Toward Efficient Post-Quantum Bitcoin Bridges

Stephen Duan*[1, 2], Vanhger Pham[1], Blake Hou[1], Ethan Zhu[2], and Jingping Yin[1]

[1]ZKM Team
[2]GOAT Network

Abstract

BitVM-style protocols verify succinct proofs on Bitcoin by garbling a proof verifier off-chain. What reaches the chain in a dispute is the proof itself: its bits under one-time keys, the scripts that authenticate them and the signatures that bind them to transactions. Existing designs garble elliptic-curve succinct non-interactive argument of knowledge (SNARK) verifiers, which a quantum adversary breaks. Our goal is a post-quantum alternative: a hash-based scalable transparent argument of knowledge (STARK) verifier, garbled and fed with hash-based one-time keys, with no number-theoretic assumption anywhere in the verifier; its price is a far larger proof, and this paper measures and reduces that on-chain data. We compile the complete verifier for WHIR-based STARK proofs from Ziren into a Boolean circuit using free-XOR gadgets for binary-tower arithmetic and Blake3. Our implementation uses Ziren's binary-field proof stage and narrow recursion. For an Ethereum block proved through four recursion levels, the final recursive WHIR-proof verifier has 175.4M non-free gates and 520,832 input bits, corresponding to 2.81 GB of garbled tables, and publishing that input with translated Winternitz keys takes 3.93 million virtual bytes (MvB), in 41 standard transactions. The costs are coupled: on-chain cost grows linearly with the input bits, off-chain storage with the gate count, and proving time exponentially with the grinding that shortens the input. The WHIR schedule, the Merkle cap depth and the input encoding trade these costs against one another, and a model of proof size and proving time selects the setting, a step toward practical post-quantum Bitcoin bridges.

Keywords: Bitcoin bridges; post-quantum security; garbled circuits; STARKs; WHIR; zkVM.

1 Introduction

Bitcoin cannot verify a succinct proof economically within current Script and policy limits, but a trust-minimized bridge needs such verification. Consider a withdrawal from a rollup: an operator has paid out a user on Bitcoin and must prove, to a Bitcoin script, that the rollup's state authorizes the payment; the proof is a STARK over the rollup's execution. Our system verifies off-chain computation on Bitcoin by checking its proof in a garbled circuit. Our target computation is the complete verifier for WHIR-based [ACFY25] STARK proofs from Ziren [DYH+26], which we call *WHIR proofs*. These binary-field proofs are generated by Ziren's binary stage and narrow recursion. Ziren first compresses a zkVM execution proof into a *WHIR proof*. The binary stage proves the execution of that WHIR-proof verifier using a binary-field proof system. Four recursion proofs follow, each proving the execution of the preceding proof's verifier. The first two use Ziren's plain recursion machine; the last two are *narrow-recursion* proofs, made in a machine with fewer trace columns, which reduces the verifier input consumed by the garbled circuit.

*Corresponding author: `stephen.d@zkm.io`

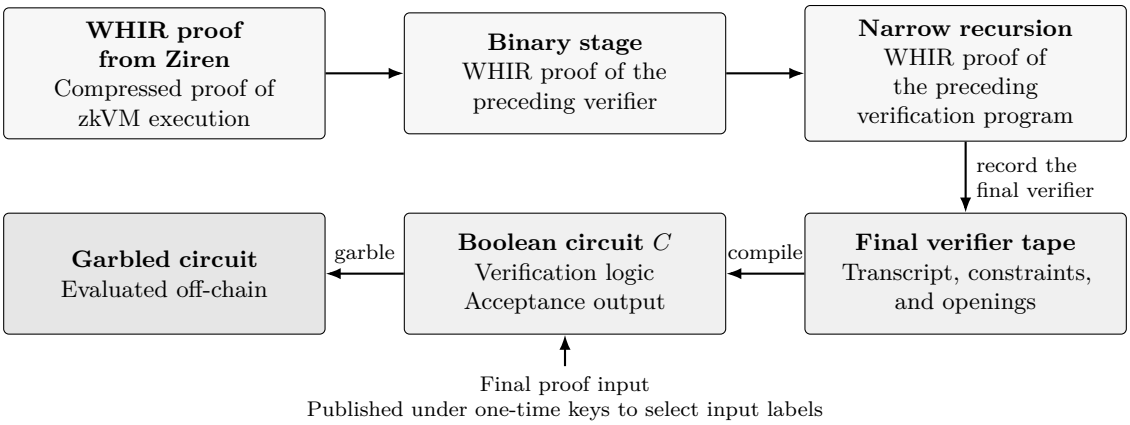

Figure 1: WHIR proofs of successive verification programs, followed by compilation and garbling of the final verifier. Arrows in the top row denote proof generation; those in the lower row denote program transformations.

A SNARK verifier written in Script takes hundreds of megabytes [GKS+26]. The BitVM line of work therefore verifies *optimistically*: a prover claims a result, and a challenger disputes it only if it is wrong. BitVM2 [WAA+26b] settles a dispute by checking a challenged verifier chunk on-chain, which can be expensive: a mainnet experiment put the unhappy path at over $14,000 [GKS+26]. BitVM3 moves the verifier off-chain into a *garbled circuit* [WAA+26a, Che25]. At setup the verifier circuit is garbled. At dispute time the proof is published under one-time signatures that give the evaluator the circuit's input labels, and the outcome depends on which output label the evaluation yields. For the Groth16 [Gro16] instances reported by BitVM3 and BABE, only the signed proof and a hashlock remain on-chain, for tens of dollars [GKS+26].

What a dispute publishes is therefore the proof: its bits under one-time keys, the scripts that check them and the signatures that bind them to the transaction. This paper is about the size of that data when the proof is a STARK, which is a thousand times larger than a Groth16 proof, and about reducing it: fewer input bits, less script per bit and fewer signature bytes per bit.

Key idea. We compile the verifier of the last WHIR proof in this pipeline into a Boolean circuit and garble it, using Plonky3's existing Wiedemann tower representation [Plo26, Wie88]. Under free-XOR [KS08] a garbled circuit pays only for non-linear gates, and in a field of characteristic two every addition is an XOR. A STARK over the binary tower field $\mathbb{F}_{2^{128}}$ with Blake3 commitments therefore costs, as a garbled circuit, its hashes and field multiplications, with additions free. Its algebraic checks rest on WHIR over $\mathbb{F}_{2^{128}}$, while Blake3 instantiates the commitments, the Fiat–Shamir transcript and the garbling hash, modeled as circular correlation robust (§ 6). This gives a hash-based outer verifier without an additional pairing-SNARK wrapper. We price input publication with hash-based one-time keys, and treat the adaptor-signature path as a baseline. Figure 1 distinguishes the proof stages from the verifier circuit that is garbled.

Challenges. Two problems stand between this idea and a dispute that Bitcoin can settle.

C1 *Garbling the verifier.* Compiling the complete verification program must preserve its transcript, multi-table constraints and commitment checks in a circuit of fixed shape. The recorded verifier's assertions and operand types must remain enforced on circuit inputs, and

the Fiat–Shamir schedule must match the reference implementation. Merkle authentication and transcript hashing also account for substantial circuit cost.

C2 *On-chain data.* A million input bits take millions of virtual bytes, against thousands for Groth16-based designs.

Contributions. We address both on-chain input cost (C2) and off-chain garbling and verification cost (C1). The binary-field verifier and garbling use hash-based assumptions; the underlying Ziren proof and the transaction layer have the separate assumptions stated in § 6.

- **Bitcoin integration and on-chain cost (C2).** We place the verifier in a BitVM3-style dispute, state the label-selection task that publishing its input must solve, and price four one-time input schemes using signed transaction serializations for the final verifier input. The operator garbles the verifier once and proves the garbling correct, so one input-key set selects its labels (§ 2.1.2, § 4.2). We measure the input-publication component; the transaction graph and the proof of correct garbling are not constructed. The component measures 1.11 MvB with the 8-bit adaptor baseline and 3.93–8.64 MvB with hash-based keys for the final Ziren verifier's 520,832-bit input, against a few thousand virtual bytes for Groth16. We also serialize a hash-based construction that would match the adaptor cost under a proposed batch hash-verification opcode.

- **Circuit-input reduction (C2).** Narrow recursion over Ziren's binary stage yields a final verifier input of 520,832 bits for a real Ethereum block. The construction combines aligned narrow layouts, fixed-data specialization, compressed linear-form openings, a closing column sumcheck that sends no opened value, shared reductions and commitment-parameter tuning (§ 4.3, Appendix B.1). A model derives proof size and the distribution of proving time from the WHIR schedule (§ 4.3.7).

- **Circuit compilation of the recursive verifier (C1).** We compile the verifier produced by Ziren's binary stage and narrow recursion from a recorded program over $\mathbb{F}_{2^{128}}$ into a Boolean circuit with one acceptance output. The compiled circuit implements its arithmetic and hashing operations, preserves its equality and nonzero assertions, and constrains byte and selector operands. The recorded program includes the transcript, multi-table constraint verification and Boolean WHIR openings; correctness relies on Ziren's execution recorder, which emits the fixed straight-line program that we compile (§ 4.1.1, § 5).

- **Boolean arithmetic gadgets and Blake3 optimization (C1).** We implement Plonky3's existing Wiedemann tower arithmetic as Boolean circuits for free-XOR garbling, using recursive Karatsuba multiplication: 3^k AND gates per 2^k-bit product, with squaring and multiplication by the tower generator implemented using only XORs. We also optimize bitvm-gc's Blake3 circuit [GOA26] from 20,657 to 10,281 AND gates per compression and add tree mode for long inputs (§ 4.1.2).

For the post-quantum goal, § 6.1 analyzes the STARK and garbling components in the quantum random-oracle model (QROM), including Grover search on the garbling offset, and distinguishes those bounds from the soundness of the garbling proof, the underlying Ziren assumptions and the non-post-quantum transaction layer. Section 5 estimates the final WHIR-proof verifier's garbling and input-publication costs, and compares the results with BitVM3's Groth16-based verification and the post-quantum Bitcoin-bridge cost analysis of Tiwari and Feickert [TF26b].

2 Related Work

We group prior work by how it verifies a proof on Bitcoin: by garbling a verifier, by witness encryption, or through small pairing-based SNARKs that both approaches rely on. We also discuss locally refutable verification and the garbling techniques we use.

BitVM. BitVMX [LAM+24] located a disputed step of a general CPU's trace (RISC-V or MIPS) by an n-ary search over a hash chain of the trace. ColliderVM [KLN25] left the optimistic model: it placed the computation on-chain in validity form and passed state between transactions through hash-collision puzzles, without fraud proofs, at a script length its authors estimate as nearly practical for STARK verification. Delbrag [Rub25] proposed replacing BitVM's fraud proof by the off-chain evaluation of a garbled circuit whose outcome is revealed as a secret, proving the circuit's correctness in zero knowledge. BitVM3 took up that idea and garbled the verifier [WAA+26a, Che25], defining BitVM3-CORE as a sound and complete on-chain proof system with on-chain costs of about $9.

Garbled verifiers. The Boolean-garbled Groth16 verifier [Bit25] has about 2.7×10^9 non-free gates [GKS+26]. Garbling Groth16 with native elliptic-curve group operations reduced the garbled program to 2.4 MiB [KFLT26]. Glock [Eag25] formalized garbled locks and garbled a designated-verifier variant of Pari; BABE reported its garbled circuit at about 12 MB on a binary elliptic curve [GKS+26]. GOAT's bitvm-gc [GOA26] garbles Groth16 and a designated-verifier form of Pari [DMS24] over BN254 and over the binary curve sect233k1.

We garble Ziren's final recursive verifier with bitvm-gc, and we contribute our Blake3 circuit back to it. BitVM3's comparison of garbling schemes already listed a Yao-style garbled STARK verifier, at about 10 GB of off-chain storage and as an estimate [WAA+26a, Table 9]; § 5.3 compares it with our measured garbled tables. TRAPGC-DV [FBF+26] verified Groth16 by designated evaluation: the prover wraps its proof in an outer Groth16 proof under a per-session trapdoor structured reference string (SRS) and garbles a small circuit that checks it in scalar form, with cut-and-choose and a universal-composability analysis. It too rests on pairings.

Garbling on Bitcoin Futoransky, Larotonda and Barbara [FLB25] broke the authenticity of the RSA-based garbling proposed for BitVM3 and of linear adaptors: coprime public exponents let the evaluator forge labels by extended Euclid, and linear adaptors fall to the Franklin–Reiter related-message attack. Our hash-based garbling has no such label algebra. Mosaic [KTA+26] made cut-and-choose maliciously secure with an on-chain footprint independent of the number of garbled copies, and Antichain Winternitz [TF26a] published input labels on-chain at a cost up to 52.9% lower than Lamport signatures. Both are independent of the garbled circuit; § 4.2 prices both for our final verifier input.

Post-quantum Bitcoin bridges. Tiwari and Feickert [TF26b] analyze the post-quantum cost of garbled-circuit Bitcoin bridges, including the proof verifier and the on-chain encoding of its input labels. Their estimates identify a tradeoff: hash-based STARK verification can reduce the garbled tables relative to a pairing-based verifier, while the larger proof increases on-chain publication costs. We compare against a representative STARK configuration from their analysis in Table 6. Our implementation addresses both costs by compiling the complete WHIR-proof verifier from Ziren and reducing the proof input consumed by that circuit. QSB [Lev26] constructed a hash-based, non-standard authorization path under existing consensus rules, mined on mainnet in August

2026 [Sta26]; a MuSig2-authorized, timelocked dispute graph [NRJ22] remains outside its current design.

Direct hash-based verification. Bitcoin Wildlife Sanctuary's Circle-STARK verifier [Bit24] implements field arithmetic, Fast Reed–Solomon interactive-oracle-proof-of-proximity (FRI) checks, Merkle paths and the Fiat–Shamir transcript in Bitcoin Script, using `OP_CAT` with `OP_SHA256` for hashing. StarkWare reports a Fibonacci-related proof verified on an `OP_CAT`-enabled Signet [Sta24]. This provides an implemented alternative for hash-based proof verification. Its on-chain Script execution and opcode requirement differ from our off-chain garbling and input-publication approach. The reported demonstration does not provide costs for the same final Ziren verifier, so we compare the architectures without assigning it a numerical cost advantage. The STARK verification uses hash-based assumptions; the covenant and transaction-authorization mechanisms must be assessed separately for post-quantum security.

Locally refutable verification. FlatChecks [Ler26] publishes a proof first and supplies the verifier's intermediate trace on challenge; the dispute checks a named local condition. This offers a different way to distribute verification across Bitcoin transactions. Our measured publication mechanism authenticates every input label needed to evaluate a complete garbled verifier. Comparing costs requires the same proof statement, challenge path and publication boundary; we do not infer a numerical advantage from the different applications reported in the two works.

Witness encryption. Witness encryption was introduced by Garg, Gentry, Sahai and Waters [GGSW13]. BABE [GKS+26] replaced the garbled Groth16 verifier with witness encryption for linear pairing relations [GKPW24], making the one quadratic pairing linear with a small garbled circuit built from decomposable randomized encodings on the Argo message-authentication code (MAC) [EL26]: 22 MiB of off-chain setup data per instance, on-chain cost comparable to BitVM3 [GKS+26]. Deferred binding [Pha26] extended it to public inputs known only at dispute time with two further garbled circuits, at 0.92 GiB per finalized instance, 768 input labels, an Assert of 2,154 bytes and a 14,312-byte ChallengeAssert witness. Bitcoin PIPEs v2 [ACG+26] witness-encrypted a Schnorr signing key under an NP statement to emulate covenants without a soft fork, with arithmetic affine determinant programs [SRG+26, BIJ+20] whose ciphertexts are cubic in the constraint count, about 338 TB for a 14,371-constraint pairing-SNARK verifier, under a heuristic security analysis. All witness-encryption designs are smaller off-chain than a garbled verifier, but all need a pairing SNARK, so a STARK would first be wrapped in one, and none is post-quantum.

Binary-field proof systems and garbling. Binius [DP25] built succinct arguments over binary towers, and ring switching [DP26] turned a claim about a polynomial's bits into one about its packed form; Plonky3's Boolean WHIR commitment [Plo26], which we garble, uses both with WHIR [ACFY25] as the proximity test, and its cheap additions are what make a garbled verifier small.

2.1 BitVM3-style verification

In BitVM3 [WAA+26a], the operator acts as the garbler and a challenger as the evaluator. At setup, the operator garbles a proof-verification circuit. At dispute time, the operator publishes the proof under one-time keys that reveal the selected input labels. Any party holding the garbled circuit and these labels can evaluate the verifier off-chain. An invalid proof yields a reject-output

label that unlocks an on-chain hashlock and prevents reimbursement. The proof is public, so the garbling needs authenticity rather than privacy: the evaluator may know the wire values but must not derive the label for an incorrect output. The garbler's construction must itself be checked. BABE [GKS+26] and Mosaic [KTA+26] use cut-and-choose: the garbler prepares many instances and opens most of them. BitVM3 also names a proof that the setup is correct [WAA+26a]. Our construction takes the second route: the operator garbles the verifier once and proves the garbling correct. It keeps the division between off-chain verification and on-chain input publication and refutation.

