XHAVIC — Technical Whitepaper v1.0

Abstract

Ethereum can handle about 15 transactions per second. When gas spikes during a token launch or an NFT drop, fees climb past $100 and half the network's users get priced out.

Xhavic is a Layer-2 execution network that sits on top of Ethereum. We take the computation off-chain, batch the results, and settle them back on Ethereum's mainnet. The architecture breaks down into six independent layers — execution, sequencing, settlement, data availability, oracle infrastructure, and governance — and each one can be upgraded without touching the others.

The numbers: over 2,000 transactions per second, average cost of $0.04, and end-to-end latency under 200 milliseconds. Full EVM bytecode compatibility, so existing Solidity and Vyper contracts work without any changes.

Two things make Xhavic different from everything else out there. First, the **Dual Wallet System**. It splits transactions into two paths at the protocol level: an Instant Wallet for composable DeFi (fast, irreversible) and a Secured Wallet with a 24–hour reversal window for high-value treasury operations. This isn't a UI toggle. It's baked into the sequencer. Second, a **native oracle and AI-agent layer** that provides on-chain data feeds and supports autonomous execution without bolting on third-party middleware after the fact.

---

1. Introduction

Ethereum changed what blockchains could do. Smart contracts turned a digital ledger into a programmable platform, and that platform now secures hundreds of billions of dollars in value. The ecosystem works. The problem is that it doesn't scale.

Fifteen transactions per second. That's the ceiling. When the network gets busy — and it gets busy often — gas prices spike, confirmation times stretch, and users who can't afford $50 for a simple transfer are shut out. This isn't a future problem. It's happening right now, every day, and it's holding back entire categories of applications that should be running on-chain but can't.

We built Xhavic to fix this specific problem. Transactions happen on our execution layer. Results get compressed and committed to Ethereum as state roots. Ethereum handles the finality and the security guarantees. We handle the speed.

What we're bringing to the table that the current crop of L2s doesn't have:

- **Dual Wallet System.** Two separate execution paths at the protocol level. One for speed-critical DeFi (irreversible, composable). One for high-value operations with a 24–hour reversal window. The sequencer routes them differently. A $50 DEX swap and a $5M treasury move shouldn't go through the same pipeline.
- **AI-native execution.** Autonomous agents get first-class treatment — deterministic ordering guarantees, direct oracle precompile access, and programmatic wallet control. No off-chain coordination needed.
- **RWA-ready architecture.** Compliance-friendly by design. Supports tokenized assets, institutional settlement, and enterprise workflows alongside permissionless DeFi.
- **Honest performance claims.** 2,000+ TPS, sub-200ms latency, $0.04 average fee. These are testnet measurements, not theoretical projections. Current architecture caps at around 12,000 TPS; sharding takes it to 120,000+.

2. Background: Ethereum and Layer-2 Scaling

2.1 The Scalability Problem

Ethereum runs on a monolithic design. Every full node processes every transaction. Execution, consensus, and data availability are all bundled together. This is great for decentralization and security. It's terrible for throughput.

A $20 gas fee is tolerable if you're swapping $50,000 on Uniswap. It's completely impractical if you're buying a $3 in-game item, updating a supply chain record, or running an automated trading bot that makes fifty transactions an hour. The cost structure shuts out most of the applications that would actually benefit from being on-chain.

2.2 Layer-2 Solutions

The idea behind L2 is simple: move the heavy computation somewhere else, keep the security where it already is. Execute transactions on a separate chain, compress the results, post cryptographic proofs back to Ethereum, let anyone verify that nothing went wrong.

Xhavic is an optimistic rollup. That means we assume transactions are correct unless someone proves otherwise. Fraud proofs handle the edge cases. This keeps gas costs down during normal operation while preserving the ability to catch and punish bad actors.

3. Design Philosophy

Five principles guide every design decision we make. They're not marketing copy — they're actual constraints we hold ourselves to.

Ethereum alignment. Full EVM bytecode compatibility. If your contract runs on Ethereum, it runs on Xhavic. Same tooling, same compiler output, same RPC calls. We're not asking anyone to learn a new stack.

Security inheritance. Every state transition settles on Ethereum. We don't ask you to trust a new validator set or a novel consensus mechanism. Xhavic's security is Ethereum's security. Period.

