SECURITY AUDIT

# Unlink

## Contract and Circuit Review

*Independent review of Unlink's smart contracts and zero-knowledge circuits by LFG Labs. The core safety properties are proved as machine-checked Lean proofs in Verity.*


| | |
|---|---|
| Version | v1.0 |
| Date | May 2026 |
| Commit | `7617b3eebcf37ab42124fe570eb7e065cf8c8461` |
| Reviewers | LFG Labs |


Mathematical proofs of correctness for smart contracts.

lfglabs.dev

## Disclaimer

**Final deliverable**

This document combines the manual contract/circuit review with the completed machine-checked formal-audit summary for the modeled Solidity properties. It is the final report for the repository snapshot identified on the cover.

This report is a manual security review of the Unlink codebase at the repository snapshot identified on the cover (`github.com/unlink-xyz/monorepo` at `7617b3eebcf37ab42124fe570eb7e065cf8c8461`). The scope is the round-1 contract and circuit surface described in Unlink's internal pre-audit packet; it is not a backend, SDK, dashboard, deployment, or operations audit. Items outside this scope are not covered by this report and require a separate backend/product review before any value-bearing launch.

**A security review is not absolute assurance.** Even with machine-checked Lean evidence for the modeled Verity surface, this report cannot rule out vulnerabilities in unmodeled code, external systems, operational processes, future upgrades, or assumptions listed explicitly in Section 5. The formal audit proves only the stated modeled properties under those boundaries; cryptographic hardness, ERC-20 behavior, EVM execution semantics, chain availability, artifact availability, privacy, and storage-layout migration review remain external to the Lean model.

This document does not constitute investment, legal, or tax advice, nor an endorsement of the project, its team, or its tokens.

# Contents

# 1 Executive summary

**Audit scope and confidence**

**System model.** A user deposits an ERC-20 token into one shared pool contract, holds it privately as a note, and later withdraws it to any address.

**Verdict.** In the audited scope there is no way to lose or duplicate funds on the contract or circuit surface (zero findings there, no High or Critical exploit path), and all findings concerned build and release tooling.

**Remediation status.** All eleven scored findings have since been remediated and verified Resolved against the merged code (MED-03, the last to close, by PR #983) (Section 6).

**Evidence basis.** We proved the core money rules as machine-checked Lean math and ran the real contract against that model on one shared conformance suite.

**Caveats.** Freeze-resistance and withdrawal availability, cryptography, circuit internals, privacy, backend and off-chain operations, and future upgrades are named caveats, not guarantees (one explicit boundary: Section 5).

## Guarantee summary

| Security question | Status | Basis |
|---|---|---|
| No double-spend / nullifier integrity | Proved | A recorded nullifier cannot be spent again. |
| Verify-before-mutate ordering | Proved | Proof verification precedes any state change. |
| Merkle root bookkeeping | Proved | A successful insertion stores the root it computed. |
| Exact token transfer / deposit deltas | Proved | Token-ledger deltas match the spend exactly, under the stated ERC-20 assumption. |
| Cross-note value conservation | Boundary | $\Sigma_{in}=\Sigma_{out}$ enforced by the spend circuit, not statable over contract state. |
| Cryptography, tokens, availability | Assumed | The 9 external trust assumptions (Groth16, Poseidon, EdDSA, ERC-20, Permit2, EVM liveness). |

Per-row tier in Section 5; evidence atoms in Section F, gate in Section E.

## Findings summary

| Security bucket | Crit | High | Med | Low |
|---|---|---|---|---|
| **Contract / circuit security** | 0 | 0 | **0** | **0** |
| **ZK release-engineering / CI / tooling** | 0 | 0 | **4** | **7** |
| **Total** | 0 | 0 | 4 | 7 |

As originally scored, the four release-blockers were MED-01, MED-02, MED-03, and LOW-04 (LOW-04 carries a Medium severity cell: identifiers are stable labels that do not encode

severity, so read severity from the severity cell). Following the Round-1 remediation closeout, all four are verified Resolved; MED-03 was the last to close (PR #983 makes strict source-git provenance the default and pins an immutable, permanent-SHA-anchored artifact bundle). Full enumeration and per-finding closeout status in Section 6 (Section 6) and the remediation chapter.

## Verdict and release gate

The contract and circuit surface is clean (no scored finding, no High or Critical exploit path), and every scored finding sits in the ZK release-engineering, CI, and tooling layer instead. A Medium blocks reliance on the affected release or artifact; a Low weakens a CI, test, or release guarantee.

**Release condition.** Production Groth16 keys must not be promoted from the local single-contributor path: require a documented multi-contributor phase-2 ceremony, verified against the audited circuit, before any value-bearing launch.

**Out of scope.** Material production risk sits beyond this review's scope (backend signing reconstruction, hosted secret custody, dashboard tenant authority, proving and relay operations, deployment controls). These surfaces were not audited here and should be covered by a separate backend/product review before any value-bearing launch.

LFG Labs performed this review over the contract and circuit surface at commit `7617b3e` (Section 3). We suggest also weighing what the proof assumes, how the model relates to the deployed code, how the theorems are counted, and why token-value conservation is left to the circuit; these are addressed in Section G.

**What the verdict rests on**

- **Proof-grade items:** 8 behavioral theorems plus 15 structural checks, 23 in total.
- **Trust map:** 9 external trust assumptions and 12 tracked boundaries. Assumptions are load-bearing premises the guarantee leans on; boundaries are fenced-off areas, neither proved nor assumed.
- **Axiom footprint:** `propext` and `Quot.sound`, with *no* `ofReduceBool` (kernel-clean, the shortest standard trust base).
- **Compiler trust:** every structural check is discharged by kernel `decide`, not `native_decide`, so no compiler-trust axiom is present.
- **Model-to-code link:** test-vector parity on the shared suite, not a proof of full semantic equivalence.
- **Value conservation:** note-level contract behavior and exact token-ledger deltas are proved on-chain; cross-note token-value conservation is a circuit boundary.
- **Severity and status note:** the originally release-blocking Medium set was MED-01, MED-02, MED-03, and LOW-04, while MED-04 carries a Low cell. Identifiers are stable labels that do not encode severity; read severity from the severity cell. Per the Round-1 closeout (Section 6), every scored finding is verified Resolved.

Full mechanics in Section 5; the evidence manifest in Section C.

# 2 Protocol overview and architecture

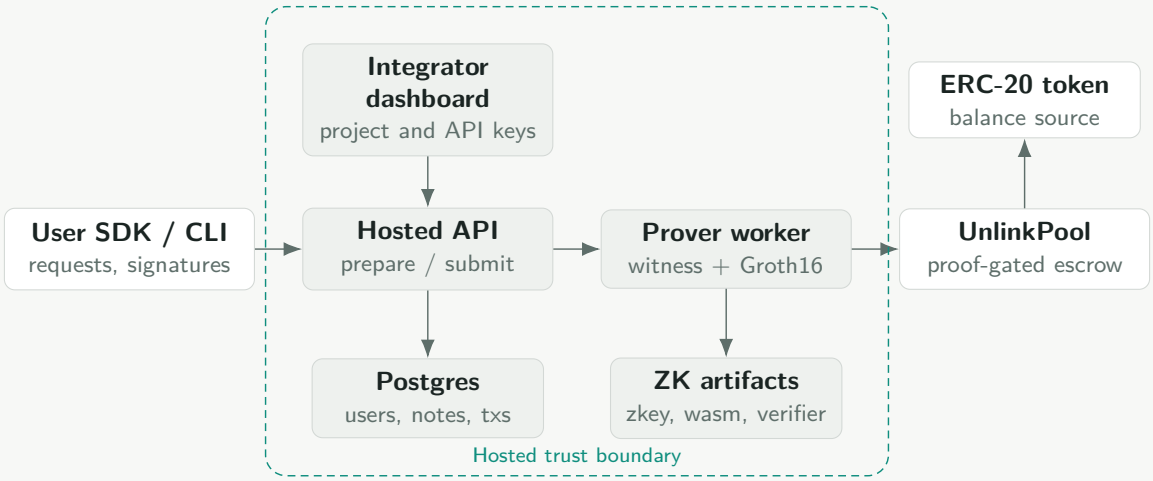
Unlink is a backend-assisted private UTXO protocol for ERC-20 tokens on EVM chains. Users deposit assets into a single `UnlinkPool` contract and receive private notes represented by commitments in an append-only Merkle tree. Later, a user can transfer value privately by consuming old notes and creating new encrypted output notes, or withdraw value by proving ownership of a note and settling the corresponding ERC-20 amount to a public EVM recipient.

The on-chain pool does not maintain an internal accounting ledger. Its solvency boundary is the ERC-20 balance held by the pool address, while the private state is represented by commitments, historical Merkle roots, and spent nullifiers. The contract accepts state transitions only after a registered Groth16 verifier accepts the proof for the selected circuit shape.

The deployed product is not only a smart contract. It includes a hosted backend, dashboard BFF, Clerk organization model, API-key management, note scanner, transaction-preparation service, proving workers, gas sponsorship, and ZK artifact deployment path. These components form the practical privacy and availability boundary for users even when the Solidity pool itself preserves the expected state-machine invariants.

The reviewed architecture deliberately splits the protocol into a minimal on-chain state machine and a larger hosted execution environment. The pool is responsible for enforcing proof-gated state transitions and exact ERC-20 settlement. The backend is responsible for reconstructing spend witnesses, discovering notes, selecting inputs, dispatching proof jobs, and relaying transactions.

## Component map




## On-chain state machine

- **Deposit.** The pool receives ERC-20 tokens via Permit2, checks exact balance deltas, and inserts commitments derived from the deposited note values.
- **Transfer.** A relayer submits a proof-backed batch that spends nullifiers and inserts new commitments. No ERC-20 transfer occurs.
- **Withdraw.** The pool verifies the proof and public withdrawal slot, marks nullifiers spent, inserts any change commitments, and transfers exactly the proven amount to the public recipient.

- **Emergency withdraw.** Any caller can submit the withdrawal proof directly to the pool without the relayer path, assuming a valid circuit remains active.

## Hosted trust boundary

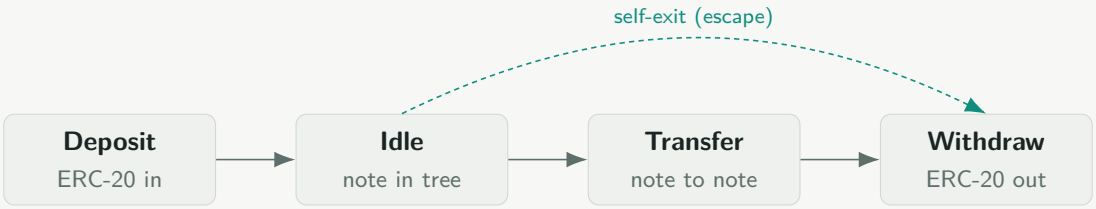
The backend is not expected to hold the user's spending private key. It does, however, handle viewing keys, nullifying keys, note ciphertexts, note selection, prepared transaction payloads, witness construction, and proving dispatch. That means the hosted system can preserve on-chain fund safety while still breaking privacy, availability, or user intent if the signing and storage boundaries are wrong.

In the expected product model, Unlink hosts this backend for integrating applications. Integrators and their end users interact with the hosted API and dashboard, while the `UnlinkPool`, verifier router, and ERC-20 token contracts are deployed publicly on-chain. This creates three different security surfaces:

- public contracts, where errors can directly affect pooled ERC-20 assets;
- Unlink-hosted services, where errors can expose privacy keys, witnesses, tenant credentials, relayer funds, or transaction availability;
- tenant integrations, where a compromised API key or malicious integration can abuse the hosted API without compromising Unlink infrastructure.

This backend/product boundary is outside the contract/circuit scope of this report, and risks that live here can be production relevant without affecting the audited surface: blind signing can let a backend-selected transaction remain internally valid while violating user intent; plaintext key and witness storage can break privacy without breaking Groth16; and dashboard/API-key bugs can change who can access the hosted control plane. These surfaces were not scored here and belong to a separate backend/product review.

## User money flow




The completed formal audit in Section 5 reasons over the modeled Verity contract surface for these phases: deposit credit, idle theft-resistance surfaces, transfer conservation and single-use spend surfaces, withdrawal structure, and the boundary between modeled contract facts and external EVM/liveness assumptions.

# 3 Audit scope

## Target

| | |
|---|---|
| Repository | `github.com/unlink-xyz/monorepo` |
| Repository snapshot | `7617b3eebcf37ab42124fe570eb7e065cf8c8461` |
| Unlink pre-audit context | tag `audit/verity-round1-2026-05-07`<br>SHA `7617b3eebcf37ab42124fe570eb7e065cf8c8461`<br>entry point `docs/internal/audits/README.md` |
| Protocol type | Backend-assisted private UTXO protocol for ERC-20 tokens on EVM chains |
| Primary contract | `UnlinkPool`, with verifier routing through `VerifierRouter` |
| Primary circuit | `spend_10x4_v1`, used for transfer, withdraw, and emergency-withdraw proofs |

In Unlink's internal packet, the contract and circuit scope was locked on 2026-05-07 under the tag `audit/verity-round1-2026-05-07` at commit `7617b3e`: the on-chain contracts under `protocol/contracts/` and the zero-knowledge circuits under `protocol/zk/`. The backend, dashboard, and the deleted DeFi adapter are out of scope here. That locked surface is this report's scope; the other surfaces are treated only as scope boundaries and are not scored as findings.

## How to verify the package

Unzip `unlink-audit-sources-7617b3e-dda647c.zip`, read `MANIFEST.json`, verify the file hashes in `SHA256SUM`, and use the pinned commits recorded in the manifest to reproduce the audited Solidity snapshot and Verity model state. This is an evidence-package integrity check, not a replacement for the build and test gates referenced later in the report.

## Included in this audit

- `protocol/contracts/src/**`: every Solidity source file under `src/`, including `UnlinkPool.sol`, `VerifierRouter.sol`, the generated `spend_10x4_v1` verifier, `src/lib/**`, and `src/interfaces/**`.
- `protocol/zk/circuits/**`: the `spend.circom` top-level circuit and imported Merkle-proof subcircuit.
- `protocol/zk/scripts/**`: verifier export and hardening scripts, including `codegen.sh` and `harden_verifier.py`.
- `protocol/zk/tests/**`: circomkit and Vitest test vectors used as validation evidence.
- Contract/circuit invariants from Unlink's internal pre-audit packet: nullifier uniqueness, Merkle-root validity, conservation, authorization by spending key, withdrawal settlement, emergency-withdraw liveness, and verifier routing for the canonical `(10,4,32)` spend shape.

At the audited snapshot (`7617b3eebcf37ab42124fe570eb7e065cf8c8461`), the `src/**` tree contains only `UnlinkPool`, `VerifierRouter`, the generated `spend_10x4_v1` verifier, and the `src/lib/**` and `src/interfaces/**` support files, the same set enumerated above. The ERC-4337 account, paymaster, and factory contracts (`UnlinkExecutionAccount`, `UnlinkExecutionAccountFactory`, and `UnlinkPaymaster`) are *not present at this commit*; they were added to `src/` only after the audited snapshot. No ERC-4337 source review was performed and no account-abstraction finding is claimed; that surface, together with backend/paymaster operations and product authorization, is out of scope for this review.

## Excluded or deferred

- `protocol/backend/**`. Backend trust boundaries are out of scope; `protocol/backend/crates/crypto/**` was only adjacent/on-request validation signal.
- `protocol/sdk/**`, except as test-vector evidence for the in-scope circuit and contract boundary.
- Dashboard and product UI, including the Next.js dashboard, dashboard BFF, Clerk organization integration, API-key management UI, and dashboard backend routes.
- Contract tests, deployment scripts, generated build output, caches, `node_modules`, vendored submodules, and deleted or planned-only product surfaces.
- Trusted setup / phase-2 ceremony execution, transcript review, production artifact promotion, and related release operations.
- Owner, governance, role hardening, and storage-layout collision review before mainnet deployment.
- Backend viewing-key custody, because the pre-audit packet explicitly treats that custody as part of the stated trust model rather than as a hardening item.
- Formal proof of unmodeled external systems and boundaries listed in Section 5, including cryptographic hardness, external ERC-20/Permit2 behavior, EVM transaction atomicity, chain liveness, production storage-layout migration review, privacy, and backend/product operations. The modeled Verity contract-surface proof is included in this final report.
- Cryptographic proof of Groth16 soundness, Poseidon collision resistance, and correctness of external ERC-20 implementations.

> **Scope consequence**
>
> Backend, SDK, dashboard, deployment, and operational issues may still be security-relevant for production tenants. They are outside this report's scope and are not scored as audit findings; they should be covered by a separate backend/product review.

# 4 Threat model

The threat model is the contract/circuit threat model from Unlink's internal pre-audit packet. A relayer may censor or reorder transactions, but cannot spend without the user's spending key. The backend holds viewing and nullifying material under the stated trust model, but cannot sign for the user. The spending key remains the non-custodial root, and permissionless `emergencyWithdraw` is the intended escape path if the relayer or backend disappears.

We did not assume the adversary can break Groth16 soundness for the deployed phase-2 verifier key under the stated external setup assumptions, forge EdDSA signatures without the spending key, bypass Solidity nullifier checks, or compromise a properly governed production multisig.

## Review focus

Within scope, review focused on whether the Solidity contracts and canonical spend circuit enforce the audited invariants:

- deposits create commitments backed by received ERC-20 value;
- spends require valid Merkle membership and an unspent nullifier;
- private transfers conserve value across input and output notes;
- withdrawals and emergency withdrawals settle only the value authorized by the proof;
- verifier routing accepts only the intended circuit identifier and verifier.

## Out-of-scope threat surfaces

The broader system has additional threat surfaces: malicious tenant integrations, stolen tenant API keys, compromised dashboard/backend components, plaintext operational databases or backups, raced hosted API requests, shared prover saturation, and artifact promotion mistakes. Those surfaces are important for production, but they sit outside the contract/circuit audit and are not covered by this report; they should be addressed in a separate backend/product review before any value-bearing launch.

"Artifact promotion mistakes" above refers to human operator decisions about which artifact bundle to promote, ceremony execution, and release-policy judgment. The script-level pipeline that supports those decisions (input validation, manifest identity binding, checksum-contract consistency, and CI gate wiring under `protocol/zk/scripts/**`) is explicitly in scope under Section 3, because that surface is amenable to script-level review. The ZK-script findings live in that in-scope surface; production risk in the surfaces listed above was outside this review's scope and was not enumerated here.

# 5 Formal audit

This section separates the formal-audit result into three parts: contract-side facts proved over the Verity model and nontrivial Lean evidence files, external facts accepted as trust assumptions, and production surfaces left to other review workstreams. The corresponding Verity model files and generated contract artifacts are included in the companion audit source and evidence package under `verity-model/`; supporting circuit, verifier-key, and ZK regression evidence is included under `zk-evidence/`.

Verbatim Lean source, axiom output, and the per-fact manifest are in the appendices (Section F, Section C); this section states results and points there for the mechanics.

> **The formal result**
>
> In machine-checked Lean (axioms `propext` and `Quot.sound` only, *no* compiler-trust axiom) this review proves **8 behavioral theorems** about the contract state machine and its token interface. Three are a *total* specification of the spend helper `_spend`: a fresh in-field nullifier is marked spent, a re-spent one reverts (**no double-spend**), and an out-of-field one reverts. A fourth: a successful non-empty `_insertLeaves` stores the root it computed and marks it seen. The remaining four hold over an executable `(token, account)` ledger: a transfer moves *exactly* the transferred amount from sender to recipient, a Permit2 deposit credits the pool by *exactly* the deposited amount, the withdrawal balance-delta guard holds by construction, and `balanceOf` is read-after-write coherent. So no-double-spend and the exact token deltas are *proved*, not assumed.
>
> Around these sit **15 kernel-decidable structural checks**, discharged by kernel `decide` with no `native_decide`, for a proof-grade total of **23**, alongside **9 external trust assumptions** and **12 tracked boundaries**. No proved fact rests on the compiler-trust axiom.

This is a contract-surface proof, not a full-system proof. Cryptography, circuit correctness, token and Permit2 behavior, EVM execution semantics, liveness, operations, privacy, and future upgrades are recorded below as explicit external assumptions or tracked boundaries, not as proved facts.

## How the formal result is counted

In this section, *the gate* is the single Lean proposition the package ships and re-checks (`formal_audit_ready_for_delivery`): it conjoins every evidence atom listed below, so it kernel-checks only if each enumerated fact holds, and one successful check certifies the whole set at once. This is the formal gate; the separate CI and ZK release gates discussed in Section 6 are unrelated workflow checks.

Color convention used throughout this section: **green** = proved in Lean, **yellow** = external trust assumption, **blue** = tracked boundary (full definitions under *How to read the guarantee*).

| Class | Count | Meaning |
|---|---|---|
| **Kernel-checked behavioral theorems** *(proof-grade)* | **8** | The state-machine and token-ledger theorems stated in the box above, enumerated as evidence atoms in the gate; axioms `propext`, `Quot.sound` only, no compiler-trust axiom. The four token-ledger deltas narrow the external-token assumption `AP-Ax4` without relabeling it. |

| Class | Count | Meaning |
|---|---|---|
| Structural / constant checks *(kernel-decidable)* | 15 | Boolean checks over modeled entrypoint metadata, write-sets, call paths, and compiled constants: verify-before-mutate ordering, zero-slot filtering, owner gating, scalar-field equality, and so on. All 15 are discharged by kernel `decide` (proof-grade, axioms `propext/Quot.sound`); with the 8 behavioral theorems this is a proof-grade total of 23. None of them rely on `native_decide`, so the structural tier carries no `ofReduceBool` compiler-trust axiom. |
| Proof-grade facts the gate enumerates | 23 | The 8 behavioral theorems plus the 15 structural/constant checks. The gate's `concreteEvidenceAtoms` manifest holds 27 atoms: these 23 load-bearing facts, three delivery-only atoms, and a separately-counted entrypoint reentrancy-revert theorem. All kernel-clean; see the assurance-tier table below. |
| External trust assumptions | 9 | The `AP-Ax1`–`AP-Ax9` premises (Groth16, circuit relation, Poseidon/EdDSA, token/Permit2, key custody, chain liveness, artifact and data availability). If one is false, a stated production interpretation of a proved fact may fail. |
| Tracked boundaries | 12 | Security-relevant scope intentionally not claimed and not used as proof evidence (circuit internals, privacy, future upgrades, full liveness, general EVM semantics). |

The full row-level index is the AP/IC manifest in Section C. That manifest is an index, not a second count of guarantees: its rows expand these 23 facts into the AP/IC labels used throughout the report, including presentation aliases and conjunctions that reuse a fact under more than one label. One of the 23 is the Merkle root-bookkeeping theorem (shared across `AP-S4`, `AP-L2`, `IC-S2`); it carries the narrow meaning set out in the Merkle root handling note.

