ZYRPC

TECHNICAL WHITEPAPER

Multi-chain RPC infrastructure, security tooling, and on-chain research

---

Version 0.1 — Draft
zyrpc.com

Abstract

Zyrpc is infrastructure for teams building on EVM-compatible chains. It starts with a single problem worth solving well — RPC access that does not go down when one provider does — and is designed to grow into a small set of adjacent services that share the same underlying principle: route around single points of failure, verify rather than assume, and keep the surface area a developer has to trust as small as possible.

This document describes what Zyrpc is building, in three parts. The first is the RPC routing layer itself: a multi-provider, multi-node architecture with continuous health checking and automatic failover, designed so that a single upstream outage never becomes the user's outage. The second is a planned security layer: automated contract scanning and a risk-scoring tool for any deployed address, positioned honestly as automated tooling rather than a substitute for manual audit. The third is a planned research layer: structured, on-chain-sourced reports on token distribution, deployer history, and real usage, intended to replace marketing claims with data a reader can verify.

> **A note on status**
>
> This whitepaper describes the system Zyrpc is building toward, not a system that is fully live today. Sections describing architecture and mechanisms use “will” deliberately, to be accurate about what exists as design versus what exists as running infrastructure. Section 8 (Status and Roadmap) gives a plain account of what is built, in progress, and planned.

1. The problem with RPC today

Every application that reads from or writes to a blockchain depends on an RPC endpoint — the door between the application and the chain itself. In practice, almost all of that traffic flows through a small number of commercial providers. That concentration creates three recurring failure modes.

1.1 Single-provider dependency

A team typically integrates one RPC provider directly into its codebase. When that provider has an outage, a regional disruption, or a degraded response time, the application inherits the problem with no fallback. Provider status pages document this pattern regularly: incidents are infrequent for any single provider, but the number of applications with no fallback path makes the aggregate impact large.

1.2 Switching cost as lock-in

Even when a team is aware of the risk, building and maintaining a second integration — a different SDK surface, a different key management flow, different rate limit behavior — is enough overhead that most teams don't do it until after an outage has already cost them. The result is that redundancy is treated as a post-incident fix rather than a default.

1.3 Opaque risk in the contracts being called

RPC access solves how a request reaches the chain, but says nothing about what a developer is calling. Teams routinely interact with contracts — their own, a counterparty's, a new integration — without a fast, structured way to check for the small set of patterns that account for most preventable losses: unrestricted mint functions, missing reentrancy guards, ownership that was never renounced, liquidity that was never locked.

Design principle: each of these is a problem of trusting a single source — a single node, a single integration, a single team's unverified claims about their own contract. Zyrpc's architecture is built around removing that single source wherever the cost of doing so is reasonable.

2. RPC routing architecture

The core of Zyrpc is a routing layer that sits between a developer's application and a pool of underlying RPC nodes — a mix of nodes Zyrpc operates directly and capacity sourced from established infrastructure providers. The design goal is simple to state and nontrivial to execute well: a single request should reach the healthiest available node for the target chain, and a node going from healthy to degraded should be invisible to the application making the request.

2.1 Request lifecycle

1. A request arrives at a single Zyrpc endpoint, scoped to one chain and one API key.
2. The routing layer consults a continuously updated health table for that chain — per-node latency (p50/p95), recent error rate, and block-height lag relative to the chain's current tip.
3. The request is forwarded to the highest-ranked healthy node. If that node fails to respond within a defined timeout window, the request is retried against the next-ranked node, transparently to the caller.
4. The response is returned to the application in the same JSON-RPC shape it would receive from a single node directly — no custom wrapper, no proprietary response format.

2.2 Health checking

Each node in the pool is checked on two independent tracks:

- Liveness: a lightweight, low-cost call (e.g. eth_blockNumber) run on a short interval to confirm the node is responding at all.
- Correctness: block-height comparison against a quorum of other nodes for the same chain, to catch a node that responds quickly but is stale or has fallen out of sync — a failure mode that liveness checks alone do not catch.

A node that fails either check is removed from the active rotation immediately and re-admitted only after it passes a recovery threshold over a defined observation window, to avoid flapping a marginal node in and out of rotation under load.

2.3 Failover targets

The target for failover latency — the time between a node becoming unhealthy and traffic being fully routed away from it — is under 200 milliseconds for liveness failures. This is a target the architecture is designed to meet, not a number drawn from production telemetry yet; Section 8 states current status plainly.

2.4 Why this is harder than a simple load balancer

A generic load balancer distributes load. Zyrpc has to additionally reason about chain-specific correctness — a node can be “up” in the networking sense while returning state that is several blocks behind, which is a correct response to the wrong question. The routing layer's health model

treats these as distinct failure classes because the right response differs: a liveness failure means stop sending traffic; a sync-lag failure means deprioritize but keep available for read calls where slight staleness is tolerable, and exclude entirely from calls (like eth_sendRawTransaction or anything depending on current nonce) where it isn't.

