIR in the Johnson list-decoding regime, planner target 272 bits [X], one out-of-domain sample per commitment, no grinding. The list size at rate ρ is $L = 10/\rho$, 40 for the first oracle at $N = 2048$ [X]. We claim a classical bound only.
- **Family U (unique decoding)**, profiles `0xfff912xx`: WHIR in the unique-decoding regime, planner target 286 bits [X], no out-of-domain sample, no grinding; the list size is 1. This family also carries the quantum claim of Section 11.7.

In both, the query count of every phase equals the library's planner formula for its regime and rate, which the ledger asserts, and no capacity conjecture is used anywhere. The library limits the final polynomial to six variables [X], and the schedules respect it. The chain-range variants are `0xfff911xx` and `0xfff913xx`. We assume **A-CHAIN**: at each listed size, the ledger assigned to a chain-range variant is the corresponding standalone ledger with only the profile identifier changed. The evidence for it is that they run the same round schedule and round table, hence the same census, as the standalone profiles of the same family and size, since the verifier derives the schedule from the family and N alone; no separate chain-range registry is exported. Under A-CHAIN the numerical census and error-budget calculations transfer unchanged. Applying Theorems 2 and 3 to the chain verifier additionally requires H2 for its transcript, parsing and acceptance behaviour; the standalone Lean transcript theorems do not establish that correspondence. The production chain verifier rejects every laboratory profile (Section 9).

N	Fam.	WHIR schedule $(r_0, \text{rates}, \text{folds})$	Queries per phase	Rounds k	$-\log_2 \varepsilon_{\text{sum}}$
128	J	(4, [7, 9, 11], [6, 5, 4, 4])	141, 80, 62, 51	2,785	270.53
128	U	(2, [5, 8, 10], [6, 5, 4, 4])	422, 300, 288, 287	2,781	284.41
2,048	J	(2, [5, 8, 10], [6, 5, 5, 5])	293, 112, 70, 56	3,154	270.36
2,048	U	(2, [5, 8, 10], [6, 5, 5, 5])	422, 300, 288, 287	3,150	284.24
4,096	J	(2, [4, 7, 10], [6, 5, 5, 5])	293, 141, 80, 56	3,279	270.24
4,096	U	(2, [5, 8, 10], [6, 5, 5, 5])	422, 300, 288, 287	3,275	284.07
16,384	J	(1, [2, 5, 8, 11], [6, 5, 5, 5, 5])	634, 293, 112, 70, 51	3,486	270.14
16,384	U	(2, [5, 8, 10, 11], [6, 5, 5, 5, 3])	422, 300, 288, 287, 287	3,481	283.20
65,536	J	(1, [2, 5, 8, 11], [6, 5, 5, 5, 5])	634, 293, 112, 70, 51	3,695	269.26
65,536	U	(1, [2, 5, 8, 11], [6, 5, 5, 5, 5])	690, 422, 300, 288, 287	3,690	282.69

Table 2: Registered schedules of the two families, as read from the verifier registries [X], with the ideal-IOP error of the ledger (sum over rounds, rounded down) [A]. The starting rate is 2^{-r_0}; the stacked polynomial has 23, 26, 27, 29 and 31 variables at the five sizes.

8 Implementation and measurements

8.1 Setup

The prover and verifier are one Rust crate on Plonky3 0.8.0 (`p3-whir`, `p3-sumcheck` and their dependencies, pinned exactly) and libcrux-ml-dsa 0.0.11 as the reference signer and verifier [X]. No elliptic-curve primitive, pairing or trusted setup is used. Statements use real signatures from distinct keys in pure mode with contexts of 1 to 255 bytes [X]; chain tests use a laboratory chain with a state registry and rotated keys. The hidden-u prover has two implementations of the Keccak GKR: a reference path, and a packed-trace path that stores the ± 1 traces packed and recomputes inner layers. They produce byte-identical certificates (Section 9).

8.2 Certificate size

Component	J, $N = 128$	J, $N = 2048$	U, $N = 2048$	QLYAUTH7, $N = 2048$
manifest	512	512	512	512
public $\tilde{c}$ / public u	8,192	131,072	131,072	2,097,152
hint counts ν	128	2,048	2,048	2,048
lift quotients	70	280	280	280
commitment root	64	64	64	64
LogUp GKR	18,400	23,920	23,920	23,920
leaf pair, table claim	120	120	120	120
master sumcheck	5,760	7,040	7,040	3,520
table sumcheck	1,520	1,520	1,520	1,520
masked output (openings, sumcheck, claim)	2,120	2,440	2,440	–
Keccak GKR, 72 layers	341,760	384,000	384,000	–
input link, bit claims, bit wiring	3,520	4,000	4,000	–
WHIR proof + 4 B length	499,057	871,177	2,168,217	553,809
total	**881,223**	**1,428,193**	**2,725,233**	**2,682,945**
bytes per signature	6,884.6	**697.4**	1,330.7	1,310.0
below the raw signatures	none (1.49 × raw)	84.9 %	71.2 %	71.7 %

Table 3: Measured certificate bytes [M]. Families J and U: standalone laboratory statements; QLYAUTH7: a laboratory chain range. Family J: one transcript per size, every component printed [M]. Family U: the measured total; the components outside the WHIR proof are fixed by the round table and equal family J's, and the WHIR proof is total minus fixed [D]. QLYAUTH7: the frozen public-u profile [M]. The raw signatures are 9,476,096 bytes at $N = 2048$ [X]. Sizes are deterministic for a given statement; the WHIR proof varies by a few KB between statements because Merkle sibling deduplication depends on the query indices.

At 2,048 signatures the family-J certificate takes 1,428,193 bytes, 697.36 bytes per signature, 84.93 % less than the raw signatures and 1.88 times smaller than QLYAUTH7 [D]. Its WHIR proof is 61 % of it, the Keccak GKR 27 % and the public $\tilde{c}$ 9 % [D]. Family U pays for its larger query counts in the WHIR proof alone, and at 2,725,233 bytes, 1,330.68 bytes per signature, it is about the size of the public-u certificate [D]. At $N = 128$ the fixed cost of the Keccak GKR dominates, and the certificate is larger than the raw signatures (592,256 bytes) [D]. A first build of the hidden-u certificate, whose transcript predates the two fixes of Section 6, measured 1,435,425 bytes at $N = 2048$; the difference lies in the WHIR proof, whose query indices change with the node hashing [M]. Certificates produced with the codewords on local disk instead of memory were byte-identical to the in-memory ones at $N = 128$ and $N = 2048$ [M].

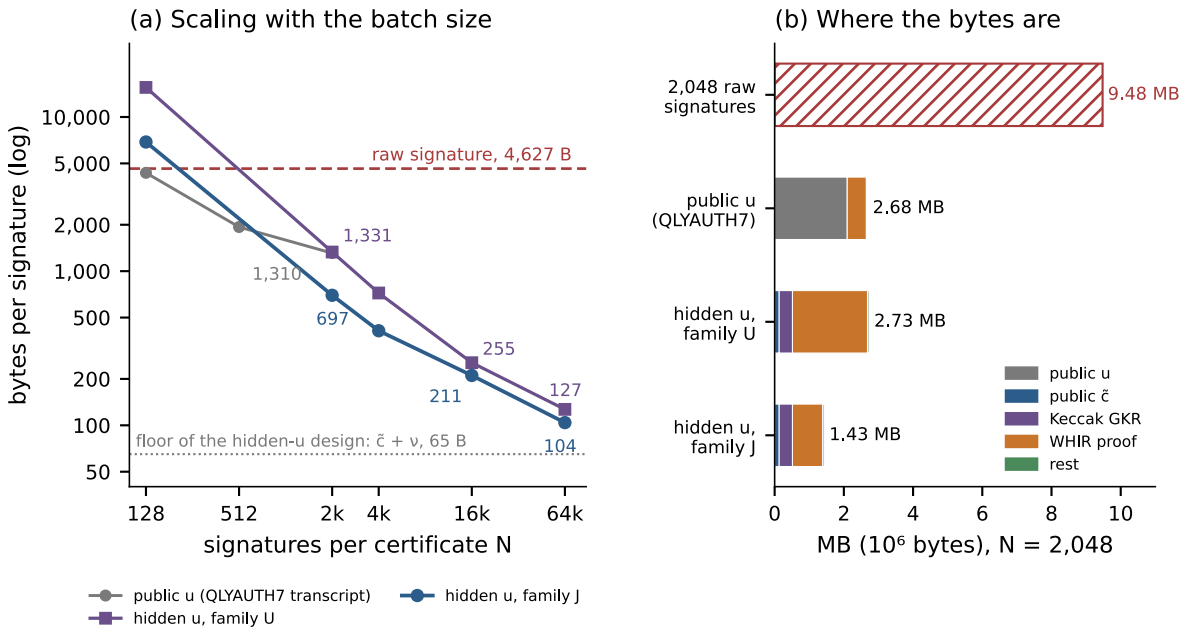


Figure 3: (a) Bytes per signature against the number of signatures N in one certificate, measured, on logarithmic axes, for the public-u certificate and the two hidden-u families: standalone statements up to $N = 2048$, chain ranges at 4,096, 16,384 and 65,536. The dashed line is one raw ML-DSA-87 signature (4,627 bytes); the dotted line is the 65-byte public floor of the hidden-u design ($\tilde{c}_i$ and ν_i). Family J has a classical bound only; family U also has the quantum bound. (b) Composition of each certificate at $N = 2048$, in MB, against the raw signatures.

8.3 Larger ranges

Table 4 lists chain-range certificates at 4,096, 16,384 and 65,536 signatures: 32, 128 and 512 full blocks of the laboratory chain, each verified in a fresh process. At 16,384 and 65,536 signatures, arrays of at least 64 MiB live in unlinked files on local disk; at 65,536 the first codeword, too large for memory, is encoded by a four-step DFT, whose certificates are byte-identical to those of the library's transform at $N = 128$, 2,048 and 4,096 [M]. Every hidden-u certificate of Table 3 and Table 4 has a recorded SHA-256 digest [X].

N	Fam.	Profile	Bytes	B/sig	Below raw	Prover, s	Verifier, s	Opt. prover, s
4,096	J	`0xfff9110c`	1,682,041	410.7	91.1 %	40.6–49.5	0.97–1.00	–
4,096	U	`0xfff9130c`	2,955,921	721.7	84.4 %	73.3[a]	0.96–0.98	–
16,384	J	`0xfff9110e`	3,449,918	210.6	95.4 %	322.2, 358.4	3.49–3.97	–
16,384	U	`0xfff9130e`	4,174,710	254.8	94.5 %	601.5[a]	3.69–3.92	300.9, 320.4[b]
65,536	J	`0xfff91110`	6,821,406	104.1	97.8 %	3,075.5 / 3,096.5	17.3–20.0	–
65,536	U	`0xfff91310`	8,315,630	126.9	97.3 %	3,095.3 / 3,115.5	16.6–17.5	677.4, 749.7[b]

Table 4: Chain-range certificates at large N [M] on the 24-vCPU server, current defaults (histogram rounds of Section 8.4 on) unless marked [a] (an earlier run with them off); the profiles are the chain-range variants of the schedules of Table 2. B/sig: bytes per signature; below raw: size reduction against the raw signatures. Bytes exclude the 64 bytes of header accumulator per block (0.5 bytes per signature). The raw signatures are 18,952,192, 75,808,768 and 303,235,072 bytes at $N = 4096$, 16,384 and 65,536 [D]. Prover times are `prove_stmt9` only, as in Table 6; at 65,536 the prover column also gives, after the slash, the time over all prover phases. Verifier times exclude regenerating the input chain. Ranges give all runs measured. [b] Opt. prover: the optimised prover of Section 8.5 (commit `2aced2d`), proving only, one run on each of two other hosts with the same quota; it wrote certificates byte-identical to those of the same rows, so bytes and bounds are unchanged [M]; four later reruns on three other hosts wrote the same 65,536 certificate again, in 782.5 to 1,967.9 s of `prove_stmt9` (Section 8.6) [M]. Hosts differ, so the two prover columns are not a speed-up factor; same-host comparisons are in Table 6. –: not run.

Size and speed at 65,536 signatures. On the reported shared machine with a 24-vCPU quota and a 24-GB memory limit, the Johnson profile produced a 6,821,406-byte certificate for 65,536 real ML-DSA-87 signatures [M], corresponding to 104.086 bytes per signature and a 97.750 % reduction relative to raw signature bytes, 44.45 times smaller per signature [D]. Proving took 3,075.5 seconds, preprocessing plus proving took 3,096.5 seconds, and fresh-process verification took 17.3 to 20.0 seconds [M]. Verification timing excludes regeneration of the input chain. Added block headers contribute 0.5 bytes per signature, giving 104.586 bytes per signature with them [D]; ordinary transaction and state data are outside this byte comparison. The Johnson family carries the classical theorem only. The quantum bound of Section 11.7 belongs to the unique-decoding family, whose 65,536-signature certificate is 8,315,630 bytes, 126.89 bytes per signature and 36.5 times smaller than a raw signature, proved in 3,095.3 seconds (3,115.5 with preprocessing) and verified in 16.6 to 17.5 seconds [M]; its prover moved 1,049.5 GiB read and 1,109.1 GiB written [M]. That certificate has been regenerated six times since, by the optimised prover of Section 8.5, on five hosts, always with the recorded SHA-256 digest `c9fbf852…`, and every fresh-process verification accepted it [M]. The difference, 1,494,224 bytes, is entirely in its WHIR proof (3,554,610 against 2,060,386 bytes, without the 4-byte length field) [M].

Size against throughput. Bytes per signature fall with N because the Keccak GKR and the WHIR proof grow far more slowly than the range, but proving time grows faster than the range once the arrays exceed memory. Full blocks bring 64 signatures per second [D], so a range of 65,536 arrives every 1,024 seconds. One 24-vCPU worker keeps up at 4,096 signatures (411 bytes per signature; three proofs in 40.6 to 49.5 s, 82.75 to 100.89 signatures per second); at 16,384 (211 bytes) the prover does 45.7 to 50.9 signatures per second and two workers are needed [D]. At 65,536 (104 bytes) the measured times of this first prover give 3,075.5 / 1,024 = 3.003 worker-equivalents for proving alone and 3,096.5 / 1,024 = 3.024 with preprocessing, so four workers, assuming that each has its own CPU, RAM and SSD bandwidth (at the measured I/

O volume, about 2.23 GB/s of aggregate read-plus-write traffic) [D]. That prover is disk-bound: it read 1,030.4 GiB and wrote 1,095.4 GiB (about 1.106 and 1.176 TB) and averaged 3.9 busy cores of its 24 [M]. Its 163.6 GiB is the peak of logical live allocations, of which 158.5 GiB was file-backed; the RAM requirement is the peak resident set, 21.88 GiB [M]. The verifier also grows, from about 1 s at 4,096 to about 17 to 20 s at 65,536, against about 4.6 s of native verification of 65,536 signatures [D].

The optimised prover of Section 8.5 changes the worker count without changing a byte of the certificate. For the unique-decoding range of 65,536 it took 677.4 and 749.7 s on two hosts, 96.8 and 87.4 signatures per second [M], that is 0.66 and 0.73 of the 1,024 s in which such a range arrives [D]; these ratios are not worker counts [D]. Rerun from a fresh build on 26 October 2026 [X], the same prover did not reproduce those times: its best run took 782.5 s, 83.8 signatures per second and 0.76 of a range interval, and the others 1,035.9 to 1,967.9 s (`prove_stmt9` in every case), a variation consistent with slower disks and busier CPUs on those hosts (Section 8.6) [M] [D]. The 677.4 and 749.7 s are therefore measurements on hosts B and C at the time, not a reproduced figure. At 16,384 it took 300.9 and 320.4 s, 51 to 54 signatures per second [M], more than the 256 s of such a range [D]. These are single-range runs, and they overstate what one worker sustains. Over consecutive ranges in the first five devnet campaigns (Section 8.8) the same prover took 847 to 1,888 s per range of 65,536 on busier hosts, and a worker also fetches the range, rebuilds the statement and waits for the node's verdict before it asks for the next range [M]. In the tested campaigns two 24-vCPU, 24-GB workers kept up at both 16,384 and 65,536 signatures, and single-worker keep-up was not demonstrated: at 65,536 one worker needed 1,135 s from one start to the next while a range closed every 1,101 s, and its wait grew [M] [D]. A later single-worker campaign at 65,536, on a faster placement, kept pace with eligible ranges without meeting its pre-registered margin rule (Section 8.8) [M]. Proof times varied 2.2 times between hosts, and the successful pair of Section 8.8 combined a faster and a slower host, so a deployment should size its workers by the slower placement's complete worker cycle; an all-slow pair at the ceiling was not measured [D]. Smaller certificates are therefore bought with machines, not with security: the schedules of all sizes have the same bounds (Table 11).

At 65,536 signatures the Johnson certificate is 4,194,304 bytes of public $\tilde{c}$, 2,060,390 bytes of WHIR proof, 436,800 bytes of Keccak GKR and 129,912 bytes of everything else [M]. Of its 104 bytes per signature, 65 are the floor of $\tilde{c}$ and ν and 31 are the WHIR proof [D]. At 4,096 signatures the packed Keccak path, with or without histogram rounds, and the reference path produce byte-identical certificates [M]. 49 verifier-side attacks on the 65,536-signature Johnson certificate, two changed bytes in every component, trailing and removed bytes, a relabelled profile, swapped records or blocks, another parent and a substituted key, all behave as expected [M]. These runs are chain ranges and the $N = 2048$ figures of Table 3 are standalone statements; the per-signature trend of Figure 3 mixes the two.

8.4 Time and memory

Prover phase, family J, $N = 2048$	**ms**	**peak live heap, MiB**
witness, sponge traces, multiplicities	227.0	469
raw instance, transcript initialisation	99.6	530
WHIR commit, 2^{26} cells	1,278.9	3,840
projections, integer lifts	206.3	4,037
LogUp leaves, fractional GKR	807.0	5,317
master weights, master and table sumchecks	2,638.1	5,565
Keccak trace, masked output	474.5	5,767
Keccak GKR, 72 layers	20,717.2	4,219
input link, bit wiring	2,227.9	6,024
WHIR opening, 29 values	1,361.5	5,149
total	**30,037.9**	**6,024**

Table 5: Prover phases on the 24-vCPU server, packed Keccak path with histogram rounds off, one run [M] (rows that merge two or three phases: times summed, heap the maximum [D]). The verifier took 544.4 ms in the same process, with a peak live heap of 533 MiB.

On the 24-vCPU server the whole family-J prover at $N = 2048$ took 30.0 s, 20.7 s of it in the Keccak GKR, with a peak live heap of 5.88 GiB and a maximum resident set of 6.04 GiB [M]. In the first three sumcheck rounds of every Keccak layer a folded value depends on only a few committed bits, so the packed path can compute those round polynomials from weighted histograms of the possible bit patterns instead of from field values. These *histogram rounds* change no message, give byte-identical certificates and are now on by default. On a second 2,048-signature statement, three proofs (one with the protocol's challenges, two with injected exceptional challenges) took 30.2 to 34.7 s with them off and 19.1 to 19.3 s with them on, against 97.4 to 125.3 s for the reference path on the same proofs [M]. With the codewords on local disk the prover took 39.3 s and wrote 32.4 GiB [M]. On the shared laptop the first hidden-u build took 163.9 to 301.8 s (median 164.9 s) with the reference path, 131.4 to 267.2 s of it in the Keccak GKR, with a peak footprint of 7.19 GB, and its fresh-process verifier took 615.9 to 649.8 ms [M]. For comparison the provers of the public-u profiles (QLYAUTH3, QLYAUTH4 and QLYAUTH7) took 7.0 to 11.9 s on the same laptop (QLYAUTH7: 10.1 to 10.4 s) and their verifiers 0.36 to 0.70 s [M].

Native libcrux verification of 128 signatures took 8.97 ms (median), so 2,048 natively verified signatures cost about 0.14 s [D]: checking a certificate is about three to five times slower than checking the raw signatures it replaces. It saves bytes, not verification time. Against the 32-second span of a 2,048-signature range, the default prover on 24 vCPUs (19.1 to 19.3 s) keeps pace, while the same prover with histogram rounds off (30.0 to 34.7 s) has little or no margin; at 16,384 signatures, 256 seconds of full blocks [D], the 322 to 358 s prover of Table 4 does not, nor does the optimised prover of Section 8.5 at 300.9 to 320.4 s [M]; in the sustained runs of Section 8.8 two such machines kept up at that size, and single-worker keep-up was not demonstrated. These are contended timings on shared hardware, not production latency figures.

8.5 A faster prover with the same certificates

After the measurements above we changed the prover, and only the prover. The verifier, the profile table, the transcript and the WHIR library are those of Table 4 [X]. Each of seven changes

computes the same field elements in a different order or from a different representation: (1) each folded round of the Keccak GKR folds the previous round's rows inside the task that reads them, instead of in a separate pass; (2) the LogUp GKR recomputes its leaves from the witness, and its largest layers from the leaves, instead of storing them; (3) multilinear evaluations go through a tensor product of two half-size equality tables; (4) the master and input-link sumchecks read their public columns as products of two factors or from generators, instead of from materialised arrays; (5) the columns T_g are read back from the committed states after the commitment; (6) the four-step DFT of the first codeword runs in place, without a second array of codeword size; and (7) the prover gives the kernel page-cache hints around the strided passes of that DFT. Changes that would alter a message were left out, among them a packed base-field sumcheck for the Keccak layers and any change of schedule, rate, folding, query count or of the WHIR library [X]. Field arithmetic is exact, so equal values are equal bytes; we nevertheless checked the bytes.

Same certificates. We made 67 SHA-256 comparisons of certificates from the optimised prover against those of the first prover, and all were identical: 56 at $N = 128$ and 2,048 in both families, with the default build, with forced memory budgets and with a forced four-step DFT, and 11 at 4,096, 16,384 and 65,536, one per timed run [M]. At 16,384 and 65,536 the digests are the ones recorded for the unique-decoding certificates of Table 4, so the optimised prover reproduced the measured 4,174,710- and 8,315,630-byte certificates byte for byte [M]. On the final commit the acceptance suite of Table 9 passed 223 of 223 in each family [M]; the 49 verifier-side attacks of Section 8.3, run on the new unique-decoding certificate at 65,536, behaved as expected, 49 of 49 on each of two hosts [M]; and a script that compares every verifier function between the two commits found none changed [M]. These comparisons are between the optimised build `c731c31b…` and the first prover's build `4a1ff4f0…`; the SHA-256 digests are artifact checksums and play no part in the certificate protocol, which uses SHAKE256 [X]. They are regression evidence on the tested corpus, not a proof that the two programs agree on every input, and the security argument does not need one. The knowledge bounds of Section 11, classical and quantum, hold for every prover algorithm against a fixed verifier, relation, profile table and transcript interpretation. Those are unchanged, so replacing the honest prover adds no term to either bound; a defect in the new prover could break completeness, availability or reproducibility, but not soundness against the unchanged verifier proven (conditional on that verifier).

Family U, chain range	Host	First prover	Optimised prover	Same host
$N = 16{,}384$, proving, s	B	876.8	300.9	2.91 times faster
$N = 16{,}384$, proving, s	C	1,213.1	320.4	3.79 times faster
$N = 65{,}536$, proving, s	A	3,357.2	–	no pair
$N = 65{,}536$, proving, s	B	stopped[c]	677.4	no pair
$N = 65{,}536$, proving, s	C	–	749.7	no pair
$N = 16{,}384$, peak live arrays, GiB	B, C	51.2	35.9	
$N = 65{,}536$, peak live arrays, GiB	A; B, C	163.6	83.1	
$N = 65{,}536$, disk read / written, GiB	A; B	1,053 / 1,124	197 / 306	different hosts

Table 6: The first and the optimised prover on shared hosts with a 24-vCPU, 24-GB quota: host A an Intel Xeon Platinum 8581C, hosts B and C two AMD EPYC 9655P machines [X]. Proving times (`prove_stmt9` only), one run each [M]; ratios only between runs on the same host [D]. Peak live arrays: the logical live allocations, heap and disk-backed mappings together, not physical RAM; equal on every host that ran the same prover [M]. Disk traffic: first prover on host A, optimised prover on host B (195 / 318 GiB on host C) [M]. [c] Stopped twice while still inside its commitment, after 50 and 11 minutes [M]. –: not run.