2.1.1 Dispute graphs

We describe transaction dependencies and execution conditions using Bitcoin's unspent-transaction-output (UTXO) model [WAA+26b, §2.2 and Appendix B.2].

Definition 1 (Dispute graph)**.** *A dispute graph is $\mathcal{G} = (V, E, \perp)$, where V is a finite set of transaction skeletons. A skeleton $T = (I_T, O_T)$ specifies the set I_T of input outpoints and the indexed collection O_T of outputs, each with a value and locking predicate. An edge $(T, i, T') \in E$ means that T' consumes output i of T; these dependencies are acyclic. For distinct transactions, $T \perp T'$ means $I_T \cap I_{T'} \neq \varnothing$. Let U_0 be the initial set of unspent outpoints. An execution of n transactions is a sequence $(T_k, \omega_k, h_k)_{k=0}^{n-1}$, where $T_k \in V$, ω_k is its witness and h_k its confirmation height. Let U_k denote the set of unspent outpoints after the first k transactions. The execution is valid if, for every $0 \leq k < n$,*

$$U_k \xrightarrow{T_k, \omega_k, h_k} U_{k+1} = (U_k \setminus I_{T_k}) \cup \mathrm{Out}(T_k),$$
$$I_{T_k} \subseteq U_k, \qquad \mathsf{Valid}(T_k, \omega_k, h_k; H_k) = 1.$$

Here H_k is the confirmed history before step k, $\mathrm{Out}(T)$ is the set of newly created outpoints, and Valid *includes value conservation with fees, witness checks, authorization and absolute or relative timelocks. Initial funding outputs may be created by transactions outside V.*

This definition distinguishes a dependency graph from an execution: a transaction with several parents requires all its inputs, and independent publication branches may execute in either order. Conflicting transactions cannot both occur in one valid history. A timeout is a lower bound on the height at which a competing spend becomes valid; it does not itself cancel another branch. Cancellation requires consuming an input needed by that branch. Covenant emulation must additionally prevent unauthorized spends of the protected outputs; graph notation alone does not enforce that property.

Refutation predicate. The refutation witness is a reject-output label of the operator's garbled verifier (§ 2.1.2). Let ℓ_0 denote that reject-output label, and let $\eta = \mathrm{SHA256}(\ell_0)$ be its committed hash. For a candidate label ℓ, the refutation condition is

$$\mathsf{Reject}(\ell) := \mathbf{1}\{\mathrm{SHA256}(\ell) = \eta\}.$$

Here $\mathbf{1}\{\cdot\}$ is 1 when its condition holds and 0 otherwise. Circuit authenticity and the proof of correct garbling relate this hashlock witness to an invalid verifier input (§ 6); the transaction graph controls how its disclosure prevents reimbursement.

transition	required guard and effect
Kickoff → Take1	$h \geq h_K + \Delta_C$ and the challenge connector remains unspent; reimburse the operator.
Kickoff → Challenge	Spend the challenge connector and supply the challenge bond; disable Take1.
Challenge → completed Assert	Publish all authenticated chunks so that $\mathsf{Complete}(x) = 1$, and create the completion output while its required inputs remain unspent.
Challenge → AssertTimeout	$h \geq h_C + \Delta_A$ and a connector needed for completion remains unspent; penalize the operator and disable completion.
completed Assert → Disprove	Supply ℓ with $\mathsf{Reject}(\ell) = 1$ and consume a connector required by Take2; prevent reimbursement.
completed Assert → Take2	$h \geq h_A + \Delta_D$ and every required refutation connector remains unspent; reimburse the operator.

Table 1: Transition guards and effects for the garbled-verifier dispute. The publication subgraph must enforce completion before either terminal branch.

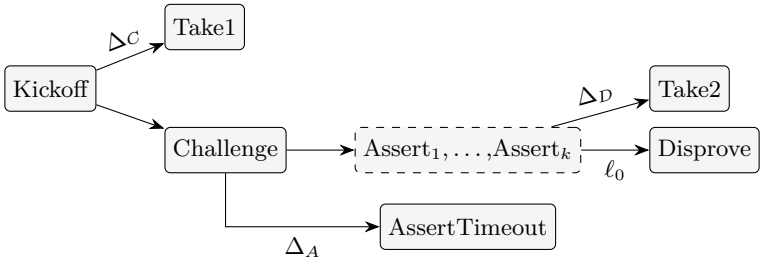

Figure 2: Dispute graph. The dashed Assert$_1, \ldots,$Assert$_k$ node groups the input-publication transactions: a funding transaction, the reveal transactions that publish the input chunks, and a join that spends the reveals' outputs and creates the completion output. Only after the join can the protocol reach the mutually conflicting Take2 and Disprove outcomes.

Dispute transitions. Figure 2 shows the dispute graph. Let h_K, h_C and h_A denote the confirmation heights of Kickoff, Challenge and completed input publication, and let h denote the height at which a transition transaction confirms. Let Δ_C, Δ_A and Δ_D be the challenge, input-publication and refutation delays, respectively, measured in blocks from these three heights. For an encoded verifier input x partitioned into k chunks $x^{(1)}, \ldots, x^{(k)}$ of fixed layout, define

$$\mathsf{Complete}(x) := \bigwedge_{i=1}^{k} \mathsf{Auth}_i(x^{(i)}),$$

where Auth_i requires publication and authentication of the prescribed chunk and its binding to the same dispute instance. The chunks must cover every input wire. Table 1 specifies the required transition guards. A *connector* is an output that conflicting transactions both spend, so confirming one disables the others [WAA+26b]; conditions on unspent connectors encode conflicts, rather than asking Script to prove the absence of a transaction.

The challenge and refutation windows provide time for a competing spend to confirm; after a timelock matures, unresolved branches may race for the same input. Safety and progress therefore also require ledger-confirmation and honest-monitoring assumptions.

2.1.2 Setup and dispute

Measurement scope. We use *total on-chain footprint* to mean the sum of the virtual transaction sizes, in virtual bytes (vB), of every transaction in the dispute graph, including ordinary transaction signatures, authentication signatures or preimages, witness framing, revealed tapscripts, control blocks, transaction inputs and outputs, anchors, timeouts and fee-management transactions. A transaction's virtual size is its weight divided by four, rounded up to an integer [LLW15]; it differs from the transaction's serialized size in bytes. We measure the input-publication component of that footprint: bounded publication slices, each a funding transaction and reveals that spend it, for all four schemes, and a signed join for Lamport. The challenge, timeout, Disprove, anchor and fee-management branches, the proof of correct garbling and the complete transaction graph are not constructed, so the figures are components of that footprint and conditional projections.

Setup. Let C be the proof-verification circuit with one acceptance output. The protocol is BitVM2 [WAA+26b] with the on-chain verifier replaced by a garbled one. The operator garbles C once. It commits the hash of the garbled tables, the reject-label hash η, the hashes of its one-time input keys, and the label ciphertexts that map each key secret to an input label (§ 4.2). It then gives a zero-knowledge proof that these commitments come from one correct garbling of C: the tables garble C under a single offset, η is the hash of its reject-output label, and the ciphertext for value b of input i encrypts the label of value b on wire i under the committed key for that value. Anyone can check this proof, and it replaces cut-and-choose: there is one garbled instance and no opened ones. To keep the verifier's assumptions, the proof must itself be hash-based. We neither construct this proof nor measure its proving cost. Only after it is checked is the transaction graph signed, by a committee as in BitVM2, whose enforcement requires at least one honest signer to delete its signing key after setup.

Publishing bit b of input i then gives a challenger exactly the label of value b on wire i. With one garbled instance, one input-key set suffices, and no correlation of labels across instances is needed.

Intended dispute. The operator's Kickoff claims a reimbursement; a challenger may spend its challenge output within Δ_C, after which the operator must publish every authenticated input bit through Assert or lose its stake on timeout. A challenger then reads the published bits, derives the input labels and evaluates the garbled verifier off-chain, one Blake3 call per non-free gate. An invalid proof yields ℓ_0, which opens Disprove. The public-value digest, eight field elements that commit to the zkVM program's verifying key and outputs, binds the verified off-chain computation to its program and public outputs; for a bridge, these outputs include the rollup-state commitment and withdrawal. Section 4 reduces the bits Assert publishes and the bytes each bit costs.

3 Setting

3.1 The garbling scheme

We use privacy-free garbling [FNO15] with free-XOR [KS08] and one ciphertext per non-free gate, in the syntax of Bellare, Hoang and Rogaway [BHR12b]. The evaluator may learn every wire's value. The required property is *authenticity*: given the labels of the published input x, the evaluator obtains the label for $C(x)$ but cannot derive the label for $1 - C(x)$.

Let Δ be a random secret 128-bit offset. Every wire w has a false label w_0, and its true label is $w_1 = w_0 \oplus \Delta$. Let $H(\ell, g) = \text{Blake3}(\ell \,\|\, g)$ truncated to 128 bits, where g is the gate's index. An

XOR gate sets $c_0 = a_0 \oplus b_0$ and has no ciphertext. An AND gate $c = a \wedge b$ sets

$$c_0 = H(a_0, g), \qquad ct = H(a_1, g) \oplus H(a_0, g) \oplus b_0.$$

The evaluator recovers the output label from its two input labels, the ciphertext and one hash [ZRE15, §6]; an OR gate is symmetric. The garbled circuit is one 16-byte ciphertext per non-free gate.

Two details matter in practice. First, Δ must be secret and fresh for each garbling: with a public Δ any label yields its complement, and the output's other label is free to compute. Second, a hash modeled as circular correlation robust [CKKZ12] (§ 6) gives labels no algebraic structure, the structure through which the RSA-based proposals were broken [FLB25].

3.2 The target verification program

The target computation is the complete verifier of the final WHIR proof: its AIR and cross-table checks and the WHIR opening proof that authenticates polynomial evaluations. This binary-field verification backend is implemented in Plonky3 [Plo26] and uses fields in the Wiedemann tower [Wie88]

$$\mathsf{T}_0 = \mathbb{F}_2, \qquad \mathsf{T}_{k+1} = \mathsf{T}_k[X_k]/(X_k^2 + X_{k-1}X_k + 1), \qquad X_{-1} = 1,$$

up to $\mathsf{T}_7 = \mathbb{F}_{2^{128}}$. An element of T_{k+1} is stored as its low half (the coefficient of 1) and its high half (the coefficient of X_k), recursively down to bits. The trace is committed as bits: a Boolean column is packed into $\mathbb{F}_{2^{128}}$ elements, and WHIR uses Reed–Solomon proximity testing to authenticate the packed multilinear polynomial on an additive (Cantor) evaluation domain [ACFY25, Can89, LCH14]. Binary towers and ring switching come from Binius [DP25, DP26].

Verification predicate. The circuit C computes the acceptance bit of the final WHIR proof's verification program, which comprises the following stages:

1. *Transcript reconstruction.* Absorb the preprocessed and main trace commitments and the declared public values, and derive Fiat–Shamir challenges in the prescribed order throughout verification. The AIRs specify polynomial constraints on current and next trace rows; these constraints, the table dimensions and the verifying key determine the expected proof structure.

2. *Cross-table consistency.* Verify the binary-bus product argument, which matches the values each table of the recursion machine (§ 4.1.1) sends with those another receives, together with any lookup reductions required by the configured AIRs. The resulting evaluation claims must subsequently be bound to committed trace values.

3. *Constraint reduction.* Verify the shared *zerocheck* sumcheck [CBBZ23, LFKN92], which reduces the claim that AIR and bus constraints vanish on every row to a check at a common random evaluation point. Each table uses the coordinates corresponding to its trace height; any indexed lookup reduction is verified at its prescribed points.

4. *Opening reduction.* Reduce the required evaluations of main and preprocessed columns, or their linear forms, to claims on the packed trace polynomials. This includes the column-batching sumchecks and the bit ring-switch arguments over $\mathbb{F}_{2^{128}}$.

5. *Commitment verification.* Verify the Boolean WHIR openings for those claims, including the sumcheck rounds, grinding checks, STIR queries [ACFY24] that open folded codewords at random positions, Merkle authentication paths and final-polynomial checks. Main values are bound to the proof's trace commitment, and preprocessed values to the commitment fixed by the verifying key.

6. *Constraint closure.* Evaluate the AIR and bus constraints using the authenticated openings, discharge the remaining lookup claims, and check consistency with the sumcheck's terminal claim. The circuit returns acceptance only if these checks and all assertions introduced by the Boolean compilation hold.

All challenges come from a Blake3 [OANWO20] sponge over bytes, and all Merkle trees use Blake3 compressions.

4 Construction and optimizations

Two quantities determine the principal costs of our garbled WHIR-proof verifier: the number of published input bits and the number of non-free gates. The former controls input authentication on Bitcoin; the latter controls garbling, evaluation and garbled-table storage. Within the BitVM3 framework of § 2.1, we develop Boolean compilation and gate reductions (§ 4.1), input publication (§ 4.2), and recursion and proof reductions (§ 4.3). These optimizations trade prover work, circuit size and input length; we state those tradeoffs explicitly. Appendix A details the input-authentication schemes, and Appendix B.1 gives the algebraic arguments.

4.1 Boolean compilation and off-chain verification cost

The circuit sets the first quantity: it reads the proof's bits, so its input layout is the bit count that Assert publishes. Its size is also what the operator garbles and proves, and what challengers store. This subsection turns the verifier of § 3 into a Boolean circuit with one output wire. We first describe how the recursive verifier is recorded and compiled, then the arithmetic and hash gadgets that reduce its non-free gate count.

4.1.1 Verifier recording and circuit compilation

The recorder executes the verifier of a Blake3-committed WHIR proof once and serializes its resulting straight-line program as a *binary-field verifier tape*. The tape contains an ordered instruction stream whose operands refer to proof inputs, constants, or earlier results. Its instructions cover field arithmetic and assertions, byte conversion, selection, Blake3 compression, and Merkle-path verification. "Binary" here identifies Ziren's binary-field recursion stage; the tape is neither a bit string nor a Turing-machine tape. Ziren proves its execution with seven AIR tables, which form the recursion machine: the ledger table tracks values and their uses, the arithmetic table checks field operations, two rewiring tables check representation changes, one for whole elements and bytes and one for the gathers of transposes, the hash table schedules hash calls, the Blake3-round table checks their internal rounds, and the public table binds the program's public values. The circuit we garble is compiled from the tape of the final recursive proof's verifier.

We compile this tape instruction by instruction into a Boolean circuit. Field addition and squaring use XORs; multiplication uses the tower gadget, and inversion uses an addition chain with a nonzero check. Equality and nonzero assertions become check bits. Byte conversions and selectors enforce their operand ranges, while hash instructions use the Blake3 circuit and Merkle

instructions select the order of the two child digests before hashing. Inputs are encoded according to their use as bits, bytes or 128-bit field elements. All assertion and range-check bits are conjoined into one acceptance output. The circuit shape is fixed by the recorded tape and its input types.

Streaming execution. We use bitvm-gc's streaming backend [GOA26] for the compiled verifier. A planning pass determines wire lifetimes; the streaming pass reuses storage after a wire's final use, retaining the acceptance output. This reduces the live-label memory required to execute the circuit without changing its gates or input. Planning metadata and the stored garbled tables remain separate costs. In particular, streaming does not remove the challenger's obligation to retain the garbled tables.

Why a binary field. Boolean WHIR commits over $\mathbb{F}_{2^{128}}$, whose addition is XOR and free under free-XOR; field multiplication is the only non-free arithmetic (§ 4.1.2).

Why Blake3. One hash serves three roles: the Merkle commitments and the Fiat–Shamir sponge of the proof system, and the garbling hash H of § 3.1. Blake3 is an add-rotate-XOR design, so its only non-free operations are 32-bit additions; a width-16 Poseidon2 permutation over the 31-bit KoalaBear field [GKS23] costs about 120 Blake3 compressions (Table 2). A prover that already commits and samples with Blake3 needs no change, and the circuit's transcript can be checked byte for byte against it. A single hash also keeps the verifier's assumptions to those of § 6, Blake3 as a random oracle with circular correlation robustness, with no elliptic-curve or lattice assumption, which is what makes the verifier's own security claim post-quantum (§ 6.1).

4.1.2 Gadgets

Tower arithmetic. We retain Plonky3's Wiedemann tower representation [Plo26, Wie88] and compile its reference arithmetic into Boolean gates. A wire vector for an element of T_k is its bit pattern, so gadget outputs can be read back directly as field elements. Multiplication follows the existing tower algorithm recursively, using Karatsuba at each level. For $a = a_0 + a_1 X_k$ and $b = b_0 + b_1 X_k$ in T_{k+1}, with $a_0, a_1, b_0, b_1 \in \mathsf{T}_k$, define the intermediate products

$$z_0 = a_0 b_0, \quad z_2 = a_1 b_1, \quad z_1 = (a_0 + a_1)(b_0 + b_1) + z_0 + z_2, \qquad ab = (z_0 + z_2) + (z_1 + X_{k-1} z_2)\, X_k.$$

Additions are free. Multiplication by the generator X_{k-1} is a rewiring plus XORs, and squaring is linear over $\mathbb{F}_2$. The only AND gates are the base-field products, three per level, so a 2^k-bit multiplication costs 3^k AND gates: 2,187 in $\mathbb{F}_{2^{128}}$. A multiplication by a constant folds to XORs entirely.