2.5 Multi-chain key model

A single API key is valid across every supported chain; the chain is selected by URL path, not by provisioning a separate key per chain. This is a product decision as much as a technical one — it removes a step that has no real security benefit (a leaked key scoped to one chain is not meaningfully safer than a multi-chain key with the same usage limits) but adds real integration friction.

2.6 What Zyrpc does not do

Two scope boundaries, stated explicitly because they shape what teams should and shouldn't expect:

- Zyrpc does not operate as a validator or participate in consensus for any chain it provides RPC access to. It is a routing and access layer, not a staking or validation service.
- Zyrpc does not hold, custody, or have signing authority over any user's funds or private keys at the RPC layer. Requests are proxied; signing happens client-side, as with any standard RPC integration.

3. Planned: MEV-protected transaction routing

> **Status: planned, not yet available**
>
> Everything in this section describes a planned addition to the routing layer. It is included here because it follows directly from the architecture in Section 2 and shapes the roadmap in Section 8, not because it is live today.

Maximal extractable value (MEV) — most visibly, sandwich attacks against pending swaps — is a cost that standard public mempools expose by design: a transaction broadcast to a public mempool is visible to searchers before it's included in a block, which is what makes front-running and sandwiching possible in the first place.

3.1 Approach: route, don't operate

Zyrpc's planned approach is to offer an opt-in endpoint that routes transactions through private or protected mempools — infrastructure already operated by established players in this space — rather than building and operating a competing private mempool from scratch. The reasoning is the same non-custodial principle that shapes the rest of the product: Zyrpc is well-positioned to be a reliable, well-integrated routing layer in front of MEV protection infrastructure; it is not well-positioned, as a small team, to independently operate the searcher relationships and block-builder integrations that make private mempool infrastructure effective. Routing to proven infrastructure gets users real protection sooner and more reliably than reinventing it would.

3.2 What this will and will not change

- Will change: the path a signed transaction takes from submission to block inclusion, when a developer opts into the protected endpoint.
- Will not change: custody. Transactions are still signed client-side before they reach Zyrpc; this is a routing decision, not a custodial one, consistent with the boundary stated in Section 2.6.

3.3 Why this is scoped as routing, not as a trading service

An earlier direction considered for Zyrpc was a custodial bot service — holding user funds and executing trades with profit-sharing. That direction was deliberately ruled out. Custodying user funds and promising a share of returns functions as an investment or fund-management activity in most jurisdictions, with licensing and compliance obligations a small infrastructure team is not positioned to meet responsibly. MEV-protected routing avoids this entirely: Zyrpc never holds funds, never executes trades on a user's behalf, and never makes a return-on-investment claim. The product is access to better infrastructure, priced like infrastructure.

4. Security tooling

> **Status: planned**
> Described here at the level of approach and scope, not implementation detail that doesn't exist yet. See Section 8 for current build status.

Zyrpc's security tooling is two related but distinct tools, both automated, both explicitly scoped as a first filter rather than a final answer.

4.1 Contract scanning

Static analysis run against a Solidity codebase, built on established open-source analysis tooling rather than a proprietary detection engine built from scratch — the value Zyrpc adds is in tuning, false-positive reduction, and presentation, not in reinventing static analysis. The scan targets the pattern classes that account for the large majority of real-world exploit post-mortems:

- Reentrancy and unprotected external calls
- Access control gaps (missing or incorrect onlyOwner-equivalent guards)
- Unchecked arithmetic and integer overflow/underflow in older compiler targets
- Unsafe delegatecall usage
- Uninitialized or unprotected proxy implementation slots

Findings are ranked by severity and mapped to the specific line and function, with a plain-language explanation of why the pattern matters — written for a developer who needs to act on the finding, not for a security researcher who already knows the vocabulary.

4.2 Token and contract risk scoring

A second tool, oriented at evaluating a contract someone else deployed rather than one's own codebase. Given a deployed address, it produces a risk score from a defined, inspectable set of signals: minting and pausing permissions still held by an externally-owned account, liquidity lock status where applicable, ownership renouncement status, and pattern-matching against known honeypot and rug-pull contract templates.

The score is a starting point for due diligence, not a verdict. It is built to be transparent about what it checked and what it didn't — a contract can score well on every automated signal and still carry risk the tooling has no way to see, such as off-chain promises the team has no on-chain obligation to keep.

4.3 The honest boundary

> **What automated scanning is not**
> Automated scanning catches known, checkable patterns quickly and consistently. It is not a

> certified audit, and a clean scan result is not a guarantee that a contract is free of vulnerabilities. Teams shipping a contract that will hold meaningful value should treat Zyrpc's scanning as a fast first pass — useful for catching the common mistakes early and cheaply — and budget for a manual audit by a security firm before a mainnet launch with real funds at stake. This boundary is stated plainly on the product itself, not just in this document.

5. Research and on-chain reporting

> **Status: planned**
>
> Same status note as Section 4 — described at the level of approach, not finished implementation.

The research layer exists to answer a specific, narrow question well: for a given project, what does the chain itself actually show, independent of what the project's marketing says. Three categories of report, each built from on-chain and publicly available data rather than self-reported figures:

5.1 Supply and distribution

Circulating versus total supply, vesting schedules where they're encoded on-chain or in publicly verifiable contracts, holder concentration, and emission curves — checked against what the project's own documentation claims, with discrepancies flagged rather than silently reconciled.