Table 6 gives the measurements. Same-host pairs exist only at 16,384, where the optimised prover was 2.91 and 3.79 times faster on two hosts [D]. At 65,536 there is no same-host pair: on host B the first prover was stopped twice before its commitment had finished, and the first prover's 3,357.2 s come from host A [M]. The interrupted run on host B gives only a censored same-host lower bound [D]. We do not turn the cross-host figures into a speed-up factor, because the hosts differ widely: the same first prover took 621.5 s at 16,384 on host A and 1,213.1 s on host C [M]. What does transfer across hosts is the memory: the peak of logical live allocations at 65,536 fell from 163.6 to 83.1 GiB [M]. These include disk-backed mappings and are not physical RAM [M]: in the sustained runs of Section 8.8 the optimised prover's resident set peaked at 20.2 to 20.4 GiB per proof, within the 24-GB limit [M]. The optimised prover is CPU-bound except in the WHIR commitment and opening: at 65,536 on host B the whole process used 9,258 CPU-seconds, 13.2 busy cores on average, where the first prover averaged 5.0 on host A [M]. The new 65,536 certificate was verified in a fresh process by the binary of the first prover in 8.8 and 9.0 s on hosts B and C [M], against 16.6 to 17.5 s in Table 4: the verifier is the same, so this too is a difference between hosts.

8.6 Reproduction at 65,536 signatures

We rebuilt the optimised prover from the committed sources and ran the unique-decoding range of 65,536 again, four times, in fresh containers with the same 24-vCPU, 24-GB quota. The sources were an export of the prover-speed branch at `bc4788b`, whose prover code is that of Table 6 (`1e5432b`), compiled at container start with rustc 1.96.0, `--release --locked` [X]. The provider chose the hosts: host D, an Intel Xeon 6975P-C, and hosts E and F, two AMD EPYC 9655P machines, the CPU model of hosts B and C; host E ran twice [X]. Each run used the recorded command with disk-backed arrays and then verified the certificate in a new process.

The certificate reproduces exactly. All four runs wrote 8,315,630 bytes, 126.89 bytes per signature, with the recorded SHA-256 digest `c9fbf852…` [M]. Every fresh-process verification accepted it: on host D in 17.4 s with the optimised build and in 18.0 s with the build of the first prover (`0e7a1c9`), on host E in 16.0 s after each run, and on host F in 11.3 s [M]. These timers exclude rebuilding the input chain, as in Table 4; the whole verifier process, chain rebuild included, took 20.1 to 25.6 s on hosts E and F [M], and 27.2 s on host D, including a 9.0 s chain

rebuild outside its 17.42 s timed region [M]. These are fresh processes; we do not claim a cold page cache. On host D the 49 verifier-side attacks of Section 8.3 behaved as expected on the rerun certificate, 49 of 49, and the unique-decoding acceptance suite at $N = 128$ passed 223 of 223 [M]. Hosts E and F built the binary digest `c731c31b…` recorded on hosts B and C; host D built `ade44b0b…` from the same sources, so the binary depends on the build host while the certificate does not [X]. Peak live array storage, the logical live allocations including disk-backed mappings and not resident RAM, was 83.14 GiB in every run, as recorded; the maximum resident set on host D was 21.8 GiB, within the 24-GB limit [M].

Family U, $N = 65{,}536$	**B**	**C**	**D**	**E, run 1**	**E, run 2**	**F**
when	recorded	recorded	rerun	rerun	rerun	rerun
`prove_stmt9` only, s	677.4	749.7	1,967.9	1,035.9	1,048.4	**782.5**
all prover phases, s	690.7	763.8	1,990.4	1,057.2	1,068.7	810.2
prover process wall, s	700.2	774.4	2,002.7	1,069.5	1,080.9	821.8
P1 WHIR commitment, s	276.6	333.3	1,444.6	530.2	531.0	298.5
P1 busy cores	5.2	4.0	0.8	2.4	2.5	5.2
P1 disk read / write, MiB/s	426 / 685	–	–	210 / 364	218 / 372	417 / 643
P5c Keccak GKR, s	158.4	163.8	213.0	204.9	207.7	199.2
certificate	`c9fbf852…`	`c9fbf852…`	identical	identical	identical	identical

Table 7: The optimised prover at 65,536 signatures on five shared hosts with the same quota: the recorded runs of Table 6 (hosts B and C) and the four reruns (host D an Intel Xeon 6975P-C; hosts E and F two AMD EPYC 9655P) [M]. Phase times and busy cores [M]; P1 rates are the process's disk traffic over the commitment window of its 5-second samples [D]. Three timer boundaries, never mixed in one row: `prove_stmt9` only, as in Table 6; all prover phases, which add the replay and witness phase P−1 (22.1 s on host D) and P0 [M]; and the whole prover process, from the operating system's elapsed time, which also covers building the input chain (8.6 to 9.3 s on hosts D to F) [M] [D]. –: not computed.

The proving time does not reproduce; it varies with the host. All times in this paragraph are `prove_stmt9` times unless stated. The rerun on host F, 782.5 s (810.2 s over all prover phases), is 4 % above host C and 16 % above host B [M] [D]; the other three took 1,035.9 to 1,967.9 s [M]. Against host B, 21.9 s of host F's 105.1 s gap is in the commitment P1 and 83.2 s outside it; against host C, host F's P1 was 34.8 s faster and the rest 67.6 s slower [D]. The phase telemetry places the variation in the disk-backed WHIR commitment and in the CPU-bound phases, consistent with differences in available I/O throughput and CPU resources on the shared hosts (Table 7); it does not isolate either as the sole cause. First, P1 tracked the disk. Host E sustained about half of host B's read and write throughput in P1, and P1 took 530 s against 277 s; its two runs agree to 0.2 % in P1, so its lower P1 throughput was a property of that host at that time, not a one-off disturbance [D]. Host F sustained about host B's rate, and its P1, 298.5 s, lies between those of hosts B and C [M]. On host D, whose load average was 13 to 38 during the run, P1 took 1,444.6 s with 0.8 busy cores: it waited on the disk-backed arrays [M]. Second, the CPU-bound phases ran 20 to 30 % slower than recorded on every new host: the Keccak GKR took 199 to 213 s against 158 to 164 s, on hosts more loaded than host B, and the containers reported no steal time, so the loss is not visible from inside them [D]. Without P1, host F took 484.0 s against 400.8 s on host B and 416.4 s on host C [D], and host D 523.3 s [D], all four from `prove_stmt9`. The recorded 677.4 s was measured on a host whose disk sustained about 0.4 GiB/s read and 0.7 GiB/s write during the commitment and whose CPUs were less shared than those we were given on 26 October 2026 [D]; a host with that disk

rate is not thereby guaranteed the same time. What these reruns reproduce is the certificate, its size, its verification and the memory. They do not reproduce the recorded prover time: the best of the three new hosts gave 782.5 s, 4 % and 16 % above the recorded figures, and host D 2.9 times the time of host B [D]. None of this touches a bound: the certificates are the same bytes, and the security argument of Section 11 does not depend on the prover.

8.7 In a node

We also ran the unique-decoding certificate inside the nodes of a three-node proof-of-work devnet, in rented containers. Each node computes $A[b]$ for every block it validates, cuts the chain into ranges and hands eligible ranges to prover machines over its peer-to-peer transport. The provers of the first runs below ran the first prover of Table 4; the sustained runs of Section 8.8 also use the optimised one of Section 8.5. A node verifies a returned certificate with a production build of the verifier, without test hooks, whose verifier source is the revision reviewed in Section 11 extended with the devnet's profile identifiers, against a statement built from its own blocks, then stores and gossips it; two nodes keep the archive, and the third compacts certified ranges and prunes them from its execution client [X]. As Section 11.12 requires, certificates never enter block validity or fork choice, and certificate-only bootstrap does not exist [X]. In this version every node computes the accumulator, but block headers do not commit to it, so a party that holds only headers cannot yet check a certificate [X].

A functional and adversarial suite with real proofs at $N = 128$ passed 45 of 45 checks: the honest flow on three nodes, compaction and pruning, 21 adversarial cases from a node that joins later (forged, tampered, wrong-range, other-network, Johnson-family and laboratory-profile certificates, ranges not yet final, malformed, oversized, flooding and lying peers) and a reorganization below certified and compacted ranges [M]. After every fix the suite passed 45 of 45 again, with a 105-block reorganization below the compacting node's pruning point [M].

At 16,384 signatures, 9 ranges (147,064 transactions) were certified on all three nodes [M]. The certificates took 4,168,566 to 4,177,142 bytes, 254.4 to 254.95 bytes per signature slot (a range is padded to its full size) [M]. Proofs took 442 s on one 24-vCPU prover and 844 to 980 s on two others with the same quota, which read about 25 times more from disk; the cause was not isolated [M]. Verification took 8.6 to 17.9 s on one node and 25 to 33 s on the two others (one outlier at 161 s), because the three nodes, the miner and the load generator shared one 8-vCPU container [M]; Table 4 gives 3.5 to 4.0 s on an idle machine. At the end of the run the compacting node held 0.91 GB of block bodies against 2.22 GB on an archive node, and 38 MB of execution-client transaction files against 1.11 GB [M].

At 65,536 signatures, 2 ranges (130,945 transactions) were certified on all three nodes, each by an 8,326,894-byte certificate, 127.06 bytes per signature slot, against 126.89 for the laboratory statement of Table 4 [M]. The two proofs took 2,824.5 s and 5,804.0 s; the slower one read 7,815.6 GiB from disk against 1,074.6 GiB [M]. All six verifications accepted, in 105.6 to 113.8 s per node and certificate, in one 24-vCPU container shared by the three nodes, against about 17 s in the laboratory [M]. The compacting node held 1.26 GB of block bodies against 2.44 GB, the open, uncompacted range included [M]. These runs show the certificate working in a node, on contended devnet hardware.

8.8 Keeping up with full blocks

The runs above certify a few ranges each. To measure whether provers keep up, we ran five devnets under full blocks for 75 minutes to 3 hours each; each run is a load window followed by drain, under light load, until every closed range was certified [X]. The load generator offered 80 transactions per second, more than the 64 per second that full 2-second blocks carry, and 99 % of blocks were full [M]. The chain carried 55.5 to 59.6 transactions per second, not 64, because the laboratory miner kept blocks at 2.13 to 2.31 s on average instead of 2 s [M]. Each devnet ran its three node processes, the miner and the load generator in one 24-vCPU, 24-GB container, where a machine-wide lock serialises verification; each prover had its own container of the same size, on a distinct host [X]. The nodes kept the production verifier `e532f0e3…` of the runs above; the provers ran either that binary or `e99d92c7…`, a build of the same crate with only the prover changes of Section 8.5 [X]. That port was not run through the paired comparisons of Section 8.5; what these runs add is that the unchanged production verifier accepted every certificate it produced [X]. The *lag* of a range runs from the block that closes it to the moment its certificate has been verified on all three nodes; the *wait* is the time an eligible range waits for a free prover. A worker's *work interval* runs from taking a range to the finished certificate: it covers fetching the range's bodies, rebuilding the statement and the proof, so it is longer than the proof timer of Table 8. Table 8 gives the five runs.

Run	N	Provers	tx/s	Ranges	Close every, s	Proof, s	Max wait, s	Lag, s	Trend, s	Kept up
1	65,536	first × 5	59.6	5[a]	1,142	5,572–6,702	–[a]	6,013–7,190	–	no
2	65,536	opt. × 1	58.5	9	1,101	893–1,081	292	1,463–1,729	+26.5	not shown[b]
3	65,536	opt. × 2	58.9	9	1,101	847–1,888	6	1,482–2,527	−9.9	yes
4	16,384	opt. × 1	57.3	15	282	292–326	649	508–1,119	+48.3	no
5	16,384	opt. × 2	55.5	15	295	285.5–365	5	464–590	−2.7	yes

Metric	Meaning
Provers	the first or the optimised (opt.) prover, and the number of workers
tx/s	transactions per second carried over the full-load window
Close every	mean interval between range closes
Proof	the prover's internal timer per range, shorter than the work interval
Max wait	the longest an eligible range waited for a free prover
Lag	from the block closing a range to its certificate verified on all three nodes
Trend	least-squares slope of the lag, in seconds per range; a description of the sample
Kept up	judged from the wait and the lag over the load window, not a stability guarantee

[a] Four ranges certified on all three nodes and a fifth on two when the run was stopped; ranges 5 to 9 found no free prover, and the count of eligible, uncertified ranges, in-flight ones included, grew from 1 to 6 [M].
[b] A start-to-start cycle of 1,135 s against a range every 1,101 s, with a growing wait: keeping up is not demonstrated [D].

Table 8: Sustained proving in three-node devnets under full blocks, unique-decoding family; every prover on its own 24-vCPU, 24-GB container [M]. Trends and keep-up verdicts are derived from the samples [D].

Two optimised provers kept up at both sizes over the tested windows. At 65,536 signatures they certified 9 consecutive ranges generated during a three-hour load window at 58.9 transactions per second, with a lag of 1,482 to 2,527 s, about 25 to 42 minutes, and no range waited more than 6 s for a prover [M]. At 16,384, 15 consecutive ranges generated during a 75-minute window at 55.5 transactions per second had a lag of 464 to 590 s, and no range waited

more than 5 s [M]. The short waits are the main evidence. The lag slopes, −9.9 and −2.7 s per range, describe the samples, and at 65,536 the pooled slope mixes the latencies of two hosts [D]. Both campaigns then drained, and their last certificates completed after the load window. This shows keep-up over the tested windows and host pairs; it is not a queue-stability guarantee [D].

Single-worker keep-up was not demonstrated in runs 2 and 4. At 16,384 one optimised prover fell behind: its proofs took 292 to 326 s while a range closed every 282 s, and the lag grew from 508 to 1,119 s over 15 ranges [M], a least-squares slope of 48.3 s per range [D]; the slope is an empirical regression, not the difference between the two means. At 65,536 one prover kept the lag between 1,463 and 1,729 s over 9 ranges, but the lag rose by 26.5 s per range and the wait grew from 3 to 292 s [M D]. Its work interval averaged 1,023 s, of which the proof timer of Table 8 shows 989 s; about 107 s more passed, mostly waiting for the node's verdict, before it took the next range [D]. From one start to the next it thus needed 1,135 s per range, while a range closed every 1,101 s: a ratio of 1.03, consistent with the growing wait [D M]. This compares service demand with arrivals; it is not a utilization figure. We do not count it as keeping up. With the first prover, five workers did not keep up at 65,536: every proof took 5,572 to 6,702 s and read about 7.8 TiB from disk, a range closed every 1,142 s, and within 100 minutes the count of eligible, uncertified ranges, in-flight ones included, grew from 1 to 6 [M]; the run was stopped before steady state.

A later single-worker campaign. After these five runs we ran one more campaign at 65,536 signatures with one optimised worker, the configuration and binaries of run 2, and one change: the worker's scheduling, so that it could take the next eligible range as soon as it had announced a certificate instead of waiting for the node's verdict [X]. Nine ranges closed during about three hours of full load at 58.92 transactions per second [M]. All nine certificates were accepted on all three nodes after the load window and drain, 27 acceptances with no rejection and no verifier error, and every range waited 1 to 6 s for the worker [M]. The worker ran on an AMD EPYC 9655P host where its work interval averaged 836.9 s, against 1,023 s in run 2: a faster placement [X D].

Registered outcome. The campaign did not satisfy its pre-registered rule, fixed before the run: every range verified on all three nodes, a start-to-start cycle shorter than the close interval, and neither the lag nor the wait growing by more than 5 s per range [X]. The cycle was 1,106.8 s against a range every 1,105.2 s, a ratio of 1.001, and the lag rose by 18.7 s per range while the wait trend was 0.0 s per range [D]. We keep the registered outcome, "not demonstrated" [M].

Post hoc interpretation. Analysis after the campaign, which we label as post hoc, found the cycle criterion unsuitable for measuring headroom in a worker that is caught up: such a worker starts each range shortly after it becomes eligible, so its start-to-start cycle follows the close interval whatever share of that interval it spends working, and the ratio stays near 1 [D]. Here the work interval was 0.76 of the close interval [D], and the lag rose mostly with the proof and verification times, whose trends were 9.8 and 7.2 s per range, not with the wait [D]. Described without the registered rule, one worker kept pace with eligible arrivals without an accumulating prover queue over this campaign, on this placement [M D].

Scheduling observation. The new scheduling path never ran: node A's verdict on each range arrived 36 to 280 s before the next range became eligible, so no range started before that verdict [D M]. The campaign therefore does not measure the benefit of removing the verdict wait.

Single-worker capacity with a specified margin, and across host placements, needs a longer pre-registered campaign.

At the 64 tx/s ceiling [D]. No run reached 64 transactions per second; in the last hour of the 65,536 runs a range closed every 1,017 to 1,100 s, within 7 % of the ceiling [M]. At 64 signatures per second a range of 65,536 closes every 1,024 s. The worker of run 2, at about 1,135 s per range, did not match that [D]. A worker with the speed of run 2 that started the next range while the node verifies would still need its work interval, about 1,023 s against 1,024 s: no demonstrated margin [D]. The later campaign deployed such scheduling but never used it, so this remains unmeasured. Its work interval of 836.9 s is shorter than 1,024 s, but it ran at 58.92 transactions per second, so no margin at the ceiling is claimed for it [D]. A range of 16,384 closes every 256 s, less than the 310 s mean proof of one worker [D]. For two workers we derive no margin at the ceiling: they ran on different hosts, and their observed waits, at most 6 and 5 s, are the evidence [D]. For illustration only, two workers each as slow as the Icelake worker of run 3, whose work interval averaged 1,964 s, with a 108 s verdict wait, would carry $2 \cdot 65{,}536 / 2{,}072 = 63.26$ signature slots per second, short of 64 [D]. That is an extrapolation, not a measured failure, and it is why the successful mixed pair does not show that any two hosts suffice. The first prover's mean proof time, 5,865 s, is 5.7 worker-equivalents at the ceiling, so six workers both at the measured rate and at 64 transactions per second [D].

Hosts. With the same binary, proof time varied 2.2 times across the rented hosts: in run 3 the worker on an Intel Icelake host took about 1,800 to 1,890 s per proof, the one on an AMD EPYC 9655P 847 to 938 s [M], and the lag of each range followed the host that proved it. Every proof of the optimised prover in runs 2 and 3, 847 to 1,888 s at 65,536, was slower than the 677 to 750 s of Table 6 on less loaded hosts [M]. The successful pair combined a faster and a slower host [X]. We therefore state a provisioning rule rather than a worker count: size workers by the slower placement's complete worker cycle, including preprocessing, proof work, verdict waiting and scheduling [D]. Two workers sufficed for the tested host pairs and achieved rates; an all-slow pair at the nominal ceiling remains unmeasured [X].

Verification and storage. The production verifier accepted every certificate it finished verifying. In runs 1 to 5, 52 certificates were accepted on all three nodes, and one certificate of run 1 on two, with the third node still verifying when the run stopped, so all 158 completed node verifications were acceptances, with no rejection and no verifier error [M]. The optimised runs 2 to 5 account for 48 certificates and 144 acceptances, and the two-worker runs 3 and 5 for 24 certificates [D]. Verification took 84 to 111 s at 65,536 and 4 to 12 s at 16,384, off the block path [M]. Runs 2 to 5 each ended with every range certified on the first archive node also certified on the two others with the same certificate, and with no block marked invalid [M], and so did the later single-worker campaign [M]. At the end of run 2 the compacting node held 3.95 GB of block bodies against 9.25 GB on an archive node, and 0.83 GB of execution-client transaction files against 4.63 GB [M]. These are sustained figures on contended devnet hardware at 55 to 60 transactions per second, with the three verifying nodes in one container, not latency figures for a production network; a geographically distributed or independently provisioned set of verifiers was not measured.

9 Adversarial evaluation

Mutation tests of a proof system only show something if a mutated witness is *otherwise valid*. We therefore built a **relaxed signer**: it draws s_1, s_2 uniformly from $[-2, 2]$, publishes a key in the standard format, and produces witnesses that satisfy the exact modular relation with a challenge from a pluggable hash, breaking exactly one condition at a time. A relaxed signature that respects every condition is accepted; so a rejection comes from the broken condition, not from the construction.

Group	What it does (hidden-u, $N = 128$, run once per family)	Result
A. controls	honest acceptance; identical prover and verifier round logs; registered calls only, no grinding; canonical extraction 128/128 accepted by libcrux; profile isolation; the unsafe no-Booleanity control is complete	8/8
B. hash side	wrong SHAKE suffix; omitted padding permutation; wrong nibble or bit order; swapped states, slots or signatures; capacity-only mutations; non-Boolean bits; non-zero padding or dummy cells; a different u through an alternative decomposition that keeps every arithmetic bound; the false product-of-evaluations shortcut	24/24
C. ported	response bound, region, hint budget, statement swaps, wrong u nibble, quotient, multiplicity, selector and last-cell mutations; ternarity in isolation ($S = 1/2$); layout; exceptional Gruen challenges; tampered bytes in every component; trailing bytes; challenge-prefix sensitivity of every statement field	113/113
D. range binding	reordered records, substituted or rotated-out key, tampered accumulator, cross-fork, shifted range, prover-side reordering, one $\tilde{c}$ bit; the production verifier rejects the profile	12/12
E. profile rules	strict projection and extension-element pools (projection shortfall makes the prover refuse; either strict-pool shortfall poisons the transcript and prevents verifier acceptance; never padded; injected pools contain exactly enough or one fewer accepted word), slack rule, per-tree node domains	22/22
F. Fiat–Shamir on GKR	coins without the instance frame; coins from a digest of the instance; statement changed after proving; records swapped after the round-1 coins; descriptor or family confusion	16/16
G. hostile gates	coordinated non-Boolean planes that cancel under a fixed sum; multiplicities moved after θ; each masked-output opening changed after the batching challenge; sign conversion on every cell; χ truth table; zero-vector terms; per-tree hash counts	16/16
H. tree contexts	verification with every routing-cell write replaced by another tree; interleaved verifiers of two profiles on two threads; a root committed under one tree checked under another; out-of-range tree ids; forced-short pools	12/12

Table 9: Hidden-u acceptance suite on the final code: 223 of 223 tests behave as expected in each family [M]. Group counts are counted from the transcript [D]. An earlier version of the suite, without groups G and H, passed 190 of 190.

The public-u profile has its own suites, all passing on the final code: 60 soundness tests, 43 transcript tests, 11 chain tests, 40 transcript gates, 74 attacks written from the specification alone with a 462-position byte-flip sweep, 24 production-profile tests at $N = 2048$ and 61 Merkle-routing tests [M]. Unchanged profiles stayed byte-identical: the certificates of QLYAUTH3 and QLYAUTH4, earlier public-u laboratory profiles, of QLYAUTH7, and of the Keccak kernel reproduce the same SHA3-256 digests before and after the hidden-u work [M].