Blake3. We start from bitvm-gc's Blake3 circuit [GOA26] and change two things. A 32-bit addition's carry is the majority

$$\mathrm{maj}(a, b, c) = c \oplus \big((a \oplus c) \wedge (b \oplus c)\big),$$

where a, b are the operand bits and c is the incoming carry. This uses one AND per bit, and the carry out of the top bit is discarded, so it is not computed. This halves a compression from 20,657 to 10,281 AND gates. We also add Blake3's tree mode (chunk chaining values on a stack, parent nodes, the root last). Tree mode supports transcript messages that exceed one 1,024-byte chunk between samples. Table 2 gives the costs. Every gadget is checked against reference implementations of the field and the hash on random inputs.

gadget	AND gates
$\mathbb{F}_{2^{128}}$ multiplication	2,187
$\mathbb{F}_{2^{128}}$ squaring, tower-generator multiplication, addition	0
Blake3 of 64 bytes (a Merkle compression)	10,281
Blake3 of 256 bytes (a leaf row of 16 elements)	41,511
Blake3 of 1,072 bytes (two chunks: a transcript flush)	186,125
Poseidon2 permutation, width 16 over KoalaBear	1,240,747

Table 2: Gadget costs.

4.2 Publishing the input

Label selection. Every input mechanism we price is an instance of a single two-party task. The *garbler* G (the operator, who also proves) holds the two labels ℓ_w^0, ℓ_w^1 of every input wire w of the garbled verifier; the *evaluator* E (a challenger) will run the garbled verifier on the proof π. Write π_w for bit w of its fixed circuit-input encoding. Publishing π on-chain must (i) *deliver* $\ell_w^{\pi_w}$ to E for every w, (ii) be *exclusive*: an evaluator given the published input labels cannot compute any $\ell_w^{1-\pi_w}$ without the garbler's secrets, since one complementary label breaks authenticity (§ 6), (iii) be *binding*: G publishes at most one value per wire, and the publication is also the authorization of the spending transaction, and (iv) be *public*: any observer can recover π and check the published object.

A malformed publication cannot be spent, a missing one forfeits the stake through AssertTimeout, and Disprove is reserved for the reject label. Lamport signatures [Lam79] achieve (i)–(iv) with one preimage per bit, at the cost of a per-bit script. The Schnorr adaptor baseline of BitVM3 and Mosaic [WAA+26a, KTA+26] achieves them with one signature per k-bit digit and no per-bit script; Appendix A.1 gives its protocol diagram (Figure 3). The hash-based alternative (Figure 4, Appendix A.2) replaces the signature by a pair of hash-chain openings and underlies the Antichain row and the hash-based projections of § 6.1.

The adaptor uses one `CHECKSIG` per digit for both selection and transaction binding, which is why its script is three bytes per digit. Hash-based schemes additionally authenticate their openings with hash ladders in Script, which hash a revealed chain node forward to its committed end, and require a separate transaction signature; only their selection is hash-based. Table 3 breaks down their costs per digit or bit, and § 5.2 gives the totals; WOTS denotes a Winternitz one-time signature. Appendix A.3 separately considers a hash-based batch-verification opcode.

One-time signature schemes. We price four one-time schemes for carrying the input bits. *Lamport* [Lam79] reveals one of two hashed preimages per bit. *Antichain Winternitz* [TF26a] and *Winternitz with a translation table* [SST+26] close the forward-hashing gap of plain Winternitz chains, in which anyone can hash a revealed node forward to the node of another digit value. The translation-table variant makes each label a Shamir secret whose threshold a signed chunk's checksum sets. *Schnorr adaptor signatures* over 8-bit digits, as in BitVM3 and Mosaic [WAA+26a, KTA+26], rest on discrete logarithms. Increasing the adaptor's digit width reduces the number of on-chain signatures but expands the off-chain choice table, the label ciphertexts of every candidate digit value, exponentially; we retain the 8-bit construction as the authenticated baseline. Appendix A.4 gives the four scripts and the signed transactions we serialize and execute for them.

Our labels are free-XOR correlated, so a hash-based preimage s cannot itself be a label: a key-derivation function (KDF) maps it to one by a label ciphertext $\mathrm{KDF}(s) \oplus w_b$, two per input bit. The garbling proof of § 2.1.2 covers these ciphertexts: it shows that each one encrypts the

scheme	selection material	script	transaction binding	vB/bit	assumption
Schnorr adaptor, $k = 8$	one 64-byte completion per 8-bit digit	one `CHECKSIG` per digit, 3 bytes	the same signature	2.13	discrete logarithm
WOTS translation, $k = 3$	one 20-byte chain node per 3-bit digit, plus two checksum chains per chunk	hash ladder in Script, 2 bytes per step	separate 64-byte Schnorr signature per input	7.55	hash; 128-bit translation chain
Antichain $(2, 16)$, $k = 4$	two 20-byte chain nodes per 4-bit digit	hash ladder in Script, 2 bytes per step	separate 64-byte Schnorr signature per input	11.38	hash
Lamport	one 16-byte preimage per bit	49 bytes per bit	separate 64-byte Schnorr signature per input	16.59	hash

Table 3: Where the four input schemes spend their bytes. Selection material and script are per digit or per bit as stated; vB/bit is the measured one-key-set component of Table 5, including each transaction's fixed fields.

label of its bit value under the committed key for that value. The one key set of Table 5 therefore selects the labels of the single garbled verifier. A cut-and-choose setup would instead keep several garblings and need a further proof that one key set selects the same bits in each, as in BABE's zero-knowledge soldering [GKS+26]; a single proved garbling has no such step.

4.3 Reducing recursive-proof inputs and verification work

The circuit consumes only the proof values read by the final verifier tape. We reduce that input in three parts: the recursion machine's layout, the representation of its opening claims, and commitment authentication. Some changes also eliminate non-free gates; others add verification work to save published bits. The following mechanisms apply to the Ziren recursion path.

The opening reductions connect the values used by AIR and bus checks to the committed trace. A column sumcheck [LFKN92] reduces claims about linear combinations of columns to column evaluations at random points. The bit ring switch [DP26] translates those claims into the packed-field representation authenticated by WHIR [ACFY25]. The optimizations below reduce the values and round messages carried between these steps. Each transmitted field element adds 128 circuit-input bits; processing it also incurs transcript and verification work. The final commitment openings remain necessary to authenticate the reduced claims.

Concretely, fix the basis $\beta_0, \ldots, \beta_{127}$ of $\mathbb{F}_{2^{128}}$ over $\mathbb{F}_2$, and pack 128 bits as $p(u) = \sum_{j=0}^{127} \beta_j t(u, j)$, where $t(u, j)$ is bit j of the element packed at u. A ring-switch proof sends a *tensor witness*

$$T = \sum_{i,j} M_{i,j}\, \beta_i \otimes \beta_j \in \mathbb{F}_{2^{128}} \otimes_{\mathbb{F}_2} \mathbb{F}_{2^{128}}, \qquad M_{i,j} \in \mathbb{F}_2.$$

The implementation transmits its 128 rows $R_i = \sum_j M_{i,j}\beta_j$, hence 128 field elements, rather than a matrix of field elements. Reading the same bit matrix by columns gives $C_j = \sum_i M_{i,j}\beta_i$. For an incoming opening point whose seven coordinates inside a packed element are r_{lo}, the verifier derives, rather than receives, the equality-weight vector $q_j = \mathrm{eq}(r_{\mathrm{lo}}, \mathrm{bin}(j))$ and checks

$$y = \sum_j q_j C_j.$$

Here $\mathrm{eq}(a, b) = \prod_k ((1-a_k)(1-b_k)+a_k b_k)$ is the multilinear equality polynomial and $\mathrm{bin}(j) \in \{0, 1\}^7$ is the binary address of position j. After T is absorbed, the transcript samples a second point

ρ and derives row weights $w_i = \mathrm{eq}(\rho, \mathrm{bin}(i))$. The initial claim of the ring-switch sumcheck is $\sum_i w_i R_i$. That sumcheck reduces it to a random evaluation $p(z)$ of the packed polynomial, which the subsequent WHIR opening authenticates. At its terminal point the verifier computes the closing weight from the equality tensor

$$E(r, z) = \sum_u \mathrm{eq}(r, u) \otimes \mathrm{eq}(z, u),$$

not from a prover-supplied weight matrix. Thus T is prover-supplied data; q, w and $E(r, z)$ are verifier-derived weights.

4.3.1 Narrow recursion and aligned trace layouts

Recursing over the verification program. Every recursion level, plain or narrow, records its single child's verifier through a wrap node, a recursion step over a single child proof. A compose node over the same child checks that the child's shards form one consecutive chain in KoalaBear arithmetic, 4,298 operations of its tape at each level; the wrap node commits to the same public values but makes that check in binary-field arithmetic, so no recorded tape carries a KoalaBear operation. A wrap node takes only one child, so a node over several children remains a compose node. The compiler therefore also translates KoalaBear operations, so a compose node's tape compiles too; none of the measured tapes uses that translation. Each narrow-recursion level proves that verifier in the machine of § 4.1.1. The final garbled circuit therefore verifies a proof of a narrow machine rather than exposing the original verification program's trace width at its input, which moves work from the final verifier into recursive proof generation.

Aligned table packing. A power-of-two table block placed at an offset aligned to its own size has a fixed address prefix and free trailing coordinates, so one restriction of the packed polynomial, and hence one ring-switch claim, describes the entire block. An unaligned block crosses an address boundary and splits into several claims, each carrying a tensor witness and its verification work. We align the narrow machine's power-of-two blocks and pad its preprocessed round-table width accordingly; the binary stage's arbitrary table layouts are not aligned.

Narrow rewiring rows. A transpose of 128 elements passes through bytes: 128 splits write the 16 bytes of each row, and 16 gathers each read byte c of every row and write the eight output columns of that byte position. Only a gather reads more than one element's 16 bytes, and a transpose has eight splits to each gather, so the gathers have a table of their own with 128-byte rows and every other rewiring uses 16-byte rows; the bus authenticates the bytes between them. Fewer columns require fewer opened field values, but the extra rows increase prover work. Thus narrowing the machine is a tradeoff between final verification cost and the size of the execution trace that recursion must prove.

Half-round Blake3 rows. The Blake3-round table normally holds a whole round per row, 4,096 columns. The machine of the final proof instead holds one half of a quarter-round per row. For lanes (a, b, c, d) and message word m, a row computes

$$a_1 = a + b + m, \quad d_1 = (a_1 \oplus d) \ggg r_1, \quad c_1 = c + d_1, \quad b_1 = (c_1 \oplus b) \ggg r_2,$$

with rotations $(r_1, r_2) = (16, 12)$ in a first half and $(8, 7)$ in a second. A row holds the lanes and word it reads, the sums and carries of its three additions, and d_1 and b_1: 416 columns, and 112 rows per compression. The table commits one and a half times the bits of the whole-round rows and opens a tenth of their values.

4.3.2 Specializing fixed data and input encodings

Typed inputs and known values. The compiler allocates one bit, eight bits or 128 bits according to a consumed input's type, while enforcing its byte and selector ranges. Unread proof values receive no input wires. The recorder also replaces known padding values and preprocessed columns constant over every row with circuit constants, after checking their values in the recording proof. Their transcript absorption and commitment-opening relations remain checked; the substitution removes input wires rather than omitting the corresponding verification constraints. Constant field multiplications then reduce to XORs. The final verifier sends no opened value (§ 4.3.4), so this substitution applies only to verifiers that read them.

Dead operations. The circuit's output is the AND of its checks, so an operation matters only if it checks something or a live operation reads its result. The compiler omits additions, products, squares, byte splits and transposes whose results no live operation reads, and an input read only by such operations receives no wires. Assertions, inversions, range-checked operands, hashes, selections and Merkle steps always remain.

A deep cap for fixed preprocessing. The preprocessed Merkle tree is fixed by the verifying key. We embed its nodes at a chosen depth as constants, checked against the key's cap, and select the appropriate node using a one-hot decoding of the query index. A query authenticates only the path below that node, so upper siblings are no longer circuit inputs and their hashes are no longer computed. The verifier still hashes the query-dependent leaf and lower path to a selected constant node. Each omitted upper sibling saves one digest from the input and one path hash; the one-hot selector enforces which embedded node the query reaches. A greater depth κ removes more siblings but adds selection gates. This specialization depends on the fixed preprocessing tree; proof-dependent main commitments cannot be treated as constants.

4.3.3 Opening only the linear forms used by verification

Packed operands and general linear forms. The ledger table's values and arithmetic operands are read by their constraints and buses as packed field elements. We first replace each such group of column openings by its packed value. We then generalize this compression: symbolic AIR and bus expressions determine a fixed matrix L of required linear forms. Instead of transmitting the full column-evaluation vector x, the prover sends the vector $y = Lx$ of linear-form evaluations. For example, if the affected checks use two columns only through $x_0 + \gamma x_1$, for a fixed coefficient $\gamma \in \mathbb{F}_{2^{128}}$, one transmitted field element can replace the two column evaluations. The extractor must retain every additional combination read elsewhere by a constraint or bus. The verifier reconstructs a representative Ry using a fixed right inverse R on the image of L, so $LRy = y$ for $y \in \operatorname{im} L$. This preserves an affected expression precisely when it factors through L, as formalized in Lemma 1. The reconstruction uses constant field coefficients and therefore only XORs. A column sumcheck binds the transmitted forms to the committed trace.

Affine relations in fixed preprocessing. For preprocessed columns, we additionally derive relations over every row of the fixed trace, including the all-ones column in the basis. Dependent forms are reconstructed from an independent subset and constants. Linearity of multilinear extension preserves these affine identities at every opening point. This removes further input values without guessing relations from a particular proof's evaluations.

One shared forms sumcheck. The forms must be checked against the committed columns. Running a separate column sumcheck for each table repeats the round messages and transcript updates. We instead batch the form claims before running one sumcheck over the widest column domain. Here a group t contains the forms of one table block at a fixed row point. Let k_t be its column-variable count, C_t the multilinear column polynomial at that row point, and $L_{t,j}$ the multilinear extension of the coefficient vector of form j. Its claimed value $y_{t,j}$ must satisfy

$$y_{t,j} = \sum_{u \in \{0,1\}^{k_t}} L_{t,j}(u) C_t(u).$$

The commitments, row points and claimed forms are absorbed before sampling $\rho \in \mathbb{F}_{2^{128}}$. Enumerate all forms with distinct consecutive exponents $e_{t,j}$ starting at zero, and define

$$Y = \sum_{t,j} \rho^{e_{t,j}} y_{t,j}, \qquad W_t(u) = \sum_j \rho^{e_{t,j}} L_{t,j}(u).$$

The combined claim is $Y = \sum_t \sum_u W_t(u) C_t(u)$. These random weights prevent a fixed collection of false form claims from cancelling except with the batching error accounted for in Lemma 2.

To share rounds across different widths, let $d_{\mathrm{col}} = \max_t k_t$ and zero-extend both W_t and C_t to $\{0,1\}^{d_{\mathrm{col}}}$. For $z \in \mathbb{F}_{2^{128}}^{d_{\mathrm{col}}}$, write z_t for its trailing k_t coordinates and define $b_t(z) = \prod_{i=1}^{d_{\mathrm{col}}-k_t}(1 - z_i)$. The shared integrand is

$$S(z) = \sum_t b_t(z)^2 W_t(z_t) C_t(z_t), \qquad Y = \sum_{u \in \{0,1\}^{d_{\mathrm{col}}}} S(u).$$

The prefix is squared because both factors are zero-extended. On Boolean points $b_t^2 = b_t$, but the terminal point is a random field point, where the square cannot be dropped. The integrand has degree at most two in each variable, so every sumcheck round uses a quadratic polynomial.

Starting from running claim $s_0 = Y$, round i supplies a polynomial $g_i(X) = A_i + B_i X + D_i X^2$ whose Boolean sum must equal s_i. The prover sends only A_i, D_i; the verifier reconstructs $B_i = s_i - 2A_i - D_i$ to enforce $g_i(0) + g_i(1) = s_i$. After absorbing the message, it samples α_i and sets $s_{i+1} = g_i(\alpha_i)$. After d_{col} rounds, the challenges form z and the verifier checks $s_{d_{\mathrm{col}}} = S(z)$. It computes each $W_t(z_t)$ from the known coefficients and batching weights, while the prover supplies the column fold $C_t(z_t)$. Each fold becomes a ring-switch claim and is subsequently authenticated against the commitment.