Modularity. Six layers, each independently upgradeable. We can overhaul the oracle system without touching the execution engine. We can swap out the data availability strategy without disrupting settlement.

Progressive decentralization. The sequencer starts permissioned. A freshly launched network needs reliability more than it needs decentralized block production. The architecture supports the transition to multi-sequencer stake-based operation, and Phase II delivers it.

Practical focus. We're building for identified workloads: trading platforms, oracle-dependent contracts, automated strategies, institutional settlement. Every design choice gets tested against "does this actually help the people building on us?"

4. Architecture Overview

The protocol breaks into six layers. Each one does one thing well. They connect through well-defined interfaces, and any layer can be upgraded independently.

1. **Execution Layer** EVM-compatible processing
2. **Sequencer Layer** Ordering & MEV mitigation
3. **Settlement Layer** Ethereum L1 finality
4. **Data Availability Layer** Calldata + redundancy
5. **Oracle & AI Agent Layer** Native data feeds + agents
6. **Governance Layer** DAO-controlled upgrades

Figure 1 — Xhavic modular protocol architecture. Each layer operates independently.

5. Core Protocol Components

5.1 Execution Layer

This is where transactions actually get processed. We run a bytecode-compatible EVM — same opcodes, same gas semantics, same everything. If it compiles for Ethereum, it runs here without modification.

OPERATION	L1 COST (GAS)	L2 COST (GAS)	REDUCTION
SSTORE	20,000	500	**40×**
SLOAD	2,100	50	**42×**
Transaction base	21,000	300	**70×**
CREATE2	32,000	800	**40×**

Table 1 — Gas cost comparison: Ethereum L1 vs Xhavic L2

The execution engine uses a 256-bit word size with a 1024 stack depth limit, dynamic memory allocation with quadratic cost scaling, and 32-byte storage slots verified through Merkle proofs. Oracle data is available through dedicated precompile addresses in the `0x00...00F0-0xFF` range.

5.2 Sequencer

The sequencer collects transactions, orders them fairly, executes them, generates state roots, and submits batches to Ethereum. It's the workhorse of the protocol.

PARAMETER	VALUE	DESCRIPTION
Block time	2 seconds	Time between block proposals
Max block size	5 MB	Compressed data limit per block
Batch window	60 seconds	Accumulation period before L1 submission
Transaction timeout	24 hours	Maximum time for guaranteed inclusion

Table 2 — Sequencer configuration parameters

During Phase I, sequencers are permissioned for stability. Phase II transitions to a stake-based multi-sequencer model with governance-controlled admission.

5.3 Validators

Validators watch the sequencer. They independently execute transactions, compare results, and challenge any discrepancies through fraud proofs. The top 100 by stake are eligible, provided they maintain 95% uptime over a rolling 30-day window.

VIOLATION	PENALTY
Offline for one epoch (2 hours)	5% of stake
Invalid fraud proof submission	25% of stake
Double-signing or state root collision	**100% of stake**

Table 3 — Validator slashing schedule

5.4 Data Availability

Compressed transaction data gets posted to Ethereum calldata. Validators keep redundant off-chain copies. Full state reconstruction from L1 data takes roughly 5–10 minutes if it's ever needed. This prevents data withholding attacks and keeps the system transparent.

5.5 Settlement

State roots are committed to Ethereum through the CanonicalTransactionChain smart contract. Once Ethereum confirms them, those transactions have the same finality as any native Ethereum transaction.

6. Technical Deep Dive

6.1 State Model

We use Ethereum's account model. EOAs hold user balances; contract accounts store code and state. The global state tracks balances, contract storage, nonces, and code hashes.

```
StateRoot(n) = Keccak256(RLP(Accounts || Contracts || Storage))
```

6.2 State Transitions

Every block applies a deterministic function:

```
S(n+1) = Apply(S(n), T(n))
```

Same inputs, same outputs, every time. That's what makes fraud proofs work — anyone can re-execute a disputed transaction and prove whether the published result was correct.

6.3 Blocks and Finality

Each sequencing round collects pending transactions, applies fair ordering rules, executes them, produces a state root, and commits it to Ethereum. Soft finality arrives in 2–5 seconds. Full Ethereum finality takes about 13 minutes.