Four results deserve a sentence each. **Ternarity in isolation:** setting $S = 1/2$ in four interior hint-free cells, with Y and the multiplicities adjusted so that every other constraint holds, is

rejected by the full verifier, while a control that drops only the ternarity weight accepts [M]: the zerocheck is necessary and enforced. **Booleanity cannot be isolated:** a non-Boolean bit that keeps the arithmetic intact is still caught by the hash link, because the Keccak input must be ± 1; the extraction argument relies on the Booleanity zerocheck, not on this test. **Packed and reference provers agree:** complete certificates from the packed and the reference Keccak paths were byte-identical in every comparison, exceptional challenges included, at $N = 32$, 128 and 2,048, with histogram rounds off and on, and at 4,096 [M]. **A real bug in the earlier profile:** the serialisation library accepted trailing bytes inside the WHIR proof when the outer length was raised to match; the verifier now re-encodes the proof and requires byte equality [M].

10 Negative results

Per-block certificates under 128 KiB are out of reach in this family. For one block of 128 signatures (2^{20} cells), the smallest WHIR proof we measured was 258.5 KiB at a composed bound of 268.5 bits; a sweep of every schedule the library accepts in the Johnson regime without grinding found nothing smaller than 257.7 KiB [M]. Only the capacity regime went lower, to about 140.6 KiB [D], and that regime rests on conjectures, in forms since disproved (Section 13). This is what moved the design from blocks to ranges; the hidden-u certificate is still larger than the raw signatures at $N = 128$.

At first, hiding u was too slow. A standalone Keccak GKR kernel proved 1,024 permutations in 4.45 to 4.85 s on the laptop, against a criterion of 1 s [M]. Inside the first hidden-u build the reference path took 7.3 s per 1,024 real permutations on the laptop [D]. The packed path on 24 vCPUs brought the Keccak GKR to 20.7 s for the 18,432 real permutations of $N = 2048$, about 890 permutations per second [D]. A binary-field prover for batched Keccak reports about 239,000 permutations per second on a 10-core laptop [19], about 270 times more, on different hardware [D]. This comparison shows a large throughput gap, but differences in field arithmetic, protocol, implementation and hardware prevent us from attributing a measured fraction of it to the field choice alone (Section 13).

Generic proving was not the answer either. Before the specialised frontend, a generic STARK over the same verification took about 20 s and 2.1 MB for 128 signatures, 1.3 MB of it for Keccak [M].

11 Security

11.1 Relation, scope and status labels

Let x be the canonical raw instance of a hidden-u certificate: the profile and manifest, the ordered records with their state-resolved keys (and the range anchors in the chain variant), the public digests $\tilde{c}_i$ and the hint counts ν_i. $\mathcal{R}_{\text{auth}}(x, \boldsymbol{\sigma})$ holds when, for every i, σ_i is a canonically encoded FIPS 204 ML-DSA-87 signature $(\tilde{c}'_i, z_i, h_i)$ with $\tilde{c}'_i = \tilde{c}_i$, ML-DSA.Verify in pure mode with context ctx_i accepts it on (pk_i, M_i), and h_i has ν_i hints. Replay, nonces and key resolution happen outside the certificate. The certificate proves knowledge of signatures satisfying $\mathcal{R}_{\text{auth}}$; it does not prove that they are byte-identical to the signatures originally broadcast.

The registered relation, verifier and extractor use the same specified sampler interpretation. Direct instantiation with a faithful total extension of the unbounded FIPS 204 samplers (Section 12) describes that extension's own protocol. For the formal bounded fallback protocol,

agreement of its authorization relation with faithful unbounded FIPS verification is proved under the per-seed within-limit conditions of Section 12. The transfer retains the fallback verifier and changes the relation on that domain; it does not establish equality of the two verifiers.

The transcript and compiler claims below are restricted to canonical instances with $|M_i| < 2^{32}$ bytes for every record. This is the proved framing domain of Section 6; the implementation does not yet enforce that limit.

Every statement below carries one label. proven: proved under the hypotheses stated with it. This section states the results and the idea of each proof; Appendix C writes the proofs out step by step, our internal review has checked them, and Section 12 lists the parts that are also machine-checked. cited: relies on a published theorem; we checked its statement number in the version we name and its hypotheses against our protocol, but did not re-check its proof. tested: established by tests or by a script, not by proof. open: not established.

Some letters of Section 2.1 are reused in this section with another meaning: k is the number of registered rounds, t a number of oracle queries, $\ell_{\max}$ the number of leaves of the largest Merkle tree, $h = \log_2 \ell_{\max} + 1$ the Merkle term of Theorem 3, and ρ a code rate. R and Q are the adversary's classical and quantum oracle queries, and E the number of verification submissions it makes.

11.2 Hypotheses

Id	Hypothesis	Status
H1	Random-oracle model with split hashing: the compiler's Fiat–Shamir and Merkle oracles are independent random functions, implemented by SHAKE256 on disjoint, injectively framed domains. The relation and the arithmetized Keccak-f use the concrete SHAKE256 and Keccak-f. H1Q: the same, with oracles that the adversary may query in superposition (quantum random-oracle model; Theorem 3).	model
H2	The prover and verifier implement the interactive protocol with the registered round table, the full-history transcript and canonical parsing.	tested; source reviewed internally
H3	Lemma 1 and the ternary cover.	proven (Lean); also tested
H4	Mutual correlated agreement up to agreement $(1 + 1/(2m'))\sqrt{\rho}$, $m' = 10$ (family J): DKT26 Theorem 5.12 [20]; fallback BCGM Definition 9.1 and Theorem 9.2 [21], at the same tradeoff m' and generator degree 1.	cited
H5	Mutual correlated agreement in unique decoding, $\delta < (1 - \rho)/2$ (family U): WHIR Corollary 4.11, from its Lemma 4.10 and its Theorem 4.8, which is the result of [22].	cited
H6	Johnson list bound: at agreement $(21/20)\sqrt{\rho}$ the list has at most $L = 10/\rho$ members: WHIR Theorem 4.3.	cited
H7	WHIR round-by-round soundness with the per-round errors of WHIR Theorem 5.2, with H4 or H5; the sharper fold error that the ledger charges is proved in Appendix C.	cited
H8	Family U transcript: per-tree node domains, strict projection pool, no out-of-domain sample, target 286, no grinding.	tested
H9	The extractor completes erased or malformed leaves with a fixed zero symbol and rejects when verification queries one.	proven

Table 10: Hypotheses of the hidden-u analysis [X]. H4 and H6 apply to family J, H5 and H8 to family U; "H3 to H7" means H3, H7, and H4 with H6 (family J) or H5 (family U). H1 is the standard idealisation and is not a statement about SHAKE256. WHIR theorem numbers refer to ePrint 2024/1586, version of 21 November 2024.

11.3 Common-list extraction

Let Π_1 be the first oracle, the stacked codeword: rows of $2^{f_0} = 64$ base-field elements, one Merkle leaf per row. Per row the library stores the evaluations, at a common domain point, of the 64 lane polynomials of the stacked message, which we read from its pinned source.

Lemma 3 (one common list). proven, given the cited decoder theorem and, for family J, H6. Read row j of Π_1 as one symbol of $J = \mathbb{F}_p[\zeta]/(\varphi)$, for a fixed irreducible $\varphi \in \mathbb{F}_p[\zeta]$ of degree 64, through $y_j = \sum_a \zeta^a \Pi_1[j][a]$. Honest rows are then exactly the codewords of one Reed–Solomon code over J with evaluation points in $\mathbb{F}_p$, and:

(1) List decoding that code (at agreement $(21/20)\sqrt{\rho}$ ✗ in family J with the deterministic decoder of Chatterjee, Harsha and Kumar [23] (Theorem 1.1), which works above the Johnson agreement over every finite field, and by Berlekamp–Welch [24] below $(1-\rho)/2$ in family U) returns in deterministic polynomial time every candidate message within the radius, at most $L = 10/\rho$ of them in family J and at most one in family U, each a complete witness of all 15 columns and the multiplicities.

(2) Every vector of 64 lane polynomials over K, each of degree below d, that meets the whole-row agreement radius of (1) has all its lane coefficients in $\mathbb{F}_p$ and hence occurs in that list.

(3) The list depends on Π_1 only, so it is fixed before every challenge.

(4) Since every frontend claim ends as one of 29 opening claims on the one stacked message, which family J joins with its out-of-domain claim, and these m functionals ($m = 30$ in family J, 29 in family U) are batched with powers of one challenge drawn after all values and points are fixed, a candidate of the list that meets the batched claim meets every opening claim at every point, except with probability $(m-1)L/\,|K|$ over that challenge.

Item (4) excludes by construction an attack in which the arithmetic and the hash are satisfied by different list candidates. Item (2) is a descent statement for the K-valued lane candidates used by the WHIR analysis. Expand each lane in the basis $1, X, ..., X^4$ of K over $\mathbb{F}_p$. Its four non-constant coordinate polynomials vanish at every agreeing domain point; the agreement set is larger than their degree, so all four vanish identically. The packed polynomial over J can still have non-zero coefficients in all 64 basis directions of J over $\mathbb{F}_p$. The field J is a device of the extractor only; it changes neither the proof field nor any assumption. The same whole-row argument applies, over an auxiliary extension of K, to every later WHIR oracle, and every fold keeps one agreement set for all the lanes of a row (Lemma C1 of Appendix C); only the first oracle needs the base-field conclusion. One interface identity is easy to get wrong: the frontend reads a table v as values on the Boolean cube, $T_v(X) = \sum_b v_b\, \mathrm{eq}(X, b)$, while the encoder reads the same cells as coefficients, $E_v(z) = \sum_b v_b z^{\mathrm{idx}(b)}$. They are related by $E_v(z) = \sum_b T_v(b) \prod_j \big((1-b_j) + z^{2^j} b_j\big)$, and the out-of-domain functional the library implements is $T_v(\mathrm{pow}(z))$, the multilinear evaluation at the power point. Writing a non-zero multilinear difference as $\sum_S a_S \prod_{j\in S} X_j$, its value at the power point is $\sum_S a_S z^{\sum_{j\in S} 2^j}$; distinct subsets S give distinct exponents, so this is a non-zero univariate polynomial of degree below 2^n, which carries the charged out-of-domain bound (proven).

Coefficients and tables. WHIR [14], and our statements of its steps in Section 11.5 and Appendix C, take a committed message to be the coefficient vector of a multilinear polynomial.

The library (`p3-whir` 0.8.0) instead commits a table v of values on the Boolean cube and reads it in two ways: as values in its folds, its constraints, its out-of-domain answers ($T_v(\mathrm{pow}(z))$) and its final sumcheck, and as univariate coefficients in its encoder and its query answers ($E_v(x)$ at a domain point x). Our analysis is written in the coefficient convention and transfers to the library's. A bijection built from the Möbius transform maps the codes, the folds and the constraints of one convention onto those of the other, the weights being carried by the inverse transpose, so the fold step, Lemma C1 and the list bounds transfer through it. The out-of-domain and shift steps do not, because under that bijection their readers differ from those of the coefficient convention; they hold for the library's readers with the same constants, the out-of-domain step by the root count above. The final query block and the tail of Lemma C2 hold for the final table. These transfers and steps are machine-checked (Section 12).

11.4 Initial extraction

Let $\mathcal{R}_{\mathrm{alg}}$ be the conjunction of the four lookups as integer ranges, ternarity, Booleanity of each plane, the hint counts ν_i (each at most 75, which the verifier checks when parsing), the modular relation of Section 5 with $z = X - 524167$ and $e = Y + \mathrm{start}(u, S)$, and the masked-output identity: the 72 Keccak layers, applied slotwise to the input that the input link builds from μ_i, the states T_g and the bits of u_i, output T_{g+1} on slot $g < 8$ and the encoding of $\tilde{c}_i$ on the first 512 cells of slot 8.

Lemma 4 (from the algebra to signatures). proven given H3, and proved in Lean (Section 12); the generation of the Keccak gather-table dump from the implementation is a reviewed external step. If a candidate w satisfies $\mathcal{R}_{\mathrm{alg}}(x, w)$, then the signatures $\sigma_i = (\tilde{c}_i, \mathrm{Pack}(z_i), \mathrm{HintEnc}(h_i))$ with $h_i = S^2$ satisfy $\mathcal{R}_{\mathrm{auth}}(x, \boldsymbol{\sigma})$.

The proof is an induction over the nine sponge blocks. Booleanity makes u an integer in $[0, 15]$ and d_0, d_{15} exact indicators; the input link makes every slot input a valid row that encodes the previous state XORed with the block; the layers compute Keccak-f on valid rows; the mask forces T_{g+1} to be the output of slot g and the first 512 output cells of slot 8 to encode $\tilde{c}_i$, so $\tilde{c}_i = H(\mu_i \,\|\, \mathrm{w1Encode}(u_i))$ with the same committed bits that the arithmetic uses; Lemma 1 then gives $\mathrm{UseHint}(h, w') = u$ coefficientwise. The algebraic content of each layer is proved. The index part, the θ neighbours, ρ offsets, π permutation and χ neighbours, is a finite table that we checked against a reference Keccak-f on eight random states, and the whole sponge against an independent SHAKE256 implementation [M]. Our source review then derived every table from the FIPS 202 formulas and compared it entry by entry with the code: all 307,200 active gather entries (of 589,824 table cells [D]), every active set and every ι sign match [M]. The formal algebraic and sponge result is proved in Lean. The generated implementation-table dump is also proved to agree with the formal FIPS 202 tables. Generation of that dump from the implementation remains a reviewed external step.

The extractor list-decodes Π_1, builds $\boldsymbol{\sigma}$ for each candidate in a fixed order, skips a candidate that cannot be canonically packed, runs native verification, and returns the first candidate that passes. It reads Π_1 only. Its native-verification part is at most LN verifications, 81,920 at $N = 2048$ in family J, about 5.7 s at the median of 70 µs per verification measured in Section 8 [D], plus decoding: in family U, Berlekamp–Welch uses fewer than $n_0^3 + 3n_0^2$ multiplications in J for n_0 rows, plus comparable additions, at most n_0 inversions, and the system-construction cost

detailed in Section C.6; in family J the cited decoder's polynomial running time has no stated exponent, so that part has no explicit constant (Section C.6).

11.5 The interactive protocol

Theorem 1 (interactive IOP; conditional efficiency). proven, with the two-stage knowledge state below, which our internal review checked; K1–K4 are also proved in Lean for the registered protocol (Section 12); an external review is outstanding (Table 12). For every registered profile of families J and U at the five audited sizes of Table 2, under H3 and H7, the stated decoder-correctness hypotheses, and the family-specific proximity hypotheses, H4 and H6 for family J or H5 for family U, the specified interactive protocol satisfies conditions K1–K4 of relaxed round-by-round knowledge soundness for $\mathcal{R}_{\text{auth}}$ in the sense of [18] (Definition 3.6; the conditions are listed in Appendix C), with the per-round errors κ_i of the ledger. Its deterministic semantic extractor reads Π_1 only. If exact candidate decoding and the construction and native verification of one candidate batch have polynomial-cost implementations, the extractor also satisfies K5. The size measure includes the encoded public instance and profile together with $n_0 + d + N$. Section 12 states the two implementation premises precisely.

The implementation's Keccak gather tables enter through the generated table dump of Lemma 4, whose generation from the implementation is a reviewed external step.

The knowledge state. The state has two stages. Its first version had one stage and was wrong; the remark after the construction explains why.

Stage I: a fixed candidate list. Through the application frontend the state keeps the fixed list: some $W \in \Lambda(\Pi_1)$ satisfies every pending frontend obligation.

Handoff at the batching challenge. At the batching challenge α the state hands off to $\exists W \in \Lambda(\Pi_1) : \ell_\alpha(W) = v_\alpha$, the batched opening constraint, at a cost of $(m-1)|\Lambda(\Pi_1)|$ / $|K|$ with $m = 30$ (J) or 29 (U) functionals (the 29 opening claims, plus the out-of-domain claim in family J), which is the ledger's batching entry. This handoff is a proximity predicate; when it is false the constrained list is empty, which is exactly the premise of WHIR's first-fold argument.

Stage II: phase-local WHIR state. Stage II is WHIR's phase-local state, the existence of a constrained codeword close to the *current* folded oracle, followed, after the final coefficient vector and its query block, by the truth of the remaining sum claim for that vector and then an ordinary sumcheck state through the library's final degree-2 rounds (Lemma C2); it does not refer to $\Lambda(\Pi_1)$. When a new WHIR oracle is committed, the preceding phase's state is kept until the verifier's next coins, conjoined with any rejection flag the commitment determines: a commitment alone never replaces the preceding proximity predicate by the non-emptiness of the new oracle's list, and the state moves to the new phase only through the out-of-domain or shift transitions charged below.

An original valid candidate is recovered by tracing an accepting run back when no charged bad transition occurred. The stages agree at their boundaries, so their error vectors concatenate and the ledger's entries are unchanged [A].

Why the original invariant fails. The first version of this argument defined one state for the whole transcript: some candidate of the fixed list $\Lambda(\Pi_1)$ satisfies every pending obligation,

together with WHIR's state. Our internal review found that this definition is wrong. WHIR can recover from a bad state at an exceptional fold challenge, and its proof charges that recovery at the fold; a state that keeps requiring a member of the *original* list cannot see the recovery. Concretely, adding X^d to one odd lane leaves every whole-row candidate with agreement at most ρ, so $\Lambda(\Pi_1)$ is empty; a first fold challenge $r = 0$ discards that lane, and an honest continuation accepts. The old state stays 0 on every proper prefix and is overwritten by acceptance, so its last bad-transition probability is about 1 although the event is charged in the ledger. A toy instance over $\mathbb{F}_{17}$ reproduces the core by exhaustive enumeration: maximum agreement 4 of 16 before the fold, 16 of 16 after it [M]. This is not a forgery; it is a misplaced state transition.

Error charges. Within Stage I every algebraic phase has the identity as backward extractor, and its bad event is that a non-zero polynomial of known degree, fixed by the candidate before the challenge, vanishes at the challenge; a union over the fixed list gives $Ld_j/\,|K|$. The degrees are those of the registries: 2 per LogUp GKR round, 8 per master round, $\deg f$ and $\deg f + 1$ per Keccak round, 2 per wiring, mask and bit-wiring round, 3 per input-link round, and one joint term for the ternarity, Booleanity, count and projection obligations, whose independent weights are variables of one polynomial. The projection trials are one atomic coin block with no prover message between them, so the product bound $\left(2/q - 1/q^2\right)^{14}$ applies; the multiplicities are committed before θ; all 29 opened values are fixed before the batching challenge. The sum of the frontend degrees, projection and lookup excluded, is 9,024 to 11,985 across the registered sizes, below 2^{20}, and the coin-decoder distance is at most $2^{-572.75}$ in every registered shape [A]. After the projection, the projected obligations are carried as field equalities while the range obligations stay exact, so the no-wrap bound turns each into an integer identity. The WHIR rounds are charged as in WHIR Theorem 5.2 (cited), except the fold term, which the ledger charges in the sharper form proved in Appendix C (proven). Family U has no out-of-domain sample, and for it we prove the shift step directly (proven): before a shift's coins the new oracle has at most one nearby codeword, fixed before the query indices, so a false state survives with probability at most $(1 - \delta_{i-1})^{t_{i-1}} + (t_{i-1} + 1)/|K|$. At the strict radius $\delta_{i-1} = (1 - \rho_{i-1})/2 - 2^{-60}$, with n and d the length and dimension of that phase's code, both powers of two, a codeword farther than δ_{i-1} from the preceding folded oracle differs from it on more than $(n - d)/2 - n2^{-60}$ rows, hence on at least $(n - d)/2$ whole rows. A query then lands on a row where they agree with probability at most $(n + d)/(2n) = (1 + \rho_{i-1})/2$, and the first term is at most $((1 + \rho_{i-1})/2)^{t_{i-1}}$ [X]. The bound is therefore below the ledger's charge $((1 + \rho_{i-1})/2)^{t_{i-1}} + 2(t_{i-1} + 1)/2^{319}$; the final query block of Lemma C2 uses the same integer argument. Unique decoding supplies the selection that the out-of-domain sample provides in the Johnson regime.

The ledger reports $\varepsilon_{\text{sum}} = \sum_i \kappa_i$, the knowledge error of one interactive run, and $\varepsilon_{\max} = \max_i \kappa_i$, which is what the compiled statements below take; the ledger adds to each κ_i its round's coin-decoder distance δ_i^{dec} (not to be confused with the proximity radius δ_i), so both figures include those distances. At $N = 2048$ in family J, $\varepsilon_{\text{sum}} < 2^{-270.36}$ and $\varepsilon_{\max} < 2^{-272.11}$, the largest round being a WHIR query round [A]. Every security exponent in this section is rounded down, so that each stated inequality holds literally.

11.6 Classical compiled bound

Lemma 5 (state restoration, classical). proven, in the random-oracle model of H1. Consider the Fiat–Shamir transform of the interactive protocol of Theorem 1 with the full-history coins of Section 6, in the state-restoration game of [17] (Definition 9.3). For every classical prover making t queries to the coin oracles, the probability that the verifier accepts its output while the extractor fails is at most $(t + k)\varepsilon_{\max}$.

The coins of round i are a function of the full prefix, so each new prefix receives a fresh uniform block. If extraction fails on an accepted output, some prefix on its path has a dead state that its fresh coin turns alive, and a union over the at most $t + k$ prefixes gives the bound. A prefix here is a sequence of interactive prover messages and coins; Merkle roots take the place of the oracles only in the compiler of Theorem 2. Appendix C gives the proofs of Theorem 1 and Lemma 5 in full.

Partial oracles (H9). The compiler extractors of Theorems 2 and 3 recover oracle strings in which some leaves have no preimage among the queries, or a malformed one. The application extractor decodes each recovered leaf against its expected profile, round, kind, length and canonical field encoding, replaces every failure by the fixed zero symbol of its type, and runs the extractor of Lemma 4 on the completed Π_1. The completion map is fixed independently of the coins; any dependence of the recovered partial oracle on the oracle transcript is the compiler theorem's to handle. The verifier rejects when a query hits a missing or malformed symbol, and the knowledge state is evaluated on the completed oracles with an absorbing flag for failed checks, so it is not cleared merely because an unqueried symbol is missing. Acceptance on the partial oracles implies acceptance on their completion, so completion adds no bad-transition probability (proven).

Theorem 2 (classical). cited [17] (full version, ePrint 2024/724, revision posted 5 September 2024), Lemmas 9.12 and 7.15; proven Lemma 5 and Theorem 1; under H1 to H3, H7 and H9, with H4 and H6 for the Johnson family and H5 and H8 for the unique-decoding family, and the two extractor-efficiency premises of Theorem 1. Under the stated split-hash random-oracle model, implementation-correspondence hypotheses and cited proximity results, the audited Johnson and unique-decoding profiles are arguments of knowledge for the relation $\mathcal{R}_{\text{auth}}$: knowledge of canonically encoded FIPS 204 ML-DSA-87 signatures accepted for the bound public keys, pure-mode messages and contexts, with the specified challenge digests and hint counts. For at most 2^{64} adversarial oracle queries (R) and 2^{20} verification submissions (E), the classical extraction error is below $2^{-188.07}$ for Johnson and below $2^{-201.99}$ for unique decoding, uniformly over the audited sizes $N \in \{128, 2048, 4096, 16384, 65536\}$; with the summed per-round error ε_{sum} in place of $\varepsilon_{\max}$ the figures are $2^{-185.26}$ and $2^{-198.69}$.