For n_{grp} groups, the reduction sends $2d_{\mathrm{col}} + n_{\mathrm{grp}}$ field elements, excluding the forms themselves, instead of $2\sum_t k_t + n_{\mathrm{grp}}$ for separate sumchecks with the same round encoding. For example, 16-column and 8-column groups share four rounds instead of running four and three rounds separately. Lemma 2 gives the conditional algebraic bound, with commitment-authentication errors accounted for separately.

4.3.4 Closing the zerocheck without opened values

The closing check as a column sumcheck. The shared zerocheck ends with a closing check: the verifier evaluates every table's AIR constraints and bus terms on the column values at the sumcheck point and compares the batched result with the final claim. Even as linear forms, those values are sent, so the input grows with the tables' widths. The final proof instead proves the closing check against the committed columns and sends no opened value.

The check is a polynomial of degree at most three in the column values, one polynomial per table, with the zerocheck's equality and batching weights included. For table t with column values

v_t, we split it by degree as

$$P_t(v_t) = c_t + L_t(v_t) + Q_t(v_t, v_t) + T_t(v_t, v_t, v_t),$$

where L_t, Q_t and T_t are multilinear in one, two and three vector arguments. A product places its left factor's arguments before its right factor's, so every part stays multilinear without symmetrization, which characteristic two would not allow. Index the table's main and preprocessed columns in one space of n_t coordinates, preprocessed columns in the upper half, and let $V_t(u)$ be the multilinear extension over column indices of the column values, so $V_t(u)$ is column u's value for Boolean u. We extend every table by zeros to the widest space, d_{cl} coordinates, write L_t, Q_t and T_t also for the multilinear extensions of their coefficient tensors, and let $\delta(z) = \mathrm{eq}(0, z)$. The claim $Y = \sum_t P_t(v_t)$ is then

$$Y - \sum_t c_t = \sum_{z_0, z_1, z_2 \in \{0,1\}^{d_{\mathrm{cl}}}} \sum_t V_t(z_0)\Big[L_t(z_0)\delta(z_1)\delta(z_2) + V_t(z_1)\big(Q_t(z_0, z_1)\delta(z_2) + V_t(z_2)T_t(z_0, z_1, z_2)\big)\Big].$$

Three sumchecks of d_{cl} rounds each, one per point z_0, z_1 and z_2, prove this identity. Each round polynomial has degree two in its variable and is sent as two values. Once z_0 is bound to its challenges, $V_t(z_0)$ is a fixed factor of the remaining sum, and likewise z_1. The last round leaves each table's columns at the three points, which the prover sends as one fold per aligned column block and side; the folds become claims of the joint main/preprocessed ring switch below and its paired WHIR opening. The verifier evaluates L_t, Q_t and T_t at the three points by running the table's constraints and bus terms once over the points' equality tables.

Cost and scope. The closing replaces the opened values by $6d_{\mathrm{cl}}$ round values and one fold per aligned column block, side and point. In exchange the verifier evaluates every table's constraints over three equality tables, which adds non-free gates. The closing requires that no table read the next row and that the batch have no lookup or indexed argument; the final machine meets both. Lemma 3 gives the algebraic error.

4.3.5 Combining ring-switch claims and shortening bus rounds

Merged ring-switch tensors. Several claims can transmit separate tensor witnesses even when they use the same absorbed coordinates. In this case their column-reading equality vector q and the verifier-derived column-reading functional $\ell_q(T) = \sum_j q_j C_j$ are identical. For tensor witnesses $T_0, \ldots, T_{m-1}$ with incoming claims $y_i = \ell_q(T_i)$, bind the points and claims before sampling $\mu \in \mathbb{F}_{2^{128}}$, and transmit

$$T = \sum_{i=0}^{m-1} \mu^i T_i, \qquad \ell_q(T) = \sum_{i=0}^{m-1} \mu^i y_i.$$

Linearity makes the second equality the randomly weighted combination of the original incoming-claim checks. After the merged witness is absorbed, the transcript derives its row weights and the ring-switch sumcheck binds that row reading to the packed polynomial. The implementation checks the shared-coordinate condition and excludes successor views, the openings of the next row, whose shifted reading needs additional tensors. Sending one tensor witness saves repeated input elements and verification work. The batching error and subsequent authentication are treated in Appendix B.1.

A joint main/preprocessed ring switch. A trace has proof-dependent main columns and fixed preprocessed columns. Opening them through separate ring switches repeats the reduction transcript. We combine equal-size main and preprocessed packings, whose multilinear extensions are P_0 and P_1, respectively. For their common coordinate vector z and a new selector variable s, define

$$P(s, z) = (1 - s)P_0(z) + sP_1(z).$$

The selector s identifies which half of the combined packing is read: the original main and preprocessed claims enter at $s = 0$ and $s = 1$. One ring switch over this enlarged domain reduces the combined claims to a terminal point, where the paired WHIR opening authenticates both P_0 and P_1 and the verifier checks their selector-weighted sum. This removes a second ring-switch transcript while adding one selector coordinate. This operation complements tensor merging: tensor merging combines compatible readings, while the joint switch combines the main and preprocessed packing domains.

Eq-factored bus rounds. The bus product argument reduces a claim about products to claims about their children. In a radix-four layer, where each node of the product tree multiplies four children, the product of four multilinear child values has degree at most four in the current sumcheck variable. Multiplication by the known equality weight raises the round degree to five. We avoid transmitting the extra polynomial data contributed by that known factor. The round polynomial has the form $g(X) = e_r(X)h(X)$, with known linear factor $e_r(X) = (1 - r)(1 - X) + rX$ and $\deg h \leq 4$, where r is the current coordinate of the equality-polynomial challenge point. A degree-five round normally needs five transmitted field elements after using the Boolean-sum constraint. Factoring out e_r leaves a degree-four polynomial, which needs four transmitted elements once the same constraint is used. From the running claim $s = g(0) + g(1)$, the verifier reconstructs the omitted value as

$$h(1) = \frac{s - (1 - r)h(0)}{r}.$$

Since $e_r(0) = 1 - r$ and $e_r(1) = r$, this reconstruction enforces the original Boolean-sum relation. The circuit requires $r \neq 0$. After absorbing the round message, it samples the next sumcheck challenge $\alpha \in \mathbb{F}_{2^{128}}$ and advances the claim to $e_r(\alpha)h(\alpha)$. Each round sends one fewer field element, at the cost of reconstruction arithmetic, including the inverse of r. This saves 128 circuit-input bits per affected round, but need not reduce the non-free gate count. The verified polynomial still has degree five; the shorter message does not lower its soundness error. Appendix B.1 gives the algebraic checks for these reductions.

4.3.6 Commitment parameters and accelerated grinding

Per-round WHIR scheduling. WHIR [ACFY25] repeatedly folds the committed polynomial, authenticates queried codeword values and reduces verification to a smaller polynomial. Its proof input includes opened leaf rows, Merkle siblings, out-of-domain (OOD) answers, sumcheck messages and the final polynomial. OOD answers bind evaluations at points outside the code's evaluation domain. Reducing one component can enlarge another, so we select folding factors, code rates and grinding budgets jointly for the recursive proof's shape.

A larger fold eliminates more variables in one stage and can remove later stages, but can also widen the row opened by each query. A lower code rate adds redundancy, which can reduce the query count required by the security assessment, while enlarging the evaluation domain and its Merkle tree. More grinding requires the prover to search for a transcript nonce, trading hash trials for fewer queries under the chosen assessment. Every removed query saves its leaf payload and

authentication path; the garbled verifier also avoids the corresponding path hashes. We score the complete opening schedule, including round messages and terminal data, rather than minimizing the number of queries alone. Since polynomial arity decreases through the protocol, different folding factors can be preferable in different rounds.

Section 4.3.7 states the rules that derive each round's queries and grinding from its rate and fold. A schedule must fit the actual polynomial arity; the schedules of Table 4 are not a universal setting for other recursion traces.

Shorter Merkle digests. We reduce each Blake3 Merkle digest from 32 bytes to 25 bytes (256 bits to 200 bits). This directly reduces proof input; it does not imply a proportional reduction in hash gates. It also caps generic collision resistance at 100 classical bits and about 66.7 quantum bits [BHT98]. The collision bound that the configuration records for its digests, and the remaining composition terms, are stated in Appendix B.1.

GPU grinding. To reduce the wall-clock cost of the increased grinding budget, the prover prepares the candidate-independent Blake3 transcript state on the host and tests nonce candidates in a CUDA kernel. Every returned witness is rechecked by the CPU verifier. A search spreads over up to four cards: each range of candidates is cut into consecutive slices, one per card, and the smallest passing index of the range is kept. Every index below the range has failed, so the witness, and hence the proof, is the one a single card finds. This accelerates the search without changing the acceptance predicate or removing its logical work. It does not address the memory cost of the enlarged proving domains: a commitment to v variables at log inverse rate ρ has a codeword of $2^{v+\rho}$ field elements.

4.3.7 A model of proof size and proving cost

The commitment parameters fix the WHIR schedule. The schedule fixes the final verifier's input exactly, and the grinding its prover performs in expectation. We state these dependencies as rules; § 5.1 instantiates and validates them.

Schedule. A commitment packs v variables. Its stages $i = 0, \dots, r$, r intermediate and one terminal, fold k_i variables each, leaving $f = v - \sum_i k_i$ for the final polynomial. The prover stops folding at the first stage that leaves at most six variables, so $f \le 6 < f + k_r$. Write $v_i = v - \sum_{j<i} k_j$. Stage i reads a codeword of $2^{v_i+\rho_i}$ positions at log inverse rate ρ_i, with $\rho_{i+1} = \rho_i + k_i - 1$. A query opens a coset of 2^{k_i} positions, which is a leaf of a Merkle tree of height $h_i = v_i + \rho_i - k_i$. The schedule prices every WHIR error term at $\lambda = 108$ bits under the Johnson bound, against a composed target of 100 bits. The proximity parameter is $1 - \delta = (1 + \frac{1}{2m})2^{-\rho/2} = \frac{21}{20}2^{-\rho/2}$ for list parameter $m = 10$, so a query at log inverse rate ρ buys $a_\rho = \rho/2 - \log_2(21/20)$ bits. Each folded variable is ground against the proximity-gap error of the codeword it folds. That error is a line bound of $2^{v_i+2\rho_i+11.59-126}$, where $11.59 = \log_2(8 \cdot 10.5^3/3)$ for the same m and the analysis prices $\mathbb{F}_{2^{128}}$ at 126 bits, two below its size. Each of stage i's k_i folding grinds therefore takes

$$b_i = \lceil v_i + 2\rho_i + 11.59 - (126 - \lambda) \rceil = \lceil v_i + 2\rho_i - 6.41 \rceil \tag{1}$$

bits, and the paired opening adds one more grind at b_0. The allowance $g = \max_i b_i$ leaves $\lambda - g$ bits to the queries:

$$t_i = \left\lceil \frac{\lambda - g}{a_{\rho_i}} \right\rceil, \qquad p_i = \lceil \lambda - t_i a_{\rho_i} \rceil, \tag{2}$$

where p_i is stage i's query grind. A lower rate or a larger early fold raises the later ρ_i and so lowers every later t_i, but by (1) it also raises the late folding grinds, two bits per unit of ρ_i less one per variable folded before, and these dominate the grinding.

Proof input. At the first stage the paired opening reads both the main and the preprocessed tree, so $c_0 = 2$, and $c_i = 1$ afterwards. The WHIR opening carries E_W field elements and D_W digests:

$$E_W = \sum_{i=0}^{r} (c_i t_i 2^{k_i} + s_i) + 2v + n_g + 2^f, \qquad D_W = S(h_0, t_0) + t_0(h_0 - \kappa) + \sum_{i=1}^{r} S(h_i, t_i) + r, \quad (3)$$

$$S(h, t) = \sum_{c\,:\,t_c = 1} 2^c (h - c) \quad \text{for} \quad t = \sum_c t_c 2^c. \quad (4)$$

The elements are the opened cosets, the $s_i = 2$ OOD answers of each commitment, two coefficients per folded variable, one nonce for each of the n_g grinds, and the final polynomial. The queries are drawn in strata: each summand 2^c of t draws one query in every subtree c levels below the root, so the top c bits of that query's index are constants. The verifier reads the query's path up to the node its stratum fixes, $h - c$ siblings, and hashes the tree above those nodes once. The largest summand's nodes fill their level, so the top needs no further input, and S counts the siblings read. The deep cap makes the preprocessed tree's nodes κ levels below its root constants, so each of its queries reads $h_0 - \kappa$ siblings; the roots of the r folded codewords complete D_W. A digest enters as one field element and nine bytes, the 200 bits it keeps, so the final verifier reads

$$I = I_A + 128\, E_W + 200\, D_W \text{ bits}. \quad (5)$$

I_A is the algebraic part: the bus product reduction, the shared and closing sumchecks, their column folds and the ring-switch claim. The machine fixes it, and it does not depend on the WHIR schedule. The serialized proof is not fixed: it carries Plonky3's pruned multiproof, whose shared siblings depend on the drawn indices, and writes field elements as variable-length integers. The verifier takes each query's whole path as hints recovered from it, so its input is the same for every proof.

Proving time. A b-bit grind tests candidates in order until one passes. The witness it returns, the smallest passing index N, is geometric with mean and standard deviation about 2^b. On c cards that each test ν candidates per second, a level takes

$$T = T_{\text{set}} + T_{\text{host}} + \frac{W}{c\nu}, \qquad W = \sum_j N_j, \quad \mathbb{E}[W] = \sum_j 2^{b_j}, \quad \text{Var}[W] \approx \sum_j 4^{b_j}. \quad (6)$$

Each nonce is the number of candidates its search tested, so a proof's nonces calibrate ν in place.

Coupled costs. The input length I sets the on-chain cost: an input-publication scheme spends a fixed γ virtual bytes per input bit, so publication costs γI. Off-chain, a challenger stores 16 bytes per non-free gate and the scheme's translation table, τI bytes. Proving time follows (6). Shortening I moves cost into the other two: the schedule buys fewer queries with exponentially more grinding, a deeper cap removes t_0 digests a level while its selection gates grow as 2^κ, and a wider input digit lowers γ while τ grows exponentially with the digit width.

5 Evaluation

We estimate the cost of the complete WHIR-proof verifier from Ziren's binary stage and narrow recursion using compiled circuit counts, then account for publishing the final verifier input (Table 5). We compare its on-chain data and off-chain storage with BitVM3 and the post-quantum Bitcoin-bridge estimates of Tiwari and Feickert [TF26b] (Table 6). Appendix B gives the recursion details.

We report circuit-input lengths in bits and serialized data sizes in bytes. The prefixes kB, MB and GB denote powers of 10^3; KiB, MiB and GiB denote powers of 2^{10}. Transaction costs use virtual bytes, with $1\,\mathrm{kvB} = 10^3\,\mathrm{vB}$ and $1\,\mathrm{MvB} = 10^6\,\mathrm{vB}$.

5.1 WHIR-proof verification in Ziren

The statement is Ethereum mainnet block 26,206,588, executed by Ziren in 110.8M cycles over 49 shards, segments of the execution proved separately. Ziren's GPU prover proves the shards and compresses them into one binary-stage root proof. Four recursion levels follow (Table 4); each proves an execution of the verifier for the preceding proof. The first two use the plain recursion machine on the GPU, and the last two use narrow recursion with the reductions of § 4.3. We compile and garble the verifier for the final proof.

Verifier cost model. We estimate garbling storage from the compiled verifier's non-free gate count at 16 bytes per gate, and derive input-publication costs from its input encoding. Following the recording and compilation procedure of § 4.1.1, after four recursion levels and the optimizations of § 4.3, the verifier of the last proof has 520,832 input bits, comprising 79,360 bits from byte-valued inputs and 441,472 bits from field-valued inputs, in 175.4M non-free gates and an estimated 2.81 GB of garbled tables. The public-value digest adds at most 1,024 bits (0.20%) to this input. This verifier is compiled from a recorded execution on the block's real final proof. The archived run accepts that proof, rejects three single-input changes, records agreement on all 10,121,640 intermediate values and performs a full streaming garbling of the 175,411,008 non-free gates, in 121 s on one core, with acceptance output 1. These checks exercise the recorded execution and selected mutations. Small synthetic-tape regressions separately check zero inversion, selector and byte ranges, and Blake3 slot padding.

Effect of the final reductions. The dead-operation pass omits 300,771 operations of the final tape, and the deep cap at depth $\kappa = 18$ removes 40,656 input bits from the uncapped final verifier. The closing column sumcheck sends 126 column folds and the round values in place of 1,744 opened values and shrinks the final proof from 102,766 to 79,219 bytes. Without the deep cap, it lowers the final verifier's input from 768,592 to 561,488 bits and raises its size from 125.4M to 176.7M non-free gates.