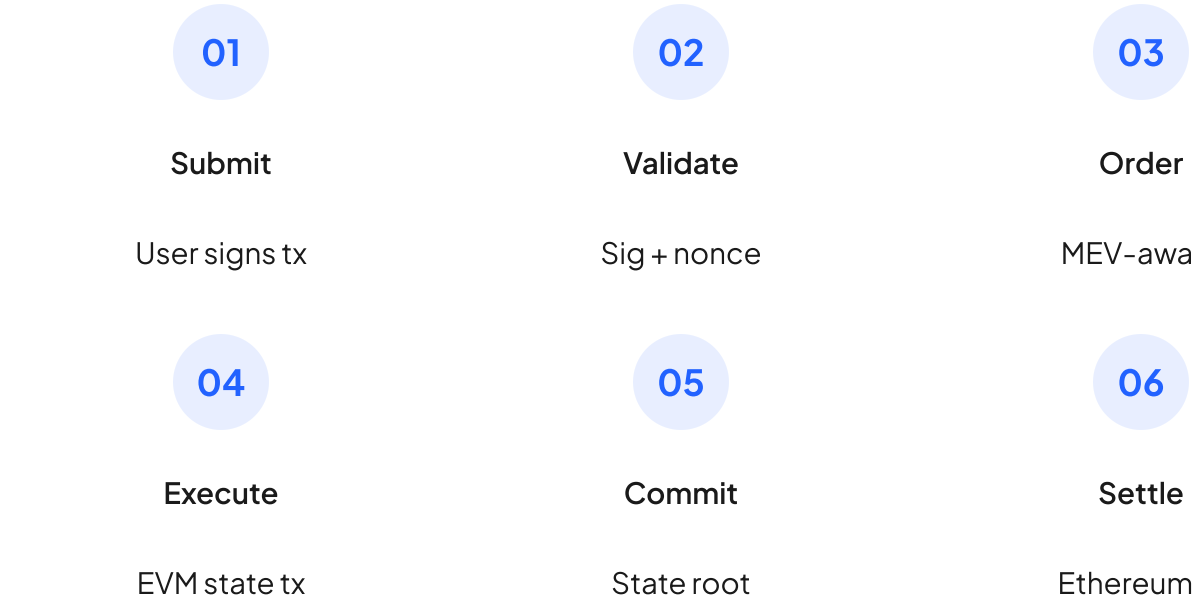
6.4 State Commitment

We use a modified Merkle-Patricia trie. Account proofs are O(log n). Storage slot proofs are O(log m). RLP encoding cuts proof size by about 40%. Batch commitments stack state roots in a Merkle tree, so validators can verify efficiently without redoing all the math.

---

7. Transaction Processing

7.1 L2 Transaction Flow



~200ms end-to-end · 2–5s soft finality · ~13 min Ethereum finality

User signs, sequencer validates (signature, nonce, balance), orders the transaction with MEV-aware rules, executes it on the EVM, generates a state root, and submits the batch to Ethereum.

Censorship resistance. If the sequencer blocks your transaction, you can submit it directly to Ethereum L1. The sequencer then has to include it within a defined window or get slashed. It costs about 10× more than the normal path, but nobody can permanently censor you.

7.2 Cross-Chain Transactions

Deposits (L1 → L2). Send assets to the bridge contract on Ethereum. The contract locks them and queues a minting instruction for L2. ETH becomes WETH (ERC-20) on our chain — this helps with replay protection. Typically done within minutes.

Withdrawals (L2 → L1). Burn your L2 tokens, wait for the batch to post on Ethereum, create a Merkle inclusion proof, then wait the 7-day challenge period. Or use a liquidity provider who'll pay you on

L1 immediately and collect from the bridge when the window closes.

7.3 Data Efficiency

The sequencer batches calldata from consecutive transactions and delivers them to the CanonicalTransactionChain contract. The contract builds a Merkle tree from the transaction hashes and stores only the root on-chain. This is dramatically cheaper than storing each transaction individually.

8. Dual Wallet System

Every other L2 treats every transaction the same way. Same pipeline, same finality model, same risk profile. But a $50 token swap and a $5 million treasury transfer have nothing in common. One needs speed. The other needs safety. Routing them through identical infrastructure is lazy design.

The Dual Wallet System separates these at the sequencer level. When a transaction arrives, the sequencer checks which wallet originated it and routes accordingly. This is a protocol-level mechanism, not a wrapper or an application feature.

