Blockchain Checkpoint Publication Under Uncertainty: Separating Relay Stalls from Interface Rejection

Jonathan Oleszkiewicz
BaaS.sh
France
jonathanolesz@gmail.com

***Abstract*—Cross-chain bridges and rollup settlement interfaces have suffered repeated high-impact attacks that exploit the control plane (relayer keys, governance upgrades, and verifier semantics) rather than the underlying cryptography. When a validity proof is visible but the corresponding on-chain checkpoint is still missing, the defender faces two opposite responses: wait, because the relay is merely slow, or quarantine the settlement interface, because it may be compromised. A passive monitor observes the same symptom in both cases, leaving a Security Operations Center (SOC) with a single alert and no principled way to choose between the two. This paper introduces *evidence-guarded triage*, a deterministic detection primitive that closes this blind spot. Beyond observing proofs and records, the monitor reconstructs the canonical checkpoint call, probes whether the live settlement interface would accept it, and routes every alert to one of five operator-actionable states. We evaluate a prototype in a 110-run live campaign on the Sepolia Ethereum testnet, covering healthy operation on two host profiles as well as two injected incident classes. All 110 runs reach their expected outcome: relay stalls are held for escalation and injected interface rejections are flagged on the first probe, while probing overhead stays between 0.002% and 0.003% of proving time.**

***Index Terms*—blockchain security, decentralized security, intrusion detection, resilient networks, security operations center, zero-knowledge proofs**

I. Introduction

Rollups, bridges, and settlement services split execution, proving, relaying, and checkpoint publication across components and trust domains, expanding the attack surface of the rollup control plane [1]–[4]. Escape hatches and blob-based data availability further decouple off-chain validity from on-chain exposure [5], [6].

Once a proof is visible, a missing checkpoint can mean two very different things: a stalled relay, which calls for escalation, or a live interface that rejects the canonical call, which calls for retry suppression. A Security Operations Center (SOC) limited to proof and record visibility cannot tell the two apart, and both mistakes are costly: waiting can extend attacker dwell time, while quarantine can cause avoidable unavailability.

We propose *evidence-guarded triage*, a deterministic detection primitive that closes this blind spot by adding an admissibility witness A, an evidence-bound control-plane signal. At each iteration, the monitor reconstructs the canonical sender, value, entrypoint, and calldata, evaluates the call at the same pinned settlement view used for the record lookup, and deterministically routes the alert to the matching Service-Level Objective (SLO) policy. This paper makes three contributions: (1) an evidence-guarded triage primitive that maps every iteration to one of five operator-actionable labels; (2) a formalization of the passive-monitoring blind spot and of the minimal signal that closes it; and (3) a 110-run live Sepolia campaign covering primary and secondary hosts as well as two injected incident classes, relay stall and interface rejection.

This paper is organized as follows. Section II presents the threat model, and Section III reviews related work. Sections IV–V formalize the failure modes and introduce the triage primitive. Section VI evaluates the prototype, Section VII discusses operational use and reproducibility, and Section VIII concludes.

II. Threat Model and Trust Assumptions

The attacks that motivate this work compromise the publication path rather than the proofs themselves. This section makes that adversary precise and states which components we trust.

Adversary capabilities. We model an adversary who can (a) take hostile control of the relayer or publisher key path to delay, suppress, or replay the canonical checkpoint call; (b) push a malicious upgrade of the manager or verifier contract (logic substitution, role rotation, or adversarial pause toggle) that turns the canonical call from accepting to rejecting; (c) drive an Application Binary Interface (ABI) pinning or calldata-encoding attack that causes the reconstructed canonical call to silently drift; or (d) tamper with the monitor's view of settlement state, by (d.1) serving inconsistent or unavailable Remote Procedure Call (RPC) responses, or (d.2) eclipsing a single endpoint with internally consistent fabricated state. The attacker's goal is to make the SOC misroute: either suppress retries on a legitimate stall, amplifying a denial of service, or keep retrying against a compromised interface, extending attacker dwell time.

Trust assumptions. The monitor binary and operator configuration are trusted. The pinned settlement view is taken as ground truth at confirmation depth k. Proof-log integrity is mediated through the evidence guard G introduced in Section V. We rely on the proving stack for prover soundness, the settlement

chain for consensus integrity, the monitor host for operating-system-level isolation, and the monitor's read-only signer for key safety.