Both conventions are reported for both families in Table 11; the minimum over sizes is taken in each [A]. With the BCGM fallback in place of DKT26 Theorem 5.12, the Johnson bound is below $2^{-146.33}$ [A]. The relation does not itself prove state execution, nonce correctness, key-resolution correctness or chain selection; those remain the surrounding replay and chain-verification obligations. The numerical theorem covers exactly these five registry-backed sizes per family; any other accepted size needs its own registry export and ledger run before the claim extends to it [X]. The bounds come from

$$\kappa_{\mathrm{cl}}(R,E) \le E\big[(t+k)\big(\varepsilon_{\max}+\delta^{\mathrm{dec}}\big) + (3/2t^2 + 3k(\log_2 \ell_{\max}+1)t)/2^{512}\big], \quad t = R + E(q_V + k - n_{\mathrm{real}}),$$

where $\ell_{\max}$ is the number of leaves of the largest tree, q_V the verifier's oracle calls (counting the singleton hashes twice, which is conservative) and n_{real} the number of real trees, the witness tree and one per WHIR fold commitment: 4 at $N \le 4096$ and 5 at 16,384 and 65,536, in both families [X]. The summed figures use $\varepsilon_{\mathrm{sum}}$ in place of $\varepsilon_{\max}$. Here δ^{dec} is the total decoder allowance summed over the registered rounds. Since $\varepsilon_{\max}$ already includes the per-round decoder distances, the additional allowance is conservative; it is retained to match the compiler budget used in the ledger, as explained in Section C.8.

The specialisation has no simulation queries and 512-bit digests. E counts verification submissions in the experiment, including failed attempts and alternative certificates, not distinct ranges, and the query budget includes the oracle calls that produced any honest certificate given to the adversary. With the final profiles' tagged node hashing, the classical compiler is applied to a Merkle oracle whose inner-node domain is partitioned by tree. This is a short adaptation of the published proof, not its literal statement (proven): giving the extractor each tree's own context and counting the child digests of well-formed node queries, at most twice the number of queries, leaves the collision and extraction counts of Lemma 7.15 and their constants unchanged.

N	Fam.	$-\log_2 \varepsilon_{\max}$	Classical	with $\varepsilon_{\mathrm{sum}}$	BCGM constant	Quantum
128	J	272.07	188.07	186.53	151.83	not claimed
2,048	J	272.11	188.11	186.36	152.83	not claimed
4,096	J	272.07	188.07	186.24	150.83	not claimed
16,384	J	272.11	188.11	186.14	150.33	not claimed
65,536	J	272.11	188.11	185.26	146.33	not claimed
128	U	286.14	202.14	200.41	–	**129.82**
2,048	U	286.14	202.14	200.24	–	**129.82**
4,096	U	286.14	202.14	200.07	–	**129.82**
16,384	U	286.14	202.14	199.20	–	**129.82**
65,536	U	285.99	201.99	198.69	–	**129.67**

Table 11: Compiled bounds, as $-\log_2$ of the extraction error rounded down, at $R = Q = 2^{64}$ queries and $E = 2^{20}$ covered submissions [A]. "Classical" is Theorem 2 with $\varepsilon_{\max}$; the next column uses $\varepsilon_{\mathrm{sum}}$, the convention of the public-u analysis; "BCGM constant" replaces the DKT26 proximity constant by that of BCGM. "Quantum" is Theorem 3, for family U only, with $\varepsilon_{\max}$ as in the CDHZ theorem. For one submission ($E = 1$) the classical figures with the DKT26 constant are 20 bits larger, and the quantum ones are 149.82 and 149.67. The $\varepsilon_{\max}$ column is rounded down like the others; every compiled figure is computed from the exact ledger values, not from the printed exponents.

At $R = 2^{64}$ and $E = 2^{20}$ every row of Table 11 meets 2^{-128}. The DKT26 theorem is recent; with the older BCGM constant, whose error grows with the square of the code length, the family-J bound falls to $2^{-146.33}$ at $N = 65{,}536$ and stays below 2^{-128} at every audited size [A].

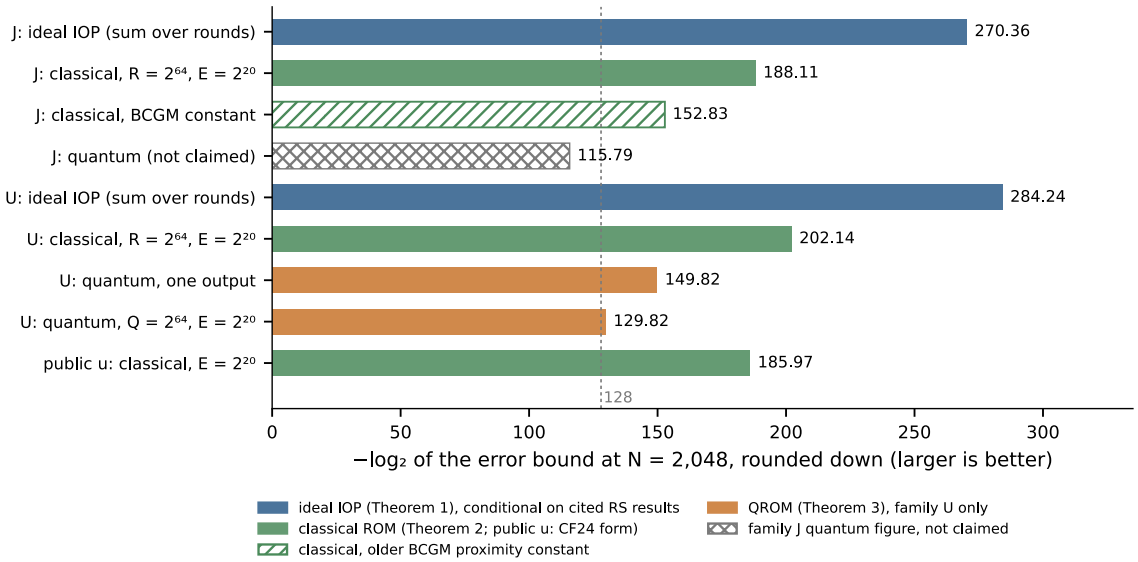


Figure 4: Security ledger at $N = 2048$, as $-\log_2$ of each bound, rounded down. Blue: the ideal IOP (Theorem 1), conditional on cited Reed–Solomon results. Green: classical random-oracle bounds (Theorem 2; for the public-u profile, the form of our earlier analysis); hatched green: the same with the older BCGM proximity constant. Orange: quantum bounds (Theorem 3), family U only. Grey, crossed: the family-J quantum figure, shown for comparison and not claimed. Compiled bounds are at $R = Q = 2^{64}$ queries and $E = 2^{20}$ submissions, except the one-output bar ($E = 1$); the dotted line marks 128.

11.7 Quantum compiled bound (family U)

Theorem 3 (quantum, family U). cited [18] (version of 2 March 2026), Theorems 11.3, 6.10 and 8.3 with Construction 11.7; proven Theorem 1 and H9; under H1Q, H2, H3, H5 and H7 to H9, and the two extractor-efficiency premises of Theorem 1. In the stated split-hash quantum random-oracle model (H1Q), the audited unique-decoding profiles admit the CDHZ straightline selected-output extraction guarantee. For an adversary making at most 2^{64} quantum oracle queries (Q), the extraction reduction covering at most 2^{20} verification submissions has failure probability below $2^{-129.67}$, uniformly over $N \in \{128, 2048, 4096, 16384, 65536\}$.

The figure is an extraction-failure probability at a stated query budget, not a work factor: it does not say that an attack needs 2^{129} operations. Its leading term grows with the square of the number of queries, so at 2^{128} quantum queries the same bound is about $2^{-1.67}$ at $N = 65{,}536$ and $2^{-1.82}$ at the smaller sizes, which gives no meaningful cryptographic security guarantee [D]. The quantum statement also does not establish simultaneous non-disturbing extraction from every output, standard-model security of concrete SHAKE256, or equivalence to ML-DSA-87 category-5 security. The reported bound uses the selected-output experiment and conservative factor-E allowance defined in Section C.8, with preceding verifications charged to the query budget; the bound is $2^{-129.82}$ at $N \le 16{,}384$, and below $2^{-149.67}$ for one submission ($E = 1$), as in Table 11 [A]. It belongs to the unique-decoding family only, whose certificates are larger (126.89 bytes per signature at 65,536); the 104.09 bytes per signature of the Johnson family at the same size carry the classical theorem only.

The specialisation is

$$\kappa_Q(t) \le 320(t+k+1)(t+k)\varepsilon_{\max} + 4(t+k+1)k/2^{r_{\min}} + [1280t(2t+1)^2 + 960T^3 + 480q_V^2(T+1) + 128q_\Pi h]/2^{512},$$

with $T = t + 1 + q_V + k$, $h = \log_2 \ell_{\max} + 1$, $q_\Pi = (k - n_{\text{real}}) + \sum_i t_i$ the singleton rounds plus every proximity query (it is not q_V), $r_{\min} = 2368$ bits, and a factor E for the selection, with $t = Q + E(q_V + k - n_{\text{real}})$ [A]. The binding term is the first: $\varepsilon_{\max}$ is the largest WHIR shift (query) round, about $2^{-286.15}$, and at $N = 65{,}536$ the first fold's proximity term, about $2^{-286.00}$ [A]. Here $r_{\min}$ is the minimum declared length of a complete raw coin block, not the entropy of a decoded challenge; its term is below 2^{-2000} [A]. We checked these hypotheses against the profile: an interactive reduction to the trivial, monotone relation with relaxed round-by-round knowledge (Theorem 1); one Fiat–Shamir oracle per round, through the injective round framing; distinct commitment indices, fixed by the registered table; salts fixed to zero, which only restricts the accepted set; one vector-commitment oracle per tree, which is the per-tree node domain of Section 6; and a relation decider that makes no compiler-oracle query, since $\mathcal{R}_{\text{auth}}$ uses the concrete SHAKE256 (H1). Leaves carry no salt: the Merkle extraction proof of Theorem 8.3 uses no entropy from the leaf salt, so with empty salts its constants are unchanged, a short adaptation (proven). A classical union bound does not give simultaneous extraction of every quantum output, and we do not claim it.

Why family J carries no quantum claim. At $Q = 2^{64}$ and $E = 2^{20}$ the quantum bound needs every round below about $2^{-284.32}$, at every size of family J. In the Johnson regime the mutual-correlated-agreement term of a fold of m variables at rate 2^{-r} grows as 2^{m+2r}, so the largest round whose error queries cannot lower is about $2^{-275.41}$ at $N = 2048$ [A] and $2^{-273.41}$ at $N = 65{,}536$ [A]. The family-J quantum figure would be about 115.8 bits, and raising the query target to 286 or 300 leaves it at about 119.1 at $N \le 16{,}384$ [A] and 117.1 at $N = 65{,}536$ [A]. Increasing the WHIR proximity-query counts at fixed adversarial Q and E cannot bring this bound to the target; that is a limit of the bound and the schedule, not an attack (a smaller adversarial budget would meet it, but that is a different claim). The unique-decoding regime does meet it, as would a larger challenge field. We report the 115.8 for comparison and claim nothing from it.

Category 5 is itself a resource comparison with AES-256 key search, not a bit count [25], so no bit count of this section can be read as a category.

11.8 The public-u profile

For QLYAUTH7 the same method gave an ideal-IOP error of $2^{-269.97}$ with a list of 160 candidates at rate 1/16, conditional on the same cited Reed–Solomon results as family J (H4 with the DKT26 constant, H6 and H7) [A]. With ε_{sum} in place of $\varepsilon_{\max}$, 354 rounds, $q_V = 7559$ verifier oracle calls and a largest tree of 2^{22} leaves [A], the classical theorem of our earlier analysis, which compiles relaxed round-by-round knowledge in the random-oracle model of H1 with a single Merkle oracle and has the same form with ε_{sum}, gives $2^{-185.97}$ at $R = 2^{64}$ and $E = 2^{20}$, and $2^{-205.97}$ for one submission [A], with decoder distance below $2^{-727.20}$. That analysis used the same single-list knowledge state as the first version of the hidden-u argument, and the same counterexample applies to it. We have repaired it in the same way (proven): the list of 160 candidates is kept through the frontend and the batch of the six opening functionals (five openings and the out-of-domain claim), at a cost of $(m-1)L/\,|K| = 5 \cdot 160/\,|K|$; WHIR's

phase-local state follows, then a four-round final sumcheck tail. Every transition is already charged by a term of the earlier ledger, so its figures stand unchanged: rerunning both ledger scripts reproduces $2^{-269.97}$ and $2^{-185.97}$ [A]. QLYAUTH7 shares its node hashing across trees, so the quantum compiler of [18] does not apply to it as written, and no quantum bound is claimed for it; family U is its quantum-ready successor. Interpreting these figures as knowledge-extraction bounds also requires polynomial-cost implementations of the public-u extractor; the registered cost-model result of Section 12 concerns the hidden-u profiles.

11.9 Fiat–Shamir in the standard model

The theorems above are random-oracle theorems. Khovratovich, Rothblum and Soukhanov give, for every hash function, a GKR instantiation that is unsound under Fiat–Shamir, through a circuit that evaluates the Fiat–Shamir hash on its own transcript [26], and Fenzi extends the attack to generated statements, with the mitigation of deriving the first challenge from the generated statement [27]. No standard-model proof exists for our transcript (open). Four structural facts (proven) place it outside the known attack pattern; that they suffice is a judgement, not a theorem. The circuit is fixed by the registered profile, not chosen by the prover or by the statement. Every coin block, the first included, is SHAKE256 over the *raw* instance, so the statement is bound before the first challenge; the framing of Section 6 encodes the profile, the instance and the full root history injectively into each coin input, as H1 requires, for statements whose messages are shorter than 2^{32} bytes (Section 6). Each arithmetized sponge hashes exactly 1,088 bytes from the all-zero state, while every coin input starts with a fixed domain string and contains the whole instance, so no circuit evaluation equals a coin derivation [X]. And coins depend on the full root history, not on a chained state that a circuit could track. Group F of the test suite checks the corresponding rejections (Table 9); those tests show that the implementation detects these deviations from the specified transcript, not that Fiat–Shamir is sound in the standard model.

11.10 What is proven, cited, tested and open

Statement	Status
Lemma 1 (interval form) and the ternary cover	proven in Lean; independently tested, exhaustive
Lemma 2 (equivalence of the certified relation and FIPS 204 verification)	proven given Lemma 1
Keccak layer reductions, masked output, input link, bit wiring, Booleanity zerocheck, degree-8 master sumcheck, projections, lookup, LogUp GKR, table sumcheck, batched opening, coin decoders	proven
Lemma 3 (common list); backward reduction of each phase; frontend-degree and decoder gates	proven; gates computed from the registries
Lemma 4: algebraic and sponge part / generated Keccak gather-table dump	proven given H3, in Lean / proven agreement with FIPS 202 in Lean; generation from the implementation a reviewed external step
Canonical signatures rebuilt and accepted by libcrux (128/128, 2,048/2,048) [M]	tested
Two-stage knowledge state (repairs the single-list state of the first version, which was wrong; the same repair for QLYAUTH7); coefficient/multilinear functional identity; normalized lanes at every WHIR phase	proven; checked by the internal review

Statement	Status
Theorem 1, conditions K1–K4 (relaxed round-by-round knowledge), Lemma 5 (state restoration)	proven given H3 to H7 and the cited decoder theorem; K1–K4 also in Lean
Theorem 1, condition K5 (extractor efficiency)	proven in a defined bit-cost model from two implementation premises, which are open
Partial-oracle completion (H9)	proven
Family-U decoder multiplication count (Berlekamp–Welch); explicit constant for family J's list decoder	proven / open
WHIR rounds	cited WHIR Thm 5.2 with DKT26 Thm 5.12 or BCGM Thm 9.2 (J), WHIR Cor 4.11 (U), WHIR Thm 4.3; the fold bound proven (Appendix C)
Shift step with zero out-of-domain samples (family U)	proven, short adaptation of WHIR Thm 5.2
Theorem 2 (classical)	cited CF24 Lemmas 9.12, 7.15, and proven; tagged-node adaptation proven; conditional on the K5 premises
Theorem 3 (quantum, family U only)	cited CDHZ Thms 11.3, 6.10, 8.3, and proven (empty-salt adaptation); H8 tested; conditional on the K5 premises
Implementation correspondence (H2): every registered row mapped to its statement and lemma; honest runs match bodies and coin prefixes	tested
Internal review of the verifier source against the proof: 36 items, including the LogUp GKR verifier and the table sumcheck	tested (internal reading): 36 match, 0 mismatch, 0 unclear
External review of the proof and the code	open
Standard-model Fiat–Shamir soundness	open
Reduction from extraction to multi-user unforgeability (schematic inequality of Section 11.11)	open
Proofs of the cited theorems (DKT26, BCGM, BCIKS20, WHIR, CF24, CDHZ, the deterministic decoder)	cited, not re-checked

Table 12: Status of every statement of the hidden-u analysis. Proven: proved under the stated hypotheses (full proofs in Appendix C); cited: rests on a published theorem whose statement and hypotheses we checked but whose proof we did not; tested: established by tests or scripts; open: not established. Section 12 lists the statements that are also machine-checked.

The registry is produced by the code itself, so we also read the code against the proof. An internal review went through 36 items of the verifier source against the security document: every terminal-identity function, the opening order, the multilinear functionals and the out-of-domain functional the library implements ($T_v(\mathrm{pow}(z))$, confirmed), quotient parsing and the no-wrap bound, decoder exhaustion, tree-context routing, the Keccak tables, and the LogUp GKR verifier and table sumcheck (`gkr::verify`, `sumcheck::table_verify`). All 36 items match, with no mismatch and no unclear item, and nothing was found that could let an invalid signature pass [M]. The verifier source is unchanged from the reviewed revision to the one that produced the 65,536-signature measurements [M]. This meets our internal acceptance criterion for reading the code against the proof. It is not the independent review that Section 11.12 requires: no independent expert has yet reviewed the proof against the code, and that review remains outstanding; it should cover the state boundaries of the knowledge argument and the transcript correspondence [X]. Certificate-only bootstrap therefore stays off (Section 11.12).

The two extractor-cost implementation premises and the authorization reduction of Section 11.11 remain open, and independent review is outstanding. Section 12.7 lists the remaining formalization obligations, including concrete compiled games and correspondence with the Rust implementation. Standard-model soundness of our Fiat–Shamir instantiation is not established. The cited proximity theorems are assumptions of this analysis; we have not independently rechecked their proofs. Two earlier reproducibility gaps were closed by the final measurement commit: every hidden-u certificate in the size tables has a recorded digest, and the standalone round registries were re-exported from the measured default build with identical rows [X].

11.11 From extraction to authorization

The following is a schematic statement, not a theorem of this paper (open): we have neither defined the game nor proved the inequality. For a suitably defined multi-user authorization game,

$$\mathrm{Adv}_{\mathrm{unauthorized}}(\mathcal{A}) \le \kappa_E + E \cdot \mathrm{Adv}^{\text{MU-EUF-CMA}}_{\text{ML-DSA-87}}(\mathcal{B}) + \varepsilon_{\mathrm{binding},E}.$$

Here κ_E is the extraction allowance for at most E submissions from Theorem 2 or 3. The reduction $\mathcal{B}$ selects one of those submissions uniformly and pays the extractor's time and oracle queries. The term $\varepsilon_{\mathrm{binding},E}$ bounds the probability of any relevant anchor, message-commitment or state-commitment failure across the entire E-submission game; a per-selected-submission binding bound would instead acquire the same factor E. This remains a proposed reduction shape, not a proved inequality. The binding term also needs anchors that bind both endpoint heights of a range: with anchors that hash only accumulator values, a cycle of the accumulator, not only a collision, could let two contents of different lengths share their anchors. A reduction to single-user security must also pay for user selection. Prior public verification of a raw signature does not replace this argument when a later verifier receives only the certificate.

11.12 Deployment policy

The analysis fixes how the certificate may be used today.

- Every fresh block **must** be natively verified, all ML-DSA-87 signatures, before its state transitions are accepted [X].
- Certificate-only bootstrap **must** stay disabled until the activated profile has an adequate quantum knowledge argument, a proved reduction for the intended authorization and header-state game, *and* an independent review of those arguments against the implementation. Family U has a quantum bound, conditional on its hypotheses, that is a candidate for the knowledge condition; the authorization reduction and the independent review remain outstanding, and for the review only the internal one exists. While the rule holds, historical bootstrap obtains and natively verifies raw signatures from the authenticated anchor; certificate rejection or absence never invalidates a natively accepted block nor changes fork choice; failure to retrieve raw signatures stops synchronisation and never falls back to certificates.
- Nodes that validated history may prune their own copies, but complete raw history must remain retrievable for native bootstrap.

During this gated deployment the reductions measured above (84.9 % at 2,048 signatures [M], 97.8 % at 65,536) therefore describe the certificate representation and the evidence a pruned node retains. They do not yet remove raw signatures from the network or from what a new node downloads.

12 Formal verification

We have formalized parts of this paper in Lean 4 (version 4.35.0) [X], using Lean core for the pilot and Mathlib for the second project. This section states what is checked, at which scope, and what remains open; the results are conditional statements about the formal protocol and the stated compiler models, not about the Rust implementation.

12.1 Verification process and checker coverage

The development uses a statement-review-and-freeze process. Some proofs were developed against draft statements and checked against the same statements after freezing. Frozen statement files and their local imported definition dependencies are tracked by SHA-256 manifests. The check scripts verify the hashes and check coverage of the local import closure using Lean's parsed dependencies. Publishing the reviewed source snapshot and its final manifests remains release work.

The supplied builds check the proofs with Lean's kernel. The checks fall into four groups.

- **Library checks.** Axiom audits of the exported declarations report only `propext`, `Classical.choice`, and `Quot.sound`, and the checked source scans contain no `sorry` or `native_decide`. On 8 November 2026 the check scripts of both projects passed on the library, including its replays with `leanchecker` and nanoda.
- **Standalone replay of the theorem dependencies.** The chain of the registered interactive theorem, the decoder-distance budget and the compiler allowance vector described below was also replayed on its own: `leanchecker` accepted its 46 modules [M] and nanoda its 42,568 declarations [M], without error.
- **AXLE-covered bundles.** AXLE has verified Lemma 2, the concrete composition of Lemma 4, the generic partial-oracle completion, the decoder-distance budget and the compiler allowance vector, the bundles of every item of Section 12.2 except the short claims, and those of the generic composition of Theorem 1 and of the conditional compositions of Theorems 2 and 3. The requests for the larger bundles of the registered interactive theorem exceeded the service's time or memory limits, so those bundles have not been verified with AXLE.
- **Modules added after those runs.** Modules added or revised after these runs, for the compiler binding, the Fiat–Shamir transcript, the extractor costs, the FIPS 204 sampler instantiation and the corollaries that use them, are so far checked by the kernel builds and the axiom audits only.

12.2 Checked algebraic and codec results

The following are machine-checked, each at the scope stated.

- **Lemmas 1 and 2.** Lemma 1 and the ternary regions of Table 1, by proof rather than by enumeration. Lemma 2, for one record and for a list of records: for every record (public key, message, context, challenge digest and hint count) and every signature string, FIPS 204 verification accepts the signature, with the record's digest and hint count, if and only if the relation of Section 3 has a witness that encodes to it. This is a byte-level equivalence for the formal transcription of FIPS 204, including the key and signature encodings, NTT operations, `UseHint`, and challenge-hash binding. The hash H and the samplers `ExpandA` and `SampleInBall` remain abstract functions in this theorem; fidelity of the transcription is part of the specification trust boundary.