Proving cost. From the block to the final proof the prover takes 1,174 s of wall time on at most four cards (Table 4), or 1.21 card-hours: 0.10 for the shards and compression and 1.11 for the recursion job. Most of the final proof's time is proof-of-work grinding, whose length depends on where each search's smallest passing nonce falls. The transcript fixes those nonces, so a rerun of the same proof repeats the same searches: an earlier run of the final proof took 738.2 s. The rerun reproduced every recursive proof byte for byte, since the prover, including its multi-card grinding, is deterministic. Another statement draws new nonces, so we give the cost as a formula of the schedule.

stage	schedule	proof	setup	proving
shards and compression	Ziren GPU prover, 2 cards	511,766 B	–	177.7 s
recursion	rate 1/32, fold 3, $\leq$ 33-bit grinding	329,412 B	3.9 s	2.3 s
recursion	rate 1/32, fold 4, $\leq$ 36-bit grinding	321,492 B	2.0 s	2.2 s
narrow recursion	rate 1/256, fold 4, $\leq$ 41-bit grinding	191,241 B	37.5 s	120.4 s
narrow recursion	rate 1/1024, folds 4-5-4-4, $\leq$ 44-bit grinding	79,219 B	70.9 s	752.4 s

Table 4: Proving Ethereum block 26,206,588 to the final proof on RTX 5090 cards and 64 host cores, one row per proof in order. Setup is the time to build a proof's proving keys. The shards-and-compression row is one run on two cards with proving keys built cold (162.4 s with keys reused). The four recursive proofs are one job on four cards, 996 s of wall time including recording and checking each one. The rate-1/32 proofs are made on the cards. The narrow ones commit on the host: their first codewords, 2^{31} and 2^{33} field elements, take 32 and 128 GiB, and the rate-1/1024 one alone would fill all four 32 GB cards. Both grind on all four cards.

Size and cost from the schedule. For the final proof, with $v = 23$ and $\kappa = 18$, (5) and the grinding work of (6) become

$$\begin{aligned} I &= 164{,}328 + 456\,r + 128\Big(\sum_{i=0}^{r} c_i t_i 2^{k_i} + n_g + 2^f\Big) \\ &\quad + 200\Big(S(h_0, t_0) + t_0(h_0 - 18) + \sum_{i=1}^{r} S(h_i, t_i)\Big), \qquad (7) \\ \mathbb{E}[W] &= \sum_{j=1}^{n_g} 2^{b_j}, \qquad 2^g \leq \mathbb{E}[W] \leq n_g\, 2^g, \end{aligned}$$

where W counts Blake3 compressions, one per tested candidate. The constant is $I_A + 128(2v+2) = 158{,}184 + 6{,}144$: the algebraic part, which the machine fixes, the $2v = 46$ sumcheck coefficients and the first commitment's two OOD answers. Each folded codeword adds two OOD answers and a root, 456 bits. The count is exact: it gives the input of every final verifier we compiled, at cap depths 10, 14 and 18 and for both alternative schedules below. Since $t_i = \lceil (\lambda - g)/a_{\rho_i} \rceil$, I is affine in $\lambda - g$, while $\mathbb{E}[W] = \Theta(2^g)$.

The final proof has $\rho_i = 10, 13, 17, 20$, $b = 37, 39, 42, 44$, so $g = 44$, and $t = 13, 10, 8, 7$. Its 22 grinds give $\mathbb{E}[W] = 7.35 \cdot 2^{44} \approx 1.29 \cdot 10^{14}$, 82% of it in the six grinds at 2^{44}, and its nonces came to $0.78\,\mathbb{E}[W]$. They were tested at $4.14 \cdot 10^{10}$ compressions a second per card; at that rate, with both narrow proofs' grinding at its expectation, the block takes 1,369 s and 1.42 card-hours, and 2,136 s at the 99th percentile. A query at stage i costs $128 c_i 2^{k_i}$ bits for its coset and 200 bits a sibling, at least $h_i - \lfloor \log_2 t_i \rfloor$ in a tree and $h_0 - 18$ in the capped preprocessed one: 6.8 to 11.6 thousand bits here. By (2), a stage drops one query for every $a_{\rho_i} = 4.9$ to 9.9 bits by which g rises, about 5,300 input bits per bit of g on average, while $\mathbb{E}[W]$ doubles. Halving the rate makes a query buy half a bit more, but lengthens each path by a sibling and by (1) raises every b_i by two, so $\mathbb{E}[W]$ grows about fourfold. At rate 1/2048, 4-5-4-4 reads 4.6% fewer bits for 4.5 times $\mathbb{E}[W]$, and its first codeword doubles to 2^{34} field elements.

Alternative schedules. We also proved two other final schedules from the same rate-1/256 proof. Folding 4-4-4-5 moves the five-variable fold to the last stage, so by (1) g falls to 43 and the expected grinding halves. Each of the first three stages then needs one more query, so the verifier reads 21,248 more bits (4.1%); the proof took 423.9 s against 752.4 s. Folds of 4-4-4-4 leave seven

input mechanism	inputs	reveal transactions	component (kvB)
Schnorr adaptor, 8-bit baseline	67	12	1,111.863
WOTS translation	443	41	3,933.134
Antichain $(2, 16)$	393	66	5,928.181
Lamport	522	87	8,638.098

Table 5: Input-publication costs for the complete Ziren narrow-recursion verifier, with 520,832 input bits and one input-key set. WOTS translation and Antichain use all-nonzero worst-case encodings. Adaptor, WOTS translation and Antichain have two outputs per reveal transaction; Lamport has one. Funding, completion, Disprove and the remaining dispute graph are excluded. Lamport's separately serialized join adds 6.557 kvB.

variables, so the prover runs its constant schedule 4-4-4-4-4: 15,608 more bits (3.0%) for 2.1 times the expected grinding, proved in 2,953.4 s. Equation (7) gives both inputs without proving either. Over the 8,252 fold sequences of two to eight variables a stage that the prover runs as written, 4-5-4-4 reads the fewest bits at rate 1/1024, and at 1/2048 every schedule that reads fewer bits than it needs at least 1.8 times as much expected grinding.

Independent-proof recorder regression. On an earlier configuration of the recursion, with its digests' collision bound corrected, we generated two proofs of one small binary-machine program using different execution witnesses. Both recordings contain exactly the same 769,910 operations, including operands and constants, with 87,426 input field values. Each tape accepts the other proof's input vector and reproduces all 1,075,048 native intermediate values. Mutating each of the eight public-digest elements separately causes rejection. Boolean evaluation also accepts both input vectors using the first tape's operations, constants and input encoding, matching all native intermediate values for each proof. This tests the recorded program and its Boolean compilation; it does not reuse stored garbled tables.

5.2 Publishing the final verifier input

We serialize the four input-authentication schemes for the final Ziren verifier's 520,832 input bits. Table 5 includes selection witnesses, transaction-binding signatures, authentication scripts, control blocks and the reveal transactions' fixed fields. Every selected leaf is executed under the stack limit. These serializations have not been replayed in Bitcoin Core. Per input bit they cost $\gamma = 2.13$, 7.55, 11.38 and 16.59 vB for the adaptor, WOTS translation, Antichain and Lamport, and the WOTS translation table adds $\tau = 1{,}830$ bytes off-chain.

Transactions, relay and confirmation. Each reveal transaction stays below the 400,000 weight-unit limit of Bitcoin Core's standard relay policy, every witness item is at most 80 bytes, and every leaf runs under the 1,000-item stack limit. The WOTS-translation input therefore needs 41 standard transactions, one more than its size alone requires. One funding transaction creates the 443 publication outputs. The funding transactions measured in Bitcoin Core have one wallet input and one P2TR output per publication input plus change; that shape reproduces their sizes exactly and gives about 19.2 kvB for 443 outputs. In strict Bitcoin Core 31.1 replays of the adaptor, Antichain and Lamport publication slices at the 1,041,024-bit input of an earlier verifier for a Keccak-f permutation trace, Bitcoin Core rejected a full reveal transaction while its funding parent was unconfirmed, because the pair exceeded the mempool's cluster size limit, and accepted every

reveal once the funding had confirmed. It likewise accepted Lamport's join only after the join's reveal parents had confirmed. Reveal transactions near the size limit therefore cannot be relayed as a package with their parents, and they exceed the 10,000 vB limit of version-3 relay [Zha24]. The operator signs each reveal at dispute time and sets its fee then. Publication takes one block for funding and at least four full blocks of reveals, 15.73M weight units in all, since a block holds at most 4×10^6 weight units, and the completion join follows in a later block; Δ_A must cover these blocks and the competition for block space. The 97 vB Disprove transaction, a SHA256 hashlock on the 16-byte reject label next to its timeout branch, was accepted and mined under strict Bitcoin Core 31.1 policy; the challenger builds it and pays its fee. A challenged dispute thus needs about 46 transactions: Kickoff, Challenge, funding, 41 reveals, the completion join, and Disprove or Take2. Of these, the 41 reveals and Disprove are serialized, and the funding size is derived. An unchallenged claim needs only Kickoff and Take1.

WOTS translation requires a 953,151,840-byte off-chain table, and the label ciphertexts of § 4.2, two per input bit, occupy 16.67 MB. The operator garbles one instance, so these figures and the on-chain components of Table 5 are not multiplied by a number of kept garblings.

5.3 On-chain and off-chain cost comparison

Table 6 compares our approach with Boolean-garbled Groth16 verification in BitVM3 and the post-quantum Bitcoin-bridge analysis of Tiwari and Feickert [TF26b], separating on-chain data from off-chain storage. The Tiwari–Feickert column is the combined-cost example of their Section 3.5: RISC Zero's verifier with Blake3 as its internal hash, at an estimated 1.56 billion non-free gates, its proof's length as the input, and Antichain Winternitz (ACW) for input-label revelation.

Off-chain storage. The Ziren verifier's garbled tables are about 15.5× smaller than the Boolean Groth16 baseline's. The Tiwari–Feickert ACW integration also requires a post-quantum proof binding the translation table to the intended labels. Its cost is outside these table sizes. The proof systems and input-label mechanisms differ; the table compares their complete reported storage costs rather than matching individual components.

On-chain data. Even the 1.11 MvB adaptor baseline for the Ziren input remains far larger than BitVM3's accounted subtotal. Tiwari and Feickert's ACW estimate, like their 31.7 MvB for Lamport, excludes fixed transaction overheads; our serialized Antichain and Lamport publications, overheads included, take 5.93 and 8.64 MvB. Their 0.043 kvB witness contribution for a hashlock fault signal is not a complete Disprove transaction, so it is not entered in that row of Table 6. The comparison therefore identifies two distinct costs: Boolean compilation reduces the verifier's garbled tables, while the authenticated proof input remains the dominant measured on-chain component. Section 4.3 reduces both through narrower recursion, proof-layout changes and WHIR parameter selection.

Cost component	Boolean Groth16 (BitVM3) [WAA+26a, GKS+26]	PQ bridge analysis [TF26b]	**This work**
PQ-safe verifier assumptions	No (pairings)	Yes (hash-based)	Yes (hash-based)
Verifier input bits	1,019	$\approx 1.78 \times 10^6$	520,832
On-chain data: one input-key set			
Input publication (kvB)	≈ 2.4 (Assert; adaptor)	$\approx 14{,}900$ (ACW)	3,933.13 (WOTS translation)
Additional verification transactions (kvB)	0.093 (Disprove)	—	0.097 (Disprove)
Accounted transaction subtotal (kvB)	≈ 2.5	—	3,933.231
Off-chain storage			
Garbled tables per instance (GiB)	40.5	23.3 (estimate)	2.61
Input-label translation tables (GiB)	—	72.7 (ACW)	0.89 (WOTS)
Total tables per instance (GiB)	—	≈ 96.0	3.50
Garbled instances stored	7	—	1

Table 6: On-chain data and off-chain storage for Bitcoin proof verification. A dash marks an unreported or inapplicable entry. PQ-safe concerns the proof verifier and its garbling only (§ 6.1). The subtotal counts input publication and Disprove only; BitVM3 pads its proof to 1,024 bits. Garbled tables are 16 bytes per non-free gate in every column (§ 3.1). BitVM3 keeps seven of 181 garblings after cut-and-choose; this work stores the one garbling that the operator proves correct, and the cost of that proof is not included.

6 Security

This section states what the garbled verifier guarantees and on which assumptions. It distinguishes the recording and compilation assumptions of the Ziren path, combines proof-system soundness with garbling authenticity, and then accounts for the dispute of § 2.1.2.

The circuit and the verifier. For the Ziren path, the argument assumes that the recorder captures every check of the reference verifier and that the Boolean compilation preserves the tape's semantics. The compiler enforces the tape's equality and nonzero assertions and the ranges of byte and selector operands. The theorem below states recorder fidelity as an assumption; the tests of § 5 are implementation evidence for it.

In the theorem below, $\varepsilon_{\text{guard}}$ denotes the completeness loss introduced by the circuit compilation and additional protocol guards. The eq-factored bus guard is analyzed in Appendix B.1; a bound for the complete recorded verifier requires accounting for all guards.

Soundness of the proof system. We use $\varepsilon_{\text{STARK}}(t)$ for the composed soundness error against an adversary making t hash queries. This parameter includes the preceding proof layers, the zkVM proof, algebraic composition losses and hash-related errors. Section 6.1 gives the QROM accounting, and the theorem below is conditional on this composed soundness.

Authenticity of the garbling. The scheme of § 3.1 uses H = Blake3 and a fresh secret Δ per garbling. Free-XOR is secure when H is circular correlation robust [CKKZ12]; we assume this of Blake3, as is standard for hash-based garbling. In the BitVM setting the garbled circuit is published at setup and the proof is chosen afterwards, so authenticity must hold for inputs chosen adaptively, after the garbled circuit is seen [BHR12a]. Let $\varepsilon_{\mathrm{GC}}$ be the advantage of an evaluator that sees the garbled circuit, then chooses an input x, receives its labels, and outputs the label of the value $\neg C(x)$ on the output wire. We assume that our Blake3-based free-XOR garbling has adaptive authenticity in this sense, with Blake3 modeled as a random oracle; Bellare, Hoang and Rogaway give constructions proven adaptively secure [BHR12a].
The following circuit-level statement is conditional on the Blake3 circular-correlation-robustness and adaptive-authenticity assumptions stated above, and assumes that the circuit compilation is faithful to its reference verifier with completeness loss at most $\varepsilon_{\mathrm{guard}}$. It does not cover the proof of correct garbling or Bitcoin transaction enforcement.

Theorem 1. *Let C be a correctly garbled verifier of a fixed configuration, and x the encoding of a proof π for a statement.*

(i) Soundness. *If the statement is false, an evaluator holding the labels of x obtains the output label of value 0, except with probability at most $\varepsilon_{\mathrm{STARK}}(t)$, where t bounds the hash queries of the party that chose π.*

(ii) Authenticity. *If π is honestly generated, an evaluator holding the labels of x obtains the output label of value 0 with probability at most $\varepsilon_{\mathrm{guard}} + \varepsilon_{\mathrm{GC}}$.*

Proof sketch. (i) If $C(x) = 1$, the reference verifier accepts π by the previous paragraphs, which for a false statement happens with probability at most $\varepsilon_{\mathrm{STARK}}(t)$. Otherwise $C(x) = 0$, and correctness of the garbling yields the label of value 0. (ii) For an honest proof, the assumed completeness loss bounds $\Pr[C(x) = 0]$ by $\varepsilon_{\mathrm{guard}}$. When $C(x) = 1$, the label of value 0 is the label of $\neg C(x)$, which the evaluator obtains with probability at most $\varepsilon_{\mathrm{GC}}$. □

The theorem applies to the relation represented by C: validity of the final recursive proof, with the preceding verification computations represented through recursion. Applying the theorem to the underlying zkVM statement requires the composed soundness error of the entire proof chain.

The dispute. Let $\varepsilon_{\mathrm{gp}}$ bound the probability that the operator's proof of correct garbling is accepted although its commitments do not come from a correct garbling of C; it is the soundness error of that proof system. Let $\varepsilon_{\mathrm{sig}}$ bound the aggregate probability of forging any one-time or transaction-authorization signature across the entire dispute, so a reduction from a per-target bound must include the number of targets. Cut-and-choose that keeps M of N instances has error $1/\binom{N}{M}$, which is 2^{-40} at $(N, M) = (181, 7)$ [GKS+26]; the malicious-garbler term $\varepsilon_{\mathrm{gp}}$ can be set to the soundness target of the verified proof.

Subject to an enforced transaction graph, timely confirmation and an instantiated garbling proof, Theorem 1 supports two intended properties of the dispute of § 2.1.2. *A false claim is disproved:* if the operator's statement is false, an honest challenger obtains ℓ_0 except with probability $\varepsilon_{\mathrm{STARK}}(t) + \varepsilon_{\mathrm{gp}}$, since the garbling is then correct, its labels are those of the published proof, and Theorem 1(i) applies; an operator that publishes no proof loses its stake to AssertTimeout. *An honest operator keeps its stake:* if the proof is honest, no challenger obtains ℓ_0 except with probability $\varepsilon_{\mathrm{guard}} + \varepsilon_{\mathrm{GC}} + \varepsilon_{\mathrm{sig}}$, by Theorem 1(ii), provided the garbling proof reveals nothing about the labels. This is conditional accounting, not a protocol theorem in a ledger model: the garbling proof is not constructed, so $\varepsilon_{\mathrm{gp}}$ has no instantiated bound, and the transaction graph

and ledger-timing assumptions have not been implemented or proved. BitVM3 [WAA+26a] and BABE [GKS+26] analyze their own instantiations.