Instant Wallet

1. 1. Submit transaction
2. 2. Validate (sig, nonce, balance)
3. 3. MEV-aware ordering
4. 4. Execute on EVM
5. 5. Soft confirmation (< 200ms)
6. 6. L1 batch settlement

IRREVERSIBLE · COMPOSABLE

DeFi · Trading · AI Agents · Automation

Secured Wallet

1. 1. Submit transaction
2. 2. Route to escrow
3. 3. Execute on EVM
4. 4. 24-hour escrow hold
5. 5. Finalize (no reversal)

6. 6. L1 batch settlement

REVERSIBLE · NON-COMPOSABLE

Treasury · Institutional · RWA · High-value

Figure 4 — Dual Wallet execution paths

8.1 Instant Wallet

Built for speed. DEX trades, perpetuals, liquidation triggers, arbitrage, bot operations — anything where you need confirmation in under 200ms and the result needs to be immediately composable. Other contracts can reference the output within the same block.

There's no reversal option. That's a feature, not a bug. If a transaction could be reversed, every contract that built on top of it would break. Composability requires finality.

8.2 Secured Wallet

Designed for operations where the cost of a mistake outweighs the inconvenience of waiting. Transactions execute on the EVM normally, but the result sits in a 24-hour cryptographic escrow. During that window, the sender can request a reversal (authenticated by multi-sig or threshold signature). If nobody triggers a reversal, it finalizes and settles in the next L1 batch.

PROPERTY	INSTANT WALLET	SECURED WALLET
Confirmation	< 200ms soft finality	24-hour escrow window
Reversibility	None	User-initiated within 24h
Composability	Full — same-block	Non-composable until finalized
Target use case	DeFi, trading, AI agents	Treasury, institutional, RWA
Settlement	Standard L1 batch	L1 batch post-escrow

Table 4 — Execution profile comparison: Instant vs Secured

---

9. L1/L2 Interoperability

9.1 Asset Movement

Getting in. Deposit ETH or ERC-20 tokens to the bridge contract on Ethereum. The contract relays a minting instruction. Equivalent tokens appear at your L2 address within minutes. If the sequencer drags its feet, you can force-include through L1 directly.

Getting out. Standard withdrawal takes seven days (the fraud proof window). Initiate on L2, wait for the batch to post on Ethereum, submit a Merkle proof, wait out the challenge period. Or use an LP for instant liquidity: they pay you on L1 right away and collect from the bridge after the window closes.

9.2 EVM Compatibility

Bytecode-level equivalence with the Ethereum Yellow Paper spec. Your contracts, your tools, your libraries — they all work. Hardhat, Foundry, ethers.js, web3.py, Remix. No new SDKs to learn, no compiler flags to set.

9.3 Cross-Chain Contract Calls

EOAs and contracts can interact across chains using the bridge contracts for message passing. These calls are asynchronous: you initiate on one chain, execution happens later on the other. L1 → L2 calls complete in minutes. L2 → L1 calls wait for the challenge period.

---

10. Security Architecture

10.1 Security Model

Figure 6 — Defense-in-depth security model

The model is layered, and each layer reinforces the others. At the outermost ring, Ethereum's proof-of-stake consensus provides settlement finality. Inside that, fraud proofs catch invalid state transitions within a 7-day window. Validators actively monitor the sequencer. Deterministic execution means anyone can independently verify the math. Standard cryptographic primitives handle the mathematical guarantees.

10.2 Fraud Proofs

The system only needs one honest participant to stay secure. If even one validator catches an invalid state root and submits a fraud proof, the bad state gets rejected and the submitter gets slashed. The bisection protocol narrows a dispute down to a single EVM instruction, which gets verified on-chain.

10.3 Threat Mitigation

Sequencer censorship. Can't permanently happen. Users submit directly to L1, and the sequencer must include those transactions within a defined window or face rotation and slashing.

State root manipulation. Validators independently compute state roots. Mismatches trigger fraud proofs and 100% slashing of the malicious sequencer's stake.

Data withholding. Compressed data is posted to Ethereum calldata with Merkle commitments. Full state reconstruction from L1 takes 5–10 minutes.