- **Lemma 3**, for both families, under the cited decoder-correctness hypotheses and, for family J, H6, together with the stated domain and layout conditions. The development includes the radius of item (2) and the batching bound of item (4). The algorithmic formulation uses an abstract polynomial-time predicate and a closure-under-filtering assumption; it does not establish an explicit running-time bound for the complete application extractor.
- **Lemma 4.** The algebraic part; the hash part, that the arithmetized sponge computes SHAKE256 as FIPS 202 specifies it; and the composition from $\mathcal{R}_{\text{alg}}$ on a candidate's cells to signatures that satisfy $\mathcal{R}_{\text{auth}}$, with SHAKE256 as the hash and the samplers abstract. The generated Keccak gather-table dump is proved to agree with the formal FIPS 202 tables. Generation of that dump from the implementation is performed by a script outside Lean and is not itself machine-checked. That the verifier enforces $\mathcal{R}_{\text{alg}}$ belongs to Theorem 1 for the registered protocol.
- **The WHIR steps** in the library's table convention (Section 11.3): the correspondence with the coefficient convention, through which the fold step, Lemma C1 and the list bounds carry over; the out-of-domain step and the shift steps of both families, given the list bound of the new oracle; and the final block and tail of Lemma C2. In the coefficient convention, Lemmas C1 and C2, the fold bound and the zero-OOD shift step of family U are proved as written, with WHIR's out-of-domain and shift steps cited.
- **The five claims of Sections 3 to 5**: the hint count is checked per signature (an epoch-wide budget admits fixed public records, with their published hint counts, that no accepted signatures match); the paired LogUp leaf identity, under both of its non-vanishing conditions; the lookups do not imply ternarity ($S = 1/2$ passes them at every interior u); the coefficient-domain adjoint of the projection, and that the projected value of a trial is $\alpha^{\top} R\eta \bmod q$; and the multilinear extension of d_0 is not $\prod_j (1 - \tilde{B}_j)$, although the two agree on the cube.
- **Short claims of Section 5 and Appendix C**: the identity F4, with its active-gather premise; the lookup challenge θ that the registered decoder returns, fallback included, lies outside $\mathbb{F}_p$, so no leaf denominator of a base-field witness vanishes; the lookup no-wrap argument, as the pair bound and the count bound; the LogUp root criterion and the degree sum of the LogUp GKR rounds; the bound on a false hint count; and the degree of the master summand, at most 8 and exactly 8 when $\gamma_Q \neq 0$.
- **The counterexample of Section 11.5** over $\mathbb{F}_{17}$ shows that the original qualifying list can be empty while the folded word has a qualifying codeword. It invalidates the withdrawn state invariant that requires later candidates to remain tied to that original list.
- **Field and domain facts**: p is prime; $X^5 - 3$ is irreducible over $\mathbb{F}_p$, so K is a field with p^5 elements; the implementation's coordinates of K, as transcribed, form a basis, and its product of coordinates is the field product; the domain generators have their stated two-adic orders, and the first oracle's evaluation points are distinct and form a smooth subgroup.

12.3 Registered interactive theorem and partial oracles

Generic composition of Theorem 1, in both conventions. From the stated Stage I rules and bounds of Appendix C, the handoff, the WHIR transition obligations, and the final-tail obligations, the two-stage state of Section 11.5 yields K1–K4 of relaxed round-by-round knowledge soundness at the ledger's per-round errors. The extractor is a deterministic function whose first-step output depends only on the instance and the first oracle. The full extractor's polynomial running time, K5, is not established by this theorem.

Generic partial-oracle completion. Completing failed leaves with fixed symbols and recording queried failures in an absorbing latch preserves K1–K4 and the run-error bound, under the stated acceptance laws.

Registered interactive theorem. We formalized the registered Geode interactive verifier in Lean and proved the K1–K4 component of its relaxed round-by-round knowledge soundness. For every registered profile, the theorem applies to the specified sampler parameter and every exact candidate lister, assuming H4 at the selected proximity constant and H6 for family J, or H5 for family U. The extractor reads only the first oracle. Its per-round error is the registered ledger charge plus the actual statistical distance of the round's coin decoder; the corresponding single-run knowledge-error bound is the sum of those errors. All operational assembly premises P1–P16 are discharged for the registered model, and H3 and the table-convention H7 are proved within the development.

Scope. This result concerns the Lean interactive model with independent uniform raw coin blocks and explicit decoders. It does not establish the full extractor's polynomial-time bound K5, identify the model with the compiled Fiat–Shamir experiments, or prove end-to-end correspondence with the serialized Rust verifier. The sampler interface remains parametric. The FIPS sampler package supplies bounded error-preserving APIs and conditional refinement to the unbounded sampling algorithms upon success; it does not prove successful termination for every seed. Registered partial-oracle completion is proved at the decoded-oracle level, as described below. Recovering that partial oracle from Merkle openings and discharging the remaining compiler assumptions are separate obligations.

Registered partial-oracle completion. The partial-oracle completion result is proved for the decoded oracle of the registered protocol: the failure latch and its laws, filling invariance, the identification of the declared queries with the verifier's actual oracle reads, and the transfer of the registered interactive theorem to the protocol with partial oracles. Recovering the partial oracle from Merkle openings belongs to the compiler and is not covered.

12.4 Ledgers, transcripts and node encodings

Numerical and compiler-ledger results. We proved the registered decoder-distance budget and constructed the chronological compiler allowance vector, which dominates the registered knowledge-error vector round by round and has the required ledger arrangement. The registered-model interface requires exact equality between the compiler's interactive knowledge predicate and that of the registered formal-block experiment; identification from this defining equality and the resulting transfer are proved. The compiler model's adversary, resource and failure-probability fields remain abstract. Identifying those fields with concrete Fiat–Shamir and Merkle experiments remains open.

Registered error ledgers. For the registered protocol, with its coin decoders included, the $\varepsilon_{\mathrm{sum}}$ column of Table 2 and the figures of Section 11.5 at $N = 2048$ in family J hold as stated, as strict inequalities on the exact ledger values. Exact arithmetic on the registered ledgers also establishes the reported compiled figures (Table 11 and Section 11.7), the census values (Appendix B), and bounds on the ledger's decoder-error allowance. The chain-range profiles additionally use A-CHAIN for their numerical ledgers and H2 for their implementation correspondence, as distinguished in Section 7.

Fiat–Shamir transcript and node inputs. For a transcription of the registered build's coin derivation (Section 6) and of the arithmetized sponge's possible inputs, we proved the structural facts of Section 11.9, the faithfulness of the raw instance and of every coin input, and the injectivity of the complete statement serialization, for every registered profile and for all of them in one transcript. These results hold on canonical statements, whose public keys have exactly 2,592 bytes [X] and whose messages are shorter than 2^{32} bytes; Section 6 explains why both conditions are needed.

The transcript uses the formal coin block of Section 6, and its prefix consistency and decoder correspondence with the shorter prefix the implementation reads are proved. The transcription itself is reviewed, not proved. For the per-tree node hashing of Section 6, the input has 135 bytes [X] and fits one SHAKE256 block, its encoding is injective, and distinct trees and profiles give distinct inputs; the binding of the hash itself remains hypothesis H1.

12.5 Extractor costs

The formalization proves the registered interactive knowledge-soundness theorem, including the run-error form, for the Lean protocol and its stated error ledger, under the cited hypotheses retained by the registered theorem. For the extractor's efficiency condition K5, we define a deterministic first-order program language with explicit bit-cost semantics and counted access to the first oracle. Its transcript interface exposes the first-coin-prefix flag and the first oracle in canonical form; coin values and later-oracle contents are not accessible. Returning an unchanged extractor state is represented by state reuse.

We construct a common extractor wrapper, instantiated with one decoder program per family and a candidate-processing program uniform over the registered profiles. We prove that its output is exactly the output of the semantic round extractor, including the order in which candidates are tested and the choice of the first successful candidate. This also gives exact agreement with the extractor on formal coin-block transcripts. From decoder and candidate-processing bounds with constants (C_d, C_c) and exponents (k_d, k_c), we derive the bound

$$10000(C_d+1)^2(C_c+1)^2(s+1)^{2(k_d+k_c+1)},$$

where s is the encoded instance and profile size together with $n_0 + d + N$. K5 is therefore derived, in this cost model, from two named implementation premises:

- DecoderCost requires a program returning exactly the common candidate list, without repetition and in canonical form, with a polynomial cost bound.
- CandidateCost requires a program that constructs and natively verifies one candidate batch, including signature encoding, SHAKE256 and the samplers, with a polynomial cost bound.

These implementation premises remain open. In particular, the formalization does not yet connect the abstract polynomial-time predicates of the cited CHK and Berlekamp–Welch results to DecoderCost, and it does not supply the candidate-processing program. The cost results concern the fixed registered profiles and fields; they do not exclude finite lookup-table implementations, establish useful implementation constants, or prove asymptotic bounds beyond those profiles. Their costs are those of the stated semantics, rather than measured or formally derived runtimes of the Lean or Rust implementations.

12.6 FIPS 204 sampler instantiation

We instantiate the registered theorem with the bounded sampler definitions and their auxiliary total fallback instance. We also define faithful total extensions of the unbounded FIPS 204 sampler semantics: such an extension returns the prescribed result whenever the corresponding unbounded algorithm terminates, and is unconstrained where that algorithm has no result. We prove that a faithful total extension exists, using classical choice, and instantiate the parametric registered theorem with every such extension. This construction establishes neither an executable unbounded sampler nor its termination or running time.

Under explicit per-seed predicates asserting that `SampleInBall` and all `ExpandA` entries finish within the specified limits, we prove agreement between the fallback instance and every faithful extension for signature verification, witness extraction and the authorization relation. Consequently, the fallback verifier's knowledge-soundness statement transfers to the faithful authorization relation on that domain, with the same extractor and error ledger. This transfer retains the fallback verifier; it does not prove equality of the two IOP verifiers' acceptance predicates. The within-limit predicates remain hypotheses. No universal termination result, adaptive sampler-failure bound or additional probabilistic error allowance is derived here.

12.7 Compiler corollaries and remaining obligations

Theorems 2 and 3 are checked as conditional compositions over an abstract model of the compiled argument: given the compiler's Theorem 1 premise, including its extractor-efficiency requirement, the cited compiler theorems, Lemma 5 where required, and the stated adaptation, selection, and query-mass hypotheses (Section 11 and Appendix C), the displayed bounds follow. Applying these figures to the registered operational protocol requires the registered results above and the compiler-binding obligations that remain open.

Compiler corollaries. We prove registered forms of Theorems 2 and 3, including the BCGM variant of Theorem 2, without a standalone K5Registered premise. They derive that premise from DecoderCost and CandidateCost at the defined cost class. The corollaries retain the registered-model identification, the stated compiler premises and the adversary resource bounds. The general classical corollary takes decoder-cost premises for both families; the BCGM and quantum corollaries take the premise for their respective family. The existence of a faithful FIPS sampler extension does not discharge the candidate-processing cost premise.

Remaining obligations. The following are not yet machine-checked.

1. **Cost implementations:** the two implementation premises of K5, DecoderCost and CandidateCost.
2. **Concrete compiler games and recovery:** the concrete compiled games, with the random-oracle models they use, and Lemma 5 on them; the compiler's recovery of the partial oracle from Merkle openings; the identification of the abstract compiler's adversary, resource and failure-probability fields with concrete Fiat–Shamir and Merkle experiments; and the retained compiler adaptations.
3. **Certificate parsing:** the certificate-byte parser and the round trip of a whole certificate.
4. **Rust correspondence:** the correspondence between the implementation's DFT order and the formal protocol, and an end-to-end correspondence between the Rust verifier and the Lean protocol.

5. **Honest-prover and authorization results:** the honest-prover claims and the authorization reduction of Section 11.11.

Smaller remaining claims have the status recorded in the coverage inventory. Cited external theorems enter as hypotheses and are not re-proved.

The machine-checked results concern the specified mathematical objects under their explicit hypotheses. In the compiler compositions, H1 and H1Q are hypothesis fields intended to represent the classical and quantum random-oracle assumptions; those oracle models and the compiled security games are not themselves formalized here. Correspondence with the Rust implementation is supported by reviewed transcription of the pinned source, together with the proved comparison against the generated Keccak table dump. There is no end-to-end refinement proof from the Rust verifier and certificate bytes to the Lean protocol. The formalization does not replace the independent implementation review required by Section 11.12.

Writing and reviewing the statements exposed claims in the first version that were wrong or unsupported as written. The revision history records the corrections made and the issues that remain open. The Lean-source release and its commit identifier remain pending.

13 Related work

Aggregating lattice signatures. Lattice signatures do not aggregate natively: sequential half-aggregation of Fiat–Shamir-with-aborts signatures, the family of ML-DSA, achieves only small compression [28], and synchronized multi-signatures such as Lemur [29] require new key material and a common message. The dominant approach is therefore to prove knowledge of valid signatures. Aardal et al. [30] aggregate Falcon with LaBRADOR [31], keep each signature's salt public so that hashing stays outside the proof, and give a complete knowledge-soundness analysis of non-interactive LaBRADOR; an implementation compresses 10,000 Falcon-512 signatures into 74 KB [32] [X]. Orthus [33] reduces verification cost for batched lattice relations.

Aerie [4] proves the full Falcon verification relation, including the encoding checks, over the Akita lattice polynomial commitment [5], in the line of LaBRADOR and Greyhound [34]. On a 16-thread laptop it aggregates 32,768 Falcon-512 signatures in 0.65 s, 42.1 bytes per signature of which 40 are salts, and 65,536 Falcon-1024 signatures in 1.86 s, about 41 bytes per signature, with a verifier faster than native verification and no recursion [X]. Its knowledge soundness is proved in the classical random-oracle model; for Akita, its authors name a quantum random-oracle analysis as the main open soundness issue [X]. The same commitment replaces Dory in Lattice Jolt [35], a general-purpose zkVM targeting 128 bits under Module-SIS [X]; it could run ML-DSA verification as a guest program, but no ML-DSA figures are published.

Our construction follows Aerie's pattern, one non-recursive proof of the algebraic relation and its bounds with extraction of the signatures, and is much weaker on performance: per signature it is about 2.5 to 3 times larger with salts counted, its prover is two to three orders of magnitude slower on different hardware, and its verifier is slower than native verification (Section 8.3). It differs in three respects. ML-DSA's challenge is a hash of a value derived from the signature, so the hash cannot be left to the verifier without publishing that value, and proving it inside the proof is what our hidden-u certificate adds. Our only cryptographic assumption is the hash function, modelled as a random oracle, next to the cited Reed–Solomon proximity theorems.

And for the unique-decoding family we state a quantum random-oracle bound, through a compiler for hash-based interactive oracle proofs [18] that does not apply to lattice folding.

On ML-DSA, LayerZero's announcement of Aerie states that ML-DSA aggregation exists, with performance and implementation details still to be published [6], while the Aerie paper covers Falcon only and the accompanying a16z post treats ML-DSA as future work, noting that its hash depends on the signature [36] [x]. A lattice ML-DSA aggregate with the hash proven inside would very likely be smaller and faster than ours; we make no claim of priority over it.

Proving exact ML-DSA. Hekate [7] proves the exact FIPS 204 `ML-DSA.Verify_internal`, Keccak included, with a hash-based proof over binary fields. It batches ML-DSA-87 signatures in one proof [x], reports a 1,604 KiB proof for its ML-DSA-87 benchmark in the v0.38.0 release, targets 100 bits, and is unaudited. It shows that proving exact ML-DSA-87 verification with SHAKE can be done; we found no comparable archive-compression measurement for it. Earlier work proves a STARK-friendly Dilithium variant [37] rather than the standardized scheme.

Hash-based aggregation. Ethereum's post-quantum roadmap pairs hash-based signatures designed for proving [38] with leanVM [39], whose README at commit 4af2cdd reports aggregation of 1,400 such signatures into 358.5 KiB and states a 128-bit random-oracle and 64-bit quantum random-oracle soundness target [x]. EIP-8292, "Post-Quantum Attestation Aggregators", reports about 227 KiB for 1,000 signatures [x] at about 124-bit classical and 62-bit quantum security for one leanVM instantiation [40], and EIP-8288, "In-mempool signature and proof aggregation", aggregates at the mempool and builder layer, with one recursive STARK per block [41]. These systems replace the signature scheme and rely on recursion. The cited recursive analyses use idealized models [42]. The impossibility in [43] rules out succinct arguments for relations that themselves query the random oracle, and therefore that route to recursive composition; it does not rule out every recursive construction in the random-oracle model. Our construction avoids recursion. We keep unmodified FIPS 204 signers and stay off the fresh-block path. History pruning [44], [45] removes data after validation but gives a new node no replacement for the signatures.

Proof aggregation in a post-quantum chain. Quantus [46] signs transparent transfers with ML-DSA-65 by default, or ML-DSA-87 [x]. Its whitepaper bounds ML-DSA-87 transfers by block space at about 43 per second, with 3.75 MB blocks every 12 s [x]. Its encrypted transfers replace per-transfer signatures with Plonky2 proofs over the Goldilocks field with Poseidon2, which show knowledge of hash preimages rather than signatures, and two layers of recursion merge up to seven proofs and then up to 53 batches, for a stated rate of about 430 transfers per second [x]. These rates are block-space bounds, not measurements, and the cited passages give no numerical soundness bound for the proof system [x]. Quantus raises the rate of new transfers by changing how they are authorized. Our certificate does not change how transactions are authorized: it compresses the history of blocks already validated, proves unmodified ML-DSA-87 signatures without recursion, and leaves the fresh-block path, and so the transfer rate, unchanged.

Proof-system foundations. Our commitment is WHIR [14]. In the Johnson regime, mutual correlated agreement is proven for all polynomial generators [21], and Dao, Kominers and Thaler give sharper constants and provable bounds beyond the Johnson radius [20]. Our Johnson family rests on its Theorem 5.12 (H4); with it, that family's quantum figure would be about $2^{-115.8}$, short of 2^{-128}, which is why our quantum claim is made for unique decoding

only (Section 11.7) [A]. The up-to-capacity conjectures on which smaller parameters relied have been disproved in the forms those parameters used [47], [48], [49], and extension codes over small base fields are especially exposed [50]; Goldilocks[5] is such an extension. We use only the proven Johnson and unique-decoding bounds: every query count is the planner's Johnson or unique-decoding formula, and no list size beyond the Johnson bound enters. We compile with the full-history BCS transform [16] and state concrete classical bounds following Chiesa and Fenzi [17]. Quantum security of such compilers was established asymptotically [51], and with concrete constants and adaptive straightline extraction for interactive oracle reductions [18], which we specialise. The attacks in [26], [27] exploit specific forms of self-reference or binding of generated statements. Section 11.9 explains the structural differences of our fixed circuit and transcript; those differences do not establish standard-model Fiat–Shamir soundness. Hash-based accumulation [52], [53] could merge range certificates; in our construction that would bring back a recursive verifier. Binary-field provers for Keccak [19] are two orders of magnitude faster than our prime-field Keccak argument, at the cost of a link between bits committed over $\mathbb{F}_2$ and the Goldilocks witness, for which we do not provide a security argument in this work.

14 Limitations and conclusion

14.1 Limitations

Prototype and chain assumptions. The hidden-u profiles are laboratory profiles. The laboratory chain has synthetic headers, no proof of work and no transaction roots; key rotation inside one range and bootstrap are not exercised. In the devnet of Section 8.7 block headers do not yet commit to the accumulator.

Performance and reproducibility. All timings are contended figures on shared machines whose speed varies widely from host to host, so only same-host comparisons are speed-up factors. Above 4,096 signatures the arrays exceed the 24 GB of a worker: at 65,536 the first prover took about 52 minutes and moved about 2.1 TiB to and from disk. The optimised prover of Section 8.5 took 677 and 750 seconds for the unique-decoding range on two hosts and still reads and writes about 0.5 TiB, mostly in the WHIR commitment and opening. Four reruns from a fresh build reproduced its certificate byte for byte but not those times: 782.5 s at best and up to 1,967.9 s in the proof routine, a variation consistent with the disk throughput during the commitment and the CPU resources of the shared hosts (Section 8.6) [M]. It has no same-host comparison at that size, and was not run on the Johnson family above 2,048 signatures. A certificate is several times slower to verify than the raw signatures, and the public $\tilde{c}$ and ν set a floor of 65 bytes per signature [D].

Sustained-load evidence. In devnets under full blocks, two workers with the optimised prover kept up over finite load windows at 16,384 and 65,536 signatures, and with one worker keep-up was not demonstrated in the first campaigns; a later single-worker campaign at 65,536, on a faster placement, kept pace with eligible ranges without meeting its pre-registered margin rule. But those devnets carried 55 to 60 transactions per second, no run was made at exactly 64, the three nodes shared one container, and keep-up over a window is not a stability guarantee. Proof times with the same binary varied 2.2 times between rented hosts and the successful pair had one fast and one slow host, so provisioning must use the slower placement's complete worker cycle; an all-slow pair and a worker that proves the next range while the node verifies were not measured.

Security and formal-verification scope. The analysis rests on stated hypotheses. Lean checks the registered interactive theorem's conditions K1–K4 and the table-convention WHIR steps under the declared coding hypotheses. Its cost-model result derives K5 from two implementation premises that remain open. The compiler corollaries identify the interactive predicate and ledger within an abstract compiler interface; concrete compiled games and execution correspondence with Rust are not established by these proofs, and implementation correspondence is tested, not proved. The tagged-node and empty-salt compiler adaptations remain separate arguments. The cited external theorems are assumed rather than re-proved, the family-J list decoder's cost has no explicit constant, the authorization reduction remains schematic, and independent review is outstanding.

Deployment requirements. Certificate-only bootstrap remains disabled under all the conditions of Section 11.12, so the certificate does not yet reduce what a new node downloads.

14.2 Conclusion

Within those limits the result is concrete. ML-DSA-87 verification is an exact bounded relation once the high bits are fixed. A specialised proof of that relation, with the challenge hash proven by a Keccak GKR over the same commitment, stores 2,048 authorizations in 1.43 MB [M] instead of 9.48 MB, and 65,536 authorizations in 6.82 MB instead of 303 MB, with a stated classical extraction bound under a published compiler; a unique-decoding family, 8.32 MB at 65,536 signatures, also has a stated quantum extraction bound under another. The next steps are the external review, committing the accumulator in block headers, streaming the WHIR commitment, which is still partly disk-bound, and a sustained run at the 64 transactions per second of full 2-second blocks, before any further compression.

Acknowledgments

The author thanks the maintainers of Plonky3 and libcrux. Every measurement in this paper comes from code at a recorded commit, with its transcript, and every analytic bound was recomputed by a script (Appendix A). An external review of the analysis against the implementation is still pending.

References

[1] National Institute of Standards and Technology, "Module-Lattice-Based Digital Signature Standard," Federal Information Processing Standards Publication FIPS 204, Aug. 2024. doi: 10.6028/NIST.FIPS.204.

[2] Cryspen, "libcrux-ml-dsa, version 0.0.11: formally verified ML-DSA implementation." [Online]. Available: https://crates.io/crates/libcrux-ml-dsa/0.0.11

[3] Y. Alloul, "Qlyphs EVM: an Ethereum-compatible blockchain where every signature is ML-DSA-87 (manuscript in preparation)." 2026.

[4] A. Centelles, Q. Dao, M. Badakhshan, and J. Thaler, "Aerie: Efficient Aggregation of Falcon Signatures." [Online]. Available: https://eprint.iacr.org/2026/2501

[5] Q. Dao *et al.*, "Akita: A High-Performance Lattice-Based Polynomial Commitment Scheme." [Online]. Available: https://eprint.iacr.org/2026/1983

[6] LayerZero Labs, "Introducing Aerie." [Online]. Available: https://layerzero.network/blog/introducing-aerie

[7] Oumuamua Labs, "Hekate, version 0.38.0." [Online]. Available: https://github.com/oumuamua-labs/hekate/releases/tag/v0.38.0

[8] National Institute of Standards and Technology, "SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions," Federal Information Processing Standards Publication FIPS 202, 2015. doi: 10.6028/NIST.FIPS.202.

[9] C. Lund, L. Fortnow, H. Karloff, and N. Nisan, "Algebraic Methods for Interactive Proof Systems," *Journal of the ACM*, vol. 39, no. 4, pp. 859–868, 1992.