## Conservation across contract and circuit

The conserved quantity lives at three layers, and two of them are now contract theorems. *Note conservation* over the contract state machine is proved and kernel-clean: the three `_spend` theorems characterize the spend transition completely, so a successful spend marks *exactly* its one declared in-field nullifier, mutating no other state, while a re-spent or out-of-field nullifier reverts. This is the contract-state form of "nothing is created or destroyed," with axioms `propext`, `Quot.sound` only.

*Exact token deltas* at the external-token interface are also proved. This review models that interface as an executable `(token, account)` balance ledger: `safeTransfer` debits the sender and credits the recipient (reverting on insufficient balance), `permit2Pull` credits the pool on deposit, and `balanceOf` reads the ledger after writes. Over that model four

behavioral theorems hold as counted gate atoms: (i) a transfer decreases the sender balance and increases the recipient balance by *exactly* the transferred amount; (ii) a Permit2 deposit credits the pool by *exactly* the deposited amount; (iii) the withdrawal balance-delta guard's exact equalities hold by construction when the transfer is run, not just matched on source shape; and (iv) `balanceOf` is coherent with the preceding write (read-after-write). These narrow the external-contract assumption (`AP-Ax4`) to the residual that a production ERC-20 conforms to this ledger model; they do not relabel it, and the assumption count is unchanged.

*Token-value conservation* ($\Sigma_{\text{in}} = \Sigma_{\text{out}}$) across the shielded pool is a distinct, circuit-level property and is **not** statable as a contract theorem. The conserved note values live in circuit commitments, not in the pool's `StateStorage` (which holds nullifiers and Merkle roots) nor in the per-call token ledger (which holds external ERC-20 balances, not private note amounts), so there is no cross-note quantity in contract state to sum, and the pool keeps no internal accounting ledger (Section 2). That arithmetic is enforced by the spend circuit's balance constraint, a tracked boundary, not a contract-control-flow fact. Whole-protocol token-value conservation is therefore the conjunction of these contract-side exact-delta facts with the circuit balance constraint and ERC-20 conformance, named here rather than asserted as a single contract proof.

## Assurance grading of the formal facts

Both kinds of gate fact are validated entirely inside Lean's kernel and add only `propext` and `Quot.sound`; they differ in *kind*, not in trusted base. The distinction the gate is careful about is `native_decide`, which would compile the decision procedure to native code and accept its answer through the `Lean.ofReduceBool` compiler-trust axiom, widening the trusted surface to the Lean and C compilers and the native runtime. The gate keeps the structural scans in kernel-reducible form instead, so kernel `decide` evaluates them directly and no gate fact carries `ofReduceBool` (the per-check re-derivation is in Section F).

| Assurance tier | Count | Axiom footprint and meaning |
|---|---|---|
| Kernel-checked behavioral | 8 | `propext`, `Quot.sound` only, **no `ofReduceBool`**. The eight theorems stated in the box above, proved by structural tactics over `StateStorage` and the executable `(token, account)` ledger. Strongest tier: genuine state-machine theorems, not decidable shape checks. |
| Kernel-decidable structural and constant | 15 | `propext`, `Quot.sound` only, **no `ofReduceBool`**. Boolean checks over modeled metadata, write-sets, call paths, and compiled constants (including the `SNARK_SCALAR_FIELD`, `MAX_NOTE_VALUE`, circuit ID, and lazy-IMT zero-subtree manifest-parity equalities). All discharged by plain kernel `decide` from kernel-reducible definitions, with no `native_decide`. |

Both tiers are genuinely proved (all green) and uniform in axiom footprint: the 8 behavioral theorems are the headline, and the 15 kernel-decidable structural and constant checks bring the proof-grade total to 23, none carrying `ofReduceBool`.

## Guarantee map

This is the decision-oriented view: what a reader can rely on, and the limitation to keep visible for each answer.

| Decision question | Formal answer | Limitation to keep visible |
|---|---|---|
| Can a modeled spend mutate pool state before proof verification succeeds? | Proved no, for the modeled verifier-routing and proof-before-mutation surfaces. | The cryptographic meaning of "proof succeeds" still depends on Groth16 and circuit correctness. |
| Are nullifiers protected against direct double marking in the modeled helper? | Proved both directions for concrete `_spend`: a fresh in-field nullifier is marked; a re-spent nullifier reverts (no double-spend) and an out-of-field nullifier reverts (field bound). The latter two were proved kernel-clean by this review. | Full spend validity still depends on circuit relation semantics and honest witness construction. |
| Can admin/upgrade paths directly rewrite spend accounting state? | Proved no for the modeled write-set summaries and owner-gated UUPS authorization hook; current-snapshot storage anchors are pinned by the delivery gate, and a Foundry storage-layout snapshot gate fixes the current pool/router layout baseline. | Future implementations still require an upgrade-pair storage-layout and upgrade review. |
| Are relayer and emergency paths separated as intended? | Proved structurally: relayed paths contain the relayer-authorization error surface; emergency withdrawal lacks that relayer error surface. | End-to-end escape still depends on liveness, gas, route availability, proving artifacts, token availability, and user data availability. |

| Decision question | Formal answer | Limitation to keep visible |
|---|---|---|
| Does the proof guarantee exact solvency for all tokens in production? | Partially: contract-side balance guards are proved behaviorally over an executable token ledger (exact transfer, deposit, and read-after-write deltas), and token matching is proved structurally. Cross-note value conservation ($\Sigma_{in} = \Sigma_{out}$) is not statable over the model. | Exact asset conservation assumes conforming ERC-20 behavior, Permit2 semantics, circuit correctness, EVM revert atomicity for withdrawal paths that mutate before token settlement, and no unsafe upgrade. |
| Does the proof guarantee user privacy? | No; privacy is intentionally outside this contract-surface proof. | Privacy requires a separate protocol, backend, relayer, metadata, and operational analysis. |

The proved statements are AP/IC predicates over Verity entrypoints, contract metadata, constants, and state helpers. These predicates are not the model itself: the model is the hand-written Verity representation of functions such as `transfer`; an AP/IC predicate is a specific claim checked about that model, such as "verification occurs before spend-state mutation". In this section, **AP** means an audit property: a security or correctness claim the audit wants to establish. **IC** means an implementation check: a lower-level fact about the Verity model, compiled constants, or Lean helper lemmas that supports one or more APs. Abstract names such as `Sigma`, `Step`, `SpendStep`, `WithdrawStep`, and `DepositStep` are used only as report vocabulary; the delivered proof obligations are stated over the Verity contract surface.

## How to read the guarantee

The classification is based on proof dependency. A **proved** item is established inside the Verity/Lean development. An **external trust assumption** is a premise the formal result depends on: for example, if Groth16 soundness were false, the production interpretation of a proof-ordering theorem would fail. A **tracked boundary** is different. It is not used to prove anything and the theorem does not depend on it; it is a named exclusion so the reader does not infer that Lean covered it. For example, the formal model proves the modeled owner gate for upgrades, but future implementation compatibility is still a tracked boundary because a later upgrade can change code and storage outside this audited model.

| Status | Plain meaning | Examples in this audit |
|---|---|---|
| **Proved in Lean** | Machine-checked from the Verity model, compiled artifact facts, or small concrete state lemmas. | Proof-before-mutation ordering, modeled active-route checks through the typed verifier-router ABI surface, relayer guard error-surface presence, owner-gated upgrade authorization, admin write-set exclusion, concrete `_spend` nullifier behavior, structural zero-slot filtering, and scalar-field literal equality. |
| **External trust assumption** | Required by the stated guarantee, but justified outside this Verity contract model. In proof terms it is axiom-like; in audit terms it is a named boundary condition. | Groth16 soundness, circuit relation correctness, Poseidon/EdDSA security, ERC-20/Permit2 conformance, spending-key custody, chain/proving availability, ABI/byte-packing semantics, transient reentrancy-guard semantics, router conformance, deployment-owner validity, EVM revert atomicity where later calls can revert, and event/ciphertext availability. |
| **Tracked boundary** | Security-relevant scope that is deliberately outside the formal claim. The proof does not rely on it; the row is there to stop the report from implying coverage. A Lean/report name for such an item is only an audit index entry. | General EVM/network execution semantics beyond the explicit withdrawal revert-atomicity premise, storage-layout compatibility across future upgrades, full privacy-game claims, backend custody and availability, and deployment process controls. |

Reading convention: green rows are contract-surface evidence; yellow rows are explicit premises that must be supplied by cryptographic, circuit, token, or operational evidence; blue rows are tracked exclusions from the formal claim, retained so the report does not overstate what Lean proves. The Merkle root guarantee is green but narrow; it is stated separately in the Merkle root handling note below.

## Proof files and validation gate

The formal audit lives in the Verity/Lean modules checked into the LFG fork of the Unlink monorepo. The source tree used for reporting is:

- `protocol/contracts/verity/UnlinkXyz/Pool/FormalAudit.lean`
- `protocol/contracts/verity/UnlinkXyz/Pool/Contract.lean`
- `protocol/contracts/verity/UnlinkXyz/Pool/State.lean`
- `protocol/contracts/verity/UnlinkXyz/Pool/Proofs.lean`
- `protocol/contracts/verity/UnlinkXyz/Pool/Compile.lean`

This monorepo path is the canonical Section 7 re-issue artifact; its exact delivery commit on `lfglabs-dev/unlink-forked-monorepo` is recorded in `FORMAL_ARTIFACT_MANIFE`

ST.md, with the delivered Verity model aligned to the audited `7617b3e` behavior. The manifest also pins `lfglabs-dev/verity@4101073bc7eff980b3e9fda3afe889c4c30f4199`, Lean `leanprover/lean4:v4.22.0`, and the Unlink audit target `7617b3eebcf37ab42124fe570eb7e065cf8c8461`.

Within this pinned tree, the two revert proofs sit in `State.lean` beside `spend_success_of_fresh_in_field`, and the structural and constant checks use kernel `decide` in `FormalAudit.lean`. Both are kernel-verified against the prebuilt model oleans (`#printaxioms`: `propext/Quot.sound` for the reverts, `propext/Quot.sound` or less for the structural and constant checks; no `ofReduceBool` in any), and the two revert theorems are enumerated as AP/IC evidence atoms in the gate.

The monorepo exporter stages this source tree into a local Verity/Lake build harness when producing generated artifacts. The checked model gate is:

```
cd protocol/contracts
npm run check:verity-model
```

The delivered local final run is recorded as successful for the artifact named in Section E. This is a local delivery attestation, not a fresh-clone reproducibility claim unless the accompanying source manifest, pinned Verity checkout, and build log are supplied together. The active audit/model files contain no `sorry`, no `admit`, and no active Lean `axiom`. Each reported item is classified as proved over the Verity model and nontrivial Lean evidence files, an external trust assumption, or a tracked boundary. The listed `Proofs.lean` target is only a build-green placeholder containing `unlinkPool_compiles:True:=trivial`; it should not be read as a proof-bearing delivery target. The substantive AP/IC evidence is in `FormalAudit.lean` and the modeled contract/state files.

## Verification methodology

The proof checks the hand-written Verity model, not the Solidity source by automatic semantic equivalence. The Solidity-to-Verity relation is supported by two independent checks: a static source crosswalk (Section B), and an executed dual-implementation differential in which the curated Foundry conformance suite is run twice over the same vectors (once against the hand-written Solidity contracts, once against EVM bytecode compiled from the Verity model) and required to produce identical pass/fail outcomes. That differential was executed and passed with zero failures on both sides; the run is recorded in Section E. It establishes behavioral parity over the suite's concrete vectors, not exhaustive semantic equivalence: input regions that no vector exercises are outside its reach. Within that scope, the formal evidence has three forms: structural metadata checks over modeled entrypoints, concrete state-helper theorems over `StateStorage`, and compiled-artifact parity checks. Report-level AP/IC names are labels for these checks; they are not a second model.

Formal model provenance: the audit report target is `7617b3e`; the visible upstream markers in the Unlink Verity model are pinned to the same target. The dual-implementation differential was executed in both configurations that matter for provenance, with the guard-free model on the Verity side in each: against the production `UnlinkPool.sol` at `dda647c` (which retains the two post-audit balance guards), and against the audited `7617b3e` `UnlinkPool.sol` (`VerifierRouter` bytecode is identical between the two commits, so only `UnlinkPool` differs), and it passed with zero failures in both (Section E). The delivered proof package still relies on the source crosswalk in Section B plus the generated artifact wrappers named in Section E; it is not an automatic semantic-equivalence proof from Solidity to Verity.

### What was modeled in Verity

The Verity model is a hand-written Lean/Verity representation of the Solidity-facing pool contract. It covers contract-side control flow and local state behavior, not the entire production protocol. The formal audit then proves Lean predicates over that model, over small state-helper lemmas, and over compiled artifact facts such as manifest/constant parity. The modeled surface includes:

- public and internal pool entrypoint metadata, including role guards, relayer guards, proof-verification ordering, revert/require placement, and internal-call structure;
- admin and ownership write-set summaries, including owner-gated UUPS authorization and direct-write exclusion from protocol state;
- deposit-path structural checks: presence of note validation, token matching, commitment hashing, insertion shape, Permit2 call-path evidence, and balance-mismatch guard surfaces;
- transfer/withdraw/emergency-withdraw structural checks: active circuit route lookup, proof-success gate before mutation, zero-slot filtering path presence, withdrawal-slot binding, and relayer authorization error-surface presence;
- concrete state-helper behavior for `_spend`, plus empty insertion root preservation for `_insertLeaves`;
- generated artifact wrapper resolution and modeled `SNARK_SCALAR_FIELD` equality against the BN254 scalar-field order.

The resulting Lean statements stay close to the Verity model. The report uses short AP/IC labels for readability; the long theorem names remain in the Lean source.

The formalization follows the Solidity source closely, but there are two different kinds of Lean definitions in play. `Contract.lean` contains the Verity model of the pool. `FormalAudit.lean` contains audit predicates: small named claims about that model. Some predicates are direct state theorems; others are structural checks over the modeled function metadata. The full crosswalk for the modeled surface is in Section B.

```
// UnlinkPool.sol
_verifyProof(c.verifier, txn.proof, txn.merkleRoot, txn.contextHash,
             txn.nullifierHashes, txn.newCommitments);
_spendNullifiers(txn.nullifierHashes);
uint256[] memory leaves =
  _realCommitments(txn.newCommitments, type(uint256).max);
uint256 newRoot = _insertLeaves(leaves);
```

```
-- Contract.lean, Verity model of transfer
verifyProof c.verifier txn.proof txn.merkleRoot txn.contextHash
  txn.nullifierHashes txn.newCommitments
spendNullifiers txn.nullifierHashes
let leaves := realCommitments txn.newCommitments UINT256_MAX
let newRoot <- insertLeaves leaves
```

```
-- FormalAudit.lean, AP-S2 as a report-level label.
-- The Boolean check scans the modeled transfer and executeWithdrawal metadata.
def verifyBeforeMutateGenerated : Bool :=
  transferVerifyBeforeMutate && executeWithdrawalVerifyBeforeMutate

def AP_S2_VerifyBeforeMutateGenerated : Prop :=
  IC.ContractFacts.verifyBeforeMutateGenerated = true
```

```
theorem ap_s2_verify_before_mutate_generated :
    AP_S2_VerifyBeforeMutateGenerated := by
  exact IC.ContractFacts.verifyBeforeMutateGenerated_true
```

In plain language, this check asks whether the modeled transfer and withdrawal-execution paths contain the expected control-flow pattern: the circuit route must be registered and active, proof failure must revert, and the call that marks nullifiers spent must appear only after those checks. The final line does not prove cryptographic soundness; it proves that the Verity model has the required contract-side ordering. The `AP_S2_VerifyBeforeMutateGenerated` definition is therefore a label: it connects the report item AP-S2 to the lower-level implementation check `verifyBeforeMutateGenerated`. The full Boolean predicate body is in `FormalAudit.lean`.

A second predicate is a direct state-helper guarantee rather than a metadata scan: for any modeled storage state and any fresh, in-field nullifier, `State._spend` succeeds and the resulting state records that nullifier as spent, paired with a structural scan showing the generated nullifier loop skips zero slots and reaches real nullifier writes (`AP-S3`). It is the behavioral core of the no-double-spend result; the verbatim predicate and its companion revert theorems are in Section F.

## Merkle root handling

The Merkle guarantee is narrow, so it is stated here as final state: what is proven, what is not proven but expected, and why the gap is safe for spends.

| Status | Statement |
|---|---|
| **Proven** | On a successful non-empty `_insertLeaves`, the modeled pool appends each leaf through the concrete `InternalLazyIMT._insert` helper and writes the root produced by the concrete `InternalLazyIMT._rootWithDepth` into `merkleRoot`, marking that same root in `rootSeen` (`AP-S4`, `AP-L2`, `IC-S2`). This bookkeeping theorem is kernel-checked (axioms `propext`, `Quot.sound`). |

| Status | Statement |
|---|---|
| **Delegated boundary** | That the stored root equals the canonical incremental-Merkle root of the appended leaf set. This review does *not* prove root-value correctness; it *explicitly delegates* the property to a named boundary. The tree is re-implemented concretely in the Verity model (`InternalLazyIMT.lean` mirrors the vendored zk-kit library: index arithmetic, right-spine fold, the zero-subtree constants `Z_0`–`Z_32`, and dynamic-depth root selection); it is not treated as an opaque external return value. The single opaque primitive inside that arithmetic is the Poseidon compression function `PoseidonT3.hash`. The bookkeeping theorem binds the computed root abstractly, so it does not by itself establish root-value correctness; closing that would require structural and collision properties of `PoseidonT3.hash` plus induction over the spine. That residue is delegated to two named owners: the Poseidon hash assumption, and the circuit's Merkle-membership re-constraint at spend time (`IC-C2`). The delegation is consistent with the architecture in Section 2: the pool admits a state transition only after the registered Groth16 verifier accepts the proof for the selected circuit shape, so root-value correctness is enforced at the circuit boundary rather than reconstructed in contract control flow. |
| **Why the gap is safe** | Spend safety does not depend on any particular old root staying individually visible. It rests on one invariant: `rootSeen` only ever accumulates roots produced by a successful pool insertion, so a spend proof can only reference a root the pool itself created. Availability of a specific historical root is a liveness/UX property, tracked under data-availability and operational assumptions, not a safety property. |

## Formal property overview

The complete AP/IC manifest is in Section C. The main formal result is summarized below by security question rather than by Lean name.

| Area | Status | Checked claim | Representative evidence |
|---|---|---|---|
| Spend ordering | Proved | Modeled spend and withdrawal-execution paths require an active verifier route through the typed verifier-router ABI surface, and proof-success check before nullifiers or leaves are mutated. | `AP-S2`, `IC-S5` |
| Nullifier state | Proved | For a fresh in-field nullifier, the concrete modeled `_spend` helper marks that nullifier, and the generated `spendNullifiers` helper checks field bounds and freshness before marking nonzero nullifiers. | `AP-S3`, `IC-S1` |

| Area | Status | Checked claim | Representative evidence |
|---|---|---|---|
| Deposit and withdrawal structure | Proved | The modeled deposit and withdrawal surfaces contain the expected note/token validation, Permit2 call path, balance-mismatch guard, named structural zero-slot filtering surfaces, and withdrawal-slot binding surfaces. | AP-S1, AP-S5, AP-S6, AP-S7, IC-TB1 |
| Admin and upgrade authorization | Proved | Modeled admin paths do not directly write protocol accounting state, and the UUPS authorization hook is owner-gated. | AP-T1, AP-T2a |
| Constants and artifacts | Proved | The modeled SNARK_SCALAR_FIELD constant equals the BN254 scalar-field order, and generated artifact wrappers resolve to the generated Verity specs. | IC-X1 |
| Merkle/root-history insertion surface | Proved | Successful real _insertLeaves executions store the returned root as merkleRoot and mark that returned root in rootSeen. This is a concrete insertion guarantee, not a broad theorem that every historical root remains visible across arbitrary abstract transitions. | AP-S4, AP-L2, IC-S2 |
| Cryptography, circuits, tokens, and availability | External assumptions | These are required premises for production interpretation of the proved contract-surface claims; the appendix also names external-call and deployment model assumptions. | AP-Ax1–AP-Ax9 plus appendix rows |
| Privacy, future upgrades, circuit internals, and full liveness | Tracked boundary | These are named workstreams, not claims proved or assumed by this contract proof. | AP-T2b, IC-C1–IC-C3, IC-PR1 |

## External assumptions

External assumptions are required premises for interpreting the proved contract-surface facts as production security guarantees. They are axiom-like from Lean's perspective, but each one corresponds to an ordinary audit boundary that must be justified by cryptographic review, circuit review, integration review, or operations.

**What these assumptions do and do not gate.** None of the nine assumptions is used to prove the eight kernel-checked behavioral theorems: those are established in State.lean and the executable token-ledger model, neither of which imports the assumption module, with a propext/Quot.sound footprint. The assumptions are consumed only where the report lifts structural facts into abstract production-interpretation bundles (proof

soundness, fund safety, exit, authority), so the trust surface for the behavioral results is empty.

**Trust-surface disclosure (effective shrink).** Two of the nine premises, Poseidon/EdDSA hardness (`AP-Ax3`) and spending-key custody (`AP-Ax5`), plus two circuit-internal opaque premises tracked as boundaries rather than counted among the nine (circuit bookkeeping, circuit balance/range) are currently referenced by no proved fact in the delivered development: each appears exactly once, as its own `opaque` declaration. This does *not* make them unimportant for production (Poseidon hardness and key custody are essential real-world assumptions); it means the current Lean development does not yet wire them into any theorem, so a reader should treat them as declared scope, not as load-bearing proof premises. The remaining premises below are genuinely consumed by at least one guarantee bundle.