6.1 Post-quantum analysis

The binary-field verifier uses hash-based commitments, a hash-based transcript and hash-based garbling. We analyze these components first, then address input publication and Bitcoin transaction authorization separately. The Ziren recursion path uses 200-bit Merkle digests; its soundness must also account for the earlier proof layers. The estimates are conditional on the stated random-oracle, round-by-round-soundness and adaptive-authenticity assumptions. Pairing-based verifiers (Groth16, Pari and their designated-verifier variants), the witness-encryption schemes built on them (§ 2) and Bitcoin's own Schnorr signatures fall to Shor's algorithm [Sho94]. Avoiding discrete-logarithm and pairing assumptions is necessary for quantum resistance here, but a hash-based verifier is not sufficient by itself: a quantum adversary still speeds up generic preimage and collision search [Gro96, BHT98]. Table 7 gives component-level estimates; Appendix C derives them. We write λ_Q for the desired quantum security level in bits.

The proof system and the garbling. For a suitable round-by-round-sound protocol, a Fiat–Shamir reduction in the quantum random-oracle model bounds success after t queries by $O(t^2\varepsilon_{\mathrm{rbr}} + t^3/2^\lambda)$ [CMS19], where $\varepsilon_{\mathrm{rbr}}$ is the interactive protocol's round-by-round soundness error and λ is the hash-output length in bits for the hash modeled in the reduction. If $\varepsilon_{\mathrm{rbr}} = 2^{-a}$, the first term becomes order one near $t = 2^{a/2}$, suppressing constants: the scale at which that bound ceases to certify small error (Appendix C). Instantiating this bound for the complete recursive verifier requires the composition analysis in Appendix B.1. Grinding is a separate generic search problem: a b-bit predicate can be satisfied in about $2^{b/2}$ quantum queries. For the garbling, an evaluator can test candidates for the 128-bit offset Δ against gate data; generic search has an approximately 2^{64} query scale. A hashlock on a 128-bit output label has the same single-target search scale.

The surrounding protocol. Our optimization scope covers the proof verifier and authenticated delivery of its input, rather than Bitcoin's transaction-graph authorization. Current bridge graphs use MuSig2/Schnorr authorization, which falls to Shor's algorithm. A complete post-quantum bridge therefore also depends on a future post-quantum replacement for Bitcoin's MuSig2 authorization. Hash-based input keys avoid the adaptor-signature dependency, and the proof of correct garbling must be hash-based as well. The keys themselves are bounded by multi-target preimage search over the T alternative Lamport preimages of one input (Table 7, Appendix C).

component	basis	classical estimate	quantum scale	candidate changes for λ_Q bits
WHIR and algebraic reductions	composed soundness	–	–	instantiate the full recursive reduction
Merkle hash (200-bit digest)	generic collision search	100	≈ 66.7	$\gtrsim 3\lambda_Q$-bit digest and sufficiently wide internal state
Blake3 transcript (256-bit digest)	generic collision search	128	≈ 85.3	wider digest and internal state for higher targets
garbling (Δ, output hashlock)	Grover on 128-bit labels	128	≈ 64	$2\lambda_Q$-bit labels: 256 bits
Lamport input keys	multi-target preimage, $T = 520{,}832$	≈ 109.0	≈ 54.5	at least 219 bits of preimage entropy for 100 quantum bits under a 256-bit hash
proof of correct garbling	soundness of its proof system	–	–	a hash-based proof at the target error; not constructed
Bitcoin graph authorization (MuSig2/Schnorr)	Shor	–	broken	post-quantum replacement for MuSig2

Table 7: Component-level estimates for the final Ziren configuration, in bits. Dashes denote an uninstantiated composed soundness bound. Hash and garbling rows describe generic search scales; they do not certify the complete protocol. Candidate parameter changes are design targets. The WHIR-composition and Bitcoin- authorization rows identify future work outside the measured verifier.

7 Conclusion

We have presented a Boolean compilation and cost analysis for Ziren's complete recursive WHIR-proof verifier. With narrow recursion and the combined proof-layout and opening reductions and a closing column sumcheck, the final verifier for an Ethereum block consumes 520,832 input bits and contains 175.4M non-free gates and 2.81 GB of garbled tables. The measured publication cost is 3.93 MvB in 41 standard-size transactions with translated Winternitz keys. These results show that binary-field verification can reduce the stored garbled verifier, while authenticated proof input remains the dominant accounted on-chain cost. That cost is linear in the input, so the reductions that shorten the input pay for it in grinding or gates: the closing column sumcheck removed 27% of the uncapped input for 41% more non-free gates, and folding 4-5-4-4 instead of 4-4-4-5 removed 3.9% for 2.1 times the expected grinding. Choosing the schedule, cap depth and input encoding against these relations is a step toward practical post-quantum Bitcoin bridges.

The verifier supports any off-chain computation proved by Ziren, including rollup withdrawals. The operator's proof that its single garbling is correct, and the complete transaction graph, remain to be constructed; the cost of proving the garbling of 175.4M non-free gates and the input label ciphertexts is not measured here.

Future work should reduce authenticated input through segmented or locally refutable verification and reduce retained storage without introducing an elliptic-curve proof wrapper. A short-key delivery of the selected labels, as TinyLabels gives under Ring-LWE in $|x| + \mathrm{poly}(\lambda)$ bits [DLL24], would replace the per-bit reveals by the raw proof and a key of tens of kilobytes; what it lacks is public verifiability, and since the operator commits to the hash of every label at setup, a Disprove branch that decodes one wire in Script and shows the result matches neither committed hash would supply it. It should also instantiate the composed soundness bound for the recursive WHIR proof and provide a post-quantum replacement for the MuSig2/Schnorr authorization used by current

Bitcoin bridge graphs. All components must ultimately be evaluated at a common security target, including recursive-proof soundness, garbling, input authentication, the proof of correct garbling and transaction authorization.

References

[ACFY24] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. STIR: Reed–Solomon proximity testing with fewer queries. In *Advances in Cryptology – CRYPTO 2024, Part X*, volume 14929 of *LNCS*, pages 380–413. Springer, 2024. Full version: Cryptology ePrint Archive, Paper 2024/390. https://doi.org/10.1007/978-3-031-68403-6_12.

[ACFY25] Gal Arnon, Alessandro Chiesa, Giacomo Fenzi, and Eylon Yogev. WHIR: Reed–Solomon proximity testing with super-fast verification. In *Advances in Cryptology – EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 214–243. Springer, 2025. Cryptology ePrint Archive, Paper 2024/1586. https://doi.org/10.1007/978-3-031-91134-7_8.

[ACG+26] Michel Abdalla, Brent Carmer, Muhammed El Gebali, Handan Kilinc-Alper, Mikhail Komarov, Yaroslav Rebenko, Lev Soukhanov, Erkan Tairi, Elena Tatuzova, and Patrick Towa. Bitcoin PIPEs v2. Cryptology ePrint Archive, Paper 2026/186, 2026. https://eprint.iacr.org/2026/186.

[BHD24] Hunter Beast, Ethan Heilman, and Isabel Foxen Duke. BIP 360: Pay-to-Merkle-Root (P2MR). Bitcoin Improvement Proposal, 2024. Draft, accessed 12 November 2026. https://github.com/bitcoin/bips/blob/master/bip-0360.mediawiki.

[BHR12a] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Adaptively secure garbling with applications to one-time programs and secure outsourcing. In *Advances in Cryptology – ASIACRYPT 2012*, volume 7658 of *LNCS*, pages 134–153. Springer, 2012. https://doi.org/10.1007/978-3-642-34961-4_10.

[BHR12b] Mihir Bellare, Viet Tung Hoang, and Phillip Rogaway. Foundations of garbled circuits. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, pages 784–796. ACM, 2012. https://doi.org/10.1145/2382196.2382279.

[BHT98] Gilles Brassard, Peter Høyer, and Alain Tapp. Quantum cryptanalysis of hash and claw-free functions. In *LATIN 1998: Theoretical Informatics*, volume 1380 of *LNCS*, pages 163–169. Springer, 1998. https://doi.org/10.1007/BFb0054319.

[BIJ+20] James Bartusek, Yuval Ishai, Aayush Jain, Fermi Ma, Amit Sahai, and Mark Zhandry. Affine determinant programs: A framework for obfuscation and witness encryption. In *11th Innovations in Theoretical Computer Science Conference (ITCS 2020)*, volume 151 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 82:1–82:39. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. https://doi.org/10.4230/LIPIcs.ITCS.2020.82.

[Bit24] Bitcoin Wildlife Sanctuary contributors. Circle STARK Verifier in Bitcoin Script. GitHub repository, 2024. https://github.com/Bitcoin-Wildlife-Sanctuary/bitcoin-circle-stark.

[Bit25] BitVM. Garbled SNARK verifier. Source code, commit 5a2cd4dc6cb19e37adb1b3ab94414e01d1e8b338, 2025. https://github.com/BitVM/garbled-snark-verifier/tree/5a2cd4dc6cb19e37adb1b3ab94414e01d1e8b338.

[Can89] David G. Cantor. On arithmetical algorithms over finite fields. *Journal of Combinatorial Theory, Series A*, 50(2):285–300, 1989. https://doi.org/10.1016/0097-3165(89)90020-4.

[CBBZ23] Binyi Chen, Benedikt Bünz, Dan Boneh, and Zhenfei Zhang. HyperPlonk: Plonk with linear-time prover and high-degree custom gates. In *Advances in Cryptology – EUROCRYPT 2023, Part II*, volume 14005 of *LNCS*, pages 499–530. Springer, 2023. Full version: Cryptology ePrint Archive, Paper 2022/1355. https://doi.org/10.1007/978-3-031-30617-4_17.

[Che25] Weikeng Chen. SoK: BitVM with succinct on-chain cost. Cryptology ePrint Archive, Paper 2025/1253, 2025. https://eprint.iacr.org/2025/1253.

[cHK+26] conduition, Ethan Heilman, Mikhail Kudinov, Oleksandr Kurbatov, Jonas Nick, and remix7531. SHRINCS: A compact hash-based signature scheme. Draft specification, 2026. Accessed 12 November 2026. https://github.com/SHRINCS/shrincs-bip/blob/main/SHRINCS.md.

[CKKZ12] Seung Geol Choi, Jonathan Katz, Ranjit Kumaresan, and Hong-Sheng Zhou. On the security of the "free-XOR" technique. In *Theory of Cryptography Conference (TCC 2012)*, volume 7194 of *LNCS*, pages 39–53. Springer, 2012. https://doi.org/10.1007/978-3-642-28914-9_3.

[CMS19] Alessandro Chiesa, Peter Manohar, and Nicholas Spooner. Succinct arguments in the quantum random oracle model. In *Theory of Cryptography Conference (TCC 2019), Part II*, volume 11892 of *LNCS*, pages 1–29. Springer, 2019. Cryptology ePrint Archive, Paper 2019/834. https://doi.org/10.1007/978-3-030-36033-7_1.

[DLL24] Marian Dietz, Hanjun Li, and Huijia Lin. TinyLabels: How to compress garbled circuit input labels, efficiently. Cryptology ePrint Archive, Paper 2024/2048, 2024.

[DMS24] Michel Dellepere, Pratyush Mishra, and Alireza Shirzad. Garuda and Pari: Faster and smaller SNARKs via equifficient polynomial commitments. Cryptology ePrint Archive, Paper 2024/1245, 2024. https://eprint.iacr.org/2024/1245.

[DP25] Benjamin E. Diamond and Jim Posen. Succinct arguments over towers of binary fields. In *Advances in Cryptology – EUROCRYPT 2025, Part IV*, volume 15604 of *LNCS*, pages 93–122. Springer, 2025. Full version: Cryptology ePrint Archive, Paper 2023/1784. https://doi.org/10.1007/978-3-031-91134-7_4.

[DP26] Benjamin E. Diamond and Jim Posen. Polylogarithmic proofs for multilinears over binary towers. In *Advances in Cryptology – EUROCRYPT 2026, Part VII*, volume

16547 of *LNCS*, pages 3–32. Springer, 2026. Full version: Cryptology ePrint Archive, Paper 2024/504. https://doi.org/10.1007/978-3-032-25336-1_1.

[DYH+26] Stephen Duan, Jingping Yin, Blake Hou, Vanhger Pham, Ethan Zhu, and Ming Guo. Ziren: Succinct arguments for MIPS32 execution. Cryptology ePrint Archive, Paper 2026/2330, 2026. https://eprint.iacr.org/2026/2330.

[Eag25] Liam Eagen. Glock: Garbled locks for Bitcoin. Cryptology ePrint Archive, Paper 2025/1485, 2025. https://eprint.iacr.org/2025/1485.

[EL26] Liam Eagen and Ying Tong Lai. Argo MAC: Garbling with elliptic curve MACs. Cryptology ePrint Archive, Paper 2026/049, 2026. https://eprint.iacr.org/2026/049.

[FBF+26] Ariel Futoransky, Fadi Barbàra, Ramses Fernandez, Gabriel Larotonda, and Sergio Demian Lerner. TRAPGC-DV: Trapdoor garbled circuit designated verification of Groth16 on Bitcoin. Cryptology ePrint Archive, Paper 2026/1430, 2026. https://eprint.iacr.org/2026/1430.

[FLB25] Ariel Futoransky, Gabriel Larotonda, and Fadi Barbara. A note on the security of the BitVM3 garbling scheme. Cryptology ePrint Archive, Paper 2025/1291, 2025. https://eprint.iacr.org/2025/1291.

[FNO15] Tore Kasper Frederiksen, Jesper Buus Nielsen, and Claudio Orlandi. Privacy-free garbled circuits with applications to efficient zero-knowledge. In *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 191–219. Springer, 2015. https://doi.org/10.1007/978-3-662-46803-6_7.

[GGSW13] Sanjam Garg, Craig Gentry, Amit Sahai, and Brent Waters. Witness encryption and its applications. In *45th ACM STOC*, pages 467–476. ACM, 2013. https://doi.org/10.1145/2488608.2488667.

[GKPW24] Sanjam Garg, Dimitris Kolonelos, Guru-Vamsi Policharla, and Mingyuan Wang. Threshold encryption with silent setup. In *Advances in Cryptology – CRYPTO 2024, Part VII*, volume 14926 of *LNCS*, pages 352–386. Springer, 2024. https://doi.org/10.1007/978-3-031-68394-7_12.

[GKS23] Lorenzo Grassi, Dmitry Khovratovich, and Markus Schofnegger. Poseidon2: A faster version of the Poseidon hash function. Cryptology ePrint Archive, Paper 2023/323, 2023. https://eprint.iacr.org/2023/323.

[GKS+26] Sanjam Garg, Dimitris Kolonelos, Mikhail Sergeevitch, Srivatsan Sridhar, and David Tse. BABE: Verifying proofs on Bitcoin made 1000x cheaper. Cryptology ePrint Archive, Paper 2026/065, 2026. https://eprint.iacr.org/2026/065.

[GOA26] GOAT Network. bitvm-gc. GitHub repository, 2026. https://github.com/GOATNetwork/bitvm-gc.

[Gro96] Lov K. Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the 28th Annual ACM Symposium on Theory of Computing*, pages 212–219. ACM, 1996. https://doi.org/10.1145/237814.237866.

[Gro16] Jens Groth. On the size of pairing-based non-interactive arguments. In *Advances in Cryptology – EUROCRYPT 2016, Part II*, volume 9666 of *LNCS*, pages 305–326. Springer, 2016. https://doi.org/10.1007/978-3-662-49896-5_11.

[KFLT26] Nakul Khambhati, Aaron Feickert, Christian Lewe, and Mukesh Tiwari. Garbling Groth16 with native group operations. Cryptology ePrint Archive, Paper 2026/2100, 2026. https://eprint.iacr.org/2026/2100.

[KLN25] Victor I. Kolobov, Avihu M. Levy, and Moni Naor. ColliderVM: Stateful computation on Bitcoin without fraud proofs. Cryptology ePrint Archive, Paper 2025/591, 2025. https://eprint.iacr.org/2025/591.

[KS08] Vladimir Kolesnikov and Thomas Schneider. Improved garbled circuit: Free XOR gates and applications. In *Automata, Languages and Programming (ICALP 2008), Part II*, volume 5126 of *LNCS*, pages 486–498. Springer, 2008. https://doi.org/10.1007/978-3-540-70583-3_40.

[KTA+26] Nakul Khambhati, Mukesh Tiwari, Azz, Sapin Bajracharya, Manish Bista, Liam Eagen, Christian Lewe, and Aaron Feickert. Mosaic: Practical malicious security for garbled circuits on Bitcoin. Cryptology ePrint Archive, Paper 2026/812, 2026. https://eprint.iacr.org/2026/812.

[Lam79] Leslie Lamport. Constructing digital signatures from a one way function. Technical Report CSL-98, SRI International, 1979. https://lamport.azurewebsites.net/pubs/dig-sig.pdf.