COMPONENT	ALGORITHM	SECURITY LEVEL	NOTES
Hash function	Keccak-256	256-bit	Identical to Ethereum
Signatures	ECDSA (secp256k1)	128-bit	EVM standard
Encryption	ChaCha20-Poly1305	256-bit	Private transaction support
State commitment	Merkle-Patricia Trie	256-bit	Modified for batch proofs
Dispute resolution	Interactive fraud proofs	Game-theoretic	Bisection protocol

Table 5 — Cryptographic primitives

11. MEV and Fair Ordering

MEV is the money block producers can extract by reordering, inserting, or censoring transactions. It's a tax on regular users — front-running and sandwich attacks cost DeFi participants billions annually. We address it at the protocol level:

- **Threshold encryption.** Transactions stay encrypted until the sequencer commits to an inclusion order. You can't front-run what you can't read.
- **Time-weighted ordering.** Priority is a function of submission time plus gas price. First in line actually matters.
- **Oracle sequencing.** Price feeds get committed before user transactions execute. This prevents oracle manipulation attacks.
- **Transaction batching.** Grouping transactions into atomic batches reduces the surface area for ordering manipulation.
- **Governance oversight.** The community controls sequencing parameters through the DAO. Rules evolve as new attack vectors emerge.

---

12. Oracle & AI Agent Infrastructure

12.1 Hybrid Oracle Architecture

The oracle layer isn't bolted on — it's built into the protocol. Oracle nodes pull data from multiple external sources (exchange APIs, Chainlink, Band Protocol, custom feeds), normalize the formats, run outlier detection, and compute weighted averages based on reliability scores. The aggregated data lives on-chain through precompile addresses (`0x00...00F0-0xFF`). Smart contracts read it directly. No external contract calls, no additional gas overhead.

12.2 AI Agent Execution

Autonomous agents have specific requirements that most L2s ignore: they need deterministic ordering guarantees, direct data access, and programmatic wallet control.

CAPABILITY	WHAT IT MEANS
Deterministic execution	Predictable ordering and inclusion — agents can plan multi-step strategies
Native oracle access	Direct precompile reads, no external contract dependencies or extra gas
Programmatic wallets	API-first control for automated strategy execution and portfolio management
Gas-metered compute	Predictable cost model so agents can budget their operations autonomously
Sub-200ms latency	Fast enough for real-time decision loops and reactive trading

Table 6 — AI agent execution capabilities

Scope note: The protocol provides execution guarantees. Strategy design, decision logic, and multi-agent coordination are application-layer concerns.

13. Governance

Governance transitions progressively to a DAO. Token holders vote on four categories of protocol decisions:

Protocol Upgrades

Sequencer Policies

Oracle Parameters

Treasury Allocation

All decisions execute on-chain — no off-chain multisig backdoors. The mechanism is deliberately conservative: we'd rather move slowly than break things.

14. Performance & Scalability

14.1 Throughput

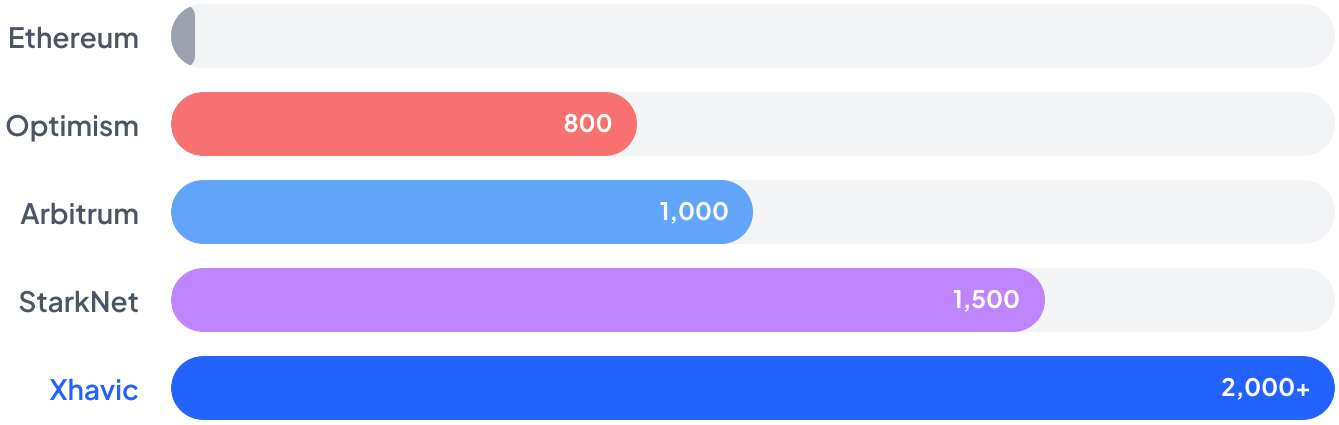


Figure 10 — Peak throughput comparison (testnet benchmarks, January 2026)