**On discharging the runtime-semantics premises.** The ABI-encoding, keccak memory-slice, Permit2/router/verifier ABI, public-signal packing, and transient reentrancy-guard rows are not discharged in this delivery. The Verity framework pins an executable EVM-semantics formalization (EVMYulLean) and a compiler-correctness bridge, but the Unlink Pool model is interpreted over the source-level `SourceSemantics` interpreter and does not route through that bridge, so these rows remain honest opaque boundaries refined against the source crosswalk. The reentrancy guard is the most tractable case: its revert-on-re-entry control flow is already proved in-model and kernel-clean (Section F), and only EIP-1153 opcode faithfulness remains assumed. A full discharge is substantial new development rather than a label change, and the keccak row cannot be eliminated this way at all; both points are detailed in Section F.

| Assumption class | Required premise | Evidence owner |
|---|---|---|
| Cryptography and ceremony | Groth16 soundness for the verification key pinned in the audited artifact manifest; Poseidon collision resistance; EdDSA unforgeability. The pinned verifier source includes scalar/base fields, alpha/beta/gamma/delta coordinates, and IC0/IC1 public-input basis coordinates, with structural VK sanity tests. Under pinned `circom compiler 2.2.3`, the local artifact check rebuilt the R1CS to the recorded digest, reproduced the downloaded prover WASM and witness-calculator hashes, verified the downloaded zkey against the rebuilt R1CS, and checked the generated Solidity verifier constants against the zkey-derived verification key. The pinned R2 artifacts identify the audited bundle, but production ceremony/transcript review remains a separate external prerequisite before deployment. | Cryptography / ceremony and artifact-provenance review. |

| Assumption class | Required premise | Evidence owner |
|---|---|---|
| Circuit relation | The deployed `spend_10x4_v1` R1CS implements the intended spend relation, including Merkle membership, authority binding, public-signal binding, balance, range, and no-wrap constraints. The ZK `npmtest` gate passes 43 circuit tests for valid witnesses, mutation rejection, sentinel-zero padding, vector parity, and proof verification, but does not replace an independent circuit audit or proof. | Circuit audit and circuit tests. |
| External contracts | Supported ERC-20 tokens and Permit2 behave according to the transfer, balance, signature, nonce/deadline, and witness semantics expected by the pool. Supported tokens must have coherent `balanceOf` reads, exact sender and recipient deltas for `transfer`, and exact Permit2-funded deposit deltas. This review proves these exact-delta and read-after-write semantics in-model over an executable `(token,account)` ledger, kernel-clean at `[propext,Quot.sound]` (Section F), shipped as four counted gate atoms rather than a relabel of this assumption's `assumed` marker; only conformance of a production ERC-20 to that ledger model remains assumed here. Fee-on-transfer, rebasing, fake-success, and other nonstandard tokens are outside the supported-token policy unless a reviewed wrapper or adapter restores these exact-delta semantics. The model checks the pool-side Permit2 argument flow, not Permit2's cryptographic acceptance relation. | Token integration review. |
| Lean/Yul external-call and byte-packing model | ABI encoding, keccak memory-slice behavior, Permit2 witness-transfer ABI, verifier-router ABI, verifier proof ABI, SHA public-signal packing, and BN254 precompile probe memory/return semantics refine the production compiler/runtime behavior. | Verity model and integration review. |

| Assumption class | Required premise | Evidence owner |
|---|---|---|
| Transient reentrancy guard | OpenZeppelin `ReentrancyGuardTransient` and EIP-1153 `tload/tstore` semantics provide the mutex assumed by `deposit`, `transfer`, `withdraw`, and `emergencyWithdraw`. This review proves revert-on-re-entry on the delivered `emergencyWithdraw` in-model and kernel-clean at `[propext, Quot.sound]` (Section F), shipped as a counted gate atom rather than a relabel of the guard's `assumed` marker; only the EIP-1153 opcode-faithfulness of `tload/tstore` remains assumed here. | Solidity dependency and EVM integration review. |
| Verifier-router governance/interface | The owner sets only a live router implementing the expected interface, and each `getCircuit` result routes to the intended verifier for the circuit ID. The delivered evidence structurally checks owner-only mutation paths, active-route and shape checks before spend mutation, shape immutability, duplicate-verifier exclusion, pause-to-inactive behavior, and disabled renounce. It still does not prove production governance process, live-code conformance of an arbitrary replacement router, or intended verifier identity across future governance actions. | Deployment and governance review. |
| Deployment owner and EVM atomicity | The initializer owner is a valid nonzero owner, and EVM revert atomicity restores earlier withdrawal mutations if later token settlement reverts. | Deployment and execution-environment review. |
| User/key custody | Spending keys are generated, distributed, and stored so only the intended user can spend the corresponding notes. | Client, wallet, and custody review. |
| Availability and operations | Chain inclusion/finality, route availability, proving artifacts, gas, emergency procedures, and event/ciphertext data availability are sufficient for the exit and usability guarantees that depend on them. | Deployment and operations. |

## Out-of-model workstreams

Out-of-model items are not formal claims in this report and are not used as proof evidence. Some have Lean/report names so the manifest can track the boundary explicitly; the name is an index entry, not a proof or semantic model of the external behavior.

| Workstream | Not claimed here | Why separate |
|---|---|---|
| Circuit internals | Authority binding, circuit bookkeeping, balance/range arithmetic, and dummy slot semantics inside the R1CS. | These are circuit facts, not Solidity-facing contract-control-flow facts. |
| Merkle root-value correctness | That the root computed by the modeled tree equals the canonical incremental-Merkle root of the appended leaf set. The tree arithmetic is re-implemented concretely in `InternalLazyIMT.lean`; what is not proved is that this computation, over the opaque `PoseidonT3.hash`, yields the canonical root. | Closing it needs structural/collision properties of `PoseidonT3.hash` plus spine induction; that residue is carried by the Poseidon assumption and the circuit Merkle-membership re-constraint (`IC-C2`), not the contract control-flow proof. |
| Execution environment | General EVM atomicity/revert semantics beyond the explicit withdrawal premise, chain censorship resistance, and complete liveness. | These require VM/network and operational analysis beyond the Verity contract model. |
| Future upgrades | Storage-layout compatibility and safety of future implementations. | The current model can prove owner gating, write-set exclusion, and snapshot layout anchors, not future code correctness. |
| Completeness and usability | Honest witness construction, no-revert completeness for every valid off-chain intent, and all user-recovery paths. | These combine client, circuit, data, chain, and runtime conditions. |

| Workstream | Not claimed here | Why separate |
|---|---|---|
| Privacy and product/backend behavior | End-to-end privacy games, metadata leakage, relayer/backend custody, hosted service behavior, deployment controls, and dashboard/API behavior. | These are protocol/product surfaces outside the contract-surface proof. |

## Axioms and boundary names

In Lean, an opaque proposition used as a hypothesis is axiom-like: Lean may reason from it without proving it inside this development. In this report, **external trust assumption** is the audit classification for the subset of those axiom-like premises that the stated security interpretation actually requires. A **tracked boundary** is different: it is a named exclusion from the formal claim, not an extra premise. Keeping those entries is useful because it prevents a contract-surface proof from being mistaken for a proof of circuit arithmetic, privacy, future upgrades, backend operations, or chain liveness.

The detailed index of opaque names, presentation predicates, external assumptions, and tracked-boundary names is in Section D.

## Formal-audit conclusion

The formal audit establishes the stated contract-surface obligations under the explicit external assumptions above. Contract-side claims that can be checked against the Verity model and substantive Lean evidence files are machine-checked; cryptographic, circuit, environmental, operational, privacy, and upgrade claims are recorded as external assumptions or tracked workstreams. The Merkle/root-history facts are deliberately narrow: they prove the returned-root and `rootSeen` update for a successful concrete `_insertLeaves` execution, not that the returned root is recomputed on-chain (see the Merkle root handling note). The remaining formally scoped risk is the explicit list of assumptions and tracked boundaries above.

# 6 Findings

Four scored Medium findings and seven scored Low findings remain after applying the audited scope. Every one of them is in the ZK release-engineering / CI / tooling bucket: they affect the layer that builds, ships, and verifies Unlink's zero-knowledge artifacts (the proving key, the witness-generation WASM, the witness calculator, and the on-chain verifier source), plus the CI gates around those scripts. The contract/circuit security bucket (the Solidity pool, the verifier router, and the `spend_10x4_v1` circuit) has no scored finding at any severity; no scored finding breaks the pool or the circuit.

**Remediation status: all scored findings Resolved**

This section states each finding as originally scored. Unlink has since submitted a Round-1 remediation closeout, and we verified each fix against the merged monorepo code. **All eleven scored findings are Resolved** (MED-03, the last open one, was closed by PR #983, which makes strict source-git provenance the default). The status cell on each finding below reflects this, and the per-finding evidence is collected in the remediation-status table (Section 6).

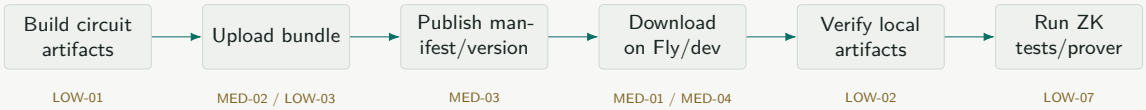
**How to read this section**

Unlink's prover needs three artifacts produced offline once per circuit: `spend_10x4_v1.zkey` (the proving key), `spend_10x4_v1.wasm` (the witness-generation program), and `witness_calculator.js` (the wrapper used by the SDK and backend). They are built locally, uploaded to Cloudflare R2, and downloaded by every Fly machine and developer. Each finding below identifies a step in that pipeline where a manual mistake, CI gap, or supply-chain compromise can cause a production outage or, in the worst case, an arbitrary file-write primitive in the operator's environment. The on-chain verifier remains the hard binding against forged proofs.

**Claim labels**

**Confirmed** means reproduced locally or in a workflow simulation. **Inferred** means derived from the inspected code path without needing an end-to-end exploit. **Trust-assumed** means accepted by the stated trust model. **Deferred** means credible production work that belongs to a later scope.

## Artifact lifecycle map




**Where each finding sits in the artifact life cycle.**

| | |
|---|---|
| Build | LOW-01 (hardener can think a partially patched verifier is fully patched) |
| Upload | MED-02 (version label is a source-derived prefix, not a durable bundle identity) |
| | LOW-03 (publisher/consumer manifest contract: WC can be omitted at upload) |
| Download | MED-01 (manifest can write files outside the artifact dir) |
| | MED-04 (the warm-boot fast path does not detect post-bootstrap volume drift) |
| Reproduce | MED-03 (drift checker compares without binding bundle source identity) |
| CI gating | LOW-04 (Medium: the ZK workflow's path filter omits scripts, tests, and lockfile changes) |
| | LOW-05 (no merge gate on Circomspect warning drift) |
| | LOW-06 (shared-vector changes do not run Solidity parity) |
| Test wiring | LOW-02 (`just test-zk` does not preflight contrary to its docstring) |
| | LOW-07 (witness test misclassifies rapidsnark proof-gen failures as a skip) |

**MED-01** · Medium · Resolved

### Remote artifact manifest names can escape the artifact directory

**Invariant**

Downloaded artifact paths must remain under the caller-provided `-output-dir`.

**What breaks**

When you run `just fetch-zk-artifacts`, the script `protocol/zk/scripts/download-artifacts.sh` is supposed to write every downloaded file inside the directory chosen by the caller via `-output-dir` (`/data/zk-artifacts` on Fly, a developer's checkout otherwise). The script keeps that promise by concatenating each circuit's name from `manifest.json` into a local path – for example, `<output-dir>/circuits/spend_10x4_v1.zkey`. The circuit name is supposed to be a bare identifier.

The script does not enforce that. `manifest.json` comes from a remote source (R2, the mutable `latest` prefix, or whatever URL `-artifacts-url` points at), and the script splices the strings it finds straight into filesystem paths. If a hostile manifest says the circuit's name is `../../pwn`, the destination becomes `<output-dir>/circuits/../../pwn.zkey`, which the OS resolves to a path two directories above the chosen output. No allowlist, no identifier regex, and no `realpath`-and-compare-prefix check is applied.


**Impact**

Anyone who controls the manifest source can write arbitrary bytes to any path the script's user can write to. On Fly the script runs as the `unlink` user before the engine starts; locally it runs as the developer. The SHA-256 check on each file does not help because the same compromised source also authors the checksum manifest. Callers lose the guarantee that downloads stay under `-output-dir`.

In production, the realistic attack path is upstream artifact-source compromise: writing to `artifacts.unlink.xyz` (R2 origin / TLS interception), since `download-artifacts.sh:20` defaults `ARTIFACTS_URL` to that origin and the Fly entrypoint plus documented operator workflows pin it. An attacker with that capability also has strictly stronger options (serving a malicious zkey, breaking proof acceptance via

crafted artifacts), so this finding's standalone severity sits at defense in depth rather than at "primary production attack path." Two factors still warrant Medium: the validation is missing, and the fix is cheap (allowlist of circuit names plus `realpath` prefix-check); and the local-dev exposure does not require any upstream compromise: a developer or CI job running with `--artifacts-url` pointed at an unknown server (a colleague's mirror, a fork's staging, a personal cache) sees the same path-traversal behavior against their own filesystem.

**Reproduce**

Confirmed end-to-end. A 127.0.0.1 HTTP server serves a manifest of `{"version":"v1","circuits":["../../pwn"]}` plus a matching checksum file. Running:

```
bash download-artifacts.sh --version v1 \
  --output-dir /tmp/pocs/med01/artifacts \
  --artifacts-url http://127.0.0.1:8765
```

exits cleanly with `All artifacts verified.`, and `find` finds the downloaded payload at `/tmp/pocs/med01/pwn.json`, `/tmp/pocs/med01/pwn.zkey`, `/tmp/pocs/med01/pwn/witness_calculator.js`, `/tmp/pocs/pwn.wasm`, and `/tmp/pocs/pwn.bin`, all outside the intended `artifacts/` subtree, including two directories up for the `wasm/${circuit}/${circuit}.wasm` layout.

**Fix and verification**

Implementation: reject any remote circuit name not in the local `codegen.sh/circuits.json` allowlist, additionally enforce `^[A-Za-z0-9_-]+$` on the name, and canonicalize every destination with `realpath -m` before writing, rejecting paths that escape the artifact directory. For production, prefer pinned versions over `latest` and authenticate the manifest with a signature independent of R2 transport.
Validation: add a shell regression test that serves the malicious manifest above and asserts the script fails before any file appears outside `--output-dir`.

---

**MED-02** · Medium · Resolved

**Artifact version is a source-derived prefix, not a durable bundle identity**

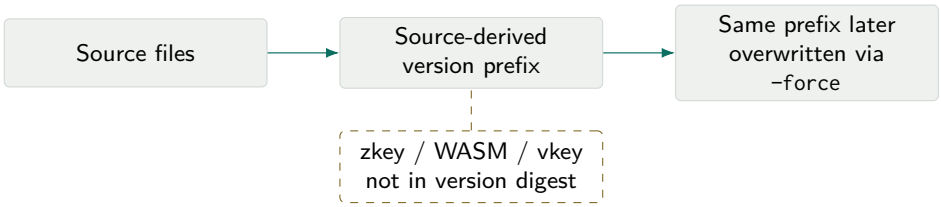
**Invariant**

A version identifier used as a release pin must either be derived from the full bundle (so the same identifier always resolves to the same bytes) or be backed by a write-once / append-only artifact prefix discipline that makes overwriting an existing identifier impossible without operator action. A mutable, source-derived label is not, by itself, a release identity.

**What breaks**

When Unlink uploads a new artifact bundle, `upload-artifacts.sh` computes a version label and publishes the bundle under `s3://unlink-zk-artifacts/v<label>/`. Fly machines pin `ARTIFACT_VERSION=v<label>` (production explicitly requires a pinned version, not `latest`: see `infra/fly/entrypoint.sh:27-44` and `fly.production.toml:10-14`) and download from that prefix.
The label is computed as `v` plus the first 16 hex characters of `sha256(circuits.json`

+ circomkit.json + sorted *.circom files) (upload-artifacts.sh:60-72). Artifact bytes – zkey, WASM, .bin, witness_calculator.js, the per-circuit checksum manifests, the exported verification key – never enter the hash. There is also a workspace-state quirk: the find sweep picks up gitignored generated wrappers under circuits/main/, so the same committed source produces different version labels on a clean checkout vs. a workspace mid-build.
Bundle bytes are not unverified. The downloader does per-file SHA-256 verification against each circuit's checksum manifest at download time (download-artifacts.sh:146-178) and runs a final verification pass over every file in the manifest, exiting non-zero on any missing or mismatched file (download-artifacts.sh:231-268). So an operator pinned to a given ARTIFACT_VERSION does receive checksum-bound bytes. The defect is one level up: the per-prefix checksum manifest is authoritative *for whichever bundle was last uploaded under that prefix*, and --force can replace both the bytes and the checksum manifest under an existing prefix (upload-artifacts.sh:101-107).




**Impact**

Two genuinely different artifact bundles (for instance, the output of two different trusted-setup ceremonies for the same circuit, or a hotfix re-upload) can occupy the same version label over time. Because the checksum manifest lives under that mutable prefix, a later download verifies the *new* bytes successfully against the new manifest.
The runtime risk is therefore not unverified proving artifacts; it is release identity ambiguity:

- **Rollback determinism.** Rolling back to a previous ARTIFACT_VERSION can silently mean different bytes than were running when that version was first deployed.
- **Forensic identity.** ARTIFACT_VERSION is not a durable handle for "the exact bundle that produced this proof," so audit-time forensics, ceremony attribution, and incident response cannot rely on it.
- **Cross-machine divergence.** Combined with the warm-boot sentinel that skips re-fetch when a per-version marker exists (Finding MED-04), two machines pinned to the same version can diverge depending on when they bootstrapped relative to a force-reupload.

The on-chain verifier remains the hard binding against forged proofs, so the runtime symptom of a divergence is proof generation failing against the deployed verifier, not silent proof acceptance.

**Reproduce**

Confirmed by reading the script's version-derivation logic (upload-artifacts.sh:60-72) and the existence of the --force overwrite path (upload-artifacts.sh:101-107). The dev-side rerun matched the audit reproduction: same source, three different artifact bundles, identical version label.

Same source, three different artifact bundles, identical label:

```
bundle A (zkey="ceremony-A", wasm="wasm-A"): v3eba9a855f53cb7d
bundle B (zkey="ceremony-B", wasm="wasm-A"): v3eba9a855f53cb7d
bundle C (zkey="ceremony-C", wasm="wasm-B"): v3eba9a855f53cb7d
```

Same committed source, different workspace, different label:

```
clean checkout:                  v3eba9a855f53cb7d
with circuits/main/<gen>.circom: v5cfbb5b2b1a393f8
```

These bundles all pass downloader byte verification under the same `ARTIFACT_VERSION` because the checksum manifest is written alongside the bundle in the same mutable prefix.

**Fix and verification**

Implementation: compute `VERSION` over the full bundle by hashing the sorted per-file checksum manifests plus the top-level manifest, then derive the version label from that digest. Drop or strongly gate `-force` behind an explicit break-glass procedure, or move artifacts to a write-once / append-only R2 prefix discipline so the same label cannot resolve to two different bundles. Additionally, embed the final bundle digest into `manifest.json` so deploy logs and incident response can compare an immutable bundle identity even when the version prefix is a release-management convenience. Restrict the `find` source set to a committed allowlist so workspace state cannot perturb the version. Have the drift checker compare the exported vkey against the committed Solidity verifier constants.
Validation: add a test that mutates zkey, WASM, vkey, and checksum bytes and requires a different bundle digest for each mutation; add a test that attempts to overwrite an existing prefix without the break-glass flag and asserts the publisher refuses.

---

**MED-03** · Medium · Resolved

### Drift check compares current source to fetched bundle without binding bundle source identity

**Invariant**

Before reporting source/artifact drift, the gate must bind the downloaded bundle to the source checkout under test – either by requiring `manifest.gitSha` to match `git rev-parse --short HEAD`, or by requiring an explicit immutable artifact version tied to the release SHA. Only then may checksum mismatches be reported as source drift.

**What breaks**

`check-zk-artifacts.sh` is the official "is the downloaded bundle clean?" command. It rebuilds the circuit from source, compares hashes against the downloaded checksum manifest, and runs `snarkjs zkey verify` to confirm the proving key matches the rebuilt constraint system.
The gate has no bundle-identity precondition. The checker requires `manifest.json` to exist, but never reads `manifest.gitSha`; its only `jq` lookup is for checksum entries

under `.files[$f].sha256`. It then verifies downloaded artifact bytes against the checksum manifest, rebuilds the current checkout, and compares current source-built WASM/witness bytes against the downloaded bundle's checksums.
The documented fetch path makes this gap routine, not theoretical. The downloader defaults `VERSION="latest"`, the `just fetch-zk-artifacts` recipe defaults its version argument to `latest`, and the nightly proof workflow regenerates current `circuits.json`, downloads `latest`, then runs the drift checker. None of these steps confirms that the fetched bundle is for the checkout under test before checksum comparison.
At the audit SHA, running this workflow against the live `latest` bundle reports WASM and `witness_calculator.js` mismatches. The zkey verifies against the source-built R1CS, but the WASM and wrapper bytes do not match the source-built outputs. The downloaded bundle's manifest records `gitSha: d8cf16ba`, not the audit target's `7617b3e`; the published `latest` bundle was built from a different commit. This observation is therefore an identity-precondition failure, not a same-source non-reproducibility proof for the audit SHA – a separate rerun at `d8cf16ba`, or with a bundle whose `manifest.gitSha` equals `7617b3e`, would be required to make the latter claim.
The same manifest also records nonsense byte sizes (e.g., `"size": 101` for a 220 MB zkey). This is the upload script's `file_size()` returning `stat -c%s` on a symlink, which on GNU coreutils returns the symlink target path length rather than the target file size. SHA-256 is the security-relevant integrity check; this is an artifact-metadata quality gap that supports the same "gate hygiene" narrative.


#### Impact

The gate is the operational signal meant to tell operators when the deployed bundle has diverged from the audited source. As wired, it can report mismatches whenever the fetched `latest` bundle was built from a different commit than the checkout under test. Real source drift is therefore indistinguishable from a version-selection error without manually inspecting `manifest.gitSha`. This produces operator desensitisation: a drift alarm in production looks the same as the expected baseline. The on-chain verifier remains the hard binding against forged proofs, so this is a release/operations signal-quality issue, not a direct soundness issue.

#### Reproduce

**Drift-check semantics.** A stub of the `check-zk-artifacts.sh` `check_hash` function against a `downloaded` WASM (hash `d9db0aec...`) and a divergent `source-built` WASM (hash `a1701606...`) prints `OK: downloaded WASM` then `ERROR: source-built WASM checksum mismatch`, returning `errors=1`. Identical control flow to the script at the audit SHA.
**Identity precondition is absent.** `rg gitSha protocol/zk/scripts/check-zk-artifacts.sh justfile` returns zero hits. The script's only `jq` use against `manifest.json` is `.files[$f].sha256`.
**Symlink-size bug.** On GNU coreutils 9.4, the script's

```
file_size() { stat -f%z "$1" 2>/dev/null || stat -c%s "$1"; }
```

on a symlink to a 2,097,152-byte file returns `48` (the symlink target path length). `stat -L -c%s` returns `2097152`. This matches the `size: 101 / 119` values seen in the published manifest.

**Not yet demonstrated.** A same-source non-reproducibility claim for the audit SHA would require rerunning the checker at `d8cf16ba` (the manifest's `gitSha`), or fetching an immutable bundle whose manifest `gitSha` equals `7617b3e`. The report does not assert this; the surviving claim is identity-precondition failure plus the symlink-size metadata bug.