[LAM+24] Sergio Demian Lerner, Ramon Amela, Shreemoy Mishra, Martin Jonas, and Javier Álvarez Cid-Fuentes. BitVMX: A CPU for universal computation on Bitcoin. arXiv preprint arXiv:2405.06842, 2024. https://arxiv.org/abs/2405.06842.

[LCH14] Sian-Jheng Lin, Wei-Ho Chung, and Yunghsiang S. Han. Novel polynomial basis and its application to Reed–Solomon erasure codes. In *IEEE 55th Annual Symposium on Foundations of Computer Science (FOCS)*, pages 316–325. IEEE, 2014. https://doi.org/10.1109/FOCS.2014.41.

[Ler26] Sergio Demian Lerner. Verifiable data on Bitcoin: FlatMerkle commitments, refutable proofs, and FlatChecks. Cryptology ePrint Archive, Paper 2026/2717, 2026. https://eprint.iacr.org/2026/2717.

[Lev26] Avihu Mordechai Levy. Quantum-safe Bitcoin transactions without softforks. Technical report, https://github.com/avihu28/Quantum-Safe-Bitcoin-Transactions/blob/main/paper/QSB.pdf, 2026. Source code: https://github.com/avihu28/Quantum-Safe-Bitcoin-Transactions.

[LFKN92] Carsten Lund, Lance Fortnow, Howard Karloff, and Noam Nisan. Algebraic methods for interactive proof systems. *Journal of the ACM*, 39(4):859–868, 1992. https://doi.org/10.1145/146585.146605.

[LLW15] Eric Lombrozo, Johnson Lau, and Pieter Wuille. BIP 141: Segregated witness (consensus layer). Bitcoin Improvement Proposal, 2015. https://github.com/bitcoin/bips/blob/master/bip-0141.mediawiki.

[NRJ22] Jonas Nick, Tim Ruffing, and Elliott Jin. BIP 327: MuSig2 for BIP340-compatible multi-signatures. Bitcoin Improvement Proposal, 2022. Deployed. https://github.com/bitcoin/bips/blob/master/bip-0327.mediawiki.

[OANWO20] Jack O'Connor, Jean-Philippe Aumasson, Samuel Neves, and Zooko Wilcox-O'Hearn. BLAKE3: One function, fast everywhere. https://github.com/BLAKE3-team/BLAKE3-specs, 2020.

[Pha26] Vanhger Pham. Deferred binding: Extending BABE for dynamic public inputs in GOAT BitVM3. GOAT Research, HackMD, 2026. Accessed 5 November 2026. Implementation: GOATNetwork/bitvm2-gc, branch feat/goat-bitvm3, commit c23130a. https://hackmd.io/@goatresearch/HkKp2g1Zfl.

[Plo26] Plonky3 contributors. Plonky3. GitHub repository, 2026. https://github.com/Plonky3/Plonky3.

[Rub25] Jeremy Rubin. Delbrag. Draft, April 2025. https://rubin.io/public/pdfs/delbrag.pdf.

[Sho94] Peter W. Shor. Algorithms for quantum computation: Discrete logarithms and factoring. In *35th Annual Symposium on Foundations of Computer Science*, pages 124–134. IEEE, 1994. https://doi.org/10.1109/SFCS.1994.365700.

[SRG+26] Lev Soukhanov, Yaroslav Rebenko, Muhammad El Gebali, Mikhail Komarov, Handan Kilinc-Alper, Michel Abdalla, and Patrick Towa. Implementable witness encryption from arithmetic affine determinant programs. Cryptology ePrint Archive, Paper 2026/175, 2026. https://eprint.iacr.org/2026/175.

[SST+26] Mikhail Sergeevitch, Konrad Staniec, David Tse, Nikhil Vanjani, and Robin Linus Woll. Non-interactive translation of Winternitz signatures to Lamport signatures via secret sharing. Cryptology ePrint Archive, Paper 2026/1684, 2026. https://eprint.iacr.org/2026/1684.

[Sta24] StarkWare. The path to general computation on bitcoin. StarkWare Research, July 2024. https://starkware.co/blog/general-computation-on-bitcoin/.

[Sta26] StarkWare. The first quantum-safe Bitcoin transaction has been mined. https://starkware.co/blog/the-first-quantum-safe-bitcoin-transaction-has-been-mined/, August 2026. Mainnet transaction: https://mempool.space/tx/305a24ffea912b9cf428f29ebf952321c96dab5bab284fc0d0801562f5abab07.

[TF26a] Mukesh Tiwari and Aaron Feickert. Antichain Winternitz: Guaranteed garbled-circuit label revelation on Bitcoin with permissionless recovery. Cryptology ePrint Archive, Paper 2026/1568, 2026. https://eprint.iacr.org/2026/1568.

[TF26b] Mukesh Tiwari and Aaron Feickert. The post-quantum cost of garbled-circuit Bitcoin bridges. Cryptology ePrint Archive, Paper 2026/2383, 2026. https://eprint.iacr.org/2026/2383.

[WAA+26a] Robin Linus Woll, Ioannis Alexopoulos, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, and David Tse. BitVM3: Efficient Bitcoin bridges via garbled circuits. Cryptology ePrint Archive, Paper 2026/933, 2026. https://eprint.iacr.org/2026/933.

[WAA+26b] Robin Linus Woll, Lukas Aumayr, Zeta Avarikioti, Matteo Maffei, Andrea Pelosi, Orfeas Stefanos Thyfronitis Litos, Christos Stefo, David Tse, and Alexei Zamyatin. Bridging Bitcoin to second layers via BitVM2. In *35th USENIX Security Symposium (USENIX Security 26)*, pages 969–988. USENIX Association, 2026. Full version: Cryptology ePrint Archive, Paper 2025/1158. `https://www.usenix.org/conference/usenixsecurity26/presentation/woll`.

[Wie88] Doug Wiedemann. An iterated quadratic extension of GF(2). *The Fibonacci Quarterly*, 26(4):290–295, 1988. `https://www.fq.math.ca/Scanned/26-4/wiedemann.pdf`.

[WNR20] Pieter Wuille, Jonas Nick, and Tim Ruffing. BIP 340: Schnorr signatures for secp256k1. Bitcoin Improvement Proposal, 2020. `https://github.com/bitcoin/bips/blob/master/bip-0340.mediawiki`.

[WNT20a] Pieter Wuille, Jonas Nick, and Anthony Towns. BIP 341: Taproot: SegWit version 1 spending rules. Bitcoin Improvement Proposal, 2020. `https://github.com/bitcoin/bips/blob/master/bip-0341.mediawiki`.

[WNT20b] Pieter Wuille, Jonas Nick, and Anthony Towns. BIP 342: Validation of Taproot scripts. Bitcoin Improvement Proposal, 2020. Defines Tapscript signature semantics and resource limits. `https://github.com/bitcoin/bips/blob/master/bip-0342.mediawiki`.

[Zha24] Gloria Zhao. BIP 431: Topology restrictions for pinning. Bitcoin Improvement Proposal, 2024. `https://github.com/bitcoin/bips/blob/master/bip-0431.mediawiki`.

[ZRE15] Samee Zahur, Mike Rosulek, and David Evans. Two halves make a whole: Reducing data transfer in garbled circuits using half gates. In *Advances in Cryptology – EUROCRYPT 2015, Part II*, volume 9057 of *LNCS*, pages 220–250. Springer, 2015. `https://doi.org/10.1007/978-3-662-46803-6_8`.

A Bitcoin integration details

This appendix details the label-selection mechanisms summarized in § 4: the Schnorr adaptor baseline, hash-based label selection, its post-quantum instantiation and the four input-publication schemes with the transactions we built for them.

A.1 Schnorr adaptor baseline

Figure 3 details the Schnorr adaptor label-selection baseline of BitVM3 and Mosaic [WAA+26a, KTA+26]. Let G be the group generator of order q, let $t_{i,v}$ be the adaptor scalar for candidate digit v, and let v_j be bit j of v. We write $\mathrm{x}(P)$ for a point's x-coordinate. As in BIP340 [WNR20], X has even y, $\sigma_v \in \{-1, 1\}$ makes R'_v have even y, and $e_v = H_{\mathrm{BIP340}}(\mathrm{x}(R'_v) \,\|\, \mathrm{x}(X) \,\|\, m_i)$. m_i is the BIP341 transaction message seen by the i-th `CHECKSIG` [WNT20a]. For fixed i, the diagram suppresses the index i on $R'_v, s'_v, \sigma_v, e_v$. Every pre-signature uses an independent nonce across candidate values, digit positions and sessions; zero nonce sums and duplicate candidate $\mathrm{x}(R'_v)$ values are resampled, and the candidate nonce list is public at setup.

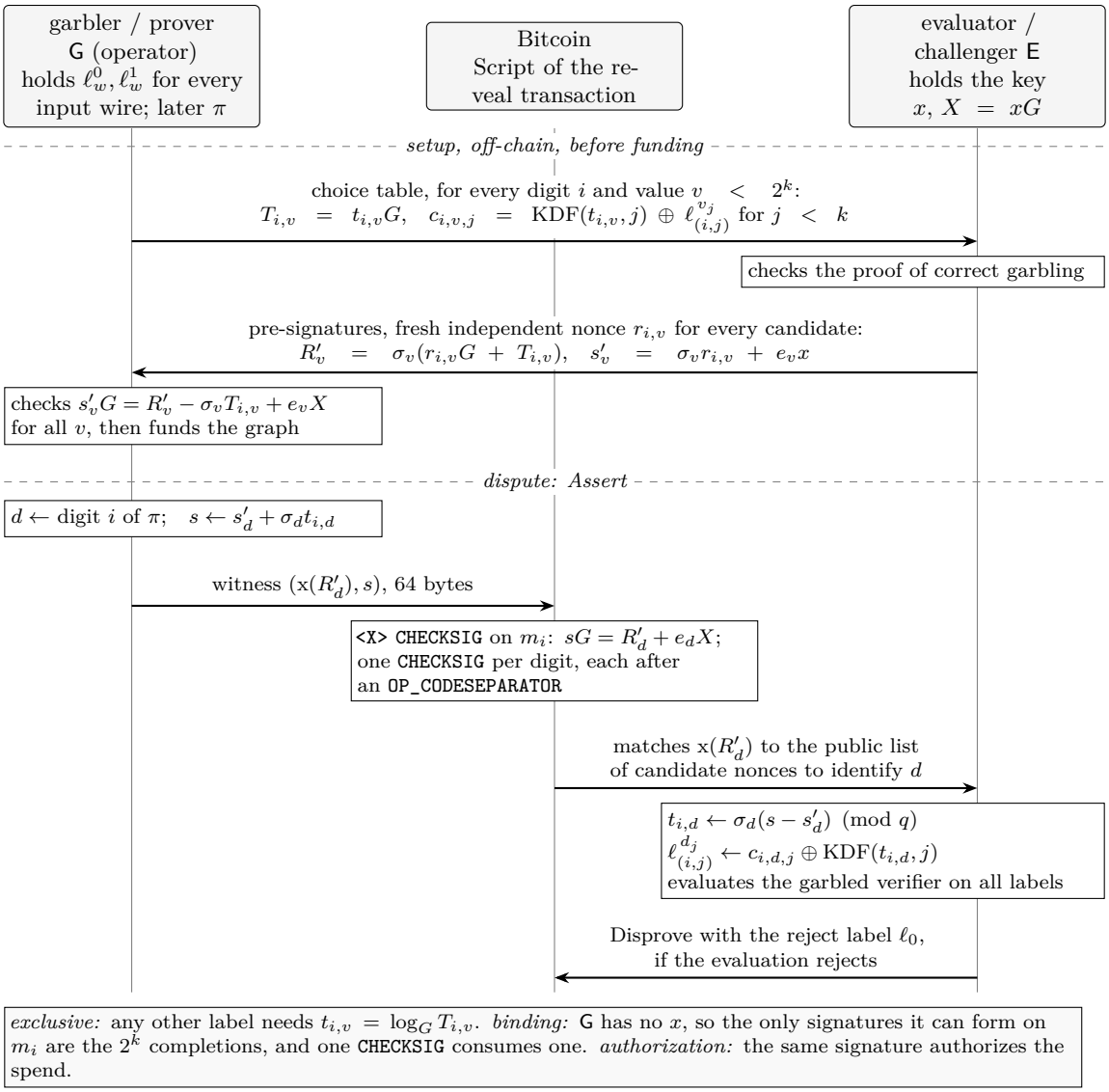

Figure 3: Schnorr adaptor label selection for one k-bit digit position i, as in BitVM3 and Mosaic [WAA+26a, KTA+26]. The same completed signature selects the digit's labels and authorizes the transaction. All setup messages above the dispute line are generated independently for every digit and candidate value.

A.2 Hash-based label selection

Figure 4 is the hash-based counterpart of Figure 3: the signature completion is replaced by a pair of hash-chain openings, the evaluator contributes nothing at setup, and the transaction binding must be added as a separate signature. Here H is the domain-separated chain hash, A and B distinguish the two chains, and v_j is bit j of candidate digit v. The diagram suppresses the digit index i on A_j and B_j.

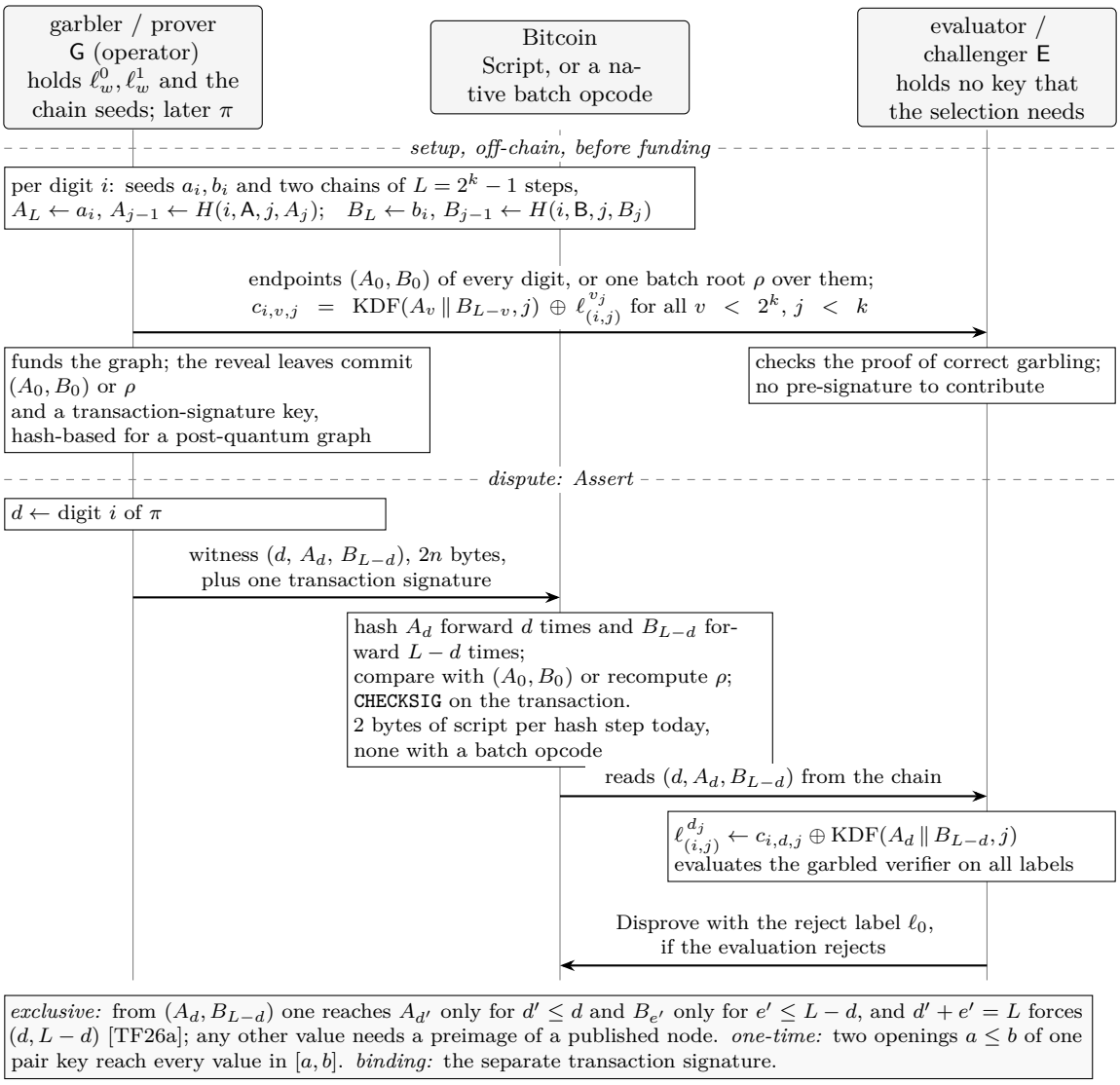

Figure 4: Hash-based label selection for one k-bit digit position i with n-byte chain nodes. Verification is a hash ladder, in Script today or in a batch opcode under the soft fork of Table 8.