NETWORK	PEAK TPS	AVG LATENCY	FINALITY	AVG FEE
Xhavic L2	**2,000+**	**200ms**	**2–5s**	**$0.04**
Ethereum L1	15	12s	13 min	$20
Arbitrum One	1,000	400ms	1 week	$0.08
Optimism	800	2s	1 week	$0.10
StarkNet	1,500	200ms	1 week	$0.06

Table 7 — Network performance comparison

14.2 Cost Analysis

OPERATION	ETHEREUM	XHAVIC	SAVINGS
Simple transfer	$20	$0.04	**500×**
Token swap	$80	$0.12	**667×**
NFT mint	$150	$0.08	**1,875×**
Contract deployment	$500	$1.50	**333×**

Table 8 — Per-operation cost comparison

14.3 Scalability Path

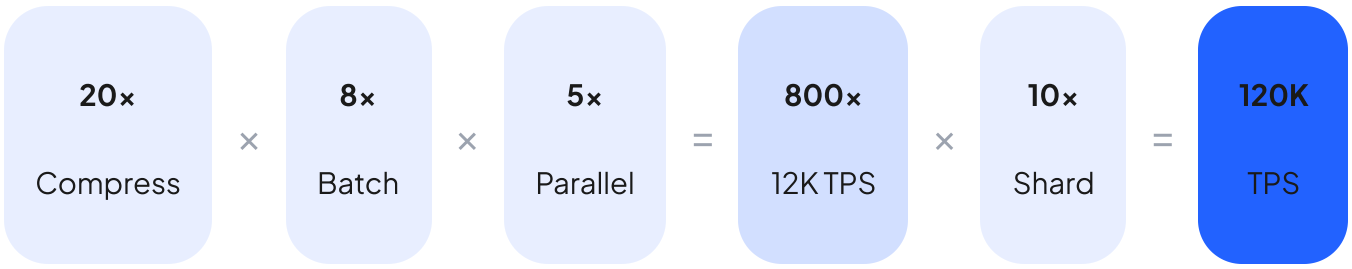


Figure 11 — Compound scalability multipliers

14.4 Infrastructure

NODE TYPE	CPU	RAM	STORAGE	MONTHLY COST
Full node (sequencer)	16 cores, 3.5+ GHz	64 GB	2 TB NVMe	$800–$1,200
Archive node (validator)	8 cores, 2.5+ GHz	32 GB	4 TB HDD	$200–$400
Light client (RPC)	4 cores	8 GB	100 GB	$50–$100

Table 9 — Infrastructure requirements by node type

15. Competitive Analysis

The L2 market is crowded and well-funded. Arbitrum, Optimism, StarkNet, zkSync — these are serious projects with real teams and real adoption. There are specific things we do that nobody else does.

FEATURE	XHAVIC	ARBITRUM	OPTIMISM	STARKNET	ZKSYNC
Peak TPS	**2,000+**	1,000	800	1,500	1,200
Native oracle	**Yes**	No	No	No	No
Dual wallet	**Yes**	No	No	No	No
MEV mitigation	**Protocol-level**	Partial	Partial	None	Partial
EVM compat	Bytecode	Bytecode	Bytecode	Transpiled	Bytecode
Proof system	Fraud proofs	Fraud proofs	Fraud proofs	ZK (STARK)	ZK (SNARK)
AI agent support	**Native**	None	None	None	None
Avg fee	**$0.04**	$0.08	$0.10	$0.06	$0.07

Table 10 — Competitive feature matrix

16. Benefits

Scalability. Over 2,000 TPS with sub-200ms latency. That's enough for high-frequency trading, live DeFi, and automated agent operations. Architecture supports growth to 120,000+ through sharding.