#### Fix and verification

Implementation: add an early bundle-identity check to `check-zk-artifacts.sh` that compares `manifest.gitSha` against `git rev-parse --short HEAD` and exits with an explicit identity error before any checksum comparison; require explicit non-`latest` artifact versions for audit and release validation in `fetch-zk-artifacts` and the nightly workflow; and patch `file_size()` to dereference symlinks (`stat -L -c%s`). If byte-for-byte WASM reproducibility is not feasible across platforms, swap the gate for deterministic semantic artifacts (R1CS, vkey, signed build attestation) and remove the WASM byte-check claim rather than leaving a baseline that depends on unbound bundle identity.

Validation: run the drift check at the same `gitSha` as the fetched bundle and require it to pass; run it against a deliberately mismatched-`gitSha` bundle and require an identity error before checksum comparison; add a symlink-size fixture proving `stat -L` reports the target file size.

---

**MED-04** · Low · Resolved

### Warm-boot sentinel does not detect post-bootstrap volume drift

#### Invariant

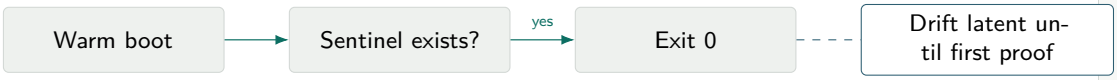
A warm-boot liveness sentinel should either be explicitly documented as non-integrity state or paired with a cheap local consistency check before reporting bootstrap success.

#### What breaks

When a Fly machine reboots, the entrypoint runs `download-artifacts.sh` with `--skip-if-complete`. The flag is intentionally a liveness optimization: the entrypoint comment says it "decouples warm-boot liveness from R2 availability" (`infra/fly/entrypoint.sh:39-44`), and the script's help text and fast-path comment frame it the same way (`protocol/zk/scripts/download-artifacts.sh:38-40`, `download-artifacts.sh:58-65`). The deployment docs (`docs/internal/deploy.md:279-284`) and Fly config (`fly.staging.toml:99-104`, `fly.production.toml:75-86`) match. So the operational gate does not promise warm-boot integrity – it promises that warm boots avoid R2 once a prior cold bootstrap completed.

The cold path does establish integrity. It fetches the manifest, fetches per-circuit checksum files, hashes each local file against the expected SHA-256 (`file_verified()` at `download-artifacts.sh:77-81`), and runs a final pass over every required file (`download-artifacts.sh:231-270`). Only after that does it copy the manifest and `touch` the per-version sentinel (`download-artifacts.sh:294-301`), with an explicit

comment that the sentinel is "written last so a crash anywhere above leaves the sentinel absent and forces a full re-run next boot." Interrupted cold-bootstrap is therefore safe by construction.
What remains is post-bootstrap volume drift. The sentinel is a normal file on the persistent volume, written without locking, file lease, or content-addressed manifest. It can coexist with stale artifacts if the volume is restored from an inconsistent snapshot, manually repaired incorrectly, modified by another same-user process, or partially refreshed under a different code path. Once that happens, the warm-boot fast path trusts the sentinel and exits 0. The engine starts and `/health` returns success; `ProverConfig::from_env()` validates path strings but not file contents, and `LocalProver` only opens the artifact files on the first proof request. So drift remains latent until then.




#### Impact

Volume drift after a successful cold bootstrap (disk fault, manual maintenance, snapshot restoration, or modification by another same-user process) is invisible to the warm-boot fast path. The engine and health checks proceed; the first proof request is where the failure surfaces, and recovery requires an operator to delete the sentinel so the cold path re-runs. This is a recovery/observability gap, not a remote artifact-supply-chain bypass. The local-write adversary required to synthesize the auditor's specific PoC is not in the contract or circuit threat model; it is in the operational handoff scope.

#### Reproduce

**Drift-state regression fixture (not a production cold-bootstrap state).** `/tmp/pocs/med04/artifacts` contains exactly one file: an empty `.bootstrap-complete-vbogus` sentinel; no manifest, no checksums, no zkey, no WASM. This state is manufactured by hand – the production cold-bootstrap path will not produce it. Pointing the script at a deliberately broken URL so that any actual fetch would fail loudly:

```
bash download-artifacts.sh --version vbogus \
  --output-dir .../artifacts \
  --artifacts-url http://127.0.0.1:9 \
  --skip-if-complete
==> Bootstrap sentinel present for version vbogus, skipping download.
exit: 0
```

Directory listing afterward is unchanged: still just the sentinel. This demonstrates that the fast path treats sentinel-presence as sufficient even when the artifacts referenced by that version are absent, which is the operational drift case we want a cheap local recheck to cover.

#### Fix and verification

Implementation: keep the sentinel as a liveness optimization but add a cheap local consistency check before treating it as bootstrap success. On cold bootstrap, persist a small local manifest of expected filenames, sizes, and hashes alongside the sentinel; on `-skip-if-complete`, re-hash every required file against that local

manifest and only exit 0 on full success. Clear the sentinel at the start of any explicit refresh and rewrite it only after the final verification pass. Consider acquiring a per-version file lock around the cold-bootstrap path so a concurrent same-version refresh cannot interleave with the sentinel write.
Validation: create only `.bootstrap-complete-vX`, run `-skip-if-complete`, and assert a nonzero exit because the recorded local manifest reports missing required artifacts.

**LOW-01** · Low · Resolved

### Verifier hardener can treat partially patched output as already hardened

#### Invariant

The verifier hardener may return unchanged only when every expected hardening marker is present.

#### What breaks

The on-chain Groth16 verifier is generated by snarkjs and then patched by `harden_verifier.py` to plug two known weaknesses in the snarkjs template (Zellic V12 #53438 and #53341): adding `returndatasize()` checks after every BN254 precompile call, and adding field-bound checks on the proof's elliptic-curve coordinates. The script is supposed to be idempotent so it can be re-run safely.
The idempotence check is one line: `if G1_MUL_NEW in src: return src`. That one substring covers only the first patch class. The script never confirms that `checkFieldQ`, the pairing-call `returndatasize` check, or the eight `checkFieldQ(calldataload(...))` coordinate checks are also present.

| Marker | Required | Partial verifier has it? |
|---|---|---|
| G1 returndata check | Yes | Yes |
| Pairing returndata check | Yes | No |
| `checkFieldQ` helper | Yes | No |
| 8 coordinate checks | Yes | No |

#### Impact

The verifier shipped with the audit target carries both patch classes, and the current contracts PR workflow runs `protocol/contracts/test/VerifierHardening.t.sol`, which mutates proof coordinates against the BN254 base field and mocks empty or overlong returndata from BN254 precompiles, asserting the actual committed `Spend10x4V1Verifier` rejects all of those cases. A verifier regenerated under `protocol/contracts/src/verifiers/` that is missing the Fq coordinate checks or the exact returndata-size checks should therefore fail this gate before being merged.
The residual risk is hardener self-validation: the script can classify a partially hardened verifier as complete and rely on those downstream behavioural tests to catch the resulting artifact. If a future snarkjs template ships the G1 returndata pattern by default, the early return on `G1_MUL_NEW` would no-op the hardener without adding the field-bound coordinate checks. That is safe only as long as `VerifierHardening.t.sol` stays present, enabled in the contracts workflow, and scoped

to every future verifier shape (additional selectors, additional precompile calls, alternative AST layouts). Out-of-PR regeneration paths – a developer running `just export-verifiers` locally and shipping the result through a hotfix branch that doesn't exercise the Forge gate, or any path that bypasses `forge test` – would no longer be caught at all. The hardener should not require an external safety net to be correct.

#### Reproduce

**Hardener self-validation.** Confirmed against the committed `spend_10x4_v1_verifier.sol`. Starting from the audited verifier, the `checkFieldQ` helper and the eight `checkFieldQ(calldataload(...))` calls were stripped, leaving the two `eq(returndatasize(), 0x40)` markers (`G1_MUL_NEW`) in place. Running:

```
python3 harden_verifier.py vfile.sol
```

prints `already hardened: vfile.sol`. Post-run grep counts: `function checkFieldQ` = 0, `checkFieldQ(calldataload` = 0. The hardener accepted a verifier missing every Fq coordinate check.

**Note on what this demonstrates.** This is a script self-validation failure, not a demonstration that the contracts PR gate would let an unsafe regenerated verifier ship. The downstream Forge hardening tests at `protocol/contracts/test/VerifierHardening.t.sol` were verified to exist at the audit SHA, to be triggered by changes under `protocol/contracts/**` in `.github/workflows/contracts.yml`, and to assert the exact properties (proof-coordinate Fq rejection, pairing/G1 returndata-size rejection) that a stripped verifier would lose. The hardener's role is to guarantee correctness on its own; today, correctness depends on those tests staying in the gate.

#### Fix and verification

Implementation: require all five marker classes before returning unchanged (two G1 returndata, one pairing returndata, the `checkFieldQ` helper, the eight coordinate-check calls); abort with an explicit partial-hardening error otherwise, so the hardener no longer relies on downstream behavioural tests. Add a script-level regression fixture that feeds a returndata-only verifier and asserts the hardener fails or inserts the missing patches.

Process: keep `protocol/contracts/test/VerifierHardening.t.sol` present and enabled in the contracts PR workflow, and treat any change that skips, weakens, or narrows those tests (including excluding the test path, mocking around it, or limiting it to one verifier variant) as security-sensitive review material until the hardener is self-validating.

Validation: run the partial-hardening fixture in CI before accepting regenerated verifier output.

---

**LOW-02** · Low · Resolved

### just test-zk does not preflight artifacts contrary to its own docstring

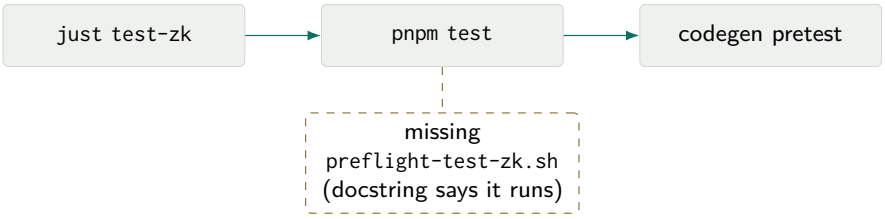
#### Invariant

`just test-zk` must either invoke `preflight-test-zk.sh` (consistent with the docu-

mented behavior in `protocol/zk/tests/spend.proof.test.ts`) or its documented behavior must match what it actually does.

**What breaks**

`preflight-test-zk.sh` is documented as the file that fails fast when prover artifacts aren't bootstrapped before `just test-zk`. It checks that the manifest, checksum file, zkey, WASM, and witness calculator all exist, then symlinks them into circomkit's build directory so the proof test consumes the downloaded release bundle.
The target does not invoke it. `just test-zk` runs `pnpm --filter "@unlink/zk" test` (`justfile:319-320`); the `pretest` hook only runs `codegen.sh circuits-json` (`protocol/zk/package.json:6-9`); and the proof test itself silently falls back to `circomkit.setup()` when the staged artifacts are missing (`spend.proof.test.ts:28-43`), generating fresh local proving keys. The proof test's own docstring (`spend.proof.test.ts:13-15`) explicitly claims `just test-zk` performs preflight: "`just test-zk` preflights the downloaded R2 artifacts and stages them into CircomKit's build layout. Direct Vitest invocations still compile/setup as a fallback when the staged artifacts are absent." That claim is currently inaccurate; the recipe does not preflight.
The actual release-artifact gate is the scheduled `.github/workflows/zk-proof-nightly.yml` workflow, which does `download-artifacts.sh` → `check-zk-artifacts.sh` → `preflight-test-zk.sh` → `pnpm vitest run tests/spend.proof.test.ts` (`zk-proof-nightly.yml:67-76`). PR-time CI does not run this proof test at all: `.github/workflows/zk.yml` runs only the `circomspect:` job and the `alias-check:` job (a specific structural test), neither of which uses the zkey. The proof test is also gated by `describe.skipIf(!!process.env.SKIP_PROOF_TESTS)` (`spend.proof.test.ts:17`), but `SKIP_PROOF_TESTS` is never set in any workflow.




**Impact**

The documented `just test-zk` flow does not preflight, contrary to the docstring at `spend.proof.test.ts:13-15`. Developers who follow the documented command see fresh-setup output (a silent fallback to `circomkit.setup()` when staged artifacts are missing) rather than a release-artifact validation, and may believe the test is exercising the staged R2 bundle. PR CI is unaffected because PR workflows do not run this proof test; the release-artifact gate is the nightly proof workflow, which does preflight. The realistic risk is developer confusion and a missing local convenience for verifying against staged R2 artifacts, not a CI release-gate bypass.

**Reproduce**

**Configured runners (read at the audit SHA):**

```
justfile :: test-zk:
    pnpm --filter "@unlink/zk" test
protocol/zk/package.json :: scripts:
    "pretest": "bash scripts/codegen.sh circuits-json",
    "test":    "vitest run --no-file-parallelism"
```

**Search:** `grep -rn "preflight-test-zk" justfile protocol/zk/package.json protocol/zk/scripts/` returns zero hits outside `preflight-test-zk.sh` itself.
**Docstring claim that conflicts with wiring** (`spend.proof.test.ts:13-15`):

```
* `just test-zk` preflights the downloaded R2 artifacts and stages them
* into CircomKit's build layout. Direct Vitest invocations still
* compile/setup as a fallback when the staged artifacts are absent.
```

**Fallback wiring** (`spend.proof.test.ts:28-43`): the `beforeAll` block runs `circomkit.compile` and `circomkit.setup` when staged WASM or zkey are absent – i.e. the silent local-fallback path.

#### Fix and verification

Either invoke `preflight-test-zk.sh` from the `test-zk` recipe (and have it re-hash staged files against the checksum manifest before staging, failing the proof test when staged artifacts are missing in release-artifact mode), or fix the docstring at `spend.proof.test.ts:13-15` to match the current behavior (no preflight; local fallback to `circomkit.setup()`) and document a distinct command for staged-artifact testing (`test-zk-staged` or similar). The nightly proof workflow should keep its preflight step regardless.
Validation: pick one of the two fixes; either way, `grep -rn "preflight-test-zk" justfile protocol/zk/package.json protocol/zk/scripts/` should agree with the proof test's docstring after the change.

---

**LOW-03** · Low · Resolved

### Artifact publisher can omit the witness calculator required by ZK test gates

#### Invariant

The publisher and all consumers must agree on the required files in an artifact bundle.

#### What breaks

A complete artifact bundle has four files per circuit: the zkey, the WASM, a native `.bin` witness graph, and `witness_calculator.js` (the Node.js wrapper around the WASM that the SDK and backend use).
`upload-artifacts.sh` validates only the first three. The fourth is appended to the checksum manifest inside an `if [ -f "$wc_js" ]` guard with no `else`; if the file is missing locally, the upload still proceeds and the manifest omits the entry. Every downstream consumer disagrees: `download-artifacts.sh` expects it, `check-zk-artifacts.sh` checks it, `preflight-test-zk.sh` requires it.
The documented release path does generate this file. `just rebuild-zk` chains `compile-zk` → `setup-zk` → `export-verifiers` → `setup-prover-artifacts` → `generate-witness-inputs` → `test-witness` (`justfile:291-300`). `compile-zk` delegates to `npx circomkit compile` (`justfile:306-308`, `protocol/zk/scripts/codegen.sh:139-146`), which produces the `*_js/witness_calculator.js` alongside the WASM, and `protocol/backend/scripts/setup-prover-artifacts.sh:66-79` copies that wrapper into `artifacts/wasm/<variant>/witness_calculator.js` when it is present. So in the normal rebuild flow the wrapper is generated and copied. The defect is that there

is no upload-time fail-closed enforcement of that contract: a bundle missing the wrapper passes upload validation, the checksum manifest then omits it, and the downloader silently accepts the manifest as authoritative even though preflight and drift checks require the wrapper.

**Impact**

Publisher and consumer disagree on the bundle contract. A bundle missing `witness_calculator.js` passes the publisher's own validation, then passes the downloader's SHA-256 verification (no checksum entry, no file requested), then fails the downstream preflight and drift checks. The realistic ways to reach this state are not the standard `just rebuild-zk` flow but bypassing or partially running it: invoking `upload-artifacts.sh` directly without `rebuild-zk`; `setup-prover-artifacts.sh` skipping the wrapper copy because the sibling file is not present in the CircomKit build (a tooling regression in CircomKit's output shape); reusing an older staging directory after a circomkit upgrade that changed file layout; or manual edits to the staging directory before upload.

**Reproduce**

**Contract-consistency fixture (not a production rebuild state).** `/tmp/pocs/low03/artifacts` contains only `spend_10x4_v1.zkey`, `spend_10x4_v1.wasm`, `spend_10x4_v1.bin`, but no `witness_calculator.js`. This state is hand-constructed; the documented rebuild flow generates and copies the wrapper.
Running the upload-side check loop (lines 132–149 of `upload-artifacts.sh`): `missing=0`, so upload would proceed.
Running the preflight-side required-file loop (lines 14–39 of `preflight-test-zk.sh`):

```
preflight-side required-but-missing:
  - artifacts/checksums/spend_10x4_v1.json
  - artifacts/wasm/spend_10x4_v1/witness_calculator.js
```

The publisher accepts the partial bundle; the consumer side rejects it.

**Fix and verification**

Implementation: require `witness_calculator.js` at upload, or formalize an optional-file marker in the manifest schema that downloader/preflight/drift checker all respect.

| Component | Requires `witness_calculator.js`? |
|---|---|
| Upload | No |
| Download | Yes |
| Drift check | Yes |
| Preflight | Yes |

Validation: add a missing-witness-calculator fixture and require upload, download, drift check, and preflight to agree on pass/fail semantics.

**LOW-04** · Medium · Resolved

**ZK script and dependency changes do not trigger the ZK PR workflow**

**Invariant**

PR-time ZK workflow filters must cover files that can change ZK script behavior, ZK tests, or resolved ZK dependencies.

**What breaks**

`.github/workflows/zk.yml` is the PR-time gate that runs Circomspect (static analysis) and the alias structural test on the ZK package. Its trigger only lists the workflow file itself, the shared toolchain action, `protocol/zk/circuits/**`, and `protocol/zk/package.json`.

Scripts (`download-artifacts.sh`, `harden_verifier.py`, all the others under `protocol/zk/scripts/`), the ZK tests, `circomkit.json`, the root `pnpm-lock.yaml`, and the EdDSA dependency patch under `patches/` are all missing from this workflow's filter. A heavier proof workflow exists, but it runs on a schedule only – no `pull_request` trigger.

For accuracy: PRs that touch `protocol/zk/scripts/**` or `protocol/zk/tests/**` are not entirely outside PR CI. `.github/workflows/backend.yml` PR-triggers on `protocol/zk/**`, and `.github/workflows/ci.yml` runs on every PR with no path filter. Those workflows do not exercise the affected ZK scripts or the gates in `zk.yml`, however: Backend CI runs Rust/SDK/CLI builds, unit tests, and the committed-vector crypto parity test, with prover wiring mocked (`UNLINK_RAPIDSNARK_BIN=/usr/bin/true`); root CI runs lint, format, actionlint, BN254 modulus sweeps, and schema/ERD diffs. None of those jobs runs Circomspect, the alias structural test, `download-artifacts.sh`, `harden\_verifier.py`, or the witness/proof smoke path.

**Impact**

A PR that only edits, say, `download-artifacts.sh`, `harden_verifier.py`, the ZK tests, `circomkit.json`, or the lockfile-resolved ZK dependency graph lands without the ZK workflow's Circomspect SARIF gate or alias structural test. Other PR workflows (Backend, root CI) may still run, but they do not execute the affected artifact-download, verifier-hardening, Circomspect, alias, or proof-smoke paths. Regressions in exactly the surfaces that produced MED-01..MED-03 and LOW-01..LOW-03 therefore rely on the nightly job (post-merge), manual review, or unrelated checks catching incidental fallout.

| Changed file | Current ZK workflow triggers? | Should trigger? |
|---|---|---|
| `protocol/zk/scripts/download-artifacts.sh` | No | Yes |
| `protocol/zk/tests/spend.test.ts` | No | Yes |
| `pnpm-lock.yaml` | No | Yes |

**Reproduce**

The ZK workflow's path filter, against a simulated PR against six relevant files. *This shows the ZK workflow does not run, not that no PR workflow runs at all (Backend triggers on `protocol/zk/**` and root CI runs unfiltered; neither exercises the gates listed above).*

```
zk.yml pull_request.paths:
  .github/workflows/zk.yml
  .github/actions/setup-toolchain/**
  protocol/zk/circuits/**
  protocol/zk/package.json
```

```
Hypothetical changed files in a PR:
  protocol/zk/scripts/download-artifacts.sh        no match
  protocol/zk/scripts/harden_verifier.py           no match
  protocol/zk/tests/spend.test.ts                  no match
  protocol/zk/circomkit.json                       no match
  pnpm-lock.yaml                                   no match
  patches/@zk-kit__eddsa-poseidon@1.1.0.patch      no match

zk.yml runs? False
```

**Fix and verification**

Implementation: add `protocol/zk/scripts/**`, `protocol/zk/tests/**`, `protocol/zk/circomkit.json`, `protocol/zk/patches/**`, `protocol/zk/Dockerfile.witness-test`, `pnpm-lock.yaml`, and `patches/**` to `zk.yml`'s filter. For script-only PRs, add a cheap PR job that lints/syntax-checks scripts and exercises the downloader/preflight/drift-check smoke path against a pinned subset.
Validation: simulate vector, script, test, lockfile, and patch-only PRs and assert at least one ZK workflow job is selected for each.

---

**LOW-05** · Low · Resolved

### Circomspect warning drift does not fail the ZK PR workflow

**Invariant**

Static-analysis gates must fail on new unreviewed warning classes, not only on error-level findings.

**What breaks**

Circomspect is a static analyser for Circom circuits. The workflow `.github/workflows/zk.yml` runs it on every PR, captures findings in SARIF, and gates the merge. Warnings are visible to reviewers. The Circomspect step invokes `circomspect circuits/ --level INFO --sarif-file circomspect.sarif` (line 69) and writes the full human report to `$GITHUB_STEP_SUMMARY` via a structured block containing the tool version, input path, exit code, and a fenced code block with the captured stdout/stderr (lines 60–87). The SARIF itself is uploaded as a 30-day run artifact via `actions/upload-artifact` (lines 86–92). It is *not* uploaded via `github/codeql-action/upload-sarif`, so warnings do not become persistent GitHub Security-tab code-scanning alerts.
The gate itself is too narrow. It filters `select(.level == "error")` and only fails on error-level findings (lines 94–104). Warning- and note-level findings appear in the step summary but never fail the workflow. The circuit currently carries three reviewed warnings: two `<--` signal-assignment findings in `spend.circom` and one non-strict `Num2Bits` call in `merkle_proof.circom`. They were audited and accepted. The gate would also pass if a future PR added a fourth warning of the exact same class on a security-sensitive signal.