[10] U. Haböck, "Multivariate lookups based on logarithmic derivatives." [Online]. Available: https://eprint.iacr.org/2022/1530

[11] S. Papini and U. Haböck, "Improving logarithmic derivative lookups using GKR." [Online]. Available: https://eprint.iacr.org/2023/1284

[12] S. Goldwasser, Y. T. Kalai, and G. N. Rothblum, "Delegating Computation: Interactive Proofs for Muggles," *Journal of the ACM*, vol. 62, no. 4, p. 27:1–27:64, 2015.

[13] A. Gruen, "Some Improvements for the PIOP for ZeroCheck." [Online]. Available: https://eprint.iacr.org/2024/108

[14] G. Arnon, A. Chiesa, G. Fenzi, and E. Yogev, "WHIR: Reed–Solomon Proximity Testing with Super-Fast Verification," in *EUROCRYPT 2025*, in LNCS. 2025. [Online]. Available: https://eprint.iacr.org/2024/1586

[15] Polygon Zero and contributors, "Plonky3: a toolkit for polynomial IOPs; crates p3-whir, p3-sumcheck and p3-challenger, version 0.8.0." [Online]. Available: https://github.com/Plonky3/Plonky3

[16] E. Ben-Sasson, A. Chiesa, and N. Spooner, "Interactive Oracle Proofs," in *Theory of Cryptography Conference (TCC 2016-B)*, in LNCS. 2016.

[17] A. Chiesa and G. Fenzi, "zkSNARKs in the ROM with Unconditional UC-Security," *Cryptology ePrint Archive, Paper 2024/724, revision posted 5 September 2024; full version of the paper in Theory of Cryptography Conference (TCC 2024), LNCS vol. 15364, pp. 67–89, doi:10.1007/978-3-031-78011-0_3*, 2024, [Online]. Available: https://eprint.iacr.org/2024/724

[18] A. Chiesa, Z. Di, Z. Hu, and Y. Zheng, "How to Prove Post-Quantum Security for Succinct Non-Interactive Reductions," *Cryptology ePrint Archive, Paper 2025/2166, revision of 2 March 2026; full version of the paper in EUROCRYPT 2026, LNCS, doi:10.1007/978-3-032-25336-1_14*, Mar. 2026, [Online]. Available: https://eprint.iacr.org/2025/2166

[19] B. Bünz, R. D. Rothblum, and W. Wang, "Flock: Fast Proving for Batch Boolean Computations." [Online]. Available: https://arxiv.org/abs/2607.27491

[20] Q. Dao, S. D. Kominers, and J. Thaler, "Reed–Solomon Codes Beyond Johnson: Efficient Decoding and Smaller Cryptographic Proofs." [Online]. Available: https://eprint.iacr.org/2026/2056

[21] S. Bordage, A. Chiesa, Z. Guan, and I. Manzur, "All Polynomial Generators Preserve Distance with Mutual Correlated Agreement." [Online]. Available: https://eprint.iacr.org/2025/2051

[22] E. Ben-Sasson, D. Carmon, Y. Ishai, S. Kopparty, and S. Saraf, "Proximity Gaps for Reed–Solomon Codes," in *IEEE Symposium on Foundations of Computer Science (FOCS 2020)*, 2020.

[23] S. Chatterjee, P. Harsha, and M. Kumar, "Deterministic list decoding of Reed–Solomon codes." [Online]. Available: https://arxiv.org/abs/2511.05176

[24] L. R. Welch and E. R. Berlekamp, "Error correction for algebraic block codes," U.S. Patent 4633470, Dec. 30, 1986 [Online]. Available: https://patents.google.com/patent/US4633470A/en

[25] National Institute of Standards and Technology, "Submission Requirements and Evaluation Criteria for the Post-Quantum Cryptography Standardization Process, Section 4.A.5 (security strength categories)," in *NIST PQC call for proposals*, 2016. [Online]. Available: https://csrc.nist.gov/projects/post-quantum-cryptography/post-quantum-cryptography-standardization/evaluation-criteria/security-(evaluation-criteria)

[26] D. Khovratovich, R. D. Rothblum, and L. Soukhanov, "How to Prove False Statements: Practical Attacks on Fiat–Shamir," in *CRYPTO 2025*, 2025. [Online]. Available: https://eprint.iacr.org/2025/118

[27] G. Fenzi, "How to prove more false statements: Fiat–Shamir limitations on (generated) R1CS." [Online]. Available: https://eprint.iacr.org/2026/1838

[28] K. Boudgoust and A. Takahashi, "Sequential Half-Aggregation of Lattice-Based Signatures," in *ESORICS 2023*, 2023. [Online]. Available: https://eprint.iacr.org/2023/159

[29] Y. Lin, M. F. Esgin, A. Sakzad, R. Steinfeld, and M.-J. O. Saarinen, "Lemur: Scalable Post-Quantum Synchronized Multi-Signatures," in *ACM CCS 2026*, 2026. [Online]. Available: https://eprint.iacr.org/2026/1161

[30] M. A. Aardal, D. F. Aranha, K. Boudgoust, S. Kolby, and A. Takahashi, "Aggregating Falcon Signatures with LaBRADOR," in *CRYPTO 2024*, in LNCS. 2024.

[31] W. Beullens and G. Seiler, "LaBRADOR: Compact Proofs for R1CS from Module-SIS," in *CRYPTO 2023*, in LNCS. 2023.

[32] D. Nevado, D. Kim, and M. Stopar, "Lattice-based signature aggregation," *Ethereum Research*, May 2025, [Online]. Available: https://ethresear.ch/t/lattice-based-signature-aggregation/22282

[33] M. Bolboceanu, J. Bootle, V. Lyubashevsky, A. Merino-Gallardo, and G. Seiler, "Orthus: Practical Sublinear Batch-Verification of Lattice Relations from Standard Assumptions," in *CRYPTO 2026*, 2026. [Online]. Available: https://eprint.iacr.org/2026/398

[34] N. K. Nguyen and G. Seiler, "Greyhound: Fast Polynomial Commitments from Lattices," in *CRYPTO 2024*, in LNCS. 2024. [Online]. Available: https://eprint.iacr.org/2024/1293

[35] M. Georghiades, J. Thaler, A. Tretyakov, and M. Zhu, "Entering the era of lattice SNARKs, with a faster, post-quantum Jolt." [Online]. Available: https://a16zfrontier.com/posts/article/lattice-snarks-jolt-post-quantum-faster

[36] A. Centelles, Q. Dao, J. Thaler, M. Georghiades, M. Zhu, and A. Tretyakov, "Fast post-quantum signature aggregation with lattice SNARKs." [Online]. Available: https://a16zfrontier.com/posts/article/fast-post-quantum-signature-aggregation-with-lattice-snarks

[37] G.-V. Policharla, B. Westerbaan, A. Faz-Hernández, and C. A. Wood, "Post-Quantum Privacy Pass via Post-Quantum Anonymous Credentials." [Online]. Available: https://eprint.iacr.org/2023/414

[38] J. Drake, D. Khovratovich, M. Kudinov, and B. Wagner, "Hash-Based Multi-Signatures for Post-Quantum Ethereum." [Online]. Available: https://eprint.iacr.org/2025/055

[39] leanEthereum contributors, "leanVM README." [Online]. Available: https://github.com/leanEthereum/leanVM/blob/4af2cdddd7cd41b9df2cc7e123e314a91b0b4462/README.md

[40] Ethereum Improvement Proposals, "EIP-8292: Post-Quantum Attestation Aggregators." [Online]. Available: https://eips.ethereum.org/EIPS/eip-8292

[41] V. Buterin and T. Coratger, "EIP-8288: In-mempool signature and proof aggregation." [Online]. Available: https://eips.ethereum.org/EIPS/eip-8288

[42] A. Chiesa, Z. Guan, S. Samocha, and E. Yogev, "Security Bounds for Proof-Carrying Data from Straightline Extractors," in *Theory of Cryptography Conference (TCC 2024)*, 2024. [Online]. Available: https://eprint.iacr.org/2023/1646

[43] A. Barbara, A. Chiesa, and Z. Guan, "Relativized Succinct Arguments in the ROM Do Not Exist," in *Theory of Cryptography Conference (TCC 2025)*, 2025. [Online]. Available: https://eprint.iacr.org/2024/728

[44] G. Kadianakis, lightclient, and A. Stokes, "EIP-4444: Bound Historical Data in Execution Clients." [Online]. Available: https://eips.ethereum.org/EIPS/eip-4444

[45] Bitcoin Core developers, "Bitcoin Core 0.11.0 release notes: block file pruning." [Online]. Available: https://bitcoincore.org/en/releases/0.11.0/

[46] C. Smith and J. Angle, "Quantus Whitepaper, version 0.4.1." [Online]. Available: https://www.quantus.com/whitepaper/v0.4.1/

[47] E. Crites and A. Stewart, "On Reed–Solomon Proximity Gaps Conjectures." [Online]. Available: https://eprint.iacr.org/2025/2046

[48] B. E. Diamond and A. Gruen, "On the Distribution of the Distances of Random Words," in *CRYPTO 2026*, 2026. [Online]. Available: https://eprint.iacr.org/2025/2010

[49] D. Krachun, S. Kazanin, and U. Haböck, "Failure of proximity gaps close to capacity." [Online]. Available: https://eprint.iacr.org/2026/782

[50] G. Fenzi and A. Sanso, "Small-field hash-based SNARGs are less sound than conjectured." [Online]. Available: https://eprint.iacr.org/2025/2197

[51] A. Chiesa, P. Manohar, and N. Spooner, "Succinct Arguments in the Quantum Random Oracle Model," in *Theory of Cryptography Conference (TCC 2019)*, in LNCS. 2019. [Online]. Available: https://eprint.iacr.org/2019/834

[52] B. Bünz, P. Mishra, W. Nguyen, and W. Wang, "Arc: Accumulation for Reed–Solomon Codes," in *CRYPTO 2025*, 2025. [Online]. Available: https://eprint.iacr.org/2024/1731

[53] B. Bünz, A. Chiesa, G. Fenzi, and W. Wang, "Linear-Time Accumulation Schemes," in *Theory of Cryptography Conference (TCC 2025)*, 2025. [Online]. Available: https://eprint.iacr.org/2025/753

A Artifact and reproducibility

The code, test vectors, transcripts and ledger scripts are public at `github.com/qlyphs/geode`, under the Business Source License 1.1. Every laboratory commit cited below is a tag of that repository, `lab-` followed by the commit identifier (for example `lab-4cc5598`), holding the code

at that commit; its `PROVENANCE.md` maps each identifier to its tag, with the SHA-256 sum of every file. The repository's README gives each command of this section.

The public-u transcripts were produced on laboratory commit `40a46c1` plus the round-8 test code, committed as `3d38ed3`, with rustc 1.96.0 on the M3 Pro [x]. The hidden-u server transcripts come from commits `3e8291b` (first transcripts), `89bd14d` (first suites, Keccak-path comparison, sizes and memory at $N \leq 2048$), `3e5e9a4` (round registries and the runs at 4,096 and 16,384), `a02cb7a` (the Johnson run at 65,536), `68cc4e1` (the unique-decoding run at 65,536), `fea3f28` (the final acceptance suites) and `4cc5598` (registries re-exported from the default build, and the certificate digests), all committed with their results by `0e7a1c9`; each transcript begins with its commit and binary digest [x]. The crate pins every Plonky3 crate at `=0.8.0` and vendors one patched file of `p3-challenger`. The binaries were built with rustc 1.96.0 (`--release --locked`): `r3-cert` without test hooks for every measurement and verification, and `r3-cert-lab` (`--features lab-hooks`) for the acceptance suites and prover comparisons [x]. From the tag `lab-4cc5598`, the main runs are below.

Baseline runs. Repository README, section Reproducing the paper.

```
cd aggregate/r3-cert
cargo build --release --locked
B=target/release/r3-cert
cargo build --release --locked --features lab-hooks \
  --target-dir target-lab
# the r3-cert-lab binary
L=target-lab/release/r3-cert
# self checks
$B selftest; $B h9self; $B r10self
# family J: bytes, phases, memory
H9_FAMILY=johnson $B h9 2048
# family U: registry + honest run
H9_FAMILY=ud $B h9registry 2048 run
# 223/223 (also with H9_FAMILY=ud)
$L h9tests 128
# chain range, N = 65,536, disk-backed arrays
H9_SPILL_DIR=/tmp/spill H9_SPILL_MIN=67108864 \
  H9_FAMILY=johnson $B h9chain 65536
# public-u profile QLYAUTH7
$B chainmeasure v7; $B soundness v7
```

The optimised prover of Section 8.5 is `lab-2aced2d`, which starts from `lab-0e7a1c9`; its prover code is that of `lab-1e5432b`, and later commits change only jobs and documents [x]. Its transcripts are complete at `lab-bc4788b`. The reruns of Section 8.6 build an export of `lab-bc4788b` (`aggregate/cert-speed/export.sh`) in a fresh container; their transcripts are at `lab-8c866e5` (host D) and `lab-3d9332b` (hosts E and F), with each run's full output and resource samples [x]. Each transcript records the SHA-256 digest of the binary it ran; the default build of `2aced2d` had the same digest on both hosts [x]. The identity check builds the two default binaries, runs the same chain statement with each and compares the certificate digests; that the verifier is unchanged is checked on the sources.

Optimised-prover equivalence. See the repository README, "Reproducing the paper". The source check is `aggregate/cert-speed/verifier-unchanged.py`.

```
# in aggregate/r3-cert
# BASE, B: default builds of lab-0e7a1c9 and lab-2aced2d
export H9_FAMILY=ud H9_SPILL_DIR=/tmp/spill \
  H9_SPILL_MIN=67108864 H9_SPILL_SEQ=1
R3_OUT=out/base $BASE h9chain 16384
R3_OUT=out/new $B h9chain 16384
sha256sum out/base/h9chain-cert-16384-fff9130e.bin \
  out/new/h9chain-cert-16384-fff9130e.bin
# sha256 c9fbf852..., as recorded
R3_OUT=out/new $B h9chain 65536
C=out/new/h9chain-cert-65536-fff91310.bin
# verifier of the first prover
$BASE h9chainverify 65536 $C
# 49/49
$B h9chainattacks 65536 $C
cd ../..
python3 aggregate/cert-speed/verifier-unchanged.py \
  lab-0e7a1c9 lab-2aced2d
```

The interval lemma: `cc -O2 -o hint aggregate/hint-check/hint.c && ./hint`. The hidden-u ledger: `python3 aggregate/r10-security/ledger10.py` (tag `lab-cbc32e4`), which reads the round registries and writes the `ledger10.txt` this paper cites. The public-u ledgers: `aggregate/r6-check/ledger.sh` and `aggregate/r8-check/check.sh`. `paper/scripts/bcgm_registered.py` evaluates the BCGM fallback at every registered Johnson schedule with the ledger's own functions, and `paper/scripts/registry_bytes.py` sums the registered round bodies. The security argument itself is Section 11 and Appendix C of this paper.

The in-node runs of Section 8.7 and the sustained runs of Section 8.8 come from laboratory commits `0c1b420`, `846f976`, `070e766`, `d7b2a9b`, `2833ab4`, `cd4f871`, `60b99e1` and `bfde9ff`; their code and evidence are not part of this release [X].

B Certificate layout and round census

Part, in order	J, $N = 2048$	U, $N = 2048$	Parsing rule
manifest	512	512	equals the verifier-derived manifest byte for byte
$\tilde{c}_0 \ldots \tilde{c}_{N-1}$	131,072	131,072	any bytes; must reproduce every accumulator
ν	2,048	2,048	each byte at most 75
commitment root	64	64	any value
round bodies (quotients, GKR, sumchecks, Keccak, links, WHIR scalars)	428,000	427,840	registered length per round; each base-field coefficient of a K element is below p
WHIR openings and framing	866,497	2,163,697	exact length; must re-encode to the same bytes

Table 13: Hidden-u certificate layout at $N = 2048$ [D], from the registries and the measured totals. A K element is 40 bytes.

N	Fam.	Rounds k	q_V, bound	q_V, measured	Merkle depth	$r_{\min}$, bits
128	J	2,785	12,200	10,060	21	1,632
128	U	2,781	28,599	17,927	19	2,368
2,048	J	3,154	17,621	13,722	22	1,792
2,048	U	3,150	32,366	21,643	22	2,368
4,096	J	3,279	19,019	–	23	1,792
4,096	U	3,275	33,913	–	23	2,368
16,384	J	3,486	33,053	–	24	1,632
16,384	U	3,481	41,797	–	25	2,368
65,536	J	3,695	35,791	–	26	1,632
65,536	U	3,690	52,362	–	26	2,368

Table 14: Compiler census from the registries [X] and the ledger [A]. q_V bound: verifier oracle calls without multiproof deduplication; measured: an honest run with deduplication [M]; Merkle depth of the largest tree; $r_{\min}$: the minimum declared length of a complete raw coin block.

C Proofs of the security statements

This appendix writes out the arguments of Section 11 step by step. It changes no bound, parameter, hypothesis or status label: every error below is the one the ledger charges, and every statement keeps the label it has in Table 12. Our internal review has checked these proofs, and Section 12 lists the parts that are also machine-checked.

Notation. K is the challenge field; the ledger charges the frontend rounds over $|K|$ and the WHIR rounds over $B = 2^{319} < |K|$, so a WHIR charge over B is at least the same bound over $|K|$ [X]. $\Lambda = \Lambda(\Pi_1)$ is the list of Lemma 3 and L its size bound: $L = 10 \cdot 2^{r_0}$ in family J (160, 40, 40, 20 and 20 at the five sizes [D]) and $L = 1$ in family U. For a candidate $W \in \Lambda$ we write $X_W, Y_W, S_W, B_{j,W}, T_{g,W}$ and m_W for the columns of its unstacked message and $u_W = \sum_j 2^j B_{j,W}$. Every derived object (the region polynomials, the LogUp leaves and tree, the Keccak input rows in_W, the 72 layer outputs, the masked-output target tgt_W, equal to $T_{g+1,W}$ on slot $g < 8$, to $1 - 2\tilde{c}_i[s]$ on slot 8 and cell $s < 512$, and to 0 elsewhere) is a function of W alone, never an extra witness. We write $\ell_{\mathrm{dc}} = 11 + \log_2 N$ for the variables of the signature cells, $n_k = 15 + \log_2 N$ for those of the Keccak cube (11 cell, 4 slot and $\log_2 N$ signature variables), d_{L} for the depth of the LogUp tree, and ξ, ξ_{out} for the zerocheck and masked-output points, ξ_{link} for the point of the claim on the permutation inputs and ξ' for the point where the input link ends (Section 5).

C.1 Four facts

(F1) Schwartz–Zippel. A non-zero polynomial of total degree D over K vanishes at a uniform point of K^v with probability at most $D/|K|$. This is the standard lemma.

(F2) One sumcheck round. Let a round reduce the claim c, for a fixed candidate whose true value is c^*, with a round polynomial of degree at most D, either in plain form (the prover gives h, the verifier requires $h(0) + h(1) = c$) or in Gruen's normalized form (the prover gives the coefficients $a_1, ..., a_D$ of the cofactor H, and the verifier sets $a_0 = c - x(a_1 + ... + a_D)$, which enforces $(1 - x)H(0) + xH(1) = c$ for the round's coordinate x without dividing by anything). The true polynomial h^* or H^* satisfies the same constraint with c^*. If $c \neq c^*$, every polynomial satisfying the constraint with c differs from the true one, so the two agree at the uniform

challenge r with probability at most $D/|K|$; outside that event the next claim, $h(r)$ or $H(r)$, is false for the candidate. This holds for every $x \in K$, including 0 and 1.

(F3) Batching. If some $\Delta_j \neq 0$, then $\sum_j \beta_j \Delta_j$ with independent uniform β_j vanishes with probability at most $1/|K|$, and $\sum_{j<m} \alpha^j \Delta_j$ with uniform α vanishes with probability at most $(m-1)/|K|$, being a non-zero polynomial of degree at most $m-1$.

(F4) Data-parallel layer identity. Let a layer compute $\mathrm{out}(p,s) = \sum_t \mathrm{sel}_t(s) f_t\big(\mathrm{in}\big(p, \sigma_{t,1}(s)\big), \ldots\big)$ on slots p and cells s, and let the gathered columns be $T_j(p,s) = \mathrm{in}\big(p, \sigma_j(s)\big)$ on the cells where gather j is active and 0 elsewhere. Suppose that a selected term reads only active gathers: if $\mathrm{sel}_t(s) \neq 0$, every gather that f_t reads is active at s. Then for every point $r = \big(r_p, r_s\big)$, $\widetilde{\mathrm{out}}(r) = \sum_{p,s} \mathrm{eq}\big(r_p, p\big) \sum_t \mathrm{eq}(r_s, s) \mathrm{sel}_t(s) f_t(T(p,s))$, which is the definition of the multilinear extension applied to the defining identity on the cube. And for every $r' = \big(r'_p, r'_s\big)$ and weights β_j, $\sum_j \beta_j \tilde{T}_j(r') = \sum_{s'} \tilde{\mathrm{in}}\big(r'_p, s'\big) M(s')$ with $M(s') = \sum_j \beta_j \sum_{s \text{ active}, \sigma_j(s)=s'} \mathrm{eq}(r'_s, s)$: expand $\tilde{T}_j(r') = \sum_p \mathrm{eq}\big(r'_p, p\big) \sum_{s \text{ active}} \mathrm{eq}(r'_s, s)\ \mathrm{in}\big(p, \sigma_j(s)\big)$ and collect the cells by $s' = \sigma_j(s)$; the multilinear extension in p commutes with a gather that acts on s only. Both identities hold for arbitrary field values of in; Booleanity plays no part.

C.2 Stage I: obligations, prover moves and the bound of each round

For a prefix tr of Stage I (from Π_1 to the batching challenge α) let $G(\mathrm{tr})$ be the conjunction of the deterministic checks that tr determines (parsing, canonical encodings, every terminal identity whose inputs are in tr), and let $\mathrm{Ob}(\mathrm{tr}; W)$ be the conjunction of the obligations pending for a candidate W. At Π_1, $\mathrm{Ob} = \mathcal{R}_{\mathrm{alg}}$, the conjunction of (A1) the four lookups as integer ranges with multiplicities m_W, (A2) ternarity, (A3) Booleanity of each plane, (A4) the hint counts, (A5) the modular relation and (A6) the masked-output identity $\mathrm{mask} \cdot \mathrm{out}_W = \mathrm{tgt}_W$ on the Keccak cube. The state is

$$S_{\mathrm{I}}(\mathrm{tr}) = G(\mathrm{tr}) \wedge \exists W \in \Lambda : \mathrm{Ob}(\mathrm{tr}; W).$$

Two rules define Ob along a transcript.

- **(O1) Prover messages.** A message m replaces Ob by an Ob' with $G(\mathrm{tr} \parallel m) \wedge \mathrm{Ob}'(\mathrm{tr} \parallel m; W) \Rightarrow \mathrm{Ob}(\mathrm{tr}; W)$ for every W. A sumcheck round message leaves Ob unchanged. A value message (the quotients $V_{c,j}$, the GKR finals, the leaf pair, the table claim, the eight o_g, the gathered values, the bit claims $U_g(\xi')$, the out-of-domain answer and the 29 opened values) adds the equalities "sent value = the candidate's value" and substitutes the sent values in the claims that depended on them; the old obligation follows by substituting back, together with the terminal check the message enables. For the quotients the new obligation is the field identity $\Phi_{c,j}(W) - qV_{c,j} = 0$ in $\mathbb{F}_p$ with (A1) to (A3) exact; the no-wrap bound $8{,}923{,}771{,}996{,}974{,}608{,}384 < p$ [M], which covers every $V \in [-2^{39}, 2^{39})$ [X], turns it into the integer identity and hence into $\Phi_{c,j}(W) \equiv 0 \pmod q$, the obligation it replaces.
- **(O2) Coin blocks.** A coin block replaces Ob by the reduced obligation of its phase, listed below.