Cost efficiency. $0.04 average fees mean you can actually build micro-transaction economies, gaming systems, and high-frequency automation on-chain without going bankrupt on gas.

Security. Everything settles on Ethereum. Fraud proofs, validator slashing, and forced inclusion provide guarantees without adding new trust assumptions.

Interoperability. Bytecode-level EVM compatibility. Your contracts work. Your tools work. Your team's existing Ethereum knowledge transfers completely.

---

17. Use Cases

DeFi. DEXs, lending, yield optimization. The Instant Wallet's composability lets you build complex multi-contract strategies that execute within a single block. Low fees make small positions viable.

Gaming. In-game economies, item trading, NFTs. At $0.04 per transaction, on-chain game mechanics actually make economic sense for the first time.

High-frequency trading. Sub-200ms confirmation, deterministic ordering, protocol-level MEV mitigation. An execution environment actually designed for algorithmic trading.

Institutional and RWA. The Secured Wallet's reversal window combined with compliance-ready architecture supports tokenized real-world assets, treasury management, and institutional settlement.

Supply chain and IoT. Low costs and native oracle integration make it practical to track supply chain events and coordinate IoT devices on-chain at scale.

---

18. Developer Guide

If you've built on Ethereum, you already know how to build on Xhavic. The RPC interface exposes standard methods: `eth_sendTransaction`, `eth_call`, `eth_getBalance`, `eth_getLogs`, plus everything else in the Ethereum JSON-RPC spec. We add a few methods for oracle data and cross-chain messaging, but the core workflow is identical.

Deploy with Hardhat or Foundry. Test on our testnet (faucet available). Interact with ethers.js or web3.py. Monitor through the block explorer.

18.1 Supported RPC Methods

CATEGORY	METHODS
Account	`eth_getBalance`, `eth_getTransactionCount`, `eth_getCode`
Transaction	`eth_sendTransaction`, `eth_sendRawTransaction`, `eth_getTransactionReceipt`
Block	`eth_getBlockByNumber`, `eth_getBlockByHash`, `eth_blockNumber`
State	`eth_call`, `eth_estimateGas`, `eth_getStorageAt`
Events	`eth_getLogs`, `eth_subscribe`, `eth_unsubscribe`
Oracle (Xhavic)	`xhv_getOraclePrice`, `xhv_getOracleFeeds`
Bridge (Xhavic)	`xhv_getDepositStatus`, `xhv_getWithdrawalProof`

Table 11 — Supported RPC methods

19. Development Roadmap

PHASE I

Foundation

Q1–Q2 2026

- Mainnet launch
- Bridge deployment
- Core DeFi integrations
- Block explorer
- Initial liquidity

PHASE II

Expansion

Q3–Q4 2026

- Multi-sequencer
- Governance DAO
- Oracle framework v2
- Developer SDK
- AI agent framework

PHASE III

Maturity

H1 2027

- Sharding

- ZK proof research
- Cross-L2 interop
- Institutional tooling
- Compliance modules

PHASE IV

Scale

H2 2027+

- 120,000+ TPS
- Full decentralization
- Enterprise adoption
- Global expansion
- Cross-chain hub

20. Challenges and Limitations

Data availability costs. Ethereum calldata is our main cost bottleneck. When Ethereum ships EIP-4844 and danksharding, we'll integrate those improvements and costs will drop further. Until then, calldata fees set the floor.

Withdrawal latency. The 7-day challenge period isn't a design flaw. It's the security mechanism that makes optimistic rollups work. LP-backed fast exits handle most use cases, and a future move to validity proofs (ZK) could eventually eliminate the delay.

Sequencer centralization in Phase I. We run a permissioned sequencer at launch. If it goes down, block production stops. Users can still force-include through L1, so funds aren't at risk, but it's a liveness dependency we're upfront about. Phase II fixes this with multi-sequencer design.

Bridge complexity. Moving assets between L1 and L2 is more complicated than transacting on a single chain. We're investing heavily in wallet integration and bridge abstraction to make this invisible to end users, but it's not there yet.

---

21. Risks and Legal Disclaimer

This whitepaper is published by the Xhavic Protocol Foundation for informational purposes only. It does not constitute an offer to sell, a solicitation to buy, or a recommendation of any security, token, investment product, or financial instrument in any jurisdiction.