**Impact**

The categories Circomspect labels "warning" are precisely the ones that produced earlier circuit bugs the team had to remediate (`ENG-469` / `Num2Bits_strict`):

unconstrained signal assignments, non-strict bit conversions, and field-comparison/overflow assumptions.
Warning visibility is good (every finding is rendered in the PR's step summary), but warning-class enforcement is reviewer-discretional, not machine-enforced. A new warning of an already-accepted class (the same `<--` pattern in a new file, a fourth `Num2Bits_strict` site) would pass the gate without an automated blocker, relying on reviewers to read the step summary; the step summary itself is not a durable record (no persistence into GitHub's Security-tab alerts), so a deferred review or a forgotten warning has no merge-time backstop.

**Reproduce**

Synthesized SARIF containing one warning and no errors, fed through the exact jq query the workflow uses:

```
$ jq '[.runs[].results[] | select(.level == "error")] | length' cs.sarif
0
GATE PASS (despite 1 warning)
```

A strict version of the same query against warnings returns 1, so a warning-aware gate would have failed.

**Fix and verification**

Implementation (primary, warning-drift gate): commit a baseline of accepted warning identities and fail on any diff, or fail on any warning and suppress the known-acceptable cases in a documented wrapper. Keep malformed/missing SARIF fail-closed as it is today.
Independent hardening (warning persistence): upload the SARIF via `github/codeql-action/upload-sarif` so warnings persist as GitHub Security-tab code-scanning alerts rather than only as 30-day workflow artifacts and ephemeral step summaries. This addresses warning persistence, not warning-drift enforcement; the baseline-diff gate above is the primary fix.
Validation: feed one new warning and zero errors through the warning-drift wrapper; the gate must fail unless the warning is explicitly added to the reviewed baseline.

---

**LOW-06** · Low · Resolved

### Shared vector changes do not trigger Solidity parity tests

**Invariant**

Any change to a shared constant vector must run every consumer-side parity test that reads that vector.

**What breaks**

Unlink keeps a few cross-language "ground truth" constants – the BN254 scalar-field prime, the canonical circuit identifier – in `protocol/tests/vectors/constants.json`. Each implementation language (Solidity, Rust, SDK) has a test that reads the JSON and asserts its local copy matches. The Solidity side is `Constants.t.sol::ConstantsTest`, which calls `vm.readFile('../tests/vectors/constants.json')`.

`.github/workflows/contracts.yml`'s PR path filter includes `protocol/contracts/**` but not `protocol/tests/vectors/**`. A PR that only edits the shared vector JSON changes the inputs the Solidity test consumes, but the contracts workflow that runs that test is not triggered.
The Rust side of the parity check is covered by a different workflow: `.github/workflows/backend.yml`'s PR filter does include `protocol/tests/vectors/**` (line 13), and its `crypto-parity` job runs the cross-language Rust parity test (`cargo nextest run --cargo-profile ci -p crypto --test parity`, lines 96–113) on every vector-only PR. SDK-side parity coverage runs inside that job's committed-vector consumption. The missing coverage is the Solidity side specifically.

| Consumer test | Missing trigger path |
|---|---|
| `Constants.t.sol::ConstantsTest` | `protocol/tests/vectors/**` |
| `just check-constants` Solidity side | `protocol/tests/vectors/**` |

**Impact**

Vector-only PRs do trigger Backend CI's `crypto-parity` job, which runs the cross-language Rust parity test against the committed vectors. The gap is specifically the Solidity side: `Constants.t.sol::ConstantsTest` is the consumer test for the Solidity copy of these constants and lives in the contracts workflow, whose path filter does not include `protocol/tests/vectors/**`. A vector-only PR therefore exercises the Rust/SDK-vs-vectors invariant but not the Solidity-vs-vectors invariant. The Solidity copy currently matches; this is a CI guardrail gap on one of three language sides, not a live bypass.

**Reproduce**

```
contracts.yml pull_request.paths:
  .github/workflows/contracts.yml
  .github/actions/setup-toolchain/**
  protocol/contracts/**

Constants.t.sol reads from:
  vm.readFile('../tests/vectors/constants.json')   (lines 21, 44)

PR changing only protocol/tests/vectors/constants.json:
  contracts.yml match:  no  (Constants.t.sol::ConstantsTest skipped)
  backend.yml  match:   yes (crypto-parity job runs against vectors)
```

This is specifically the Solidity-side coverage gap; the Rust-side parity coverage already runs.

**Fix and verification**

Implementation: add `protocol/tests/vectors/**` to the Contracts workflow path filter or add a cheap always-on job that runs `just check-constants` on any change to vectors, canonical constants, or circuit-id plumbing.
Validation: simulate a PR that changes only `protocol/tests/vectors/constants.json`; the Contracts parity job should run.

**LOW-07** · Low · Resolved

### Witness test misclassifies rapidsnark proof-generation failures as a skip

#### Invariant

A proof-generation smoke test must fail when the prover binary is present but proof generation exits non-zero; any skip must be capability-proven and explicit.

#### What breaks

`test-witness.sh` is meant to confirm three things on a freshly built artifact bundle: the native and WASM witness generators produce identical witnesses, the `rapidsnark` prover can turn the native witness into a valid proof, and it can do the same with the WASM witness. The script ends with `All variants passed. The .bin files [...] are safe to deploy.` when all three steps succeed.

The official runner for this script is the dedicated Docker image `protocol/zk/Dockerfile.witness-test`, which builds `rapidsnark` in an amd64 stage and copies it to `/usr/local/bin/rapidsnark`. So the defect described here is *not* primarily a missing-binary path; the `rapidsnark` binary is normally present on the supported runner.

The defect is in the error classifier. The first `rapidsnark` call sits in `if rapidsnark ... ; then ... ; else ... ; fi`. The `else` branch unconditionally prints `SKIP: rapidsnark unavailable (likely arm64/Rosetta, AVX not supported)`. It does not distinguish "rapidsnark is not on PATH" from "rapidsnark exited 1 because the proof was invalid, the zkey was stale, the witness layout was incompatible, OOM, or the binary itself crashed." After the if-else, the script increments `PASS_COUNT` unconditionally.

#### Impact

A local rebuild smoke test (run after artifact regeneration via `just rebuild-zk`, which chains `compile-zk → setup-zk → export-verifiers → setup-prover-artifacts → generate-witness-inputs → test-witness`) can return success without exercising proof generation for the rebuilt bundle. `rapidsnark` exits non-zero because of a stale zkey, incompatible witness layout, or other runtime failure; the script reports `SKIP`, counts the variant as pass, and prints "safe to deploy."

Published `latest` artifacts are separately covered by `.github/workflows/zk-proof-nightly.yml`, which downloads the live bundle, drift-checks, stages with preflight, and runs the full Groth16 prove+verify suite in `protocol/zk/tests/spend.proof.test.ts` (real `(10,4)` proof, sparse `(1,1)` proof, tampered public-signals rejection). That workflow is `schedule`- and `workflow_dispatch`-only, so it acts as a post-publication alarm rather than an automatic pre-merge or pre-release gate. The realistic risk is therefore loss of signal in the rebuild smoke test and a misleading "safe to deploy" summary, with slower (next nightly) catch in production, not absence of any proof-generation gate in the system.

#### Reproduce

A fake `rapidsnark` on `PATH` that exits 1 reproduces the classifier bug. This demonstrates that any first-call non-zero exit is treated as availability-skip and counted as pass, regardless of whether the binary itself is present or absent.

```
bin/rapidsnark:
  #!/usr/bin/env bash
  echo "rapidsnark: simulated error" >&2
  exit 1
```

```
# Replay of test-witness.sh:147-176
if rapidsnark "$ZKEY" "$WTNS_NATIVE" "$PROOF" "$PUB"; then ...
else echo "  SKIP: rapidsnark unavailable ..."; fi
PASS_COUNT=$((PASS_COUNT+1))
# -> PASS_COUNT=1  FAIL_COUNT=0  Final summary: PASSED.
```

Zero proofs were generated; the script reports success.

**Fix and verification**

Implementation: separate capability from execution failure. Probe for the binary with `command -v rapidsnark` and, where applicable, CPU feature requirements before invoking proof generation; if any required capability is missing, exit with an explicit error unless an opt-in flag (`ALLOW_RAPIDSNARK_SKIP=1`) is set. A non-zero exit from `rapidsnark` when invoked must fail the script by default. Also update the final summary so "safe to deploy" is printed only when proof generation actually ran for every variant; otherwise print an explicit "witness-only pass; proof generation skipped" summary.
Validation: put a fake `rapidsnark` that exits 1 on `PATH`; the witness script must fail unless the explicit skip flag and capability probe both justify a skip. Separately, remove `rapidsnark` from the test container and confirm the script fails closed with an availability error rather than a silent pass.

## Verification notes

The final in-scope recheck covered the proof/state-machine boundary, sentinel padding, public-signal packing, verifier hardening, nullifier replay, Merkle-root validation, withdrawal-slot pinning, ciphertext context binding, symmetric ERC-20 accounting, and emergency-withdraw parity. The focused contract suite passed with 307 tests, including 12 invariant tests; an extended invariant re-run also passed 12 invariants at 512 runs and 32,768 calls per invariant, and the fuzz-only contract subset passed at `FOUNDRY_FUZZ_RUNS=8192`. The ZK suite passed with 43 tests after compiling the Circom circuits locally; on this review machine the ZK run required workspace-backed temporary storage and extended Vitest timeouts because the default `/tmp` tmpfs filled during repeated Circom compilation. The adjacent Rust/TypeScript parity gate, `cargo test -p crypto -test parity`, passed 21 tests covering Poseidon, EdDSA, nullifiers, commitments, multi-stage message hashing, SHA-256 public-signal hashing, and the padded `spend_10x4_v1` witness vectors. The cross-language constants gate, `just check-constants`, also passed across Solidity, Rust, and SDK tests. The verifier-key structural test, `forge test -match-contract VerifierKeyTest`, passed 11 positive and negative checks over the committed Groth16 verification-key coordinates. The configured Slither pass, `slither . -config-file slither.config.json -fail-high`, completed without a high-severity failure; the remaining reports were triaged to the intentional batch verifier/token calls, Permit2 balance-checked deposit flow, and ERC-7201/precompile assembly blocks. These checks did not produce a new credible High or Critical in-scope attack path. After restaging the downloaded artifacts with `preflight-test-zk.sh`, the adversarial witness-mutation guard `protocol/zk/scripts/adversarial-wtns-mutation.ts` also passed: the baseline proof verified, while a witness with `publicSignalsHash` mutated by one was rejected as an invalid proof.

## Audit methodology and scope boundary

The main residual risk in this report is methodological: the review could have looked at the wrong boundary or accepted a proxy signal as proof. To control that risk, the review covered scope selection, evidence selection, tool blind spots, manual reasoning gaps, and report classification.

**Scope boundary.** Backend, SDK, dashboard, deployment, and operations risks surfaced during the review. This report scores only in-scope contract and circuit findings and does not over-score adjacent surfaces; the out-of-scope risks that surfaced fall to the owning backend/product, deployment, and operations workstreams rather than being scored here. The strict scope is defined by `docs/internal/audits/README.md` at `7617b3eebcf37ab42124fe570eb7e065cf8c8461`.

**File coverage.** An in-scope source file or helper could be omitted because it is small, vendored, generated, or only reached through scripts. The review enumerated the in-scope Solidity, Circom, ZK script, and ZK test files directly from the repository tree. The coverage set includes the contract sources, ZK circuits, ZK scripts, and ZK tests listed in Section 3; deployment, backend, SDK, dashboard, and external formal-boundary work remain explicitly outside the scored set. The modeled Verity contract-surface proof is summarized in Section 5.

**Proxy evidence.** Passing tests could be mistaken for proof of all properties, especially around cryptographic soundness or ceremony correctness. The report distinguishes executable evidence from assumptions. Foundry and ZK tests are used only for the contract/circuit boundary they exercise; Groth16 soundness, Poseidon collision resistance, production setup ceremony integrity, backend key custody, and formal proof obligations outside the modeled Verity contract surface remain stated assumptions or deferred work.

**Circuit tooling blind spots.** The manual circuit review could miss unconstrained-signal patterns, aliasing, or range-conversion assumptions. The review ran `circomspect` on the main spend circuit, its generic spend template, and the Merkle-proof helper. It produced three warnings: two `<--` bit-decomposition assignments and one `Num2Bits(levels)` Merkle-index conversion. These are non-issues: the assigned bits are immediately boolean-constrained and recomposed, and `levels` is the 32-bit Merkle depth, far below the BN254 aliasing boundary.

**Solidity low-level paths.** Assembly blocks, unchecked increments, verifier precompile wrappers, and tree insertion code could hide behavior not covered by high-level reasoning. This review searched in-scope Solidity and ZK code for low-level constructs and review markers, then tied the results back to targeted tests. The relevant low-level surfaces are verifier precompile probes, ERC-7201 storage access, bounded loops, and vendored hash/tree libraries; no new untriaged `delegatecall`, `selfdestruct`, `tx.origin`, or arbitrary low-level call path was found in the in-scope application contracts.

**Classification drift.** Without an explicit classification rule, readers could draw opposite conclusions depending on the boundary they assume. The report states the rule explicitly: four scored Medium ZK-script/CI findings and seven scored Low ZK-script/CI findings sit in the audited scope, while production risk outside that scope is excluded from this report and handed off to the owning workstreams.

Beyond the scored findings MED-01, MED-02, MED-03, MED-04, LOW-01, LOW-02, LOW-03, LOW-04, LOW-05, LOW-06, and LOW-07, the remaining unclosed surface is intentionally outside this report's claim: cryptographic proof, formal circuit R1CS proof beyond the stated assumptions, backend/product security, dashboard security, governance/deployment hardening, and production ceremony execution. Within the

strict contract/circuit boundary, these methodology checks did not change the verdict or produce a High or Critical scored finding.

### Investigated non-finding paths

- **Public-signal aliasing.** The contract hashes `merkleRoot`, `contextHash`, the nullifier array, and the output commitment array with the shape supplied by `VerifierRouter`; the circuit decomposes every slot with `Num2Bits_strict`. This closes the practical path where a prover supplies an alternate field-element encoding that hashes to a different Solidity byte string for the same circuit value.
- **All-dummy spend attempts.** Sentinel-zero input and output slots are inert: dummy inputs force zero value, real inputs force nonzero value and Merkle membership, dummy outputs force zero value, and real outputs force nonzero value plus a Poseidon commitment match. An all-dummy proof can at most produce a no-op transfer; it cannot satisfy a nonzero withdrawal because the withdrawal note is validated and pinned to the final output slot.
- **Withdrawal recipient substitution at the contract boundary.** For `withdraw` and `emergencyWithdraw`, the final output slot must equal `hashNote(withdrawal)`, and settlement pays `address(uint160(withdrawal.npk))` for the same token and amount. The contract therefore enforces the proof-bound recipient/asset tuple; any blind signing or recipient-reconstruction weakness remains in the SDK/backend preparation flow and is outside the audited scope.
- **Merkle-root and nullifier bypass.** The pool accepts only roots written by initialization or by deterministic leaf insertion, and every nonzero nullifier is range-checked and written once before output insertion or token settlement. Reusing a spent nullifier, referencing an unseen root, or supplying an out-of-field nullifier reverts before value can leave the pool.
- **Ciphertext tampering and count skew.** The proof context includes a contract-computed hash of the submitted ciphertexts, chain id, and pool address. The pool also requires exactly one ciphertext per real shielded output, excluding the public withdrawal slot. Reordering or replacing ciphertexts, replaying across pools/chains, or adding ciphertexts for dummy outputs invalidates the context or count checks.
- **Verifier export and artifact hardening drift.** The ZK verifier export path writes the snarkjs verifier through `protocol/zk/scripts/harden_verifier.py`, which aborts if its template anchors no longer match. The committed `spend_10x4_v1_verifier.sol` contains the expected base-field proof-coordinate checks and BN254 precompile returndata-size checks, so the malformed-proof acceptance class is absent in the audited snapshot.
- **Transfer-output field bounds.** Deposits validate each note before insertion, while transfer and withdrawal outputs are accepted only after a proof whose nonzero output commitments are Poseidon field elements. The LazyIMT insertion path still rejects any leaf outside the BN254 scalar field, so the on-chain tree cannot be polluted with an out-of-field commitment even if future verifier wiring regressed before insertion.
- **Calldata-offset aliasing across struct arrays.** Distinct from the field-element aliasing already locked down by `AliasedCalldata.t.sol`, an independent pass also considered whether a caller-crafted `Transaction` could place `nullifierHashes` and `newCommitments` at overlapping calldata offsets. `abi.encodePacked` inside `_buildPublicSignals` serializes whichever bytes the per-array calldata pointer resolves to; for SHA-256 to match the circuit digest, the witness must still expose two structurally sepa-

rate arrays whose per-slot Poseidon images (different arities for the nullifier and commitment derivations) coincide on those overlapping bytes. The shape gate (`length == c.inputCount / outputCount`) plus per-slot `Num2Bits_strict` keeps both arrays canonical, and cross-domain Poseidon collision requires breaking the underlying hash. No exploitable state-mutation handoff was identified here.

- **Permit2 spender binding double-defense.** The deposit witness carries an explicit `address pool` alongside the notes-and-ciphertexts hash, while Permit2's own typed-data simultaneously pins `spender == msg.sender == address(this)`. The two bindings agree at the call site because the pool invokes Permit2 directly (no delegatecall splice), so even a future Permit2 deployment with weaker implicit-spender enforcement would not let a depositor signature for pool A clear at pool B.
- **Verifier router cross-shape proof routing.** A second pass exercised `VerifierRouter.setCircuit` against the registered shape and reverse-index invariants. After the first registration, `(inputCount, outputCount)` is immutable, and a verifier address cannot be claimed by a second `circuit_id`. Combined with the per-shape Groth16 verifying-key divergence, the same proof bytes cannot be re-routed across shapes by owner re-registration mistakes.
- **Token uniformity inside a deposit batch.** `_validateAndCollectDeposit` requires every note in a single `deposit` call to set `note.token == _permit.permitted.token`; the Permit2 typed-data already binds that token to the depositor's signed permission. A relayer therefore cannot mix tokens inside one batch to lift value from one ERC-20 ledger into a shielded note attributed to another.
- **UUPS upgrade and pending Permit2 signatures.** Pending depositor signatures commit to `address(this)` both in the explicit witness field and via Permit2's spender check. UUPS upgrades preserve the proxy address, so a queued depositor signature stays valid across upgrades without widening the set of accepted signatures; storage-layout collision hardening remains an out-of-scope governance concern.

## Additional candidates reviewed

A further set of candidate issues was reviewed against the audited scope. Backend, SDK, dashboard, prover, relayer, and test-only items among them are not scored in this report because Section 3 excludes those components from scope. Where those items overlap credible production concerns, they fall to the owning backend/product workstream rather than being scored here.

The in-scope contract candidates also did not become findings. `emergencyWithdraw` is intentionally permissionless and still requires a valid proof plus an unspent nullifier. The current LazyIMT depth is fixed at 32, below the vendored `uint40` capacity; increasing tree depth or changing the note-hash arity would be a future migration/design review item, not a vulnerability in the locked snapshot. These candidates therefore remain non-findings rather than additional scored issues.

## Remediation status (Round-1 closeout)

Unlink submitted a Round-1 remediation closeout (`docs/internal/audits/round-1/verity-closeout-2026-07-10.md` at `ccbc6dec8cbc75385b48a6bed633559322293090`) mapping each finding to GitHub-accessible evidence. We reviewed each evidence item against the merged monorepo code at its referenced pull request, read against the original finding above, and re-confirmed the fix content is present on the current default branch. This is a code-evidence review of the submitted remediation, not a second full audit; the machine-checked formal model was

not affected by any change (the in-scope remediations land in ZK-script, artifact-pipeline, and CI surfaces outside the modeled contract/circuit boundary).

The findings below are **Resolved**: the merged code fully closes the finding as described, and the residual, where any, is by design or outside the finding's stated scope.

| Finding | Evidence | Verified remediation |
|---|---|---|
| MED-01 | PR #892 | Remote artifact destinations are canonicalized and rejected before any write. |
| MED-02 | PR #892 | Artifact version is derived from the required artifact bytes; the upload path blocks unsafe overwrite. |
| MED-03 | PR #983 | `check-zk-artifacts.sh` now enforces strict source-git provenance by default ( `artifact_git_precondition` runs unconditionally; `-strict-source` is a no-op kept for compatibility), and the bundle is republished as immutable version `v8dab16487468ac75` anchored to a permanent `main` SHA. |
| MED-04 | PR #892 | Warm boot re-hashes local artifacts rather than trusting the sentinel alone. |
| LOW-01 | PR #947 | Verifier hardener fails closed on partial marker/body patching. |
| LOW-02 | PR #892 | The ZK test target runs artifact preflight, matching its docstring. |
| LOW-03 | PR #892 | Publisher/consumer schema requires `witness_calculator.js` where needed. |
| LOW-04 | PR #892 | ZK workflow path filters now cover scripts, dependencies, and lockfile changes. |
| LOW-05 | PR #949 | Circomspect warning drift fails closed against a reviewed baseline; malformed SARIF is rejected. |
| LOW-06 | PR #892 | Shared test-vector changes now run the Solidity parity job. |
| LOW-07 | PR #947 | Non-zero RapidSNARK exit fails closed; a skip now requires explicit no-AVX evidence. |

All eleven scored findings are Resolved; no scored finding remains open. The out-of-scope surfaces this report does not score are handed off to the owning backend/product, deployment, and operations workstreams.

# 7 Remediation priorities

> **Status of these recommendations**
>
> The recommendations in this chapter are stated as originally issued, in the imperative. Unlink has since remediated the findings under the Round-1 closeout, and we verified each fix against the merged code; per-finding status is recorded in Section 6. As of that verification every scored finding is Resolved. The prescriptive text below is retained as the record of the recommended remediation.

MED-01 should be remediated before relying on remote ZK artifact bootstrap in CI or production. Reject remote manifest circuit names that are not in the local circuit allowlist, enforce basename-style identifiers, and canonicalize every download destination before writing it.

MED-02 should be fixed before using `ARTIFACT_VERSION` as a production release pin. Derive the version from the complete uploaded artifact bundle, including zkey, verifier key, witness artifacts, and ceremony/transcript metadata, then prevent overwrites of existing version prefixes except through an audited break-glass process.