5.2 Team and deployment history

Wallet history for deployer and treasury addresses: prior projects associated with the same addresses where that history is traceable on-chain, and patterns in how and when contracts were deployed. This is necessarily limited to what's visible on-chain — it is not an identity-verification service and does not claim to unmask anonymous teams.

5.3 On-chain activity versus reported activity

Transaction volume, unique active wallets, and liquidity depth pulled directly from chain data, presented alongside (and sometimes in contrast to) the activity numbers a project publishes itself.

5.4 What this is not

> **Not financial advice**
>
> Research reports are informational. They describe what the data shows; they do not recommend buying, selling, or holding anything. This is stated on every report Zyrpc produces, not only in this document, because the line between “here is what the data shows” and “here is what you should do” is exactly the line a research product has to stay on the correct side of.

6. Where Zyrpc fits

The RPC infrastructure market is established, not new — providers like Alchemy, Infura, QuickNode, Chainstack, and others operate mature, well-resourced platforms with broad chain coverage and enterprise customers. Zyrpc is not positioned as a replacement for that category. It is positioned as a routing layer that sits in front of it, and a small set of adjacent tools that share its customers' actual workflow.

6.1 Why route rather than compete on raw node infrastructure

Operating a global fleet of archive nodes across eight or more chains at the reliability and latency that established providers already deliver is a multi-year, capital-intensive undertaking. Routing intelligently across that existing capacity — including capacity from multiple providers — is a problem a focused team can solve well without re-deriving infrastructure the market has already built. This is the same reasoning applied in Section 3.1 to MEV protection: build the layer where a small team can credibly out-execute, route to proven infrastructure everywhere else.

6.2 The adjacency between RPC, security, and research

These three services are not a random product bundle. A team building on-chain typically needs all three at different points in the same workflow: reliable access to read and write chain state (RPC), a fast check on contracts before interacting with them (security scanning), and a way to verify a counterparty project's claims before committing capital or integration effort (research). Bundling them under one account and one API surface removes friction that exists today only because these tools are usually built by separate companies with separate signups, separate billing, and no shared context between them.

7. Operating principles

Three commitments that apply across every part of the product described in this document, stated explicitly because they constrain what Zyrpc will and won't build, not just what it currently offers.

7.1 Non-custodial by design

Zyrpc does not hold user funds or private keys at any layer — not in RPC routing, not in MEV-protected transaction submission, not anywhere in the roadmap. A custodial, profit-sharing trading product was considered early and explicitly ruled out for the reasons described in Section 3.3. This is a permanent boundary, not a current-stage limitation.

7.2 Automated tools are labeled as automated tools

Section 4.3 and Section 5.4 each state a limitation plainly rather than implying more certainty than the tooling can support. This document applies the same standard to itself: claims about architecture not yet built are marked as planned, not described as if they were already running in production.

7.3 Verifiable over self-reported

Where Zyrpc can ground a claim in on-chain data instead of an assertion — in the research layer, in security scoring — it does. The research layer's entire reason to exist (Section 5) is to substitute checkable data for marketing claims; the same standard applies to what Zyrpc says about itself.

8. Status and roadmap

Stated plainly, with no rounding up: as of this writing, the RPC routing architecture described in Section 2 is in design and early implementation. The security and research tooling described in Sections 4 and 5 are in scoping. The MEV-protected routing described in Section 3 is planned for after the core RPC product is live and stable. Nothing in this document should be read as a claim about what is running in production today — Sections 2 through 5 describe the system being built, and this section describes how close each part is.

Phase	Scope	Status
Phase 1	Core RPC routing across initial chain set; health checking; failover; API key and dashboard	In progress
Phase 2	Expanded chain coverage; WebSocket subscriptions; archive access; usage analytics	Planned
Phase 3	Contract scanning tool (Section 4.1)	Planned
Phase 4	Token/contract risk scoring (Section 4.2)	Planned
Phase 5	Research reporting (Section 5)	Planned
Phase 6	MEV-protected transaction routing (Section 3)	Planned

8.1 What changes as phases ship

Each phase that ships will be reflected in an updated version of this document and in the product's own status page — not retroactively rewritten into earlier sections as though it had always been live. The version history exists so that a reader can see what changed and when.

9. Closing note

Most of what's described in this document is, on its own, not a novel idea. Multi-provider RPC routing, automated contract scanning, and on-chain research tooling all exist elsewhere, built by capable teams. What Zyrpc is betting on is narrower: that a developer's actual workflow touches all three, that there's real friction in stitching together separate tools from separate vendors to cover it, and that a small, focused team building all three with the same operating principles — non-custodial, honest about automation's limits, grounded in verifiable data — can serve that workflow better than the sum of its separate parts.

This document will be revised as the system described in it gets built. Readers are encouraged to check the version date and the status table in Section 8 rather than assume any capability described here is live without confirming it.

---

zyrpc.com