The protocol and associated technology are under active development. What's described here reflects the design and roadmap as of January 2026. Things will change.

Participation in the Xhavic ecosystem involves significant risks, including but not limited to: total loss of funds from smart contract bugs, bridge failures, or exploits; regulatory actions restricting access; and market volatility.

The protocol is provided "as is" without warranty of any kind. Do your own research. Talk to a lawyer. Talk to a financial advisor.

---

22. Conclusion

Xhavic does a specific set of things, and we think it does them well. Over 2,000 TPS at $0.04 per transaction with sub-200ms latency, settled on Ethereum. Modular architecture where each component can be upgraded independently. Full EVM bytecode compatibility so nothing breaks when you migrate.

The Dual Wallet System is the headline feature. Two execution paths, separated at the protocol level, serving fundamentally different risk profiles within a single network. The Instant Wallet handles composable DeFi at speed. The Secured Wallet gives institutions the safety they need for high-value operations.

Native oracle integration and AI-agent execution support are our bet on where on-chain activity is headed. Autonomous systems interacting directly with decentralized infrastructure, not through duct-taped middleware.

The scaling path from 12,000 TPS today to 120,000+ through sharding is architecturally ready — Phase III delivers it. The challenges are real: DA costs, withdrawal latency, sequencer centralization during launch. We've been honest about every one of them.

Appendix: Technical Reference

A.1 Glossary

TERM	DEFINITION
EOA	Externally Owned Account — user-controlled account with a private key
EVM	Ethereum Virtual Machine — deterministic execution environment for smart contracts
Fraud Proof	Cryptographic evidence that a state transition was computed incorrectly
MEV	Maximal Extractable Value — profit from transaction reordering by block producers
Optimistic Rollup	L2 design where transactions are assumed correct unless challenged
RLP	Recursive Length Prefix — Ethereum's standard serialization format
State Root	Merkle root hash representing the complete state of the chain at a given block
WETH	Wrapped ETH — ERC-20 representation of ETH used on L2 for replay protection
Bisection Protocol	Interactive dispute resolution that narrows disagreements to a single step
Challenge Period	7-day window during which fraud proofs can be submitted for L2 → L1 operations
Precompile	Built-in contract at a fixed address providing optimized functionality
DAO	Decentralized Autonomous Organization — on-chain governance structure

A.2 Protocol Parameters Summary

PARAMETER	VALUE
Block time	2 seconds
Maximum block size	5 MB (compressed)
Batch window	60 seconds
Transaction timeout	24 hours
Challenge period	7 days
Escrow window (Secured Wallet)	24 hours
Validator set size	Top 100 by stake
Minimum validator uptime	95% (30-day rolling)
L1 state reconstruction time	~5–10 minutes
Oracle precompile range	`0x00...00F0-0xFF`
Hash algorithm	Keccak-256
Signature algorithm	ECDSA (secp256k1)
Encryption	ChaCha20-Poly1305
State encoding	RLP
State structure	Modified Merkle-Patricia Trie

A.3 Contract Addresses (Testnet)

CONTRACT	ADDRESS
Bridge (L1)	`0x...TBD`
CanonicalTransactionChain (L1)	`0x...TBD`
StateCommitmentChain (L1)	`0x...TBD`
FraudProofVerifier (L1)	`0x...TBD`
GovernanceDAO (L2)	`0x...TBD`
OracleRegistry (L2)	`0x...TBD`

A.4 References

1. Ethereum Yellow Paper — G. Wood, 2014. Formal specification of Ethereum.
2. EIP-4844: Proto-Danksharding — V. Buterin et al., 2022. Blob transaction spec.
3. "An Incomplete Guide to Rollups" — V. Buterin, 2021. Overview of rollup designs.
4. Arbitrum Nitro Whitepaper — Offchain Labs, 2022. Optimistic rollup reference.
5. Optimism Bedrock Specification — OP Labs, 2023. Modular rollup architecture.
6. "Flash Boys 2.0" — Daian et al., 2019. MEV and front-running on Ethereum.
7. ChaCha20-Poly1305 — RFC 8439. AEAD encryption specification.
8. Ethereum JSON-RPC Specification — ethereum.org documentation.

A.5 Document History

VERSION	DATE	CHANGES
1.0	January 2026	Initial publication — complete protocol specification

January 2026 · Xhavic Protocol Foundation