MED-03 should be remediated before treating `check-zk-artifacts.sh` as a reliable source/artifact drift signal. Add an early bundle-identity precondition that compares `manifest.gitSha` against `git rev-parse --short HEAD` and fails closed with an explicit identity error before any checksum comparison; require explicit non-`latest` artifact versions tied to the release SHA in `fetch-zk-artifacts` and the nightly proof workflow; and fix the upload script's `file_size()` to dereference symlinks. If byte-for-byte WASM reproducibility is not feasible across platforms, replace the WASM byte-check with a deterministic semantic attestation (R1CS, vkey, signed build attestation) rather than leaving a gate whose mismatches depend on unbound bundle identity.

MED-04 (Low: the warm-boot sentinel is a documented liveness optimization, not an integrity gate) should be remediated as a recovery/observability hardening item: pair the per-version sentinel with a local manifest of expected filenames, sizes, and hashes recorded at the end of cold bootstrap, and have `--skip-if-complete` re-hash that manifest before declaring success. Clear the sentinel at the start of any explicit refresh and lock the cold-bootstrap path to prevent concurrent same-version writes from interleaving with the sentinel write.

LOW-01 should be fixed in the verifier export tooling before the next verifier regeneration. Make the hardener validate the complete expected patch set and fail on partially hardened files instead of returning early on one marker.

LOW-02 should be fixed by aligning `just test-zk` with the docstring in `protocol/zk/tests/spend.proof.test.ts`, either by invoking `preflight-test-zk.sh` from the `test-zk` recipe (verifying local artifact hashes before staging and routing fresh-setup behavior under a distinct target) or by updating the docstring to match the current local-fallback behavior and exposing staged-artifact testing through an explicit `test-zk-staged` command. The nightly proof workflow's preflight should remain regardless.

LOW-03 should be fixed in the artifact publisher before the next bundle upload. Require `witness_calculator.js` at upload time if downstream ZK test gates require it, or publish an explicit manifest schema that marks it optional and make every downloader and checker consume that same schema.

LOW-04 should be fixed in CI by expanding the ZK workflow path filters to include `protocol/zk/scripts/**`, `protocol/zk/tests/**`, `protocol/zk/circomkit.json`, native witness-test inputs under `protocol/zk/patches/**` and `protocol/zk/Dockerfile.witness-test`, `pnpm-lock.yaml`, dependency patches under `patches/**`, and artifact/test fixtures, then

adding a cheap PR-time script smoke gate for artifact-script-only, dependency-only, or build-config-only changes.

LOW-05's primary fix is a Circomspect warning-drift gate in CI: fail the ZK workflow unless the new warning is explicitly added to a reviewed baseline. An independent hardening is to upload the SARIF via `github/codeql-action/upload-sarif` so warnings persist as GitHub Security-tab code-scanning alerts; this addresses warning persistence rather than the drift gate.

LOW-06 should be fixed in CI by making shared vector changes run the Solidity parity side of `just check-constants`, either through the Contracts workflow path filters or a dedicated cheap PR job.

LOW-07 should be fixed in the witness gate by making `rapidsnark` proof-generation failures fail by default, with any local architecture skip behind an explicit opt-in flag and concrete capability check.

With the scored findings remediated and verified (Section 6), the next in-scope work item is to keep the formal-audit boundary in Section 5 in sync with any contract, verifier-route, token-listing, or artifact-pipeline change. Any change that affects entrypoint structure, storage writes, proof ordering, token movement, Permit2 wiring, or circuit public-signal binding should update the Verity model and rerun the focused Lean build gate before being treated as covered by this report.

Surfaces outside the audited contract/circuit boundary, backend, SDK, dashboard, deployment, governance, and operations, were not scored in this report. Before any value-bearing launch, they should be audited under a separate backend/product track rather than treated as closed by this contract/circuit report.

# 8 Recommendations

For the audited scope, keep the evidence set focused on the contract and circuit boundary: forge tests for `protocol/contracts/src/**`, ZK tests for `spend_10x4_v1`, verifier-export reproducibility, and parity vectors that support the canonical `(10,4,32)` spend shape.

For formal-audit maintenance, treat the Verity/Lean build as a release gate for contract-surface changes. When the pool changes, update the Verity model, rerun the Lean build, restate the affected AP/IC predicate over the modeled entrypoints/state helpers, and update the explicit assumption/tracked-boundary table when a claim depends on crypto, circuit, token, EVM, liveness, privacy, or deployment facts outside the model.

For production readiness, open a separate backend/product audit track for the out-of-scope surfaces (backend, SDK, dashboard, deployment, and operations) that this report did not score. That track should include SDK refusal tests for hostile prepare responses, storage tests that reject plaintext secret-bearing rows, worker cleanup tests for failure paths, dashboard role tests for tenant mutations, concurrent API tests for quota/prover fairness, dependency scanning, and the acknowledged ZK ceremony release gate.

# A Issue index

Finding identifiers are stable labels and do not encode severity; read severity from the severity cell where an ID prefix and the severity column differ.

| Severity | ID | Report status | Title |
|---|---|---|---|
| Medium | MED-01 | Resolved | Remote artifact manifest names can escape the artifact directory |
| Medium | MED-02 | Resolved | Artifact version is a source-derived prefix, not a durable bundle identity |
| Medium | MED-03 | Resolved | Drift check compares current source to fetched bundle without binding bundle source identity |
| Low | MED-04 | Resolved | Warm-boot sentinel does not detect post-bootstrap volume drift |
| Low | LOW-01 | Resolved | Verifier hardener can treat partially patched output as already hardened |
| Low | LOW-02 | Resolved | just test-zk does not preflight artifacts contrary to its own docstring |
| Low | LOW-03 | Resolved | Artifact publisher can omit the witness calculator required by ZK test gates |
| Medium | LOW-04 | Resolved | ZK script and dependency changes do not trigger the ZK PR workflow |
| Low | LOW-05 | Resolved | Circomspect warning drift does not fail the ZK PR workflow |
| Low | LOW-06 | Resolved | Shared vector changes do not trigger Solidity parity tests |
| Low | LOW-07 | Resolved | Witness test misclassifies rapidsnark proof-generation failures as a skip |

# B Verity source crosswalk

This appendix maps the modeled Solidity surface to the hand-written Verity model and Lean evidence files used by the formal audit. It is a crosswalk, not a second source-code listing: the complete source remains in `UnlinkPool.sol`, `State.sol`, `Contract.lean`, `State.lean`, and `FormalAudit.lean`. The table uses AP/IC identifiers for readability; exact theorem names are in `FormalAudit.lean`.

The static mapping in this table is corroborated dynamically by the dual-implementation differential recorded in Section E: the curated conformance suite is executed against both the hand-written Solidity and the Verity-model bytecode and required to produce identical outcomes, which it did (zero failures on both sides).

| Solidity surface | Verity model surface | Formal evidence | Boundary kept explicit |
|---|---|---|---|
| `deposit` entrypoint | Verity entrypoint metadata, relayer guard, note validation calls, token matching, deposit witness construction, transfer-with-balance-check call, and leaf insertion call. | `AP-S1`, `AP-S5`, and `IC-TB1`: deposit validation, balance-mismatch guard presence, and Permit2 call-path evidence. | ERC-20 and Permit2 semantics; client-side note construction. |
| `transfer` entrypoint | Verity loop shape: circuit lookup, route registration/active checks, input and output shape checks, context validation, proof verification, then nullifier spend and leaf insertion. | `AP-S2`, `AP-S6`, and `IC-S5`: verify-before-mutate ordering and zero-slot filtering. | Groth16 soundness; circuit relation correctness; EVM atomicity. |
| `withdraw` entrypoint | Verity relayer guard and call into the shared withdrawal execution path. | `IC-S6` and `AP-S2`: relayer policy and shared verify-before-mutate ordering. | Token conformance; EVM revert atomicity for mutation-before-settlement paths; successful honest witness construction. |

| Solidity surface | Verity model surface | Formal evidence | Boundary kept explicit |
|---|---|---|---|
| `emergencyWithdraw` entrypoint | Verity public entrypoint without the relayer guard, sharing the same withdrawal execution path. | `AP-L1` and `AP-S2`: permissionless emergency entry and shared verify-before-mutate ordering. | Route availability, proving artifacts, gas, chain inclusion, and user data availability. |
| `_executeWithdrawal` internal path | Verity proof-before-mutation order, nonzero withdrawal slot check, withdrawal commitment binding, zero-slot filtering, and balance-delta guard placement. | `AP-S7`, `AP-S6`, `AP-S5`, and `IC-S4`: withdrawal slot binding, guarded zero-slot path, balance-mismatch guard surface, and filtered insertion shape. | Circuit authority, range, balance, and bookkeeping constraints. |
| `_spendNullifiers` internal path | Verity loop skips zero sentinel slots and calls the spend helper only for real nullifiers. | `AP-S3`, `AP-S6`, and `IC-S1`: real nullifier marking and zero-slot filtering. | Circuit freshness and public-input semantics. |
| `State._spend` helper | Verity state helper rejects out-of-field nullifiers, rejects already spent nullifiers, and marks the fresh nullifier slot. | `AP-S3` and `IC-S1`: concrete fresh in-field nullifier marking at the target slot. | Only the helper behavior is proved here; accepted-spend meaning depends on the circuit and proof assumptions. |
| `State._insertLeaves` helper | Verity state helper returns the current root on an empty insertion; for successful non-empty insertion, it stores the returned root and marks that root as seen. | `AP-S4`, `AP-L2`, and `IC-S2`: concrete insertion root update and returned-root `rootSeen` marking. | This does not claim full historical-root preservation across arbitrary abstract transitions; witness reconstruction remains a circuit/data-availability boundary. |

| Solidity surface | Verity model surface | Formal evidence | Boundary kept explicit |
|---|---|---|---|
| Admin ownership paths | Verity owner guard metadata, disabled renounce path, UUPS authorization hook, and admin write-set summaries. | `AP-T1` and `AP-T2a`: admin write-set exclusion and owner-gated upgrade authorization. | Nonzero deployment owner, future implementation review, and storage-layout compatibility. |
| Verifier router configuration | Verity owner-gated setter and route-use checks in spend paths. | `AP-T1`, `AP-S2`, and `IC-S5`: router write-set and verify-before-mutate surfaces. | R2 artifact manifest pinning, production ceremony/transcript review, live-router interface conformance, route governance, and circuit/verifier identity. The Verity router model preserves the Solidity `bytes32` circuit IDs, `uint16` shape counts, `bool` active flag, typed router events, and named `Circuit` return metadata used by the generated compile checks. |

| Solidity surface | Verity model surface | Formal evidence | Boundary kept explicit |
|---|---|---|---|
| Constants and manifests | `Contract.lean` scalar-field constant and generated artifact wrapper resolution. | `IC-X1`: modeled `SNARK_SCALAR_FIELD` equals the BN254 scalar-field order. | Artifact publication and deployment pinning beyond this artifact snapshot. |

# C Formal AP/IC manifest

This appendix is the exhaustive AP/IC manifest used by the formal audit. It gives a non-Lean reading of every item: green means proved over the Verity/Lean model or concrete state helpers, yellow means an external trust assumption required by the guarantee, and blue means a tracked boundary that is not claimed or relied on by the formal result.

Within green rows, the evidence type is named in the reading when relevant: behavioral state theorem, structural scan, write-set/call-graph check, or literal constant check. These should not be read as interchangeable proof strengths.

The table below is the retallied manifest. The **Lean item** column gives the theorem or predicate family that closes or indexes the row. The **underlying evidence** column names the load-bearing Boolean, state-helper theorem, assumption, or boundary. Rows with the same alias group reuse the same evidence and are not counted as independent load-bearing facts.

| Row | Lean item | Underlying evidence | Evidence type | Alias group | External assumptions |
|---|---|---|---|---|---|
| AP-G1 | ap_g1_fund_safety_surface_generated | transferAndWithdrawalVerifyBeforeSpendConcrete, permitWitnessBindingConcrete, withdrawalBalanceDeltaConcrete, ownershipAndAdminWriteSetsExcludeProtocolStateConcrete | Composite theorem | AG-FUND | Token, circuit, no unsafe upgrade |
| AP-G2 | ap_g2_spend_authority_surface_generated | transferAndWithdrawalVerifyBeforeSpendConcrete | Concrete Boolean | AG-VERIFY | Groth16, circuit relation |
| AP-G3 | ap_g3_nullifier_signal_surface_generated | transferAndWithdrawalVerifyBeforeSpendConcrete and spendNullifiersHelperConcrete | Composite theorem | AG-NULLIFIER | Circuit freshness/public-input semantics |
| AP-G4 | outOfModelItems | Emergency-exit liveness boundary | Tracked boundary | AG-OOM-G4 | Chain/prover/data/gas/route availability |
| AP-G5 | outOfModelItems | EVM transaction atomicity boundary | Tracked boundary | AG-OOM-ATOMIC | EVM execution semantics |
| AP-Ax1 | AP.Ax.groth16_soundness | Groth16 soundness for deployed phase-2 key | External assumption | AG-AX1 | Ceremony/transcript or equivalent vkey evidence |
| AP-Ax2 | AP.Ax.circuit_correctness | spend_10x4_v1 implements R_spend | External assumption | AG-AX2 | Circuit audit/tests |
| AP-Ax3 | AP.Ax.crypto_assumptions | Poseidon collision resistance and EdDSA unforgeability | External assumption | AG-AX3 | Cryptographic hardness |
| AP-Ax4 | AP.Ax.token_conformance | ERC-20 transfer/balance conformance | External assumption | AG-AX4 | Token integration review |
| AP-Ax5 | AP.Ax.key_custody | Spending-key generation/distribution/storage exclusivity | External assumption | AG-AX5 | Client/custody controls |
| AP-Ax6 | AP.Ax.chain_liveness | Bounded inclusion/finality | External assumption | AG-AX6 | Chain liveness/finality |

| Row | Lean item | Underlying evidence | Evidence type | Alias group | External assumptions |
|---|---|---|---|---|---|
| AP-Ax7 | AP.Ax.proving_artifact_availability | Proving artifact availability | External assumption | AG-AX7 | Artifact distribution |
| AP-Ax8 | AP.Ax.self_exit_readiness | Self-exit route/gas/procedure readiness | External assumption | AG-AX8 | Operational readiness |
| AP-Ax9 | AP.Ax.data_availability | Event/ciphertext availability | External assumption | AG-AX9 | Data availability |
| AP-S1 | ap_s1_deposit_validation_and_insertion_generated | depositValidationAndInsertionConcrete | Concrete Boolean | AG-DEPOSIT | ERC-20/Permit2 semantics; note construction |
| AP-S2 | ap_s2_verify_before_mutate_generated | transferAndWithdrawalVerifyBeforeSpendConcrete | Concrete Boolean | AG-VERIFY | Groth16, circuit relation, router conformance |
| AP-S3 | ap_s3_state_spend_concrete_generated | state_spend_success_concrete_nullifier and spendNullifiersHelperConcrete | Behavioral theorem + Boolean | AG-SPEND | Circuit freshness and public-input semantics |
| AP-S4 | ap_s4_concrete_insert_root_history_generated | state_insert_nonempty_records_returned_root and insertLeavesRootWriteConcrete | Behavioral theorem + Boolean | AG-INSERT-ROOT | Full historical-root preservation across arbitrary abstract transitions is not claimed |
| AP-S5 | ap_s5_balance_delta_generated | permitWitnessBindingConcrete and withdrawalBalanceDeltaConcrete | Composite theorem | AG-BALANCE | ERC-20/Permit2 semantics |
| AP-S6 | ap_s6_zero_slot_filtering_generated | spendNullifiersHelperConcrete | Concrete Boolean | AG-ZERO | Circuit dummy-slot semantics |
| AP-S7 | ap_s7_withdrawal_slot_binding_generated | executeWithdrawalSlotBindingConcrete | Concrete Boolean | AG-WITHDRAW-SLOT | Circuit withdrawal binding semantics |
| AP-T1 | ap_t1_admin_write_set_generated | ownershipAndAdminWriteSetsExcludeProtocolStateConcrete | Concrete Boolean | AG-ADMIN | Nonzero owner; no unsafe upgrade |
| AP-T2a | ap_t2a_authorize_upgrade_generated | authorizeUpgradeOwnerGatedNoWriteSetConcrete and ownableOwnerSlotMatchesOZNamespaceConcrete | Concrete Boolean | AG-UPGRADE | Deployment-owner validity |
| AP-T2b | outOfModelItems | Future storage-layout compatibility | Tracked boundary | AG-OOM-UPGRADE | Future implementation review |
| AP-T3 | outOfModelItems | Registered verifier soundness boundary | Tracked boundary | AG-OOM-VERIFIER | Groth16, circuit, route governance |
| AP-L1 | ap_l1_emergency_withdraw_generated | generatedRelayerEntrypointsConcrete | Concrete Boolean | AG-RELAYER | Route/gas/proving availability for usability |
| AP-L2 | ap_l2_concrete_root_seen_generated | returned-rootinsertion theorem | Behavioral theorem + Boolean | AG-INSERT-ROOT | Full root-seen monotonicity across arbitrary abstract transitions is not claimed |
| AP-L3 | outOfModelItems | Active route availability over time | Tracked boundary | AG-OOM-LIVENESS | Governance/operations |

| Row | Lean item | Underlying evidence | Evidence type | Alias group | External assumptions |
|---|---|---|---|---|---|
| AP-L4 | outOfModelItems | Public entrypoint atomicity boundary | Tracked boundary | AG-OOM-ATOMIC | EVM execution semantics |
| IC-T1 | outOfModelItems | Spend completeness boundary | Tracked boundary | AG-OOM-COMPLETE | Honest witness and no-revert execution |
| IC-T2 | ic_t2_accepted_spend_soundness_surface_generated | AP_G2_SpendAuthoritySurfaceGenerated and AP_G3_NullifierSignalSurfaceGenerated | Composite alias | AG-T2 | Groth16, circuit freshness/relation |
| IC-BV1 | ic_bv1_public_signal_surface_generated | transferAndWithdrawalVerifyBeforeSpendConcrete | Definitional alias | AG-VERIFY | ABI/public-signal packing semantics |
| IC-BV2 | ic_bv2_verifier_hardened_surface_generated | transferAndWithdrawalVerifyBeforeSpendConcrete | Definitional alias | AG-VERIFY | Verifier/precompile semantics |
| IC-S1 | ic_s1_state_nullifier_write_once_concrete_generated | AP_S3_StateSpendConcreteGenerated and AP_S6_ZeroSlotFilteringGenerated | Composite alias | AG-S1 | Circuit freshness/dummy-slot semantics |
| IC-S2 | ic_s2_concrete_merkle_append_root_generated | returned-rootinsertion theorem | Behavioral theorem + Boolean | AG-INSERT-ROOT | Full append-only root-history semantics across arbitrary abstract transitions is not claimed |
| IC-S3 | ic_s3_per_token_solvency_surface_generated | AP_G1_FundSafetySurfaceGenerated | Definitional alias | AG-FUND | Token, circuit, no unsafe upgrade |
| IC-S4 | ic_s4_withdrawal_shape_generated | AP_S7_WithdrawalSlotBindingGenerated | Definitional alias | AG-WITHDRAW-SLOT | Circuit withdrawal binding semantics |
| IC-S5 | ic_s5_verify_then_mutate_generated | AP_S2_VerifyBeforeMutateGenerated | Definitional alias | AG-VERIFY | Groth16, circuit relation, router conformance |
| IC-S6 | ic_s6_relayer_policy_generated | generatedRelayerEntrypointsConcrete | Definitional alias | AG-RELAYER | Route/gas/proving availability for usability |
| IC-C1 | outOfModelItems | Circuit authority binding | Tracked boundary | AG-OOM-CIRCUIT | Circuit review |
| IC-C2 | outOfModelItems | Circuit membership/nullifier/output bookkeeping | Tracked boundary | AG-OOM-CIRCUIT | Circuit review |
| IC-C3 | outOfModelItems | Circuit balance/range/no-wrap arithmetic | Tracked boundary | AG-OOM-CIRCUIT | Circuit review |
| IC-TB1 | ic_tb1_permit2_witness_path_generated | permitWitnessBindingConcrete and permit2ArgumentFlowConcrete | Concrete Boolean | AG-PERMIT | Permit2 witness/signature semantics |
| IC-X1 | ic_x1_constant_manifest_generated | snarkScalarFieldMatchesManifestConcrete, maxNoteValueMatchesManifestConcrete, circuitIdMatchesManifestConcrete, and lazyImtZ7MatchesManifestConcrete | Concrete Boolean | AG-CONSTANT | Artifact/deployment pinning beyond snapshot |

| Row | Lean item | Underlying evidence | Evidence type | Alias group | External assumptions |
|---|---|---|---|---|---|
| IC-PR1 | outOfModelItems | Privacy game boundary | Tracked boundary | AG-OOM-PRIVACY | Separate privacy/backend analysis |
| IC-E3 | outOfModelItems | Censorship escape boundary | Tracked boundary | AG-OOM-LIVENESS | Chain inclusion and user readiness |
| IC-E4 | ic_e4_non_custodial_surface_generated | IC_T2_AcceptedSpendSoundnessSurfaceGenerated and ownershipAndAdminWriteSetsExcludeProtocolStateConcrete | Composite alias | AG-E4 | Circuit authority/soundness, token, EVM atomicity |
| IC-E6 | ic_e6_ciphertext_context_surface_generated | IC_BV1_PublicSignalSurfaceGenerated | Definitional alias | AG-VERIFY | Event/ciphertext availability remains AP-Ax9 |

**Row-to-fact reconciliation.** The manifest above marks 27 positive AP/IC rows as proved. These are report-level index labels, not 27 independent guarantees. Counting the underlying Lean evidence, they reduce to **23 distinct load-bearing facts**: 8 behavioral theorems and 15 structural/constant checks, listed below. Row count and fact count differ on purpose, for two concrete reasons: one row can carry several facts (the constants row IC-X1 alone supplies four constant checks), and several rows can share one fact (the three root-history rows AP-S4, AP-L2, and IC-S2 all rest on the single returned-root insertion theorem). Every other proved row is a presentation alias or a conjunction that reuses already-counted facts under a different label; the **Alias group** column marks which rows share evidence. No proved row introduces evidence beyond these 23 facts. Of the 23, the 8 behavioral theorems are enumerated as kernel-checked evidence atoms (proof-grade, axioms propext/Quot.sound, no compiler-trust axiom) and are the conservative *behavioral* headline. The other 15 structural/constant checks are all discharged by plain kernel decide: the pinned model keeps FormalAudit.lean in kernel-reducible form, so every structural scan reduces under the kernel and none rests on ofReduceBool, the compiler-trust axiom that native_decide would introduce. All 23 facts are therefore kernel-clean, off the compiler-trust axiom, which brings the proof-grade count to 23. Of the eight behavioral theorems, four characterize the _spend helper and the returned-root insertion (the success direction, the returned-root bookkeeping, and the no-double-spend and field-bound reverts proved behaviorally), and four characterize the external-token interface over an executable (token, account) ledger (the exact sender/recipient transfer delta, the exact Permit2 deposit delta, the withdrawal-guard equalities by construction, and read-after-write balanceOf coherence) (Section F). The first four are canonical in the package's State.lean and the token-ledger four live in the package's TokenLedgerExec.lean; all eight are enumerated in the gate's concreteEvidenceAtoms manifest. A ninth behavioral atom, the *entrypoint* reentrancy-revert theorem emergency_withdraw_reentrancy_revert_generated (Section F), is counted separately: it is proved over the delivered emergencyWithdraw body rather than over a _spend state helper, so it sits beside the 23 AP/IC facts, not inside them. The manifest therefore holds 27 atoms: the 23 load-bearing AP/IC facts, three delivery-only atoms, and this one entrypoint reentrancy-revert theorem.