A.3 A post-quantum instantiation

Figure 4 becomes a post-quantum replacement for the adaptor under three changes, none of which is available on Bitcoin today. First, the forward hashing moves out of Script into a native batch verifier: one leaf per transaction carries the batch root ρ, the first digit index and the count, and an opcode re-derives every endpoint from the records on the stack. Second, the transaction-binding signature becomes hash-based. We model the draft pay-to-Merkle-root (P2MR) output type [BHD24] with a new 0xC2 leaf, 6,000-byte stack items and a 548-byte stateful SHRINCS signature [cHK$^+$26] per transaction. The leaf version and stack limit are assumptions of our serialization model; 548 bytes is the minimum stateful signature size in the SHRINCS draft. Third, the node width n must match the quantum target: forging a different digit from a published opening is a second-preimage search over the roughly 2^{22} chain nodes that the whole input makes public, so n-byte nodes give about $(8n - 22)/2$ quantum bits.

We serialized this construction exactly for $n = 16$ and $n = 24$ at $k = 4$, including the P2MR outputs, 0xC2 leaves, placeholder signatures, depth-one control blocks and canonical multi-record stack items, and tested the pair code and the batch parser against tampering, reordering, cross-batch splicing and padding; nothing in it is executable by Bitcoin Core. Table 8 gives the results together with a projection whose model reproduces both serializations within 0.03%. The operative row is $n = 32$, $k = 8$, which matches the adaptor's size at the price of 255 hash steps per byte inside the opcode, about 3.3×10^7 compressions for the input. Its setup is two seeds per digit rather than the adaptor's 2^k secrets, so widening k does not grow the choice table; only the label ciphertexts grow, to $2^k \cdot k \cdot 16$ bytes per digit, 4.3 GB at $k = 8$. The garbling proof would have to cover that table.

A.4 The four input schemes

The four one-time schemes of Table 5 trade on-chain size against assumptions. The totals in this subsection use the final WHIR verifier's 520,832-bit input.

- *Lamport.* Each bit has two hashed 16-byte preimages, and the revealed preimage is the one-time-signature witness. The leaf script per bit,

  `OP_HASH160 OP_DUP` h_0 `OP_EQUAL OP_SWAP` h_1 `OP_EQUAL OP_BOOLOR OP_VERIFY`

  uses h_0, h_1 for the hashes of the two preimages. It is 49 bytes, and the witness 17 bytes. We executed it with 998 bits per input, and it rejected wrong preimages. The transaction-bound variant prefixes each leaf with a 32-byte authorization key and `OP_CHECKSIGVERIFY`, and supplies a 64-byte `SIGHASH_DEFAULT` signature. Its initial stack has 999 items and the measured transient peak is exactly the 1,000-item Tapscript limit [WNT20b]. Its serialized size is the Lamport row of Table 5. The variant uses a one-leaf tree with a 33-byte control block; timeout siblings and their longer control paths belong to the missing complete graph.

- *Antichain Winternitz.* Plain Winternitz chains are unsafe here: from a published chain value the evaluator hashes forward to the labels of other digit values [TF26a]. Antichain Winternitz closes this and costs a cited marginal 11.31 vB per bit at parameters $(2, 16)$, counting the hash-chain opening witness and authentication script but cancelling fixed transaction overhead. Our transaction-bound variant follows the upstream construction, prepends a 64-byte Schnorr authorization, and includes two pay-to-Taproot (P2TR) outputs, input shells, leaf scripts and depth-zero controls. With at most 332 four-bit digits per input, its size is the Antichain row of Table 5, an all-nonzero worst case that zero-valued ScriptNum coordinates only reduce. The

selection	consensus	n	k	vB/bit	component	selection security, quantum
Schnorr adaptor (Fig. 3)	current	–	8	2.14	2.23 MvB	broken (Shor)
WOTS translation [SST$^+$26] in Script	current	20	3	7.59	7.90 MvB	$\approx$ 64 bits (translation-key chain; nodes $\approx$ 70)
Antichain $(2, 16)$ in Script	current	20	4	11.41	11.88 MvB	$\approx$ 69 bits
batched selector, one record per item	P2MR, 0xC2, batch opcode	16	4	2.05	2.13 MvB	$\approx$ 53 bits
batched selector, 11-record items	same	16	4	2.04	2.12 MvB	$\approx$ 53 bits
batched selector, 5-record items	same	24	4	3.05	3.17 MvB	$\approx$ 85 bits
batched selector, projected	same	32	8	$\approx$ 2.05	$\approx$ 2.13 MvB	$\approx$ 117 bits

Table 8: Hash-based label selection against the adaptor baseline for a separate 1,041,024-bit input, published as funding plus reveals. This input evaluates the selector design and is not the final 520,832-bit verifier input reported in Table 5. The first three rows are the independent publication transactions tested in Bitcoin Core at this payload size. The batched-selector rows are exact serializations of a construction that requires the draft P2MR output, a hash-based transaction signature and a hypothetical batch verification opcode; the last row is a model projection. The security column is the multi-target second-preimage cost of the chain nodes alone (the 20-byte Antichain nodes expose roughly 2^{23} chain targets at this payload size), except for WOTS translation, whose 128-bit key chain gives the lower bound; the composed protocol remains bound by the proof system (§ 6.1).

source reports 0.256 kB of off-chain table per bit. Its larger parameters save more on-chain but need up to 43.8 kB per bit off-chain [TF26a].

- *Winternitz with a translation table.* The translation gadget of [SST$^+$26] also closes the gap of plain Winternitz: each label is a Shamir secret whose threshold is set by the chunk's checksum and the bit value, so a signed chunk opens exactly the labels of its bits. In the source, one such Winternitz signature over 508 bits, its Assert, is 3,748 vB, or 7.38 vB per bit. Our transaction-bound variant uses 3-bit digits, eight per chunk, with the checksum split over two chains, and the authorization, outputs and controls of the Antichain variant. With at most 49 chunks per input, its size is the WOTS row of Table 5, again an all-nonzero worst case. The price is the off-chain table, 953,151,840 bytes at this input size. Plain 4-bit Winternitz, modeled from one executed input at 23.06 weight units per bit, would be about 3.00 MvB, but it cannot deliver garbled labels without such a table. The challenger must decode digits as consensus does and, at setup, reject keys in which one chain end lies within seven hashes of another; neither check costs on-chain space.

- *Schnorr adaptor signatures* over 8-bit digits, as in BitVM3 and Mosaic. BitVM3 reports 9,707 weight units (WU) for a 1,024-bit Assert transaction with one input and two P2TR outputs [WAA$^+$26a], or 2.37 vB per bit at that size. Its printed witness formula instead sums to 9,451 WU once the SegWit marker and flag are included, while the merged 64-byte-`SIGHASH_DEFAULT` implementation implies 9,323 WU for that shell. Our executed construction reproduces 9,323 WU, uses the BIP342 opcode index rather than a byte offset for every code separator, and reaches the exact 998-signature stack capacity. Bitcoin Core 31.1 validates

all completed signatures. The final-input serialization uses 65,104 eight-bit digits; its size is the adaptor row of Table 5. Mosaic's source construction solders all kept instances to one set of signatures by polynomial label correlation, with an on-chain footprint independent of the number of instances [KTA+26]. We test only the one-key-set transaction; we do not implement or test Mosaic's polynomial correlation or setup verification. An evaluator that holds the signing key can spend without completing the adaptor [TF26a].

B Additional measurements

B.1 Narrow recursion

The final proof is produced from the preceding verifier tape at rate 1/1024 and re-recorded. The tape machine binds up to eight of a tape's first inputs as public values, enough for the public-value digest.

Section 4.3 describes the layout, recording, opening and commitment optimizations used in the final configuration. Here we give their algebraic conditions for a future composed soundness analysis.

Algebraic reductions and their scope. The following arguments concern the interactive algebraic reductions, conditional on fixed statement shapes, correctly derived linear forms and fresh uniform challenges after the preceding messages are fixed. When a commitment admits several candidate traces, the error must also account for that candidate set.

Merged tensors and the joint switch. For the merged witness of § 4.3, with μ uniform in $\mathbb{F}_{2^{128}}$ after the points and claims are fixed, and any fixed nonzero discrepancy vector, $\sum_i \mu^i(\ell_q(T_i) - y_i)$ has degree at most $m-1$, giving error at most $(m-1)/|\mathbb{F}_{2^{128}}|$. The subsequent tensor checks and sumcheck must still bind T to the packing; their errors are additional. The outer opening transcript binds the claimed forms and column folds before the ring switch draws μ; the ring-switch transcript then binds T before drawing ρ. The standalone ring-switch routine relies on that prior binding of claims.

The joint switch's selector-weighted terminal check is an exact identity; security still includes the ring-switch error at the enlarged arity and the paired opening's error.

Lemma 1 (Linear-form reconstruction)**.** *Let L be a fixed matrix and R a right inverse on its image. Suppose each constraint and bus expression affected by the substitution factors as $P(x,u) = \widehat{P}(Lx,u)$, with u denoting the unchanged operands. Then replacing x by $R(Lx)$ preserves all those expressions. Affine identities among forms of a fixed preprocessed trace also hold at every multilinear evaluation point if they hold at every row of that trace.*

Proof. The first claim follows from $LR(Lx) = Lx$. For the second, multilinear extension is linear, and the extension of the all-ones column is the constant polynomial 1. Thus, for vectors f, g of preprocessed forms, a fixed coefficient matrix A and a constant vector c, an identity $f = Ag + c$ on the Boolean row domain extends to $f(r) = Ag(r) + c$ at every row evaluation point r. □

The implementation derives the forms from symbolic constraints and bus expressions, and checks the right-inverse identities. For preprocessed forms, the affine relations are derived from the fixed trace. This lemma explains the substitution; validating the symbolic extractor and binding the transmitted forms to committed columns remain implementation obligations.

Lemma 2 (Shared forms sumcheck)**.** *Suppose m_{form} claimed forms, their row points and the relevant commitments are fixed before a uniform challenge $\rho \in \mathbb{F}_{2^{128}}$. Let d_{col} be the largest column-variable count, and assume the commitments admit at most K candidate collections of column evaluations before ρ. A degree-two sumcheck over the zero-extended column tables reduces these claims to authenticated column folds with algebraic error at most $K(m_{\text{form}} - 1 + 2d_{\text{col}})/|\mathbb{F}_{2^{128}}|$, in addition to the error of authenticating those folds.*

Proof. For a fixed candidate collection, let δ_j be claimed form j minus its candidate evaluation. A false vector of forms gives a nonzero polynomial $\sum_{j=0}^{m_{\text{form}}-1} \rho^j \delta_j$, which vanishes with probability at most $(m_{\text{form}} - 1)/|\mathbb{F}_{2^{128}}|$. Otherwise the initial weighted claim is false. Each round checks a polynomial of degree at most two, so the usual sumcheck induction adds at most $2d_{\text{col}}/|\mathbb{F}_{2^{128}}|$ before a false terminal claim remains. Union-bounding over the K candidate collections gives the stated term. □

The implementation uses the squared prefix factor of the terminal check, binds every table's forms before drawing ρ, and absorbs each round's two coefficients before sampling its coordinate. A concrete assessment must charge the total number m_{form} of forms and the applicable candidate multiplicity; the previous column-batching term cannot be reused solely because the transcript is shorter.

Lemma 3 (Closing sumcheck)**.** *Suppose the closing relation, its target Y and the relevant commitments are fixed before the first challenge of the closing sumcheck, and the commitments admit at most K candidate collections of column values. Let d_{cl} be the widest table's column-coordinate count. The three sumchecks of § 4.3.4 reduce the claim $Y = \sum_t P_t(v_t)$ to the column folds at three points with algebraic error at most $6d_{\text{cl}}K/|\mathbb{F}_{2^{128}}|$, in addition to the error of authenticating those folds.*

Proof. For Boolean u, $V_t(u)$ is column u's value, so the sums over z_1 and z_2 against δ select the zero point, and the triple sum equals $L_t(v_t) + Q_t(v_t, v_t) + T_t(v_t, v_t, v_t)$ for every table. The identity of § 4.3.4 therefore holds exactly for the true column values. Fix a candidate collection whose closing value differs from Y. Each of the $3d_{\text{cl}}$ rounds checks a polynomial of degree at most two in its variable: in the first sumcheck the integrand is a product of $V_t(z_0)$ and a part multilinear in z_0, and in the later ones $V_t(z_0)$, then $V_t(z_1)$, is a fixed factor. The usual sumcheck induction gives at most $2/|\mathbb{F}_{2^{128}}|$ per round before a false terminal claim remains, $6d_{\text{cl}}/|\mathbb{F}_{2^{128}}|$ in total, and a union bound over the K candidates gives the stated term. □

The implementation absorbs each round's two values before drawing its coordinate, and binds the folds before the joint ring switch draws its batching scalar.

Eq-factored bus rounds. For $r \neq 0$, the reconstruction of $h(1)$ in § 4.3 enforces $g(0) + g(1) = s$ exactly. The round message is absorbed before α is sampled, so the degree-five sumcheck argument still applies. The implementation rejects $r = 0$, introducing a completeness loss of at most $n_{\text{guard}}/|\mathbb{F}_{2^{128}}|$ for n_{guard} such coordinates under uniform challenges. This guard contributes to the recursion path's completeness loss.

Collision accounting and composition. The final configuration derives its digests' collision resistance from their width, four bits per byte, so its 25-byte digests are priced at 2^{100} classical collision work, about $2^{200/3}$ quantumly. The earlier configuration reported 128 bits for the same digests; deriving the bound from the width corrects this in both the native and the recorded verifier.

This affects the security report, not the gate counts. A composed analysis will include the corrected cap, candidate-set charges, tensor and paired-opening reductions, round-by-round soundness, hash errors and recursive composition. Because the circuit checks only the last proof, the statement's soundness also composes over the earlier proof layers and over the zkVM proof itself. For the zkVM proof, Ziren reports more than 100 bits of interactive soundness per proof stage and 93.5 bits for an illustrative tree of 150 proofs in the Poseidon2 configuration evaluated in that paper [DYH+26]. These figures concern that configuration's interactive soundness.

C Security details

This appendix derives the component-level estimates in Table 7 and explains the QROM accounting used in § 6. The recursive proof's algebraic and composition obligations appear in Appendix B.1.

Hash-based input keys under multi-target search. The protocol of § 2.1.2 publishes input bits under one-time keys. Lamport, Antichain Winternitz and translated Winternitz keys rest on `HASH160` preimages. The familiar 2^{64} quantum cost for our 16-byte preimages is a single-target estimate. One Lamport key set exposes $T = 520{,}832$ alternative preimages for the final verifier input, so generic multi-target search costs about $2^{128}/T \approx 2^{109.0}$ classically and $\sqrt{2^{128}/T} \approx 2^{54.5}$ quantumly. Hash-chain alternatives have the same multi-target issue; Table 8 gives the widths that lift the selection itself above this bound. Even 20-byte preimages under `HASH160` are capped near 70 quantum bits for one set, rather than 80, once the target count is included. Translated Winternitz keys are further bounded by the 128-bit key chain of the translation table. Because each step of that chain is salted with the instance, the chunk and the step index, an inversion targets a single step, and its cost is 2^{128} classically and 2^{64} quantumly.

Fiat–Shamir against quantum adversaries. Chiesa, Manohar and Spooner [CMS19] give the QROM Fiat–Shamir reduction of § 6.1 for suitable public-coin interactive oracle proofs with round-by-round soundness. The corresponding classical accounting includes both a term linear in $t\varepsilon_{\mathrm{rbr}}$ and hash-related errors; constants and protocol-dependent factors must be retained for a concrete bound. WHIR proves round-by-round soundness [ACFY25, §5]. Extending this accounting to the implemented composition requires checking every algebraic layer and the transcript, not just invoking WHIR's theorem.

If a valid round-by-round bound is 2^{-a}, that error remains small for t sufficiently below both $2^{a/2}$ and $2^{\lambda/3}$, up to its constants. It supplies no converse: once the bound becomes uninformative, the existence of a successful attack does not follow. In particular, a Schwartz–Zippel bound is an upper bound on bad-challenge density. Grover search gives a matching attack only when a searchable set of successful challenges of the asserted density is actually available to the adversary.

The garbling. Authenticity rests on the secrecy of Δ, and Δ can be tested against a single gate. An evaluator holding the labels a, b of an AND gate with known values x, y and its ciphertext ct checks a candidate Δ' by recomputing ct from $a, a \oplus \Delta', b$ and Δ'. A wrong candidate passes with probability 2^{-128}. Grover search therefore recovers Δ with about 2^{64} quantum evaluations of Blake3, after which every label's complement is known and the output label of the other value is free. With 128-bit labels the garbling offers about 64 bits against a quantum adversary. The same holds for a hashlock on a 128-bit output label: a preimage search over 2^{128} labels costs about 2^{64}. Grover search is highly sequential, and the adversary has only the time between setup and the end of the dispute window, but we do not rely on this.