Prover monotonicity. $S_{\mathrm{I}}(\mathrm{tr} \parallel m) \Rightarrow S_{\mathrm{I}}(\mathrm{tr})$: G is a conjunction of absorbing flags, so $G(\mathrm{tr} \parallel m) \Rightarrow G(\mathrm{tr})$, and a witness W of Ob' is a witness of Ob by (O1).

Per-round bound. Let a coin block c of Stage I reduce Ob to Ob_c, and suppose that for every fixed W and fixed prefix, $\Pr_c[\mathrm{Ob}_c(\mathrm{tr} \parallel c; W) \wedge \neg\, \mathrm{Ob}(\mathrm{tr}; W)] \leq b$. Then

$$\Pr_c[\neg S_{\mathrm{I}}(\mathrm{tr}) \wedge S_{\mathrm{I}}(\mathrm{tr} \parallel c)] \le \sum_{W \in \Lambda} \Pr_c[\mathrm{Ob}_c(\mathrm{tr} \parallel c; W) \wedge \neg\, \mathrm{Ob}(\mathrm{tr}; W)] \le Lb.$$

Indeed, $S_{\mathrm{I}}(\mathrm{tr} \parallel c)$ gives $G(\mathrm{tr})$ and some W with Ob_c; with $\neg S_{\mathrm{I}}(\mathrm{tr})$ that W fails Ob. The witness may be chosen after c, but Λ depends on Π_1 alone, which is fixed before every coin (Lemma 3, item 3), so the union over Λ covers it. The decoded coins are within the decoder distance δ_i^{dec} of uniform, which the ledger adds to the round's charge.

The entries marked by a letter need more than F1 to F3.

Coin block	**Obligation before (for W)**	**After**	**Bound b**
Round 1: 14 projection trials	$\mathcal{R}_{\mathrm{alg}}$	(A1)–(A4), (A6) exact; $\Phi_{c,j}(W) \equiv 0 \pmod q$ for every chunk c and trial j	$(2/q - 1/q^2)^{14}$, (a)
Round 2: θ	(A1) as integer ranges; (A2)–(A4), (A6); projected identities as field equalities	rational LogUp identity at θ; the rest unchanged	$(7680N + 2^{20} - 1)/(\lvert K\rvert - p)$, (b)
LogUp GKR: β_0, λ_1; each round; each layer's β_k, λ_{k+1}; last β	layer claim on W's fraction tree	next layer claim	$2, 2, 2, 1$ over $\lvert K\rvert$, (c)
Leaf batch γ_P, γ_Q	$P_{\mathrm{leaf}}, Q_{\mathrm{leaf}}$ equal W's leaf extensions	their γ-combination	$1/\lvert K\rvert$, F3
Round 5: $\gamma_Z, \gamma_{B_j}, \xi, \omega, \mu, \xi_{\mathrm{out}}$	(A2)–(A4), field projections, leaf-link claim, (A6)	master sum claim; masked claim at ξ_{out}	$(\ell_{\mathrm{dc}} + 1 + n_k)/\lvert K\rvert$, (d)
Master round	sum claim	next claim	$8/\lvert K\rvert$, F2
Table round	table claim	next claim	$2/\lvert K\rvert$, F2
Mask round	masked sum claim	next claim; at the end $\widetilde{\mathrm{out}}_W(\xi_1) = v_1$	$2/\lvert K\rvert$, F2
Keccak layer: slot round, cell round, β batch, wiring round	layer output claim	next claim; at the end a claim on the layer input	$\deg f$, $\deg f + 1$, 1, 2 over $\lvert K\rvert$, (e)
Input-link round	$\tilde{\mathrm{in}}_W(\xi_{\mathrm{link}}) - \mathrm{PUB}$ = sum claim	next claim; at the end claims on $\tilde{T}_{g,W}$ and $U_{g,W}$ at ξ'	$3/\lvert K\rvert$, (f)
Bit claims γ_g; bit-wiring round	$U_g(\xi') = \tilde{U}_{g,W}(\xi')$; wiring claim	γ-combination; next claim	$1/\lvert K\rvert$; $2/\lvert K\rvert$, (g)
Initial out-of-domain point (J)	Ob	Ob (unchanged)	0, (h)
Batching α	the m opening claims	$\ell_\alpha(W) = v_\alpha$	$(m-1)/\lvert K\rvert$, F3

Table 15: Stage I, coin block by coin block. The bound is per candidate; the ledger charges L times it [A]. Every backward map is the identity $W \mapsto W$.

(a) **Projections.** If (A1) to (A4) or (A6) fails for W, it fails after the coins too, since these stay exact. Otherwise (A5) fails, and let c^* be the first chunk of 512 signatures that contains a failing signature; c^* depends on W and x only. Let $R \in \mathbb{F}_q^{512\times 2048}$ have row i equal to the residual $A_i z_i - c_i 2^{13} t_{1,i} - a u_i - e_i \bmod q$ of signature i of c^*, so $R \neq 0$. With the coefficient-domain adjoint and the term $-\langle r'_{j,i}, u_i\rangle$, $r' = \mathrm{ctr}(\alpha\eta a \bmod q)$, the projected value of trial j is $\alpha_j^\top R\eta_j \bmod q$ exactly. For R of rank $\rho \ge 1$, $\Pr[R\eta = 0] = q^{-\rho}$ and, when $R\eta \neq 0$, $\Pr[\alpha^\top R\eta = 0] = 1/q$, so one trial passes with probability $q^{-\rho} + (1 - q^{-\rho})/q \le 2/q - 1/q^2$. The 14 trials are one decoded pool of one coin block, with no prover message between them, so they are independent and the product bound $(2/q - 1/q^2)^{14}$ applies [A]; splitting them across rounds would let a prover adapt between

trials, which the registered table forbids. No union over chunks is needed, because the reduced obligation requires every chunk, in particular c^*.

(b) **Lookup.** By Lemma 3 (item 2) every looked-up value is in $\mathbb{F}_p$. If (A1) fails, some occurrence value v is not an entry of the table $\{0, ..., 2^{20}-1\}$: with (A2) and (A3) still exact, $\Delta(u_W, S_W)$ is the integer width of the region (1,048,334 for the X pair), and both members of a pair $(v, \Delta - v)$ are table entries only if $0 \le v \le \Delta$ as integers, since otherwise $\Delta - v$ is, modulo p, at least $p - 2^{20}$. The rational function $\sum_{\text{occ}} 1/(\theta - v) - \sum_t m_t/(\theta - t)$ then has a pole at v with residue the number of occurrences of v, which lies between 1 and $7680N < p$, so it is not zero. Clearing the denominators gives a non-zero polynomial of degree at most $7680N + 2^{20} - 1$; under the ideal coin law θ is uniform in $K \setminus \mathbb{F}_p$, where no denominator vanishes, which gives the bound (the decoder's deviation from that law is in its distance δ_i^{dec}). The paired-leaf form of Section 4 is the same rational function. The multiplicities are columns of Π_1, fixed before θ.

(c) **LogUp GKR.** With every leaf denominator non-zero, the rational identity at θ holds if and only if the root numerator of W's fraction tree is 0, which the root check reads from the sent layer-1 values. The interpolation coin β evaluates the line through two child claims $(P_a + \beta(P_b - P_a))$, which agrees with W's line, if the sent pair is wrong, with probability at most $1/|K|$; the coin λ batches the numerator and denominator claims (F3); each round is F2 with degree 2. Over the shape of the tree (layer k has k rounds, two coins per layer final, one for the last) the degrees sum to $d_{\text{L}}^2 + d_{\text{L}} - 1$ [A]. The table part of the leaf claim is carried as its own claim T, sent before round 5 and reduced by the table sumcheck.

(d) **Round 5.** By construction of the master summand F (Section 5), for a fixed W, $\sum_b F_W(b) - c_{\text{pub}} = \Delta_W + \Gamma_W$, where Δ_W is the error of the leaf-link claim, fixed before the coins, and $\Gamma_W = \gamma_Z \tilde{Z}_W(\xi) + \sum_j \gamma_{B_j} \widetilde{\text{Bool}}_{j,W}(\xi) + \sum_i \omega_i \big(\sum_t S_W^2 - \nu_i\big) + \sum_{c,j} \mu_{c,j} \text{Res}_{c,j}(W)$, with $Z = S^3 - S$, $\text{Bool}_j = B_j^2 - B_j$ and $\text{Res}_{c,j}$ the field residual of the projected identity. Every term of Γ_W carries a fresh coin, and the coins $\gamma_Z, \gamma_{B_j}, \omega_i, \mu_{c,j}$ are distinct variables. If any of the master-summand obligations (ternarity, Booleanity, hint counts, the projected field identities, or the leaf-link claim) fails for W, then $\Delta_W + \Gamma_W$ is non-zero: a false leaf link gives a non-zero constant term; a cell with $S^3 \neq S$ or $B_j^2 \neq B_j$ makes $\tilde{Z}_W$ or $\widetilde{\text{Bool}}_{j,W}$ a non-zero multilinear polynomial in ξ, multiplied by its own coin; with S ternary a false count is a non-zero integer of absolute value at most $2{,}048 < p$, hence a non-zero coefficient of ω_i; a false projected identity is a non-zero field residual by definition, hence a non-zero coefficient of $\mu_{c,j}$. Its total degree is at most $\ell_{\text{dc}} + 1$, so F1 bounds the event that it vanishes by $(\ell_{\text{dc}} + 1)/|K|$. Separately, if (A6) fails, $D_W = \text{mask} \cdot \text{out}_W - \text{tgt}_W$ is non-zero on the Keccak cube and fixed before ξ_{out}, so $\tilde{D}_W(\xi_{\text{out}}) = 0$ with probability at most $n_k/|K|$. The union gives $(\ell_{\text{dc}} + 1 + n_k)/|K|$. The four Booleanity weights must be independent: in $\mathbb{F}_p$ both 5 and -3 are squares, so a single weight would accept two non-bits whose $B^2 - B$ are 1 and -1. After this round the projected obligations are field equalities only; no later step infers an integer identity from them.

(e) **Keccak layer.** By the first identity of F4 the layer output claim is a sum over the layer's cube of $\text{eq}(r_p, p) \sum_t \tilde{w}_t(r_s, s) f_t\big(\tilde{T}_{j,W}(p, s)\big)$. In a slot round the cofactor of the equality factor has degree $\deg f$ (5 for θ-a, 3 otherwise) in the round variable, and the normalized form of F2 applies; in a cell round the weight is multilinear in the round variable, so the degree is $\deg f + 1$. The terminal check `main_verify` is deterministic and leaves the obligations $\tilde{T}_{j,W}(r') = \text{val}_j$ for the sent gathered values; the β batch is F3; by the second identity of F4 the batched claim is a sum over cells of $\widetilde{\text{out}}_W^{(\ell-1)}(r'_p, s') M(s')$, and 11 wiring rounds of degree 2 reduce it to one claim

on the layer input. Per layer this is $(4+\log_2 N)\deg f + 11(\deg f + 1) + 23$, and $264\log_2 N + 6408$ over the 72 layers [A].

(f) **Input link.** in_W is defined pointwise by the formula of the link summand plus the public part (state copy on cells below 1,600, the factor $1-2U_g$ on rate cells, the padding block on slot 8, μ and the zero initial state on slot 0, zero on dummy slots and padding cells), so expanding its multilinear extension over the slot variables turns the claim on $\widetilde{\mathrm{in}}_W(\xi_{\mathrm{link}})$ into the sum claim of the link. The summand has degree 3 in each variable ($W_b \cdot T_g \cdot U_g$). The terminal check uses verifier-computed weights, opening claims on $\tilde{T}_{g,W}(\xi')$ and the sent $U_g(\xi')$, which become obligations.

(g) **Bit wiring.** By the definition of the re-indexed bits and the bijection between their positions and the cells of $B_0, \ldots, B_3$, $\sum_g \gamma_g \tilde{U}_{g,W}(\xi') = \sum_y M(y)\hat{B}_W(y)$ with a public weight M and $\hat{B}_W(c,j) = \tilde{B}_{j,W}(c,\xi'_i)$, multilinear in $y=(c,j)$. A 13-round degree-2 sumcheck reduces it to one opening of the four planes.

(h) **Initial out-of-domain point.** In family J the commitment's out-of-domain point is drawn before its answer. It adds and removes no obligation, so it causes no Stage I transition; its answer is a value message (O1) that adds the claim $\ell_{\mathrm{OOD}(W)} = v_{\mathrm{OOD}}$. The ledger still charges WHIR's $\varepsilon_{\mathrm{out}}$ at this round, which is conservative here.

C.3 Stage II: commitments, folds and the final tail

The state. After the handoff the state is WHIR's phase-local state of the proof of WHIR Theorem 5.2, read on P3′s oracles in the coefficient convention (Section 11.3 explains the transfer to the tables that P3 commits): at step s of phase i (after s of the f_i fold rounds), $S_{i,s}(\mathrm{tr}) = G(\mathrm{tr}) \wedge \left[\exists c \in C_\sigma^{(i,s)} : \Delta_{\mathrm{row}}\big(c, f_{i,s}\big) \le \delta_i\right]$, where $f_{i,s}$ is the current folded oracle, a word whose rows hold $2^{f_i - s}$ lanes, $C_\sigma^{(i,s)}$ is the set of interleaved codewords of the phase whose message $\hat{c}$ meets the current constraint $\sum_b \hat{w}_{i,s}(b)\hat{c}(b) = \sigma_{i,s}$, and Δ_{row} is the fraction of rows on which some lane differs. In family J it also contains WHIR's out-of-domain sub-states. At the handoff it is the handoff predicate itself: by Lemma 3 (item 2) the codewords within δ_0 of Π_1 are exactly the members of Λ, where $\delta_0 = 1 - (21/20)\sqrt{\rho}$ in family J and δ_0 is the strict radius $(1-\rho)/2 - 2^{-60}$ of the proof of Lemma C2 in family U, so $S_{0,0} = G \wedge \exists W \in \Lambda : \ell_\alpha(W) = v_\alpha$, and the stages agree at their boundary.

Commitment half-round. When the prover sends a new WHIR root, the state is the preceding one, conjoined with the checks the root determines. It moves to the next phase only at the verifier's coins: the out-of-domain point and answer and the shift block in family J (WHIR's $\varepsilon_{\mathrm{out}}$ and $\varepsilon_{\mathrm{shift}}$, cited), the shift block in family U (the zero-OOD step of Section 11.5, proven). WHIR's out-of-domain error is at most $2^{m_i}L_i^2/(2|K|)$, and the ledger charges it as $\varepsilon_{\mathrm{out}} = L_i^2 2^{m_i}/(2B)$ at each out-of-domain point of family J [X]. A commitment is a prover move under which the state cannot turn from false to true.

Lemma C1 (one agreement set through every fold). proven, given H4 or H5. Fix phase i and fold step s, and let $f = f_{i,s}$ have $2^{f_i - s}$ lanes per row, paired as (a_p, b_p) by the variable being folded. There is a set E of fold challenges, with $|E| \,/\, |K| \le 2^{f_i - s - 1}\varepsilon^*$, where ε^* is the mutual-correlated-agreement error of one lane code $\mathrm{RS}[K, H_i, 2^{m_i - f_i}]$ at radius δ_i, such that for every $r \notin E$: if the folded word f_r, with lanes $a_p + r(b_p - a_p)$, agrees on the whole rows of a set S with $|S| \ge (1 - \delta_i)n_i$ with an interleaved codeword c', then some interleaved codeword c agrees with f on the whole rows of the same S, and c' is the fold of c by r.

Proof. For each pair p apply mutual correlated agreement (WHIR Definition 4.9, provided by H4 in family J and H5 in family U) to the line generator and the fixed pair of words $(a_p, b_p - a_p)$: outside a set E_p of r of density at most ε^*, every set S with $|S| \ge (1 - \delta_i)n_i$ on which $a_p + r(b_p - a_p)$ agrees with a codeword is a correlated agreement set, that is, there are codewords $c_{0,p}, c_{1,p}$ with $c_{0,p} = a_p$ and $c_{1,p} = b_p - a_p$ on S. Let $E = \bigcup_p E_p$. For $r \notin E$, whole-row agreement of f_r with c' on S means that every folded lane agrees with c'_p on S, so the same S serves every pair. Then $(c_{0,p}, c_{0,p} + c_{1,p})$ agrees with (a_p, b_p) on S for every p, and concatenating these lanes gives an interleaved codeword c agreeing with f on the whole rows of S. Its fold $c_{0,p} + rc_{1,p}$ agrees with c'_p on S, and two codewords of degree below $2^{m_i - f_i} < |S|$ that agree on S are equal. The form $(a, b - a) \mapsto a + r(b - a)$ needs no division, so it covers every r, including $r = 1$. There are $2^{f_i - s - 1}$ pairs. □

The union fits inside the ledger's fold charge. The ledger charges $2C(m_i, r_i)/B$ per fold, with $C(m, r) = 8 \cdot 2^{m+2r} \cdot 10.5^3/(3(1 - 2^{-m}))$ in family J and $C(m, r) = 2^{m+r}$ in family U [X], and $2L_i/B$ for the fold's sumcheck polynomial [A]. A lane code has dimension $2^{m_i - f_i}$ at the same rate, so $2^{f_i - s - 1}\varepsilon^* \le 2^{f_i - 1}C(m_i - f_i, r_i)/|K|$, which is $C(m_i, r_i)/(2|K|)$ in family U and at most $C(m_i, r_i)(1 - 2^{-m_i})/\big(2(1 - 2^{f_i - m_i})\,|K|\big) \le (4/7)C(m_i, r_i)/|K|$ in family J, because $m_i - f_i$ is at least the number of final variables, 3 or more in every registered schedule [D]. With the BCGM constant, whose error grows with the square of the code length, the lane union is $2^{-f_i - 1}$ times the single-code term. That constant is $C(m, r) = 10.5^7 n^2/\big(3\rho^{3/2}\big)$ with $n = 2^{m+r}$ and $\rho = 2^{-r}$, the count of BCGM at tradeoff $m' = 10$ for a generator of degree 1 [X]. With Lemma C1, WHIR's fold argument (an exceptional challenge is charged by the MCA term, a constraint error by the sumcheck term $2L_i/B$) applies to P3′s interleaved oracles with whole-row agreement, which is the granularity of a Merkle leaf and of a WHIR query. At $s = 0$ of phase 0 the lanes are base-field words, which are in particular words over K. In family J, the phase-local list bound is $L_i = 10 \cdot 2^{r_i}$, obtained from the Johnson bound over the auxiliary extension $J_i = K[\zeta]/(\varphi_i)$ of degree 2^{f_i}, read as in Lemma 3, which holds over every field. In family U, unique decoding gives $L_i = 1$.

The fold bound (proven). The term $2L_i/B$ is sharper than the error WHIR Theorem 5.2 states for its later folds, so we prove it here, as the argument of that theorem specialised to the degree-two interleaved constraints of one fold. Fix the prefix before the fold challenge r, with the state false, and let $\hat{h}$ be the prover's round polynomial, which satisfies $\hat{h}(0) + \hat{h}(1) = \sigma$. For a member c of the phase list, the round polynomial $h_c(X) = \sum_{b'} \hat{w}(X, b')\hat{c}(X, b')$ of its constraint has degree at most two, and $h_c(0) + h_c(1) \ne \sigma$ because the state is false, so $h_c - \hat{h}$ is a non-zero polynomial of degree at most two. If the state is true after the fold and $r \notin E$, Lemma C1 makes the new codeword the fold of a member c of the list, and the new constraint reads $h_c(r) = \hat{h}(r)$.

The list and $\hat{h}$ are fixed before r, so a union over the at most L_i members gives $2L_i/\,|K| \le 2L_i/B$, and the event $r \in E$ is the MCA term.

Lemma C2 (final tail). proven, given H7. Let the last phase M end with the folded oracle f_M, a word of one lane per row in $\mathrm{RS}[K, H_M, 2^{v_{\mathrm{fin}}}]$, and the claim σ. The prover sends the final coefficient vector p ($2^{v_{\mathrm{fin}}}$ coefficients, 32 at $N = 2048$), the verifier draws t_M query indices and checks at each that the fold of the opened row equals the encoding of p there, and then v_{fin} registered degree-2 sumcheck rounds prove $\sum_{b\in\{0,1\}^{v_{\mathrm{fin}}}} \hat{w}(b)\hat{p}(b) = \sigma$, where $\hat{w}$ is the verifier's combined constraint weight, multilinear in the remaining variables, and $\hat{p}$ the multilinear polynomial of p [A]. Define the state after the query block as $G \wedge \left[\sum_b \hat{w}(b)\hat{p}(b) = \sigma\right]$, and after each tail round j as $G \wedge \left[\sum_{b'} \hat{w}(\rho_1, ..., \rho_j, b')\hat{p}(\rho_1, ..., \rho_j, b') = h_j(\rho_j)\right]$. Then the query block turns a false state into a true one with probability at most $(1-\delta_M)^{t_M}$, each tail round with probability at most $2/|K|$, and on the full transcript the state is acceptance.

Proof. The message p is a prover move; the state before the query block is the retained $S_{M,f_M} = G \wedge [\exists c \in C_\sigma : \Delta(c, f_M) \le \delta_M]$. Suppose it is false and the state after the query block is true. Then G holds, so every queried row folds to the value of $\mathrm{enc}(p)$, and $\sum_b \hat{w}(b)\hat{p}(b) = \sigma$, so $\mathrm{enc}(p) \in C_\sigma$. Since the state before was false, $\Delta(\mathrm{enc}(p), f_M) > \delta_M$, so a uniform index lands on a row where the fold of f_M equals $\mathrm{enc}(p)$ with probability below $1-\delta_M$, and all t_M independent indices do so with probability at most $(1-\delta_M)^{t_M}$; G requires them all. The ledger charges this as $\mathrm{agree}_M^{t_M}$, with agreement $(21/20)2^{-r_M/2}$ in family J and $(1+2^{-r_M})/2$ in family U, where the strict radius $\delta = (1-\rho)/2 - 2^{-60}$ makes a δ-far word disagree on at least $(n-d)/2$ rows [X]. In the tail, $\hat{w}\hat{p}$ has degree 2 in each variable and both factors are known to the verifier, so the true round polynomial h_j^* is defined, and a false claim survives a round only on the F2 event, $2/|K| \le 2/B$ per round, the ledger's $2v_{\mathrm{fin}}/B$. After the last round the state is $G \wedge \left[\hat{w}(\rho)\hat{p}(\rho) = h_{v_{\mathrm{fin}}}\left(\rho_{v_{\mathrm{fin}}}\right)\right]$, which is the verifier's final check; together with the absorbing flags it is acceptance. Acceptance is reached through these reductions, not assigned by a terminal override. □

C.4 Proof of Theorem 1

We use the conditions of relaxed round-by-round knowledge soundness of [18] (Definitions 3.5 and 3.6) in this form: a state $\mathcal{K}(\mathrm{tr}, \boldsymbol{\sigma})$ and an extractor $\mathcal{E}$ such that

(K1) $\mathcal{K}(\varnothing, \boldsymbol{\sigma}) = 1$ exactly when $(x, \boldsymbol{\sigma}) \in \mathcal{R}_{\mathrm{auth}}$;

(K2) a prover message never turns $\mathcal{K}$ from 0 to 1;

(K3) on a full transcript $\mathcal{K} = 1$ exactly when the verifier accepts, the output relation being trivial;

(K4) for every round i and every prefix tr ending before the coins c of round i,

$$\Pr_c[\exists \boldsymbol{\sigma} : \neg\mathcal{K}(\mathrm{tr}, \mathcal{E}(\mathrm{tr} \parallel c, \boldsymbol{\sigma})) \wedge \mathcal{K}(\mathrm{tr} \parallel c, \boldsymbol{\sigma})] \le \kappa_i;$$

(K5) $\mathcal{E}$ is deterministic and polynomial time.