The Lean delivery gate machine-checks these row-state totals: 29 AP items, 19 IC items, 27 positive AP/IC rows closed from concrete evidence, 9 AP-Ax assumptions, and 12 tracked boundaries. These reconcile exactly as $29 + 19 = 48 = 27 + 9 + 12$.

The 23 load-bearing facts are: transferAndWithdrawalVerifyBeforeSpendConcrete, permitWitnessBindingConcrete, permit2ArgumentFlowConcrete, withdrawalBalanceDeltaConcre

te, ownershipAndAdminWriteSetsExcludeProtocolStateConcrete, spendNullifiersHelperConcrete, state_spend_success_marks_nullifier, state_insert_nonempty_records_returned_root, state_no_double_spend_reverts, state_out_of_field_reverts, tokenLedgerExecTransferDelta, tokenLedgerExecPermit2DepositDelta, tokenLedgerExecGuardsHoldByConstruction, tokenLedgerExecReadAfterWrite, depositValidationAndInsertionConcrete, executeWithdrawalSlotBindingConcrete, authorizeUpgradeOwnerGatedNoWriteSetConcrete, ownableOwnerSlotMatchesOZNamespaceConcrete, generatedRelayerEntrypointsConcrete, snarkScalarFieldMatchesManifestConcrete, maxNoteValueMatchesManifestConcrete, circuitIdMatchesManifestConcrete, and lazyImtZ7MatchesManifestConcrete. This inventory is not a claim that each fact is a semantically independent protocol guarantee. The delivery theorem also contains delivery-only conjuncts that are not AP/IC rows: GeneratedArtifactSurfaceImported, the four proof-state/list count conjuncts, and the four proof-state ID-list conjuncts. Those delivery conjuncts are validation gates for the artifact, not additional AP/IC property rows.

| Item | Status | Natural-language reading |
|---|---|---|
| AP-G1 | Proved | The modeled pool accounting surface contains the expected fund-safety guards: balance-delta checks, token matching, and admin write-set exclusion. Exact production asset conservation still depends on conforming token behavior, circuit correctness, and no unsafe upgrade. |
| AP-G2 | Proved | The modeled spend-authority surface requires a registered verifier route and proof-success gate before the spend path is allowed to mutate spend state. The cryptographic meaning of the proof is covered by the circuit and Groth16 assumptions. |
| AP-G3 | Proved | The modeled nullifier and public-signal surface is structurally present: nonzero real nullifiers are handled by the modeled spend path, and verifier input binding is checked at the contract surface. Freshness and relation semantics remain circuit assumptions. |
| AP-G4 | Tracked boundary | Emergency exit is a conditional liveness claim. It requires chain inclusion, available proving artifacts, user readiness, gas/route availability, and data availability, which are not facts about the Solidity body alone. |
| AP-G5 | Tracked boundary | Public batch atomicity is inherited from EVM transaction semantics. The audit therefore treats atomicity as an execution-environment boundary rather than a property of the pool body alone. |
| AP-Ax1 | External assumption | Groth16 is knowledge-sound for the verification key pinned in the audited artifact manifest. Manifest pinning identifies the audited R2 bundle, and the artifact gate checks the generated Solidity verifier constants against the zkey-derived verification key; production ceremony/transcript review is tracked outside this Verity model. In plain terms: a proof accepted by the verifier should correspond to a real witness for the intended circuit relation. |

| Item | Status | Natural-language reading |
|---|---|---|
| AP-Ax2 | External assumption | The spend_10x4_v1 circuit and its Merkle constraints implement the prose spend relation R_spend. The contract can check a proof; this assumption says the proof is about the right statement. The ZK npmtest gate passes 43 circuit tests covering valid witnesses, mutation rejection, sentinel-zero padding, vector parity, and proof verification, but this remains an external circuit-review assumption for the formal contract-surface proof. |
| AP-Ax3 | External assumption | Poseidon commitments are collision resistant and EdDSA signatures are unforgeable at the relevant security level. These are cryptographic hardness assumptions, not Solidity facts. |
| AP-Ax4 | External assumption | Supported ERC-20 tokens obey coherent balanceOf semantics, exact sender/recipient transfer deltas, and exact Permit2-funded deposit deltas. Fee-on-transfer, rebasing, fake-success, and other nonstandard tokens are not covered unless a reviewed wrapper or adapter restores those semantics. The Lean contract model does not prove arbitrary external token contracts correct. |
| AP-Ax5 | External assumption | Spending keys are generated, distributed, and stored so only the intended user can spend the corresponding notes. This is a custody and client-side security assumption. |
| AP-Ax6 | External assumption | Valid user transactions can be included and finalized within the expected time window. This covers chain liveness/finality rather than pool code behavior. |
| AP-Ax7 | External assumption | Users can obtain the proving artifacts even if Unlink-hosted storage is unavailable. This is an artifact-distribution and availability assumption. |
| AP-Ax8 | External assumption | Self-exit is operationally ready: users have the needed route, gas, artifacts, and procedure to exit without relying on a privileged relayer path. |
| AP-Ax9 | External assumption | Events and ciphertexts needed for witness reconstruction remain available to users. Without that data, a formally safe contract path may still be unusable. |
| AP-S1 | Proved | The modeled deposit body contains the expected note-field validation, token matching, leaf hashing, and insertion surfaces. The current predicate does not prove a full dominance/order theorem for validation before every later transfer or insertion effect. |
| AP-S2 | Proved | Modeled spend paths require a registered and active circuit route, then proof success, before nullifiers or leaves are mutated. |
| AP-S3 | Proved | The concrete _spend helper and modeled nullifier-spend surface mark fresh real nullifiers; the report does not claim the old nonexistent ConcreteFreshNullifierSpend predicate or preservation of unrelated nullifier slots. |

| Item | Status | Natural-language reading |
|---|---|---|
| AP-S4 | Proved | Successful non-empty concrete _insertLeaves execution stores the returned root as merkleRoot and marks that returned root in rootSeen. The report does not claim that every historical root remains visible across all arbitrary abstract transitions. |
| AP-S5 | Proved | Deposit and withdrawal balance-mismatch guard surfaces are present in the modeled helper paths. The formal-audit gate pins the modeled deposit sequence balanceOf(pool) before Permit2, Permit2 acceptance, and balanceOf(pool) after Permit2 with an exact balAfter-balBefore==totalAmount check. It also pins withdrawal settlement to exact pool decrease and recipient increase checks around safeTransfer. Exact token movement still requires the supported-token and Permit2 assumptions. |
| AP-S6 | Proved | The modeled nullifier spend loop contains a guarded zero-slot filtering path, and the manifest records separate structural surfaces for note, commitment, and withdrawal-loop handling. The scan is structural; it does not by itself prove every guard expression, commitment-loop zero-padding behavior, and write target in a full semantic state theorem. |
| AP-S7 | Proved | Modeled withdrawal execution requires a nonzero withdrawal slot, binds it to the withdrawal commitment hash, and excludes it from reinsertion. |
| AP-T1 | Proved | Modeled admin and ownership write sets do not directly write spend state, leaves, roots, nullifiers, or token movement state. |
| AP-T2a | Proved | The modeled UUPS authorization hook is owner-gated and has no storage writes. The owner slot is cross-checked against the OpenZeppelin v5 ERC-7201 Ownable namespace, and the delivery gate pins the current-snapshot State/relayer ERC-7201 roots, pool inheritance order, router Circuit member shape, and router mapping declarations. The current Foundry storage-layout snapshot is also checked against storage-layouts/audit-7617b3e.json. Broader owner-gate statements over unconstrained presentation-layer steps are outside the positive claim; the proved claim is limited to the concrete modeled UUPS authorization hook and the current storage-layout snapshot gate. |
| AP-T2b | Tracked boundary | Storage-layout compatibility across future implementations must be checked at upgrade time. It cannot be proven from the current pool model alone. |
| AP-T3 | Tracked boundary | Registered verifier soundness is discharged by the Groth16, circuit-correctness, and route-governance assumptions rather than by Solidity control-flow proof. |
| AP-L1 | Proved | Modeled emergencyWithdraw has no relayer authorization guard, so the modeled emergency path is not structurally dependent on a privileged relayer. |

| Item | Status | Natural-language reading |
|---|---|---|
| AP-L2 | Proved | Successful non-empty concrete _insertLeaves execution marks the returned root in rootSeen. This is the liveness-relevant root-availability surface proved by the model; full root-seen monotonicity for every old root across arbitrary abstract transitions is not claimed. |
| AP-L3 | Tracked boundary | Active route availability is a tracked non-claim here. The operational self-exit assumption separately requires route readiness where exit usability depends on it. |
| AP-L4 | Tracked boundary | Public entrypoint atomicity is an EVM transaction-semantics boundary. It is not proven from the Verity source itself; withdrawal non-custody/solvency interpretation relies on this premise when token settlement reverts after earlier state mutation. |
| IC-T1 | Tracked boundary | Spend completeness requires honest witness construction and no-revert execution. The Verity model proves guard structure, not that every valid off-chain intent will always produce a successful transaction. |
| IC-T2 | Proved | The modeled accepted-spend soundness surface is structurally present. The meaning of the accepted proof still depends on R_spend, freshness, and cryptographic soundness assumptions. |
| IC-BV1 | Proved | The modeled proof-before-mutate verifier surface is structurally present. Exact public-signal packing and ABI semantics remain external boundaries. |
| IC-BV2 | Proved | The modeled proof-before-mutate verifier surface is structurally present. Exact verifier input packing, truncation behavior, and precompile semantics remain external boundaries. |
| IC-S1 | Proved | Nullifier write-once behavior has concrete storage evidence and zero-skip evidence for the modeled helper and spend surface. |
| IC-S2 | Proved | Successful non-empty concrete _insertLeaves execution updates merkleRoot to the returned root and records that root in rootSeen. This proves the concrete insertion root-update surface; it does not prove a global append-only historical-root theorem over arbitrary abstract transitions. Correctness of the returned root itself is delegated to the vendored zk-kit incremental Merkle tree and the circuit Merkle-membership relation (IC-C2), not recomputed here. |
| IC-S3 | Proved | The modeled per-token solvency surface is structurally present. Exact solvency inherits token, circuit, and no-unsafe-upgrade assumptions. |
| IC-S4 | Proved | Withdrawal slot checks and filtered insertion shape are structurally present in the modeled withdrawal path. |
| IC-S5 | Proved | Verify-then-mutate ordering is structurally present for the modeled spend/withdrawal surfaces. |

| Item | Status | Natural-language reading |
|---|---|---|
| IC-S6 | Proved | Relayer authorization error surfaces are present on `deposit`, `transfer`, and `withdraw`, and absent on `emergencyWithdraw`. A full dominance theorem before every later mutation/external call is outside this proof. |
| IC-C1 | Tracked boundary | The report names the circuit-authority obligation, but the contract proof does not model the internal R1CS authority relation. It checks that proof verification is called and ordered; authority binding itself belongs to circuit proof/review. |
| IC-C2 | Tracked boundary | Circuit bookkeeping for membership, nullifiers, outputs, and dummy slots is a circuit-level property and must be covered by circuit review and tests. |
| IC-C3 | Tracked boundary | Circuit balance, range, and no-wrap facts are circuit-level arithmetic claims, not Solidity control-flow claims. |
| IC-TB1 | Proved | The modeled deposit path contains the Permit2 external-call module and calls `transferWithBalanceCheck`. The concrete scan checks pool-side flow for the Permit2 token, permitted amount, nonce, deadline, spender `address(this)`, transfer amount, owner/depositor, witness, and signature slice. Permit2's cryptographic signature, nonce consumption, deadline, and witness acceptance semantics remain external Permit2 assumptions. |
| IC-X1 | Proved | The modeled `SNARK_SCALAR_FIELD` constant equals the BN254 scalar-field order. This is not a parser-level proof that an external compiled manifest was read and bound into Lean. |
| IC-PR1 | Tracked boundary | Privacy is a separate protocol game involving timing, relayers, metadata, backend behavior, and observer capabilities. It is not claimed by this contract-surface fund-safety proof. |
| IC-E3 | Tracked boundary | Censorship escape depends on chain inclusion and user operational readiness. The modeled emergency and relayer surfaces are proved separately. |
| IC-E4 | Proved | The modeled non-custodial surface is structurally present. The full non-custody guarantee inherits circuit authority, soundness, token conformance, and EVM revert-atomicity assumptions. |
| IC-E6 | Proved | The modeled context-hash/public-signal binding surface is structurally present. Event emission shape is source-crosswalk evidence, while off-chain ciphertext/event availability remains `AP-Ax9`. |

# D Opaque names and formal boundaries

The final Lean evidence files contain `opaque` predicates. In Lean, an opaque proposition used as a hypothesis is axiom-like: Lean may reason from it without unfolding a proof inside this development. The report uses the audit term **external trust assumption** for the subset of those axiom-like premises that the security claim actually depends on and that must be justified by external evidence.

Not every opaque or axiom-like name is an external trust assumption:

- **Presentation predicates**, such as abstract `Sigma/Step` notation and report-level notions like `TokenDeltaExact`, are vocabulary for explaining the intended property. They do not discharge final concrete evidence.
- **Boundary assumptions**, such as Groth16 soundness, circuit relation correctness, ERC-20 conformance, chain liveness, and event/ciphertext availability, are explicit inputs to the security claim when the guarantee depends on them.
- **Out-of-model names**, such as privacy-game or circuit-internal obligations, are index entries for separate work. They are not proved here and are not used as evidence for the proved contract-surface AP/IC items.

Thus the formal-audit result is conditional: the modeled Verity contract surface satisfies the stated structural obligations under the explicit external assumptions above. It is not a standalone proof of the ZK relation, cryptographic primitives, token contracts, EVM semantics, operational liveness, privacy, or future upgrade compatibility.

The following index gives a natural-language reading of the axiom-like or opaque names that appear in the formal Lean files. The key discipline is that only the explicitly named boundary assumptions are accepted as external inputs to the security claim; presentation predicates are not used to close final Verity-evidence obligations.

| Name or family | Audit role | Plain-language meaning |
|---|---|---|
| `Sigma`, `Step`, `SpendStep`, `WithdrawStep`, `DepositStep` | Presentation notation | Compact report vocabulary for pool state and successful transitions. Final delivery evidence does not rely on arbitrary values of these abstractions being valid executions in the Verity model. |
| `Cm` | Presentation notation | The commitment-hash function used to discuss notes and commitments. Its cryptographic properties are covered by the Poseidon/circuit assumptions, not by Solidity proof. |
| `NonUpgradeWindow` | Boundary condition | The period being reasoned about has no relevant owner upgrade, router replacement, or verifier-route replacement. Future upgrades require their own review. |
| `ProtocolDeposits`, `ProtocolWithdrawals`, `TokenDeltaExact` | Presentation notation | Report-level accounting vocabulary for deposits, withdrawals, and exact token deltas. The final proved evidence is the modeled guard and write-set surface, plus explicit token/circuit assumptions. |

| Name or family | Audit role | Plain-language meaning |
|---|---|---|
| `RouteFor`, `Verify`, `PublicSignalHash`, `WithdrawalPublicSignalHash`, `RSpend` | Boundary notation | Vocabulary for registered verifier routes, proof verification, public signals, and the intended spend relation. The Verity model proves route/proof ordering; cryptographic and circuit meaning remain explicit assumptions. |
| `LinkedIntoPostLeaves`, `RootProducedByInsert` | Presentation notation | Vocabulary for saying commitments or roots appear after insertion. Concrete empty-insert preservation is proved; full non-empty Merkle semantics remain tied to additional generated state instrumentation and the circuit/data boundary. |
| `AP.Ax.groth16_soundness` | External assumption | Accepted Groth16 proofs are sound for the verification key pinned in the round-1 artifact manifest. The R2 artifact bundle is manifest-pinned for this round, but production ceremony/trusted-setup transcript review is external to the Verity model and must be evidenced separately before deployment. |
| `AP.Ax.spend_circuit_matches_relation` | External assumption | The deployed spend circuit really enforces the intended spend relation, including Merkle and public-signal constraints. The ZK `npmtest` gate is an executable regression check for this relation, but it is not a formal completeness proof. |
| `AP.Ax.poseidon_collision_resistant_and_eddsa_unforgeable` | External assumption | The hash and signature primitives provide their expected security properties. |
| `AP.Ax.conforming_token` | External assumption | Supported ERC-20 tokens behave according to the expected transfer and balance semantics: coherent `balanceOf`, exact pool increase on Permit2 deposit, exact pool decrease on withdrawal, and exact recipient increase on withdrawal. Fee-on-transfer, rebasing, fake-success, and other nonstandard tokens need policy exclusion or adapter review before they are supported. |
| `AP.Ax.spending_key_user_exclusive` | External assumption | Only the rightful user controls the spending key needed to authorize note spends. This is a wallet/client/custody condition. |
| `AP.Ax.chain_inclusion_liveness` | External assumption | The chain eventually includes and finalizes valid transactions under the assumed network and gas conditions. |
| `AP.Ax.proving_artifact_availability` | External assumption | Users can access proving artifacts independently enough to exercise exit paths when needed. |
| `AP.Ax.emergency_exit_operational_readiness` | External assumption | Emergency exit is operationally usable: routes, gas, artifacts, and user instructions are available. |

| Name or family | Audit role | Plain-language meaning |
|---|---|---|
| `AP.Ax.commitment_event_data_availability` | External assumption | Users can recover the event/ciphertext data needed to reconstruct witnesses. |
| `abiEncodeTwoStaticArraysModule`, `permitWitnessTransferFromModule`, `getCircuitTryModule`, `verifySpendTryModule` | External-call model assumption | ABI encoding, keccak memory-slice behavior, Permit2 witness-transfer ABI, verifier-router ABI, verifier proof ABI, and SHA public-signal packing refine the production compiler/runtime behavior. The pool-side Permit2 argument selection is checked separately by `permit2ArgumentFlowConcrete`. |
| `bn256AddModule`, `bn256ScalarMulModule`, `bn256PairingModule` | External-call model assumption | The memory layout, free-memory-pointer restoration, success flag, and return value behavior of the BN254 precompile probes match the EVM behavior assumed by the Verity compiler modules. |
| `transient_reentrancy_guard` | External assumption | The transient-storage mutex used by OpenZeppelin `ReentrancyGuardTransient` prevents re-entry across the protected state-changing entrypoints as assumed by the Verity model. |
| `VerifierRouter.getCircuit` deployment conformance | External assumption | The configured router address is a live contract implementing the expected interface and returning the intended active verifier for each circuit ID. |
| `__Ownable_init(_owner)` deployment precondition | External assumption | The deployment owner is a valid nonzero owner, either by OpenZeppelin initializer behavior or by deployment policy. |
| `IC.C.authority_binding` | Tracked circuit boundary | The circuit binds spend authority to the right secret material and public signals. This belongs to circuit proof/review. |
| `IC.C.circuit_bookkeeping` | Tracked circuit boundary | The circuit correctly handles membership, nullifiers, outputs, and dummy slots. |
| `IC.C.circuit_balance_and_range` | Tracked circuit boundary | The circuit enforces balance, range, and no-wrap arithmetic constraints. |
| `AtomicBatchStep` | Execution boundary | Atomicity of a public entrypoint is inherited from EVM transaction semantics, not from a contract metadata theorem. |
| `StorageLayoutCompatible` | Upgrade boundary | A future implementation preserves the expected storage layout. This is checked by deployment/upgrade review, not by the current artifact alone. |
| `HonestWitnessForSpend`, `NoRevert` | Completeness boundary | Honest users can construct witnesses and the resulting transaction does not revert. This combines off-chain witness construction, chain conditions, and runtime behavior beyond the structural proof. |

| Name or family | Audit role | Plain-language meaning |
|---|---|---|
| `AcceptedSpend`, `PublicSignalParity`, `VerifierInputHardened`, `SubmissionPolicy`, `Permit2WitnessBindsSpenderAndPool` | Report predicates | Names for intended AP/IC properties. Their final concrete evidence comes from modeled verifier, relayer, and Permit2-witness surfaces plus explicit external assumptions where needed. |
| `PrivacyGameHolds` | Tracked privacy boundary | The end-to-end privacy game is not part of this contract-surface proof. It needs a separate privacy analysis across protocol, backend, relayer, timing, and metadata behavior. |
| `AdminWriteSetSafe` | Presentation predicate | Report vocabulary for admin write-set safety. The delivered evidence is the Verity write-set theorem showing admin paths do not directly write protocol accounting state. |

# E Formal verification gate

The reproducible model source for this delivery is the LFG Unlink monorepo checkout carrying the delivered Verity model, aligned to the audited `7617b3e` behavior (the two production-only balance guards removed); the full delivery SHA is recorded in `FORMAL_ARTIFACT_MANIFEST.md` and the package `MANIFEST.json`. The model lives under `protocol/contracts/verity/UnlinkXyz/Pool`. The exporter/build harness uses `lfglabs-dev/verity` pinned to `4101073`. The model gate used for the formal-audit delivery is:

```
cd protocol/contracts
npm run check:verity-model
```

Lean version: `Lean 4.22.0, commit ba2cbbf09d4978f416e0ebd1fceeebc2c4138c05`.