III. Related Work

Two lines of defensive work come closest to the settlement-publication path we protect: smart-contract runtime monitoring and cross-chain attack detection. We compare them on the ability that matters here, namely disambiguating the post-proof window in which a checkpoint record is missing.

On the first line, prior work verifies smart contracts at runtime, checks business-logic conformance, and pauses cross-chain protocols automatically for defensive incident response [7]–[10]. On the second, bridge-attack detection, cross-chain anomaly monitoring, static bridge-contract vetting, and distributed runtime verification for cross-chain protocols address adjacent threat surfaces [11]–[15]. Neither tells a stalled relay from a rejecting interface once the proof is visible and the record is missing, which is precisely the decision a SOC must automate. Our primitive fills this gap by probing the canonical call at a pinned settlement view and binding each outcome to an evidence-backed response policy.

IV. Adversarial Failure Modes: Stalls vs. Interface Compromise

We now define the monitored objects and the two adversarial failure modes that produce an identical passive symptom. The monitored system is a Zero-Knowledge (ZK) rollup deployment that publishes checkpoints to its settlement chain through a canonical publication path. A *checkpoint claim* is the obligation to publish source height h. A *checkpoint record* is the manager entry keyed by h and proof-artifact identifier. *Publication* is a successful `publishCheckpoint` on the manager. Fig. 1 summarizes the claim lifecycle and routing.

Failure modes mapped to the threat model. Once a proof is visible, two execution classes present the same passive symptom—proof visible, record missing, written $(P, R) = (1, 0)$ (Section V)—yet require opposite responses. *(i) Relay stalls* (capability (a)). The publisher, relayer, or publication service delays or suppresses the canonical call. *(ii) Interface rejection* (capabilities (b)–(c)). The live settlement interface refuses the call after adversarial upgrade, role rotation, attacker-triggered pause, replay-rule change, or ABI-pinning drift; benign version skew yields the same on-monitor evidence. Capabilities (d.1)–(d.2) raise $G = 0$ and route to `EVIDENCE_FAULT`, with (d.1) caught by intra-iteration inconsistency and (d.2) caught by multi-RPC quorum disagreement when the cross-check is configured.

V. Evidence-Guarded Triage and Security Analysis

We now present the triage loop itself, then analyze the guarantees it provides under the threat model of Section II. For each claim at height h, the monitor pins a confirmed settlement block hash b_t after confirmation depth k. The same b_t serves both the record lookup and any later dry-run. The loop uses four signals over time:

TABLE I
Checkpoint-publication triage priority and operator responses.

Condition	Label	Recommended response
Required evidence failure or $R = 1 \wedge P = 0$	`EVIDENCE_FAULT`	Quarantine automation; re-check block hash, sender path, proof log, calldata, and RPC view.
$P = 0 \wedge R = 0$	`NO_PROOF`	Wait for proof visibility; publication action deferred.
$P = 1 \wedge R = 1$	`COMPLETED`	Clear the alert and archive probe evidence.
$P = 1 \wedge R = 0 \wedge A = 1$	`PENDING`	Wait; after SLO expiry inspect relay, signer, nonce, gas, and RPC health.
$P = 1 \wedge R = 0 \wedge A = 0$	`INTERFACE_REJECT`	Suppress repeated retries; inspect encoding, versioning, roles, pause flags, replay rules, and upgrades.

$P_h(t) = 1$ iff the prover artifact for h is visible in the proof log by time t and passes preflight checks,

$A_h(t) = 1$ iff a pinned-view `eth_call` (read-only simulation) of the canonical sender, call value, gas policy, entrypoint, and calldata accepts ($A_h(t) = 0$ for a reproducible contract-level revert),

$R_h(t) = 1$ iff the matching checkpoint record is readable in that same pinned view,

$G_h(t) = 1$ iff every evidence item required by the current branch is acquired consistently.

G is branch-local. It validates proof-log acquisition, pinned-block selection, and record-lookup consistency before any probe, and adds canonical-call reconstruction and revert-evidence checks when A is sampled. The admissibility signal A is evaluated exactly when

$$P = 1, \quad G = 1, \quad R = 0.$$

Any failure of the required evidence routes through $G = 0$ to `EVIDENCE_FAULT`. Table I summarizes the five labels, their triggering conditions, and the recommended operator responses.

`INTERFACE_REJECT` suppresses retries while pinned dry-runs keep rejecting. An accepting probe restores `PENDING`. The pinned view, the canonically reconstructed call, and the branch-local guard G together yield a deterministic decision per iteration. In contrast, a naive `eth_call` against `latest` could race with chain progression or use a non-canonical sender, silently masking rejection as delay.