We argue for deterministic provers; the bounds extend to randomized provers by conditioning on the prover's random tape.

The state. $\mathcal{K}(\varnothing, \boldsymbol{\sigma}) = \mathcal{R}_{\mathrm{auth}}(x, \boldsymbol{\sigma})$; $\mathcal{K}(\Pi_1, \boldsymbol{\sigma}) = \mathcal{R}_{\mathrm{auth}}(x, \boldsymbol{\sigma}) \wedge S_{\mathrm{I}}(\Pi_1)$; for every prefix after the first coin block, $\mathcal{K}(\mathrm{tr}, \boldsymbol{\sigma}) = S(\mathrm{tr})$, the two-stage state: S_{I} up to the batching coin, $S_{i,s}$ with the commitment half-round rule through the WHIR phases, and the states of Lemma C2 in the

final tail. The extractor is $\mathcal{E}(\mathrm{tr} \parallel c_1, \boldsymbol{\sigma}) = D(x, \Pi_1)$ for the first coin block and $\boldsymbol{\sigma}$ afterwards, where D is the extractor of Lemma 4.

S-init. $S_{\mathrm{I}}(\Pi_1) \Rightarrow \mathcal{R}_{\mathrm{auth}}(x, D(x, \Pi_1))$. Indeed some $W \in \Lambda$ satisfies $\mathcal{R}_{\mathrm{alg}}(x, W)$, so by Lemma 4 the signatures built from W satisfy $\mathcal{R}_{\mathrm{auth}}$; in particular they can be packed canonically, so D does not skip W, and D returns the first candidate in its fixed order whose signatures pass native verification, which exists.

(K1) holds by definition.

(K2). From $\varnothing$ to Π_1, $\mathcal{K}(\Pi_1, \boldsymbol{\sigma}) \leq \mathcal{R}_{\mathrm{auth}}(x, \boldsymbol{\sigma})$; in Stage I by prover monotonicity (Section C.2); in Stage II by the prover monotonicity of WHIR's state (H7), by the commitment half-round rule, and in the final tail because the final vector keeps the state and a tail message only extends G.

(K3) is Lemma C2.

(K5). Efficiency requires the two implementation premises stated in Section 12: a polynomial-cost exact candidate decoder and polynomial-cost construction and verification of one candidate batch. The proved extractor-wrapper composition then yields a deterministic polynomial-time extractor. The size measure includes the public messages. Section C.6 counts parts of this computation; those counts do not themselves establish the remaining implementation premises or termination of the unbounded FIPS samplers.

(K4), first coin block. $\mathcal{K}(\Pi_1 \parallel c_1, \boldsymbol{\sigma}) = S(\Pi_1 \parallel c_1)$ does not depend on $\boldsymbol{\sigma}$, and $\neg\mathcal{K}(\Pi_1, D(x, \Pi_1)) = \neg(\mathcal{R}_{\mathrm{auth}}(x, D(x, \Pi_1)) \wedge S_{\mathrm{I}}(\Pi_1))$, which by S-init equals $\neg S_{\mathrm{I}}(\Pi_1)$. The event is therefore $\neg S_{\mathrm{I}}(\Pi_1) \wedge S_{\mathrm{I}}(\Pi_1 \parallel c_1)$, bounded by $L(2/q - 1/q^2)^{14}$ (Table 15, row 1) plus the round's decoder distance: this is κ_1.

(K4), every later coin block. $\mathcal{K}$ ignores $\boldsymbol{\sigma}$ and $\mathcal{E}$ returns $\boldsymbol{\sigma}$, so the event is $\neg S(\mathrm{tr}) \wedge S(\mathrm{tr} \parallel c)$. In Stage I it is bounded by Lb with the b of Table 15; at the batching coin by $(m-1)L/|K|$ with $m = 30$ (J) or 29 (U); in Stage II by the fold bound of Section C.3 (the argument of WHIR Theorem 5.2 specialised to degree-two interleaved constraints, with Lemma C1; proven), by WHIR's out-of-domain and shift errors in family J (cited) and the zero-OOD shift step in family U, and by Lemma C2 for the final block and tail. Each of these is bounded by the ledger's κ_i minus its decoder distance, so with the distance added the bound is at most κ_i. The bound holds for every fixed prefix, including a prefix that already contains a bad transition: Stage I quantifies over the fixed list, and every Stage II state is evaluated on the current oracle. The withdrawn single-list state failed exactly here.

Concatenation and recovery. The states agree at the two boundaries (the handoff and the start of the tail), so the error vector of the protocol is the concatenation of the stage vectors, the entries of the ledger, with no factor for the number of rounds. Along an accepting run in which no charged bad transition occurs, the state never changes from false to true (not at a coin, by assumption, and not at a prover message, by K2) and it is true at the end, so it is true at every prefix: at the handoff, before α, and at Π_1, where S-init yields signatures. This is the backward recovery of Section 11.5; it is a property of runs, not an invariant of prefixes that follow a bad transition. □

C.5 Proof of Lemma 5

In the state-restoration game of [17] (Definition 9.3) the prover may extend any prefix it has seen. The coins of round i are a function rnd_i of the full prefix of the interactive protocol, the instance, the prover's messages and oracle strings up to round i, and the salts (fixed to zero); the game samples rnd_i on a prefix the first time that prefix is queried. Merkle roots play no part here; they replace the oracles only in the correspondence of Theorem 2.

Let the prover make t coin queries and output a complete transcript tr^* that the verifier accepts. Let P be the set of prefixes whose coins were sampled, either by a query or by the final verification, which recomputes the coins of the at most k prefixes of tr^*; $|P| \le t + k$. For $\tau \in P$ ending before the coins of round i, let $\mathrm{Bad}(\tau)$ be the event $\neg S(\tau) \wedge S(\tau \parallel \mathrm{rnd}_i(\tau))$.

Accept and extraction failure imply a bad prefix. Suppose $D(x, \Pi_1^*)$ is not in $\mathcal{R}_{\mathrm{auth}}$. By S-init, $S_{\mathrm{I}}(\Pi_1^*) = 0$. The verifier accepts, so the state of the full transcript is 1 by (K3). Walk along the prefixes of tr^* and take the first at which the state becomes 1. It is not a prover message, by (K2), so it is a coin block: a prefix τ of tr^* with $S(\tau) = 0$ and $S(\tau \parallel \mathrm{rnd}_i(\tau)) = 1$. Its coins were recomputed by the verification, so $\tau \in P$ and $\mathrm{Bad}(\tau)$ holds.

Each prefix is bad with probability at most $\varepsilon_{\max}$. Order the prefixes of P by the time their coins were first sampled. When the coins of the j-th prefix τ_j are sampled, τ_j is determined by everything before, and $\mathrm{rnd}_i(\tau_j)$ is a fresh uniform block, independent of it. By (K4), which holds for every fixed prefix, $\Pr\big[\mathrm{Bad}(\tau_j) \mid \mathrm{history}\big] \le \kappa_i \le \varepsilon_{\max}$, where κ_i already contains the decoder distance of the round. Taking expectations and a union over $j \le t + k$, the probability that the verifier accepts while D fails is at most $(t + k)\varepsilon_{\max}$. □

C.6 The extractor and its cost

The extractor D of Lemma 4 runs on Π_1 alone: it decodes, unstacks each candidate (a fixed permutation of cells), builds the signatures of Lemma 4 for each candidate in a fixed order, skips a candidate that cannot be packed canonically (z out of range, a non-ternary S, more hints than fit), runs native FIPS 204 verification with the count and digest checks, and returns the first candidate that passes, or $\perp$. Its parts are counted as follows. The first oracle has $n_0 = 2^{\mathrm{nv}\,+r_0-6}$ rows, the number of leaves of the largest tree in the census of Appendix B: $2^{21}, 2^{22}, 2^{23}, 2^{24}, 2^{26}$ in family J and $2^{19}, 2^{22}, 2^{23}, 2^{25}, 2^{26}$ in family U at the five sizes [D]; the code has dimension $d = 2^{\mathrm{nv}\,-6}$ over J, a field of p^{64} elements.

- **Native verification.** At most LN verifications: 20,480, 81,920, 163,840, 327,680 and 1,310,720 in family J, and N in family U [D]. At the median of 70 µs per verification measured in Section 8 [D] this is about 1.4, 5.7, 11.5, 23 and 92 s in family J, and 4.6 s at $N = 65{,}536$ in family U.
- **Family U, unique decoding.** Berlekamp–Welch at the strict radius corrects up to $e = (n_0 - d)/2 - 1$ errors: it solves $Q(x_j) = y_j E(x_j)$ for E monic of degree e and Q of degree below $e + d$, $n_0 - 2$ unknowns in n_0 equations, by Gaussian elimination, divides Q by E, and checks that the quotient is within the radius (otherwise the list is empty). Elimination takes fewer than n_0^3 multiplications and as many additions in J, and at most n_0 inversions; back substitution, the division and the distance check take fewer than $3n_0^2$ more; building the system takes fewer than $65n_0^2$ multiplications in $\mathbb{F}_p$. One multiplication in J costs at most $2 \cdot 64^2$ multiplications in $\mathbb{F}_p$ (schoolbook product and reduction modulo φ). So the decoder costs fewer than $n_0^3 + 3n_0^2$ multiplications in J. That bound is about 2^{57}, 2^{66}, 2^{69}, 2^{75} and

2^{78} at the five sizes [D], approximations of it rather than upper bounds, and at the largest size about 2^{91} multiplications in $\mathbb{F}_p$. This is a bound for the plain algorithm; faster deterministic decoders exist, and we do not state their constants.

- **Family J, list decoding.** The decoder of Chatterjee, Harsha and Kumar runs in time $\mathrm{poly}(n_0, \log|J|)$ above the Johnson agreement over every finite field ([23], Theorem 1.1), with $\log_2|J| < 4096$ here. It returns every codeword above the Johnson agreement $\sqrt{(d-1)n_0}$, a list that the theorem does not bound by L; the extractor keeps the candidates with agreement at least $(21/20)\sqrt{dn_0}$, and by the Johnson bound (H6) at most $L = 10/\rho$ of them remain. The theorem does not state the exponent of this polynomial, so we cannot give an explicit constant for this part of the extractor (open).

The cited theorem supplies an asymptotic algorithmic result for decoding. Connecting an implementation to the exact candidate list and the stated cost model remains an explicit premise, as does a polynomial-cost implementation of candidate construction and verification. The efficiency conclusion of Theorem 1, and its use in Theorems 2 and 3, is conditional on these premises. Concrete implementation constants and the authorization reduction of Section 11.11 remain separate questions.

C.7 Lemma 3 and Lemma 4 in full

Lemma 3. *(1) Decoder and list bound.* For lane polynomials $g_a \in \mathbb{F}_p[X]$ of degree below d, the polynomial $G = \sum_a \zeta^a g_a \in J[X]$ has degree below d and $G(\omega^j) = \sum_a \zeta^a g_a(\omega^j)$, because $\omega^j \in \mathbb{F}_p$. The map $(g_a)_a \mapsto G$ is an $\mathbb{F}_p$-linear bijection, since $1, \zeta, ..., \zeta^{63}$ is a basis of J over $\mathbb{F}_p$, so honest rows are the codewords of one Reed–Solomon code over J with evaluation points in $\mathbb{F}_p$, and a row agrees with a codeword in all 64 lanes exactly when the symbol y_j agrees. In family J the decoder of [23] returns every codeword with agreement above $\sqrt{(d-1)n_0}$, which is below the agreement $(21/20)\sqrt{dn_0}$ used here; the extractor keeps those that reach $(21/20)\sqrt{dn_0}$, and by the Johnson bound, which holds over every field (WHIR Theorem 4.3, stated for a smooth evaluation domain; the evaluation points of the first oracle form a two-adic, hence smooth, multiplicative subgroup of $\mathbb{F}_p$), there are at most $1/(2\eta\sqrt{\rho}) = 10/\rho$ of them for $\eta = \sqrt{\rho}/20$. In family U, Berlekamp–Welch returns the at most one codeword below half the distance. Expanding each G in the basis ζ^a gives the lanes, and the inverse of the suffix layout gives every cell of tables A and B; there is no product over independently decoded lanes.

(2) Recovery over the base field. Suppose $(G'_a)_a$ with $G'_a \in K[X]$ of degree below d agrees with Π_1 on the whole rows of a set S. In family J $|S| \geq (21/20)\sqrt{\rho}n_0 > \rho n_0 = d$, in family U $|S| > (1+\rho)n_0/2 > d$. Write each G'_a in an $\mathbb{F}_p$-basis $1, T, ..., T^4$ of K: the four non-base coordinate polynomials vanish on S, since the rows are in $\mathbb{F}_p$, and have degree below $d < |S|$, so they are zero. Hence the K-list and the J-list are the same set of base-field candidates.

(3) List fixed before challenges. Λ is a function of Π_1, which is sent before the first coin.

(4) Batched openings. This is F3 for each fixed candidate, with the union over Λ: all m values and the five points, and in family J the out-of-domain point, are fixed before α, the values in the body of the opening round and the points by earlier coins.

Lemma 4. Fix signature i and a candidate w satisfying $\mathcal{R}_{\mathrm{alg}}$.

1. **Bits.** By Booleanity each $B_{j,w}$ is 0 or 1, so $u_t = \sum_j 2^j B_{j,w}$ is an integer in $[0, 15]$, $d_0 = \prod_j (1 - B_j) = [u = 0]$ and $d_{15} = \prod_j B_j = [u = 15]$; the region polynomials of the master summand are then those of Table 1.

2. **Sponge.** Let $U_i = \text{w1Encode}(u_i)$, 1,024 bytes, and μ_i the public 64-byte message representative; $\mu_i \parallel U_i$ is 1,088 bytes, eight blocks of the SHAKE256 rate of 136 bytes, and the ninth block is the padding block. Each 136-byte block is extended by the 512 zero bits of the capacity before it is XORed into the 1,600-bit state; block_g below denotes that extension. We show by induction on $g = 0, ..., 8$ that the slot-g input row $\text{in}_w(\cdot, g, i)$ is valid (each of the 1,600 state cells is ± 1, each padding cell 0) and encodes $S_g \oplus \text{block}_g$, where $S_0 = 0$ and $S_{g+1} = \text{Keccak-f}\big(S_g \oplus \text{block}_g\big)$. For $g = 0$ the row is enc of the bits of μ_i on cells below 512, $1 - 2U_0$ on cells 512 to 1,087, 1 on the capacity and 0 on the padding: $\text{enc}(0 \oplus \text{block}_0)$. For the step, the layers map a valid row to $\text{enc}(\text{Keccak-f}(\cdot))$ with zero padding output. The algebraic part is proved: a product of encodings is the encoding of the XOR (θ-a, θ-b), and $A(1 - (1 + B)(1 - C)/2) = \text{enc}(a_0 \oplus (\neg a_1 \wedge a_2))$ for $A, B, C = \text{enc}(a_0), \text{enc}(a_1), \text{enc}(a_2)$ (χ), with the ι sign the round-constant bit. The index part (the gathers of θ, the ρ offsets, the π permutation and the χ neighbours) is a finite table. The formal algebraic and sponge result is proved in Lean, and the generated implementation-table dump is proved to agree with the formal FIPS 202 tables; generation of that dump from the implementation remains a reviewed external step. The masked output on slot $g < 8$ gives $T_{g+1,w}(\cdot, i) = \text{out}_w(\cdot, g, i)$ on all 2,048 cells, so $T_{g+1,w}$ is valid and encodes S_{g+1}. The input-link identity, pointwise for valid T and Boolean U, then gives $\text{in}_w(\cdot, g + 1, i) = \text{enc}\big(S_{g+1} \oplus \text{block}_{g+1}\big)$, with the padding block for $g + 1 = 8$.
3. **Challenge.** The masked output on slot 8 and cells $s < 512$ gives $1 - 2\tilde{c}_i[s] = \text{enc}(S_9[s])$, so $\tilde{c}_i$ is the first 64 bytes of S_9, which is $\text{SHAKE256}(\mu_i \parallel U_i, 512)$, the output of 512 bits (64 bytes). The bits used here are the committed planes $B_{j,w}$, the same columns the arithmetic uses: one candidate, one u.
4. **Bounded relation.** The lookups give $0 \le X \le 1048334$, so $|z| \le 524167 < \gamma_1 - \beta$, and $0 \le Y \le \Delta(u, S)$, so with ternarity and step 1, $e = Y + \text{start}(u, S)$ lies in region S of u. The modular relation gives $Az - c2^{13}t_1 \equiv au + e \pmod q$, and by Lemma 1, $\text{UseHint}(S^2, Az - c2^{13}t_1) = u$ coefficientwise. FIPS 204 verification therefore recomputes $w'_1 = u$ and accepts the digest by step 3.
5. **Hints and encoding.** The counts give $\sum_t S_t^2 = \nu_i$, an integer since it is at most 2,048; the verifier checked $\nu_i \le 75$ when parsing. $\text{Pack}(z)$ is the canonical packing of $z \in (-\gamma_1, \gamma_1]$, and the hint encoding writes the sorted hint positions per polynomial with cumulative counts and zero padding. The signature $(\tilde{c}_i, \text{Pack}(z_i), \text{HintEnc}(S_i^2))$ is canonical and passes every check of FIPS 204 verification: the norm bound, the hint count and the digest equality. □

C.8 Theorems 2 and 3: the specialisations

Theorem 2. With no simulator queries, Lemma 9.12 of [17] gives $\kappa \le E[\kappa_{\text{sr}}(t) + \kappa_{\text{MT}}(t)]$ with $t = R + Eq_V$, and Lemma 7.15 gives $\kappa_{\text{MT}}(t) = (3/2t^2 + 3k(\log_2 \ell_{\max} + 1)t)/2^\lambda$ with $\lambda = 512$. Lemma 5 gives $\kappa_{\text{sr}}(t) \le (t + k)\varepsilon_{\max}$. Both terms increase with t, so replacing t by $R + E(q_V + k - n_{\text{real}})$, which counts the singleton hashes twice, and $\varepsilon_{\max}$ by $\varepsilon_{\max} + \delta^{\text{dec}}$, which counts the decoder distance twice, gives the displayed κ_{cl} as an upper bound. The extractor that Lemma 9.12 needs is straightline and reads the first oracle only: D. For the tagged node hashing, the extractor of each tree parses only the node and leaf queries carrying that tree's context, and the set of child digests of well-formed node queries has at most twice as many elements as the query database, the count the proof of Lemma 7.15 uses; each query carries one context, so no factor for the number of trees appears, and a collision of the tagged function is a collision of the random oracle on distinct inputs. The constants are unchanged.

Theorem 3. Theorem 6.10 of [18] gives the state-restoration error $80(t+k+1)(w+k)\kappa_{\mathrm{RRBR}} + (t+k+1)k/2^{r_{\min}}$ with $\kappa_{\mathrm{RRBR}} = \varepsilon_{\max}$ (Theorem 1) and $w \le t$. Theorem 11.3 multiplies it by 4 and adds the vector-commitment terms of Theorem 8.3 at $\sigma = 512$: $4\kappa_{\mathrm{extract}} = 0$ because $n_y = 0$ here (Remark 11.5), $8\kappa_{\mathrm{offline}}(2t+1) \le 8(160t(2t+1)^2 + 16q_\Pi h)/2^{512}$, $4\kappa_{\mathrm{online}}(T) \le 4 \cdot 240T^3/2^{512}$ with $T = t + 1 + q_V + k$, and $2q_V^2\kappa_{\mathrm{com}}(T) \le 2q_V^2 \cdot 240(T+1)/2^{512}$, where $w_{\mathrm{VC}} \le t$, $w \le T$, and $h = \log_2 \ell_{\max} + 1 \ge \log_2 \ell_{\max}$. Collecting the terms gives the displayed $\kappa_Q(t)$ exactly [A].

Choose an independent uniform index j^* among E designated submission slots, padding unused slots by rejection, and stop after verifying slot j^*. Earlier verifications are charged to $t = Q + E(q_V + k - n_{\mathrm{real}})$. The CDHZ theorem bounds extraction failure for this one selected submission by $\kappa_Q(t)$. We report $E\kappa_Q(t)$ as a conservative allowance for a subsequent reduction that loses a factor E by selecting an externally defined successful submission. This does not define, or prove a bound for, simultaneous extraction failure over all E outputs; the authorization reduction remains open (Section 11.11). The hypotheses of Theorem 11.3 and Construction 11.7 are those checked in Section 11.7.

Scope of the appendix. The transition arguments and numerical specialisations are separated from the implementation premises needed for extractor efficiency. The initial family-J out-of-domain charge is conservative. Section C.6 supplies an explicit family-U decoder multiplication count; the full extractor's cost-model result and its remaining implementation premises are stated in Section 12.

Revision history

Version 2, 8 November 2026; first release: 3 November 2026.

New. Section 12 records the checked algebraic and codec results, the registered interactive theorem's K1–K4 conditions, partial-oracle completion, ledger and transcript results, the conditional extractor-cost construction, and the FIPS sampler instances. The formal link between the interactive predicate and the ledger is proved within an abstract compiler interface. Remaining obligations include the two cost implementations, concrete compiled security games and their compiler correspondence, and correspondence with the Rust implementation.

Corrections.

- An honest run can fail through any strict coin pool, not only the projection pool. In the experiment with independent uniform raw coin blocks, the probability that at least one strict pool is exhausted is at most the sum of the registered per-round decoder allowances, at most $2^{-572.75}$ for every registered profile [A]; the 2^{-768} bound covers only the projection pool, and perfect completeness is not claimed (Section 6).
- The chain-range profiles' numerical budgets transfer under A-CHAIN; applying the security theorems additionally requires their implementation correspondence (Section 7).
- Lemma 3 (2) now carries its agreement radius and is stated for the K-valued lane candidates.
- With higher query targets, the family-J quantum figure is about 117.1 at $N = 65{,}536$ [A], not 119.1.
- Theorem 1 separates its checked K1–K4 conclusion from K5, whose cost-model derivation retains two implementation premises; Theorems 2 and 3 carry the same premises.
- The verifier's matrix is the coefficient-domain form $A = \mathrm{NTT}^{-1}\big(\hat{A}\big)$ of the NTT-domain matrix $\hat{A} = \mathrm{ExpandA}(\rho)$ (Section 2.1).
- Lemma 1 and the generated Keccak gather-table dump of Lemma 4 are proved in Lean; the exhaustive enumeration and the table comparison remain as independent tests.
- Appendix C: the family-J list decoder is followed by a radius filter, which is what bounds the list by L; each round's error is bounded by the ledger's charge rather than equal to it; the fold bound is proved rather than attributed to WHIR Theorem 5.2; the round-5 argument names the obligations it covers, with (A6) treated separately; the quantum factor E is a conservative allowance for one selected output, not a bound on simultaneous extraction.
- In Section 11.11, the factor E is on the unforgeability term, and the binding term covers the whole E-submission game.
- The master summand has degree exactly 8 only when $\gamma_Q \neq 0$ (Section 5).
- The lookup challenge θ is decoded with a fallback rather than by exact rejection; the fallback's deviation from the ideal law is part of the coin-decoder distance.
- The layer identity F4 of Appendix C needs its active-gather premise.
- Coin-input framing is proved injective on canonical statements whose messages are shorter than 2^{32} bytes. The implementation does not yet enforce that domain restriction (Section 6).

Clarifications throughout, among them the WHIR convention of the analysis and its transfer to the tables the library commits (Section 11.3).