The artifact hashes recorded for the final report build are:

**`Contract.lean`** `77c0d0fd32c93abb59afe865fb1eaf6d2da2f8a5035bb31d31c1293160fe4f56`
**`State.lean`** `be94395c9b8dac19042217086f586ab2df040b8e2bb47217a1f141fa9819ef6e`
**`Compile.lean`** `b0d194bfb12795e982e8bdbf111f88fcb916b87c60a1ebf822ed7937cbe43e93`
**`FormalAudit.lean`** `7cd8cbe5dc4eae123f6255ef63bd6731ef4cd43870d0b552f48d0876e62327e9`
**`TokenLedgerExec.lean`** `32407543a96a38840561f38c73171c68a5c102befb535ece6480d8f6d5e4994e`
**`UnlinkPoolArtifact.lean`** `c0dc00d7e6b244389245413689e985cf770e74758d9de04e34419a311598a1e5`
**`VerifierRouterArtifact.lean`** `4dfabd70ee937e0829ddb652428707e1c7ccc4949bdee5e69744dcb0e87284ff`

## Model–code differential (dual-implementation conformance)

The Solidity-to-Verity correspondence that underlies the Section 5 behavioral theorems was checked dynamically, not only by static crosswalk. The repository ships a dual-implementation harness (`test/helpers/VerityTestArtifacts.sol`) that selects the contracts under test from the `UNLINK_TEST_IMPL` environment variable: `solidity` deploys the hand-written `UnlinkPool` and `VerifierRouter`, while `verity` deploys EVM bytecode compiled from the Verity model (`verity/artifacts/foundry/*.verity.json`, required `status == "generated"`). Only `UnlinkPool` and `VerifierRouter` differ between the two sides; the remaining test fixture is shared.

The curated 27-file conformance suite was run under both implementations:

```
cd protocol/contracts
# canonical driver (regenerates artifacts, then runs both sides):
npm run test:duality
# or, per file against the existing artifacts:
UNLINK_TEST_IMPL=solidity forge test --match-path test/<File>.t.sol
UNLINK_TEST_IMPL=verity   forge test --match-path test/<File>.t.sol
```

The model proof surface is aligned to the audited `7617b3e` behavior: the two defensive balance-accounting guards that the production `UnlinkPool.sol` carries on top of the audited code (a deposit balance-underflow guard and a withdraw over-withdraw / recipient-delta guard) are *not* present in the Verity model, so the behavioral theorems are stated over exactly the audited bytes. Regenerating the model artifacts from that source and running the suite against the checked-in production `UnlinkPool.sol` (which retains both guards at the delivery commit `dda647c`) yields, across all 27 files under both implementations, *0 failed, 0 skipped* on each side. Differential result: GREEN. `VerifierRouter` bytecode is identical between `7617b3e` and `dda647c`, so `UnlinkPool` is the only contract that differs across the two sides. Toolchain: `forge 1.6.0`, `solc 0.8.35`.

**On the two production guards (optional defense-in-depth).** The guard-free model and the guard-bearing production Solidity agree on every vector because the guards are behavioral no-ops here, not because the suite misses them. In the model `Uint256.sub` wraps modulo $2^{256}$, so on an underflow the retained equality check `(subab)==amount` fails and reverts with the *same* custom error the removed guard carried (`PoolDepositBalanceMismatch` / `PoolWithdrawBalanceMismatch`); in the production Solidity that guard and the same equality check share one combined `require`, again reverting that custom error. Unlink **may** keep the two guards in production (`dda647c` ships them in `src/UnlinkPool.sol`) as defense-in-depth, converting a would-be checked-arithmetic panic into the named custom error. This is optional hardening, not a must-fix: on that same underflow the audited `7617b3e` code already reverts (via a checked-arithmetic `Panic(0x11)`) and the model already reverts (via the wrapping-sub equality check, with the custom error), so the guards change only the revert *reason* in production Solidity, never which transactions are admitted.

**Scope of this evidence.** The differential establishes behavioral parity between the hand-written Solidity and the Verity-model bytecode *over the concrete vectors in the curated suite*. It is not an exhaustive semantic-equivalence proof: input regions that no vector exercises are outside its reach. It is the load-bearing fidelity check connecting the audited Solidity to the model over which the Section 5 theorems are stated, and it complements (rather than replaces) the static crosswalk in Section B.
The report artifact is built with:

```
cd unlink-audit
make
```

The recommended handoff package for Unlink is built with:

```
cd unlink-audit
make deliverables
```

This produces:

- `dist/report.pdf`
- `dist/unlink-audit-sources-7617b3e-dda647c.zip`

The zip contains the audited Solidity reference source, the authoritative Verity model snapshot from the LFG Unlink monorepo fork, generated model artifacts, and a package manifest with file hashes. The PDF is the narrative report; the accompanying `.zip` is the reproducible source and evidence package used to support the report's formal-audit claims.

## Package layout

The companion audit source and evidence package, `unlink-audit-sources-7617b3e-dda647c.zip`, is organized as follows:

| Path | Contents |
|---|---|
| `audited-solidity/` | Solidity contracts at audited commit `7617b3eebcf37ab42124fe570eb7e065cf8c8461`. |

| Path | Contents |
| --- | --- |
| `verity-model/` | Verity/Lean model (aligned to the audited `7617b3e` behavior), generated contract artifacts, and formal-audit reproducibility scripts; the exact delivery commit is recorded in `MANIFEST.json`. |
| `zk-evidence/` | Circuit, verifier-key, and ZK regression evidence used by the report's artifact checks. |
| `MANIFEST.json` | Machine-readable package inventory, pinned commits, source roots, and file metadata. |
| `SHA256SUM` | File hashes for package integrity checking. |

# F Auditor-verified behavioral extensions

This appendix records the behavioral Lean theorems this review proves over the `_spend` helper, the exact-delta theorems over the executable token ledger, the kernel-`decide` grading of the constant-manifest checks, and a revert-on-re-entry theorem on the delivered `emergencyWithdraw` entrypoint, all four part of the package's formal-audit gate. The material is of four kinds, graded and routed differently.

*Behavioral revert theorems, shipped and enumerated.* Two small state-machine theorems on `_spend` state the two reader-facing revert properties as behavior rather than as source-shape. Because they are kernel-clean, they **are** counted in the proof-grade behavioral headline (8 theorems: the 2 state-machine success and root theorems, these 2 reverts, and the 4 token-ledger delta theorems below). Counting them as behavioral, rather than at the weaker source-shape presence grade that a structural guard scan would carry, is the conservative reading, not an inflation. Both theorems live in the package's `State.lean`, beside `spend_success_of_fresh_in_field`, and are enumerated as AP/IC evidence atoms in the gate's `concreteEvidenceAtoms` manifest, which holds 27 atoms (the 23 load-bearing AP/IC facts, three delivery-only atoms, and the entrypoint reentrancy-revert theorem below).

*Executable token-ledger deltas, shipped and enumerated.* Four behavioral theorems characterize the external-token interface over an executable `(token,account)` ledger: a transfer moves exactly the transferred amount from sender to recipient, a Permit2 deposit credits the pool by exactly the deposited amount, the withdrawal balance-delta guard's equalities hold by construction, and `balanceOf` is coherent with the preceding write. They live in the package's `TokenLedgerExec.lean`, are kernel-clean (`propext/Quot.sound`, no `ofReduceBool`), and are enumerated as counted `concreteEvidenceAtoms` entries on rows `AP-Ax4`, `AP-S5`, and `IC-S3`. They **narrow** the external-contract assumption `AP-Ax4` to the residual that a production ERC-20 conforms to the ledger model; they do *not* relabel its `assumed` marker, and the assumption count is unchanged. The subsection below states the four theorems and their axiom footprint.

*Structural and constant checks under kernel* `decide`. The pinned `FormalAudit.lean` keeps its structural scans and `ic_x1` constant-manifest proof in kernel-reducible form, so all 15 structural and constant checks discharge with kernel `decide` rather than `native_decide`. None of the 15 rows carries the `ofReduceBool` compiler-trust axiom, which raises the proof-grade count to 23, with no change to which facts the gate evaluates.

*Boundary-narrowing, shipped as a separate atom.* The transient reentrancy-guard row is narrowed by a revert-on-re-entry theorem proved over the delivered `emergencyWithdraw` body. It splits that row into a proved control-flow layer and a residual opcode-faithfulness layer, and it is now **counted**: it is enumerated as its own `concreteEvidenceAtoms` entry (bringing the manifest to 27 atoms) and is a conjunct of `formal_audit_ready_for_delivery`, so the delivery theorem transitively depends on it. It does *not* relabel the model's `transient_reentrancy_guard` provenance string, which stays `[assumed]`: the win is a separate checked atom, not a flipped label. The assumption count is unchanged; the row is narrowed, not removed. The subsection below states the theorem and its axiom footprint.

## What the model proves about `_spend`

The model characterizes `_spend` in the *success* direction and in both *revert* directions. `State.lean` proves `spend_success_of_fresh_in_field` (a fresh, in-field nullifier runs to `ContractResult.success` with exactly that nullifier marked), and `FormalAudit.lean` carries the forward-from-success direction `AP_S3_StateSpendConcreteGenerated`: every run that ends in `success` leaves the nullifier concretely marked. A source-shape presence check, a

Boolean predicate matching the modeled statement list for a `Stmt.requireError` guard carrying `StateNullifierOutOfField` (and likewise `StateNullifierAlreadySpent`), would establish only that the source text *contains* the two guarded reverts; it would not establish that the modeled function *behaviorally* reverts, with the branch-correct message, on those inputs. The theorems below establish exactly that, and are enumerated in the gate beside the success-direction atoms.

## The audited helper

The theorems are proved over the `_spend` definition exactly as it appears in the audited model file `State.lean`, whose SHA-256 is recorded in the formal verification gate (Section E) as `be94395c...9ef6e`. The `_spend` definition (together with the error constants `errStateNullifierOutOfField` and `errStateNullifierAlreadySpent` it raises and the `SNARK_SCALAR_FIELD` bound it tests) is the audited definition, reproduced verbatim below, not a paraphrase:

```
-- State.lean (SHA-256 be94395c...9ef6e): the audited _spend helper.
def _spend (st : StateStorage) (nullifierHash : Uint256) :
    Contract StateStorage := do
  if (nullifierHash : Nat) >= (PoolConstants.SNARK_SCALAR_FIELD : Nat) then
    require false errStateNullifierOutOfField
  if st.nullifierHashes (nullifierHash : Nat) then
    require false errStateNullifierAlreadySpent
  return { st with
    nullifierHashes :=
      fun n => if n == (nullifierHash : Nat) then true else st.nullifierHashes n }
```

## Extension theorems: both revert directions

The two theorems state that the modeled `_spend` run actually reverts (returns `ContractResult.revert` with the branch-correct error string) on an out-of-field nullifier and on an already-spent in-field nullifier. Taken with the delivered success direction, they give a complete behavioral characterization of `_spend`: success exactly when the nullifier is fresh and in field, and otherwise a revert carrying the branch-correct message.

```
-- Contributed into protocol/contracts/verity/UnlinkXyz/Pool/State.lean
-- (namespace State); State._spend = the audited model.
theorem no_double_spend_reverts
    (env : ContractState) (st : StateStorage) (nh : Uint256)
    (hField : (nh : Nat) < (PoolConstants.SNARK_SCALAR_FIELD : Nat))
    (hSpent : st.nullifierHashes (nh : Nat) = true) :
    (State._spend st nh).run env
      = ContractResult.revert State.errStateNullifierAlreadySpent env := by
  unfold State._spend
  simp [Contract.run, Verity.instMonadContract, Verity.bind, require,
        Nat.not_le_of_gt hField, hSpent]

theorem out_of_field_reverts
    (env : ContractState) (st : StateStorage) (nh : Uint256)
    (hOut : (PoolConstants.SNARK_SCALAR_FIELD : Nat) ≤ (nh : Nat)) :
    (State._spend st nh).run env
      = ContractResult.revert State.errStateNullifierOutOfField env := by
  unfold State._spend
  simp [Contract.run, Verity.instMonadContract, Verity.bind, require, hOut]
```

### Axiom footprint

Both theorems are kernel-checked. `#print axioms` reports only `propext` and `Quot.sound`, the same footprint as the delivered behavioral theorems, and in particular **no `Lean.ofReduceBool`**, the compiler-trust axiom that `native_decide` would introduce. Verbatim output:

```
'Benchmark.Cases.UnlinkXyz.Pool.State.no_double_spend_reverts' depends on axioms:
↪ [propext, Quot.sound]
'Benchmark.Cases.UnlinkXyz.Pool.State.out_of_field_reverts' depends on axioms:
↪ [propext, Quot.sound]
```

### Kernel-`decide` re-derivation of the constant-manifest checks

The same harness run also confirms the assurance-grading claim of Section 5: the four constant-manifest parity checks in the pinned `ConcreteAPIC` namespace (the `SNARK_SCALAR_FIELD`, `MAX_NOTE_VALUE`, circuit-ID, and lazy-IMT zero-subtree literals) are discharged by plain kernel `decide`, which keeps those four rows off the `ofReduceBool` compiler-trust path:

```
example : ConcreteAPIC.snarkScalarFieldMatchesManifestConcrete = true := by decide
example : ConcreteAPIC.maxNoteValueMatchesManifestConcrete = true := by decide
example : ConcreteAPIC.circuitIdMatchesManifestConcrete = true := by decide
example : ConcreteAPIC.lazyImtZ7MatchesManifestConcrete = true := by decide
```

In the delivered model, the same kernel-`decide` discharge extends to the other eleven structural scans in `FormalAudit.lean`: the file carries no `native_decide` call, so all 15 structural and constant rows are off the `ofReduceBool` path and the file's `#print axioms` footprint excludes the compiler-trust axiom. This is the source of the proof-grade count of 23.

### Transient reentrancy guard: revert-on-re-entry on the delivered entrypoint

The transient reentrancy-guard row of Section 5 bundles two distinct facts: (i) the guard's *revert-on-re-entry control-flow* (that an entry through the guard while it is already held reverts, before any other effect) and (ii) *opcode faithfulness*: that the model's `tload`/`tstore` denote the real EIP-1153 transient-storage semantics. This review discharges the reader-facing revert direction of (i) over the delivered entrypoint and leaves (ii) as the residual boundary. The split **narrows** the row; it does not remove it, and the assumption count is unchanged.

The Pool's inline guard lowers, after the Verity `function` macro, to a `tload` of the guard slot, a `requireError` that the slot is zero (reverting `ReentrancyGuardReentrantCall()` otherwise), and a `tstore` of `1`, at the head of `deposit`, `transfer`, `withdraw`, and `emergencyWithdraw`, with a success-path release to `0`. Because `emergencyWithdraw` is permissionless, that guard read is the very first effect of the function, so the revert-on-re-entry direction is provable directly on the delivered body rather than on a reconstruction. The gate theorem states it: for any state whose guard slot is already set and any transaction array, the generated `emergencyWithdraw` reverts with exactly `ReentrancyGuardReentrantCall()` and returns the entry state unchanged. The proof unfolds the generated entrypoint and discharges by `simp`/`unfold` (not `native_decide`):

```
-- namespace Benchmark.Cases.UnlinkXyz.Pool.FormalAudit.ConcreteAPIC,
-- over the delivered UnlinkPool.emergencyWithdraw entrypoint (FormalAudit.lean).
def EmergencyWithdrawReentrancyRevertGenerated : Prop :=
```

```
  ∀ (env : ContractState) (transactions : Array UnlinkPool.WithdrawalTransaction),
    env.transientStorage (UnlinkPool.REENTRANCY_GUARD_STORAGE : Nat) ≠ 0 →
    (UnlinkPool.emergencyWithdraw transactions).run env
      = ContractResult.revert "ReentrancyGuardReentrantCall()" env

theorem emergency_withdraw_reentrancy_revert_generated :
    EmergencyWithdrawReentrancyRevertGenerated := by
  intro env transactions hLocked
  unfold UnlinkPool.emergencyWithdraw
  simp [Contract.run, Verity.bind, Verity.instMonadContract,
        Verity.getStorageAddr, Contracts.tload, Contracts.requireCustomError,
        Contracts.revertCustomError, Contracts.formatCustomError,
        Verity.require, hLocked]
  rfl
```

This theorem is a counted gate atom, not a scratch lemma: it is enumerated in `concreteEvidenceAtoms` (kind `behavioralEntrypointTheorem`, a ninth behavioral atom beside the eight `_spend` state and token-ledger theorems) and is a conjunct of `formal_audit_ready_for_delivery`, so the delivery theorem transitively depends on it. It does *not* relabel the model's `transient_reentrancy_guard` provenance string, which the gate still carries as `[assumed]`: that string is a source-shape provenance label, not a checked obligation, so the load-bearing win is this separate kernel-checked atom rather than a flipped label. The proof closes through `simp`/`unfold` only, so its footprint is kernel-clean (`propext`, `Quot.sound`), with **no `Lean.ofReduceBool`** (the `native_decide` compiler-trust axiom) and no `sorryAx`, and the delivery theorem's footprint is unchanged. Verbatim `#print axioms` output (reproduced by the staged probe in Section E):

```
'Benchmark.Cases.UnlinkXyz.Pool.FormalAudit.ConcreteAPIC.emergency_withdraw_reentrancy_revert_gener
↪ depends on axioms: [propext, Quot.sound]
'Benchmark.Cases.UnlinkXyz.Pool.FormalAudit.formal_audit_ready_for_delivery'
↪ depends on axioms: [propext, Quot.sound]
```

What remains assumed is layer (ii): that `tload`/`tstore` in the model denote real EIP-1153 transient storage. That is exactly the residual the EVMYulLean bridge (Section 5) would discharge, since it models `TLOAD`/`TSTORE` directly. Proving the control-flow layer in-model does not short-circuit that step; it isolates it. The atom proves the revert-on-re-entry direction, the one the preview's claim rests on; it does not separately assert the lock-taking or full mutex-composition directions over the entrypoint, which would need the guard in isolation.

## Why the EVMYulLean route is unfinished development

Section 5 records that the ABI-encoding, keccak memory-slice, Permit2/router/verifier ABI, public-signal packing, and transient reentrancy-guard rows are refined against the source crosswalk rather than discharged against an executable EVM semantics. The Verity framework does pin an executable EVM-semantics formalization, EVMYulLean (`lfglabs-dev/EVMYulLean`, pinned commit `7785a9bb`), and a compiler-correctness bridge (the `EvmYulLean` backend) exercised for the framework's own ERC20, Vault, and SimpleStorage examples. The Unlink Pool model, however, is interpreted over the source-level `SourceSemantics` interpreter, which returns `none` for external calls and does not evaluate against the Yul/EVM backend, so the rows remain honest opaque boundaries. Independent inspection of the pinned framework shows that routing the Pool through the bridge is substantial new development rather than a status change, for three concrete reasons.

- The bridge's *executed* semantic-equivalence theorem (`nativeResultsMatchOn`) is

demonstrated end-to-end only for the framework's single-slot `SimpleStorage` fixture over persistent `SSTORE/SLOAD`. The ERC20 and Vault examples reach only bridged-fragment membership and the revert path, with their success paths explicitly deferred, and *no* executed equivalence in the delivered development covers transient `TSTORE/TLOAD`: only syntactic fragment-membership (`BridgedSafeStmts`) plus low-level per-opcode frame lemmas.

- The Unlink Pool ships no native witness, and its guarded entrypoints (`deposit`, `transfer`, and the rest) contain external-call (ECM) statements that the bridge places outside its safe-body fragment, so the whole-function equivalence whose execution would discharge the guard cannot currently be stated for the functions the guard lives in.
- The framework's `local_obligations proved` status is a provenance label carrying a justification string, not a machine-checked obligation, so flipping the guard's `assumed` marker to `proved` would weaken the disclosure, not strengthen the proof.

A genuine discharge therefore requires a new SimpleStorage-style executed-equivalence theorem for a transient-storage fixture plus a Pool native witness; the row accordingly remains an honest assumption. The keccak memory-slice row cannot be eliminated this way at all: EVMYulLean's `keccak256` is itself an `opaque` FFI primitive, so discharging against it relocates the hash assumption from the Unlink model into the EVM model rather than removing it. Linking the Pool model to the existing bridge stays the documented direction for converting a subset of these rows into proved facts, but it is unfinished development of the scale sketched here, not a label change; this report does not claim to have done so.

## Reproduction and scope

The probe files are compiled with `lake env lean` against a staged Lake harness that builds the same monorepo model, under the pinned toolchain `Lean 4.22.0, commit ba2cbbf09d4978f416e0ebd1fceeebc2c4138c05`, identical to the model gate (Section E). The two `_spend` revert theorems live in the package's `State.lean` (namespace `State`), and `FormalAudit.lean`'s constant-manifest proof uses kernel `decide` for the four constant checks; both are scoped to the `_spend` definition and to the `ConcreteAPIC` constant definitions, which match the gate byte-for-byte. The `#print axioms` output above is reproduced from a standalone probe (namespace `State`, importing only `Benchmark.Cases.UnlinkXyz.Pool.State`) so a reader can re-derive the `[propext, Quot.sound]` footprint without building the whole package; the shipped theorems carry the identical name and footprint. The reentrancy revert-on-re-entry theorem lives in `FormalAudit.lean`'s `ConcreteAPIC` namespace and is scoped to the delivered `emergencyWithdraw` entrypoint; it compiles as part of the gate build and is a counted atom, re-derived at `[propext, Quot.sound]` by the same staged `lake` harness. With it, the gate's `concreteEvidenceAtoms` manifest holds 27 atoms (the 23 load-bearing AP/IC facts, three delivery-only atoms, and the entrypoint reentrancy-revert theorem).

# G Anticipated questions

Four questions recur when weighing a "we proved it" claim, each with the honest answer the report supports. We suggest reviewing them before relying on the result.

- **A proof is only as strong as its assumptions. What is this really resting on?** The assumptions are named in the open, not hidden inside the word "proven": 9 trust assumptions (the cryptography, Groth16 / Poseidon / EdDSA; the circuit relation; ERC-20 / Permit2 conformance; EVM execution and liveness) and 12 tracked boundaries. The proof is airtight *given* those; most are the premises every system of this kind leans on, and the one project-specific premise, that the spend circuit encodes the intended relation, is exactly where the circuit tests and the value-conservation assumption are focused.
- **You proved a model. Is the deployed contract the same thing?** It is why the model is not only proven but executed. We compiled it to real EVM bytecode and ran that and the actual contract through one identical conformance suite; both passed every vector with zero failures. That is strong evidence the proven model behaves like the deployed code on every case the suite covers, while staying honest that it is parity over those test vectors, not a separate proof of full semantic equivalence.
- **Only a few core theorems. Isn't that thin?** Four of the eight behavioral theorems are the *core money-rule* theorems, the ones that decide double-spend and insertion integrity; the other four pin exact token-ledger deltas, and together they sit on 15 further machine-checked structural checks, 23 in total. The number is small because the core rules are few and stated tightly, not because coverage is shallow: the spend rules are proven across all three possible cases (fresh note accepted, reused rejected, out-of-range rejected), so no case hides in the gaps.
- **What is the most important thing you did not prove?** That money-in equals money-out as on-chain arithmetic. By design the pool keeps no balance ledger in contract state, so that sum lives in the zero-knowledge circuit, which we list as an assumption and cover with circuit tests, not with this report's proofs. We say so plainly, so "no double-spend, proven" is never mistaken for "all value math, proven."