Proposition 1 (Passive-Monitor Detection Blind Spot). Let the passive observation be

$$\sigma_{PR}(t) = \big(P_h(t),\, R_h(t)\big),$$

and let τ^{stall} and τ^{rej} denote the σ_{PR}-traces of a relay-stall and an interface-rejection execution. For every t in a post-proof window with $R = 0$,

$$\tau^{\mathrm{stall}}(t) \;=\; \tau^{\mathrm{rej}}(t) \;=\; (1, 0).$$

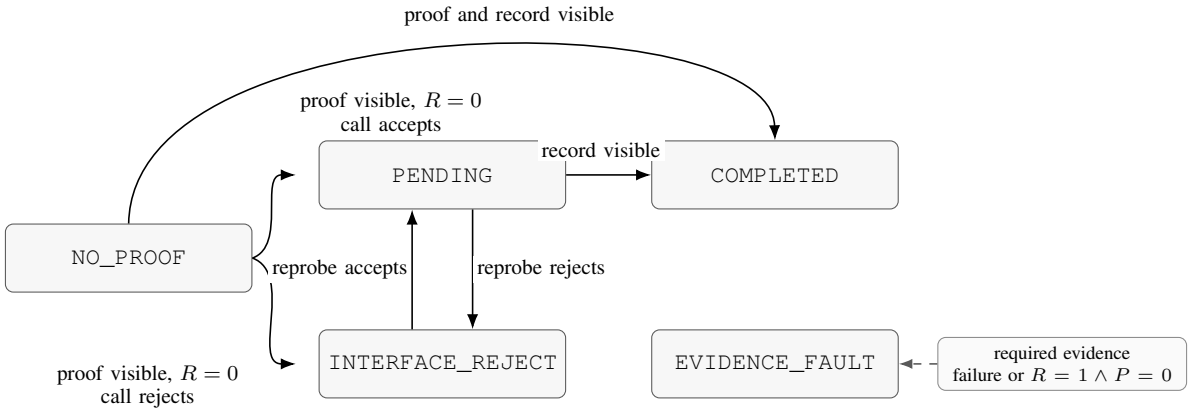

Fig. 1. Operational state machine.

Proof sketch. In a relay stall, the canonical call would accept while publication is withheld. In interface rejection, the canonical call rejects and the record remains absent. Both executions therefore expose $P = 1$ and $R = 0$ at every sampled time, so a SOC restricted to (P, R) sees a single symptom where two opposite responses are required.

Corollary 1 (A is a minimal admissibility witness). At any t with $P = 1$ and $R = 0$,

$$A^{\text{stall}}(t) = 1 \qquad \text{and} \qquad A^{\text{rej}}(t) = 0.$$

The admissibility signal A, bound to the same pinned view b_t and guard G, is a minimal addition to (P, R) that closes the detection blind spot of Proposition 1.

Probe timing. Let t_0 be the first time with consistent proof/record evidence, $P = 1$, and $R = 0$. The monitor issues one immediate dry-run against pinned view b_{t_0}. Open claims are then re-probed every p seconds, so any later interface change is captured with detection delay

$$\Delta_{\text{detect}} \leq p.$$

Security analysis. *Completeness.* The signal $A = 0$ fires on any rejection at the pinned view, whether adversarial (capabilities (b)–(c)) or benign (version skew, replay-rule change). *Soundness.* Pinning to b_t makes the dry-run deterministic against chain progression. Every rejection is reproducible from the recorded revert selector, return data, and block hash. *Trust boundary.* G natively covers (d.1) by validating intra-iteration consistency of the pinned view and proof-log acquisition; coverage of (d.2) comes from adding two or more independent settlement-RPC endpoints to G's cross-check set, with quorum agreement gating the iteration. The reported campaign uses a single endpoint per host (exercising (d.1)); production deployments enable the multi-RPC quorum for (d.2), optionally combined with append-only evidence logging.

VI. Prototype and Sepolia Campaign

We implemented the triage loop as a prototype and evaluated it in a live measurement campaign on the Sepolia Ethereum testnet, covering healthy operation and two injected incident classes.

Prototype and anchors. The loop runs on the Reth Ethereum execution client, with an Execution Extension (ExEx) consuming block updates and proof-log events [16]. An OpenVM ZK prover produces validity-proof artifacts [17]. Table II records the chain IDs, deployment anchors, and run controls of the Sepolia deployment shared by all runs.

Pinned-view and canonical-call policy. The iteration policy fixes

$$k = 2, \qquad p = 30 \text{ s}, \qquad T_{\text{SLO}} = 300 \text{ s},$$

pinning a Sepolia block hash at depth k for both the dry-run and the record lookup. Proof visibility is the first proof-log event for the run's recorded source height h (manifest field `actual_height`). The artifact must pass preflight and match the expected calldata digest. Canonical calldata is reconstructed from h, proof-artifact identifier, manager/verifier ABI version, configured publisher, and Table II anchors. The `eth_call` runs with the configured publisher as `from` against the deployed state at the pinned hash. Production deployments may replace $k = 2$ with finalized-block or multi-RPC confirmation.

Incident controls. The fault-injection harness sets controlled scenario tags before the loop decision. Relay-stall runs withhold publication after proof visibility (capability (a)). Interface-rejection runs inject a canonical claim with a payload version the live manager rejects; the returned revert data becomes `INTERFACE_REJECT` evidence. The admissibility probe observes only whether the canonical call reverts at the pinned view. The rejection evidence it collects is therefore identical whichever mechanism turns the interface from accepting to rejecting—benign version drift, an adversarial upgrade, a role rotation, or a pause toggle (capabilities (b)–(c)). The injected mismatch thus exercises, under controlled and reproducible conditions, the same detection path that these adversarial events would trigger.

Run-set integrity. The campaign manifest pins scenario tags and `actual_height` values; 73 preflight checks validate

ABI/address consistency, publisher authorization, proof availability, calldata reconstruction, pinned-block RPC support, and record lookup.

TABLE II
CAMPAIGN CONTROLS AND SEPOLIA DEPLOYMENT ANCHORS.

Item	Value
Run set	110 live Sepolia runs on the settlement deployment anchored below.
Cohorts	Healthy: 75 primary, 25 secondary; incidents: 5 relay-stall, 5 payload/version mismatch.
Integrity	110/110 slots observed; full label agreement; 73/73 preflight checks passed.
Stack	Reth ExEx observer; OpenVM v1.5.0 prover; on-chain verifier; controlled relayer.
Hosts/chains	Primary: Google Cloud n2-standard-32, 32 vCPUs; secondary: separate 32-vCPU profile. Source ID 1337; Sepolia ID 11155111.
Manager	`0xBFA11CaA9713226E7b47d073F094fa0a7c8A1759`
Verifier	`0xEbA01E929011343C1Ce39f5666e287Cd3d9b342e`
Publisher	`0x596e862b34c56d8F35b91c265e84783E1C9819B0`
Policy	$k = 2$, $p = 30\,\mathrm{s}$, $T_{\mathrm{SLO}} = 300\,\mathrm{s}$. Same pinned block hash for dry-run and lookup.

TABLE III
PRIMARY-HOST HEALTHY MEANS.

Height bucket	τ_p (s)	$\Delta_{P\to R}$ (s)	τ_d (ms)	Single-probe / proving time (%)
10	1447	62.7	32.8	0.0023
25	1409	68.8	31.6	0.0022
50	1398	70.9	33.4	0.0024
75	1470	71.0	33.8	0.0023
100	1451	73.1	35.7	0.0025

Each bucket has $N = 15$ runs, 75 total. Height bucket is an experimental grouping label spanning multiple distinct source heights. τ_p is proof-log latency, $\Delta_{P\to R}$ is proof-to-record delay, τ_d is immediate dry-run latency, and the ratio is $100 \times (\tau_d/1000)/\tau_p$.

Healthy paths. All 100 healthy runs follow the expected `NO_PROOF`, `PENDING`, and `COMPLETED` sequence through `publishCheckpoint`. The probe-to-proving ratio settles at 0.002%–0.003% (Table III), and each dry-run completes in tens of milliseconds, far below the 30 s reprobe period. The decision-relevant latencies are $\Delta_{P\to R}$ and τ_d. The average total dry-run time per run is 74.3 ms on the primary host (2.21 probes on average) and 78.6 ms on the secondary host (2.32 probes).

Detection routing under controlled incidents. Table IV summarizes scenario decisions; across the 110 runs, every loop decision matches its controlled scenario label. The passive (P, R) baseline collapses both incident classes into the same alert $(1, 0)$, while A keeps the relay stall in `PENDING` until SLO expiry and immediately suppresses retries when the canonical call rejects.

TABLE IV
CONTROLLED SCENARIO LABELS AND INCIDENT RESULTS.

Scenario	N	Passive view	Observed loop decision
Healthy	100	Record eventually visible	`COMPLETED`; observed sequence in every run.
Relay stall	5	$P = 1, R = 0$ through SLO	`PENDING` after accepting immediate probe; 11 probes, including the immediate probe; relay-owner route after SLO expiry.
Payload/version mismatch	5	Same $P = 1, R = 0$ view	`INTERFACE_REJECT` on immediate probe; one probe; interface/governance route.

SLO calibration. Healthy p95 proof-to-record delays sit at 93.9 s (primary) and 94.0 s (secondary), leaving more than a $3\times$ margin below the 300 s SLO; all healthy runs clear before SLO expiry while relay-stall injections cross the threshold (Table V). Production deployments should tune this threshold to their chain, relayer, confirmation-depth, and RPC policies.

TABLE V
HEALTHY-RUN DISPERSION.

Metric	Primary	Secondary
τ_p (s)	1291 / 1403 / 1576	1330 / 1533 / 1593
$\Delta_{P\to R}$ (s)	62.3 / 62.8 / 93.9	62.5 / 63.0 / 94.0
τ_d (ms)	28.8 / 33.3 / 38.4	26.6 / 33.6 / 38.8
Probe count	2 / 2 / 3	2 / 2 / 3

Entries report min / median / p95. Primary has $N = 75$ runs; secondary has $N = 25$ runs.

VII. OPERATIONAL USE AND REPRODUCIBILITY

The labels feed SOC automation through deterministic evidence bundles: `PENDING` records the proof artifact, call digest, pinned block, and record lookup; `INTERFACE_REJECT` adds revert evidence; and `EVIDENCE_FAULT` captures the conflicting inputs and quarantines automation. Reproducibility artifacts include per-run CSV files, the campaign and topology manifests, preflight and baseline reports, Sepolia transaction hashes, run order, and `actual_height` values.

VIII. CONCLUSION

In this paper, we exposed a detection blind spot at the core of blockchain settlement monitoring: when a validity proof is visible but its on-chain checkpoint is not, a monitor restricted to those two signals provably cannot tell a stalled relay from a rejecting settlement interface, yet the two demand opposite responses. We introduced *evidence-guarded triage*, a deterministic primitive that supplies the missing signal: it reconstructs the canonical checkpoint call, probes at a pinned block whether the live settlement interface would accept it, and routes every alert to one of five operator-actionable labels, each carrying an audit-ready evidence bundle.

In the live Sepolia campaign, the primitive routed all 110 runs to their expected outcome. Relay stalls stayed in `PENDING` through the 300 s SLO, and injected interface rejections were flagged on the first probe. Probing consumed between 0.002% and 0.003% of proving time: a deterministic verdict

at negligible cost. The labels feed existing SOC playbooks by design, and deployment requires no consensus change, no contract modification, and no trust in the relay path.

As future work, we plan to harden the evidence guard G with a multi-RPC quorum and append-only evidence logging, and to bring the primitive to admin-enabled production interfaces, where upgrades, role rotations, and pause toggles make evidence-guarded routing most valuable.

REFERENCES

[1] S. Chaliasos, D. Firsov, and B. Livshits, "Towards a Formal Foundation for Blockchain ZK Rollups," in *Proc. ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, 2025, pp. 2714–2728, doi: 10.1145/3719027.3765115.

[2] S. Chaliasos, I. Reif, A. Torralba-Agell, J. Ernstberger, A. Kattis, and B. Livshits, "Analyzing and Benchmarking ZK-Rollups," in *6th Conf. on Advances in Financial Technologies (AFT 2024)*, ser. *Leibniz Int. Proc. in Informatics (LIPIcs)*, vol. 316, 2024, pp. 6:1–6:24, doi: 10.4230/LIPIcs.AFT.2024.6.

[3] A. Augusto, R. Belchior, M. Correia, A. Vasconcelos, L. Zhang, and T. Hardjono, "SoK: Security and Privacy of Blockchain Interoperability," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024, pp. 3840–3865, doi: 10.1109/SP54263.2024.00255.

[4] T. Xie, J. Zhang, Z. Cheng, F. Zhang, Y. Zhang, Y. Jia, D. Boneh, and D. Song, "zkBridge: Trustless Cross-chain Bridges Made Practical," in *Proc. 2022 ACM SIGSAC Conf. on Computer and Communications Security (CCS)*, 2022, pp. 3003–3017, doi: 10.1145/3548606.3560652.

[5] V. Buterin, D. Feist, D. Loerakker, G. Kadianakis, M. Garnett, M. Taiwo, and A. Dietrichs, "EIP-4844: Shard Blob Transactions," Ethereum Improvement Proposal, 2022. [Online]. Available: https://eips.ethereum.org/EIPS/eip-4844. Accessed: May 11, 2026.

[6] F. G. Figueira, M. Derka, C.-L. Chiu, and J. Gorzny, "A Practical Rollup Escape Hatch Design," in *Proc. IEEE ICBC*, 2025, pp. 1–5, doi: 10.1109/ICBC64466.2025.11114670.

[7] J. Ellul and G. J. Pace, "Runtime Verification of Ethereum Smart Contracts," in *14th European Dependable Computing Conference (EDCC)*, 2018, pp. 158–163, doi: 10.1109/EDCC.2018.00036.

[8] M. Capretto, M. Ceresa, and C. Sánchez, "Monitoring the Future of Smart Contracts," in *Fundamental Approaches to Software Engineering (FASE 2024)*, ser. *Lecture Notes in Computer Science*, vol. 14573, 2024, pp. 122–142, doi: 10.1007/978-3-031-57259-3_6.

[9] M. Eshghie, C. Artho, H. Stammler, W. Ahrendt, T. Hildebrandt, and G. Schneider, "HighGuard: Cross-Chain Business Logic Monitoring of Smart Contracts," in *Proc. 39th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*, 2024, pp. 2378–2381, doi: 10.1145/3691620.3695356.

[10] B. Mateus, A. Augusto, R. Belchior, A. Vasconcelos, and M. Correia, "Automatic Cross-chain Protocol Pausing with Real-time Conformance Checking," in *7th Conf. on Blockchain Research & Applications for Innovative Networks and Services (BRAINS)*, 2025, pp. 1–5, doi: 10.1109/BRAINS67003.2025.11302954.

[11] J. Zhang, J. Gao, Y. Li, Z. Chen, Z. Guan, and Z. Chen, "Xscope: Hunting for Cross-Chain Bridge Attacks," in *Proc. 37th IEEE/ACM Int. Conf. on Automated Software Engineering (ASE)*, 2022, pp. 1–4, doi: 10.1145/3551349.3559520.

[12] R. Belchior, P. Somogyvari, J. Pfannschmidt, A. Vasconcelos, and M. Correia, "Hephaestus: Modeling, Analysis, and Performance Evaluation of Cross-Chain Transactions," *IEEE Trans. Reliability*, vol. 73, no. 2, pp. 1132–1146, 2024, doi: 10.1109/TR.2023.3336246.

[13] A. Augusto, R. Belchior, J. Pfannschmidt, A. Vasconcelos, and M. Correia, "XChainWatcher: Identifying Anomalies in Cross-Chain Bridges," in *Proc. 26th ACM Int. Middleware Conf. (Middleware 2025)*, 2025, pp. 413–426, doi: 10.1145/3721462.3770781.

[14] Z. Liao, Y. Nan, H. Liang, S. Hao, J. Zhai, J. Wu, and Z. Zheng, "SmartAxe: Detecting Cross-Chain Vulnerabilities in Bridge Smart Contracts via Fine-Grained Static Analysis," *Proc. ACM Softw. Eng.*, vol. 1, no. FSE, pp. 249–270, 2024, doi: 10.1145/3643738.

[15] R. Ganguly, Y. Xue, A. Jonckheere, P. Ljung, B. Schornstein, B. Bonakdarpour, and M. Herlihy, "Distributed Runtime Verification of Metric Temporal Properties for Cross-Chain Protocols," in *42nd IEEE Int. Conf. on Distributed Computing Systems (ICDCS)*, 2022, pp. 23–33, doi: 10.1109/ICDCS54860.2022.00012.

[16] Paradigm, "How do ExExes work? – Reth," documentation. [Online]. Available: https://reth.rs/exex/how-it-works/. Accessed: May 13, 2026.

[17] OpenVM contributors, "openvm_sdk," documentation. [Online]. Available: https://docs.openvm.dev/docs/openvm/openvm_sdk/index.html. Accessed: May 13, 2026.